MICROPROCESADORES: TEORIA Y APLICACIONES

1981

Directorio de Profesores

1. M. EN C. MANUEL ESTEVEZ KUBLI
INVESTIGADOR
CENTRO DE INSTRUMENTOS
UNAM
MEXICO 20, D. F.
550 04 16   550 52 15 ext. 5013

2. M. EN C. ANGEL KURI MORALES (Coordinador)
SUBDIRECTOR
MICROMEX S.A.
ALDAMA 78
COYOACAN
MEXICO 21, D. F.
554 75 75   658 31 25

3. ING. MARIO RODRIGUEZ MANZANERA
INVESTIGADOR
IIMAS
UNAM
MEXICO 20, D. F.
550 52 15 ext. 4573

4. ING. JORGE MARCOS BLAZQUEZ PAREDES
INVESTIGADOR
CENTRO DE INSTRUMENTOS
UNAM
MEXICO 20, D. F.
550 04-16   550 52 15 ext. 5013

5. ING. MANUEL CORREA
DEPTO. DE INGENIERIA
MICROMEX S.A.
ALDAMA 78
COYOACAN
MEXICO 21, D. F.
554 75 75   554 24 74   658 31 25

6. ING. RAUL GOMEZ
DEPTO. DE INGENIERIA
MICROMEX S.A.
ALDAMA 78
COYOACAN
MEXICO 21, D. F.
554 75 75   554 24 74   658 31 25

MICROPROCESADORES: ORIA Y APLICACIONES
1981

FECHA	HORARIO	T E M A R I O	PROFESOR
lunes 13 de julio	17 a 21 h	1. RESUMEN DEL DESARROLLO DE MICROPROCESADORES - Sistemas digitales - Unidades Básicas de Hardware - Microprocesadores - La familia'80	M. en C. Angel Kuri Morales
martes 14 de julio	17 a 21 h	2. EL SISTEMA Z80 COMO EJEMPLO DE UN MICROPROCESADOR - Procesador central Ciclo básico de operación Arquitectura básica Fuente de poder Sistema mínimo	Ing. Mario Rodríguez Manzanera
miércoles 15 de julio	17 a 21 h	SOFTWARE - Acervo de instrucciones - Compatibilidad en el 8080	M. en C. Angel Kuri Morales
jueves 16 de julio	17 a 21 h	3. UNIDAD DE COMUNICACION SERIE - El SIO Operación Programación	Ing. Mario Rodríguez Manzanera
viernes 17 de julio	17 a 21 h	4. UNIDAD DE COMUNICACION PARALELA - El PIO Operación Programación	M. en C. Manuel Estevez Kubli

MICROPROCESADORES: ORIA Y APLICACIONES
1981

FECHA	HORARIO	T E M A R I O	PROFESOR
sábado 18 de julio	10 a 14 h	5. SESION PRACTICA USANDO Z80 - Programación y uso de un sistema microprocesador	Ing. Jorge Blazquez Paredes
lunes 20 de julio	17 a 21 h	6. LENGUAJE ENSAMBLADOR - Nemotécnicas - Pseudo-operaciones	Ing. Mario Rodríguez M.
martes 21 de julio	17 a 21	- Macros - Subrutinas	Ing, Manuel Correa
miércoles 22 de julio	17 a 21	7. SISTEMAS INTEGRADOS DE INGENIERIA - El canal S-100 - Interfaz serie/paralelo - Programación de ROM's - Sistemas de una tarjeta	Ing. Raúl Gómez
jueves 23 de julio	17 a 21 h	8. BASIC DE CONTROL - Uso y aplicaciones	M. en C. Angel Kuri Morales
viernes 24 de julio	17 a 21 h	9. SISTEMAS DE DESARROLLO - Compiladores - Intérpretes - Simuladores	M. en C. Manuel Estevez Kubli M. en C. Angel Kuri Morales Ing. Mario Rodríguez M. Ing. Manuel Correa Ing. Raúl Gómez
	21 h	CLAUSURA	

EVALUACION DEL PERSONAL DOCENTE

CURSO: MICROPROCESADORES:
TEORIA Y APLICACIONES

FECHA: DEL 13 AL 24 DE JULIO 1981

	CONFERENCISTA	DOMINIO DEL TEMA	EFICIENCIA EN EL USO DE AYUDAS AUDIOVISUALES	MANTENIMIENTO DEL INTERES. (COMUNICACION CON LOS ASISTENTES, AMENIDAD, FACILIDAD DE EXPRESION).	PUNTUALIDAD	
1.	M. EN C. MANUEL ESTEVEZ KUBLI					
2.	M. EN C. ANGEL KURI MORALES					
3.	ING. MARIO RODRIGUEZ MANZANERA					
4.	ING. JORGE MARCOS BLAZQUEZ PAREDES					
5.	ING. MANUEL CORREA					
6.	ING. RAUL GOMEZ					
7.						
8.						
9.						
	ESCALA DE EVALUACION : 1 a 10					

EVALUACION DE LA ENSEÑANZA

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

	TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
1	RESUMEN DEL DESARROLLO DE MICROPROCESADORES					
2	EL SISTEMA Z80 COMO EJEMPLO DE UN MICROPROCESADOR					
	SOFTWARE					
3	UNIDAD DE COMUNICACION SERIE					
4	UNIDAD DE COMUNICACION PARALELA					
5	SESION PRACTICA USANDO Z 80					
6	LENGUAJE ENSAMBLADOR					
7	SISTEMAS INTEGRADOS DE INGENIERIA					
8	BASIC DE CONTROL					
9	SISTEMAS DE DESARROLLO					
	ESCALA DE EVALUACION : 1 a 10					

EVALUACION DEL CURSO

CONCEPTO		EVALUACION
I	APLICACION INMEDIATA DE LOS CONCEPTOS EXPUESTOS	
2.	CLARIDAD CON QUE SE EXPUSIERON LOS TEMAS	
3.	GRADO DE ACTUALIZACION LOGRADO CON EL CURSO	
4.	CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
5.	CONTINUIDAD EN LOS TEMAS DEL CURSO	
6.	CALIDAD DE LAS NOTAS DEL CURSO	
7.	GRADO DE MOTIVACION LOGRADO CON EL CURSO	

ESCALA DE EVALUACION DE I A IO

1. ¿Qué le pareció el ambiente en la División de Educación Continua?

MUY AGRADABLE	AGRADABLE	DESAGRADABLE

2. Medio de comunicación por el que se enteró del curso:

PERIODICO EXCELSIOR ANUNCIO TITULADO DIVISION DE EDUCACION CONTINUA	PERIODICO NOVEDADES ANUNCIO TITULADO DIVISION DE EDUCACION CONTINUA	FOLLETO DEL CURSO

CARTEL MENSUAL	RADIO UNIVERSIDAD	COMUNICACION CARTA, TELEFONO, VERBAL, ETC.

REVISTAS TECNICAS	FOLLETO ANUAL	CARTELERA UNAM "LOS UNIVERSITARIOS HOY"	GACETA UNAM

3. Medio de transporte utilizado para venir al Palacio de Minería:

AUTOMOVIL PARTICULAR	METRO.	OTRO MEDIO

4. ¿Qué cambios haría usted en el programa para tratar de perfeccionar el curso?

____

____

____

5. ¿Recomendaría el curso a otras personas?

SI	NO

6. ¿Qué cursos le gustaría que ofreciera la División de Educación Continua?

____

____

7. La coordinación académica fue:

EXCELENTE	BUENA	REGULAR	MALA

8. Si está interesado en tomar algún curso intensivo ¿Cuál es el horario más conveniente para usted?

LUNES A VIERNES DE 9 A 13 H. Y DE 14 A 18 H. (CON COMIDAS)	LUNES A VIERNES DE 17 A 21 H.	LUNES, MIERCOLES Y VIERNES DE 18 A 21 H.	MARTES Y JUEVES DE 18 A 21 H.

VIERNES DE 17 A 21 H. SABADOS DE 9 A 14 H.	VIERNES DE 17 A 21 H. SABADOS DE 9 A 13 Y DE 14 a 18 H.	OTRO

9. ¿Qué servicios adicionales desearía que tuviese la División de Educación Continua, para los asistentes? ______________________

____

10. Otras sugerencias:

____

____

____

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

RESUMEN DEL DESARROLLO DE MICROPROCESADORES

M en C Angel Kuri Morales

Julio, 1981

1.1 INTRODUCCION

1.1.1 Dispositivos de estado-sólido

En el desarrollo de componentes y dispositivos de cualquier clase, los objetivos básicos han sido siempre aumentar la velocidad, bajar la disipación, aumentar la complejidad funcional. Ahora esos objetivos parece que han tomado una nueva forma que nos llevan a nuevas direcciones a medida que la industria electrónica se embarca en un asalto mayor a las limitaciones fundamentales de tecnología de integración a gran escala.

El avance industrial a las memorias dinámicas de acceso aleatorio de 16 K bits (RAM) por la explotación de las técnicas de diseño del semiconductor oxido-metálico que han probado ser exitosas al cuadruplicar la capacidad de almacenamiento aproximadamente cada dos años. Cada nuevo diseño fue realizado con una celda de memoria más simple que antes, una que típicamente requiere la mitad de dibujo que la anterior.

Sin embargo, con el surgimiento de la memoria RAM de 16 K bits, el potencial para simplificaciones mayores de la celda de memoria parece que fueron terminadas. El desarrollo de la memoria dinámica RAM de 65 K bits usando los métodos corrientes de fabricación y técnicas de circuitos nos llevarían a un tamaño dado mucho muy grande para aplicaciones comerciales.

Mientras tanto en el Japón, la industria japonesa ha intensificado sus esfuerzos en un proyecto nacional que tiene como objetivo no solamente vencer los obstáculos presentes de la nueva generación de memorias, pero también crean circuitos MOS con una muy grande integración cuyas densidades serían órdenosde magnitud mayores que hoy en día. Los ingenieros japoneses ambicionan poner una computadora completa en pastillas dentro de los próximos dos años, usando técnicas de redundancia para resolver problemas potenciales de producción.

Aunque no se han trabajado aun los detalles precisos la evidencia del 'expertise' de los japoneses en semiconductores se presentó al inicio de este año cuando la Nippon Telegraph and Telephone anunció un RAM experimental de 65 K bits fabricada con técnicas fotolitográficas refinadas sobre un chip que mide solamente 6 mm por lado.

Obtener lo más posible del MOS ha sido una meta de los científicos norteamericanos. Modificaciones en el procesamiento y la estructura del dispositivo básico MOS canal n nos llevará a varias versiones de velocidad, cada una enmarcada para aplicaciones de RAM de 65 K bits y una microcomputadora completa de 16 bits en un chip, en un futuro cercano.

El producto baja velocidad-potencia de MOS complementario construído en subtratos de silicón sobre safiro han sido usados microprocesadores y circuitos relacionados fabricados por Hewlett-Packard para uso en casa. Y en el mundo bipolar la lógica de inyección integrada ha sido aplicada por Fairchild y Texas Instruments a chips de 4 K bits RAM y microcomputadoras de 16 bits.

1.1.2 Reduciendo para aumentar.

Una forma de lograr un comportamiento mayor en circuitos MOS canal n es através de reducir las dimensiones del dispositivo. Tal como lo ha aplicado INTEL, líder en la industria en esta forma de atacar el problema, el escalamiento ha reducido el producto velocidad-potencia a 1 picojoule, representando una reducción en un factor de cuatro sobre el procesamiento convencional N MOS. Un RAM estático de 4 K bits usando la técnica mantiene las mismas dimensiones como el original de la compañía el popular RAM estático de 1024 bits, pero lográndose un tiempo menor de 50 ns. o menos. Intel llama a ese acercamiento HMOS- ampliamente tenida como la primera aplicación comercial del escalamiento hacia abajo en MOS. El principio general de escalamiento tiene aplicación a dispositivos con dimensiones de submicrometro. En el principio ha sido aplicar para lograr que una componente tenga 3.5 m .

Los diseñadores MOS (y, para esa materia, los diseñadores bipolares)tienen siempre la opción de escalar la estructura de los dispositivos con objeto de reducir el producto velocidad-potencia e incrementar la densidad. En teoría, un diseñador necesita solamente reducir las dimensiones del dispositivo y otros parámetros por un factor fijo e incrementar la mezcla del sustrato por un factor, y las características eléctricas del dispositivo mejoraran -por lo menos en un factor de escala- a una aproximación de un orden. El producto velocidad-potencia, de nuevo a un primer orden, se mejorarán por un factor de escala cúbico.

Los efectos de segundo orden, sin embargo, tienden a limitar la extensión del escalamiento. Intel ha establecido en un dispositivo escalado nominalmente caracterizado por una longitud de canal de 3.5 (micrómetros) (una figura de mérito común, debido a su influencia en la velocidad del dispositivo). El producto velocidad-potencia (1 picojoule) pudo haberse mejorado mucho más, de acuerdo a INTEL, si el voltaje de alimentación se hubiese escalado. Fue mantenido en el nivel estandar de los sistemas en 5 V.

Aunque los dispositivos de INTEL requieren de tales refinamientos de fabricación como reducción del grueso de óxido y algunas mejoras en los métodos de aislamiento de dispositivo, las estructuras de los dispositivos básicos no cambian y se aplican técnicas estandar de circuitos a través del amplio campo de los circuitos integrados de alta densidad. Más aún las geometrías de patrones cada vez más finas llamadas para mayores aplicaciones de escalamiento hacia abajo vienen rápidamente para alcanzar los métodos fotolitográficos que continuamente mejoran, y aun patrones geométricos más finos se pueden esperar una vez que maduren las técnicas de fabricación con luz de electrones.

Entonces, Intel espera aplicar las'técnicas a las próximas memorias y microcomputadoras. En los trabajos, por ejemplo, es un RAM Mos estático de 16 K bits que tendrá un tiempo de acceso de cerca de 50 ns y posiblemente unida Mos de 200 mil, sobre un lado.

En un dispositivo DMOS, la difusión exterior de boro de la fuente crea un voltaje de umbral relativamente alto adyacente a la fuente mientras que la mezcla del tipo-p más ligera del material remanente entre la fuente y el sumidero dá un voltaje más bajo para esta porción del canal. Efectivamente, la región de umbral mayor es pequeña y dá una trasconductancia alta. La capacitancia se determina por la longitud total del canal.

1.1.3 NMOS con un surco.

Similar al DMOS hay aún otro, recientemente desarrollado con técnica de canal corto que emplea canales en forma de V, o surcos, y llevan el nombre de VMOS. Los transistores MOS se forman con canales V marcados en silicio. Los dispositivos están en el topo de los canales, los canales se forman a lo largo de las pendientes, y, en un rasgo particular de ahorro de área superficial, las puertas del dispositivo se encuentran en el cuerpo del silicón, en los bordes de los canales (en la sima). Como los dispositivos bipolares, VMOS requiere de una región enterrada en un crecimiento epitaxial.

De acuerdo con los abogados de la tecnología VMOS, principalmente American Microsistemas, Inc. (AMI), y Texas Instruments, el proceso evita el problema inherente en atentar fabricar canales relativamente cortos y escalar otros parámetros. La misma evaluación podría hacerse para DMOS. Sin embargo, la disponibilidad de una tercera dimensión vertical al proceso estandar NMOS, decir por caso los que apoyan la tecnología VMOS, proporciona un grado adicional de libertad que puede ser usada para aumentar la velocidad y densidad.

AMI ha usado el proceso para fabricar RAM estática de 4 K bits, con un tiempo de acceso menor que 100 ns, alrededor de la mitad del tamaño de un chip comparable en versiones NMOS directas. Acercándose a producción hay ROM de 65 K bits que serán accesibles en menos de 300 ns, y una memoria de solo-lectura, borrable y reprogramable teniendo un voltaje de programación de solamente 15 volts.

Adicionalmente al cambio estructural survo en V, el VMOS clama por un cambio en el procesamiento con objeto de mantener la longitud del canal bajo, efectivamente, a un micrómetro o algo cercano. La región del canal debe de ponérsele con un perfil doblemente difundido. El resultado neto en un dispositivo que, entre otras ventajas, puede usarse para reducir la superficie del área para lógica aleatoria sobre un tercio de lo que requieren los NMOS convencionales.

1.1.4 Enfatizando las memorias MOS.

Por mucho el segmento de las componentes orientadas a computadoras que mas crece continúa siendo en memorias basadas en varias tecnologías MOS. Los productores de memorias RAM dinámicas de 4 K bits han empezado a saturar el rendimiento y tiempos de entrega, resultando en un mercado de flujo masivo. Se han iniciado intentos de la industria de estandarizar el número de patas y la interfase.

Los RAM'S dinámicos, ambos en 4 K bits y 16 K bits, parece que han sido tomados sobre los requerimientos del procesador central de la computadora- con reducciones de todo dramáticas, más algunas mejoras en confiabilidad, los factores importantes aquí. Los sistemas basados en microprocesadores también hacen uso de RAM dinámicos de 4 K bits y 16 K bits, que son costosos en tamaños pequeños. Las memorias RAM estáticas de 4 K bits se están empezando a usar en aplicaciones de tiempo real aún cuando las memorias dinámicas cuestan menos. Sus ventajas reposan en un acortamiento al ciclo de direcciones sin multiplexar y eliminación de refrescar, que permite una transferencia más rápida en aquellos sistemas basados en discos.

Las MOS RAM estáticas han recibido mucha atención recientemente. El año pasado, MOSTEK aumentó su 4104 con tiempo de acceso de 150 ns y con un consumo de potencia muy bajo (80 mW). El 4104 dentro de ±10% de margen en 5 volts, y sus interfases son TTl para todas las entradas. La llave de este nuevo dispositivo es una celda de cuatro transistores con resistencias de polisilicio usando la técnica de implantación tónica. El uso de un dispositivo MNOS (metal-nitride-oxide semiconductor) con estructuras para obtener novolutilidad limitada ha ganado en interés, con Nitron, NCR, y Geral Instruments ofreciendo una variedad de chips. Sin embargo, problemas de fatiga y de interface compleja aún plagan en dispositivos, tal como tiempo muy bajo de escritura.

1.1.5 Las memorias CCD alçanzan el mercado.

La primer memoria de 65 K bits realizada con técnica dispositivo-acoplado por carga (charge-coupled devise, CCD) está ahora disponible de Fairchild y de Texas Instruments. Esos dispositivos en serie ofrecen la habilidad de llenar la brecha de comportamiento que existe en un extremo alta velocidad de los RAM'S, y por otro las memorias magnéticas de acceso en serie de baja velocidad. Suponiendo que los dispositivos CCD alcancen bajos costos, estos prometen extender la penetración del mercado de las memorias de semiconductores en aplicaciones de almacenamiento masivo. Esas aplicaciones incluyen sistemas basados en microprocesadores, terminales, sistemas de diversión familiar, minicomputadoras, y memorias auxiliares rápidas en computadoras grandes.

Las memorias CCD común y corrientes de 65536 bits tienen una relación de datos máxima de 5 M bits por segundo y consumen solamente 300 mW bajo las peores condiciones de operación. Los chips miden alrededor de 40 000 mil^2. La tecnología de manufactura es similar a la de los MOS-RAM'S dinámicos, y el tamaño de la celda de almacenamiento es de 0.27 nul^2 usando geometría de 4-6μm.

Se han realizado progresos a densidades mayores para las memorias CCD. Se espera que alrededor de los 80's se logren capacidades del orden de M bits.

Las memorias de burbuja, otro contendiente de estado sólido para aplicaciones de almacenamiento masivo. La primer aplicación de las memorias de burbuja fue el anuncio recientemento hecho por La Bell System, y Texas Instruments que han introducido comercialmente chips de 92 K bits y de 80 K bits.

En las aplicaciones de la Bell System, las memorias de burbuja almacenan información en un sistema que viene de una línea telefónica en casos tales como aquellos cuando un dial alcanza un número que no trabaja. El sistema, que está bajo pruebas en una oficina de Detroit, almacena cada mensaje en un circuito impreso que tiene un máximo de dos paquetes de burbujas. Los paquetes del tamaño de la mitad de una cajetilla de cigarrillos, contiene cuatro chip's de burbujas, cada una capaz de almacenar 68121 bits. Cada paquete tiene capacidad de almacenar 12 segundos de voz digitalizada.

1.1.6 Microcomputadoras en un chip

Los microprocesadores han pasado una marca importante, con la introducción de varios fabricantes de modelos que pueden describirse como microcomputadoras en un chip. Contenidos en un solo circuito LSl hay muchas de las funciones que se usan para llamar a un anfitrion de periféricos y circuitos de interfase, en adición al circuito integrado de procesamiento centrl. Ahora, en varios casos, hasta memoria -en capacidades suficientes para permitir almacenamiento de programas de sistema y datos a procesarse- han sido puestas en el chip del CPU.

La mayoría de los nuevos microprocesadores son versiones mejoradas de ofertas anteriores. Entonces, diseñadores de sistemas familiarizados con los anteriores pueden programar con, esencialmente, el mismo conjunto de instrucciones, usando básicamente las mismas ayudas para desarrollo, el mismo hardware y software y anticipar el crecimiento de un 50% y aún más alto del 'throughput', cuando se usan nuevas versiones. Como una simplificación adicional está el que se usa una fuente única de 5 volts.

Las nuevas microcomputadoras contenidas en si mismas de 8 y de 16 bits proveé al diseñador de sistemas con diseños lógicos de propósito general que pueden ser aplicados sobre un amplio rango de aplicaciones que los micros están logrando. De entre las familias de procesadores realizados con tecnología NMOS que han avanzado a una microcomputadora en un solo chip son Intel 8080, Fairchild y Mostek F8 y Motorola 6800.

Adicionalmente hay unidades de 16 bits tal como la Texas Instruments 9900, que ha sido construída en NMOS y T^2L para aplicaciones militares.

Las microcomputadoras recientes de INTEL ejemplifican un número de mejoras forjadas en la industria. Por ejemplo, la 8085, una generación siguiente que sale de la 8080, arquitecturalmente se parece a su antecesor. Aún en el chip 8085 están todas las funciones que se usan para requerir tres chips de la familia 0080, generador del reloj, controlador del sistema, y el CPU. El 8085 incrementa la relación máxima de datos a 3 Mhz y se puede formar un sistema de microcomputadora mínima con un RAM y un ROM multifunción. Como un sistema de tres chips el 8085 proveé algunas características aumentadas tales como cuatro interrupciones por hardware, 38 líneas de entrada, y timer de 14 bits.

La memoria de programa ha venido a bordo del CPU en diversos diseños. En el CPU de Intel 8748, para uno, residen 1024 bytes de ROM borrable y programable eléctricamente. Su disponibilidad se ve como que ha acelerado el desarrollo de sistemas como los programas generados por el usuario pueden ser cargados, corridos y más aún alterada, en minutos.

1.2 Tecnologías.

La tecnología para hacer electrónica integrada monolítica es muy flexible. Redes tridimensionales de conductores, aislantes y semiconductores que realizan funciones electrónicas se producen por medio de una serie de etapas de procesamiento de materiales. Hay muchas secuencias útiles que pueden emplearse para fabricar funciones electrónicas integradas. Cada una de ellas tienen nombres diferentes como su fueran tecnologías separadas, no obstante emplean las mismas operaciones generales para depositar el material semiconductor y forman capas con patrones predeterminados.

En la electrónica digital, se han desarrollado dos ramas principales. Se identifican generalmente como Bipolar y MOS dependiendo del tipo de semiconductor activo que se utilice. En la figura 1 se tiene una representación gráfica de las principales tecnologías disponibles o en desarrollo de circuitos integrados de semiconductores.

Miremos más de cerca las tecnologías más prometedoras.

A. Proceso MOS

Los dispositivos MOS fueron los primeros que se usaron para la fabricación de microprocesadores de semiconductores monolíticos. Se han usado diversos métodos en lo que respecta a tipo de canal, material de la compuerta, orientación del cristal, etc.

En la tabla 1 se presenta una lista de los factores más importantes y de las opciones más comúnes.

1) MOS canal-p. Esta es la tecnología MOS original desarrollada en la segunda mitad de los años 60 y obviamente la tecnología utilizada para el desarrollo del primer procesador monolítico. La Intel 4004 y la 8008, la Fairchild PPS25, La Rockwell PPS4 y PPS8, La National IMP, y muchos otros productos de la 1a. generación fueron realizados con un proceso canal-p. La misma se utiliza virtualmente para todos los elementos de proceso usados en las calculadores.

Por largo tiempo, la tecnología de canal-p MOS ha sido la de batalla y la más estable. Sus principales ventajas son buena respuesta, buenos records, disponibilidad entre otras. Las desventajas iniciales fueron

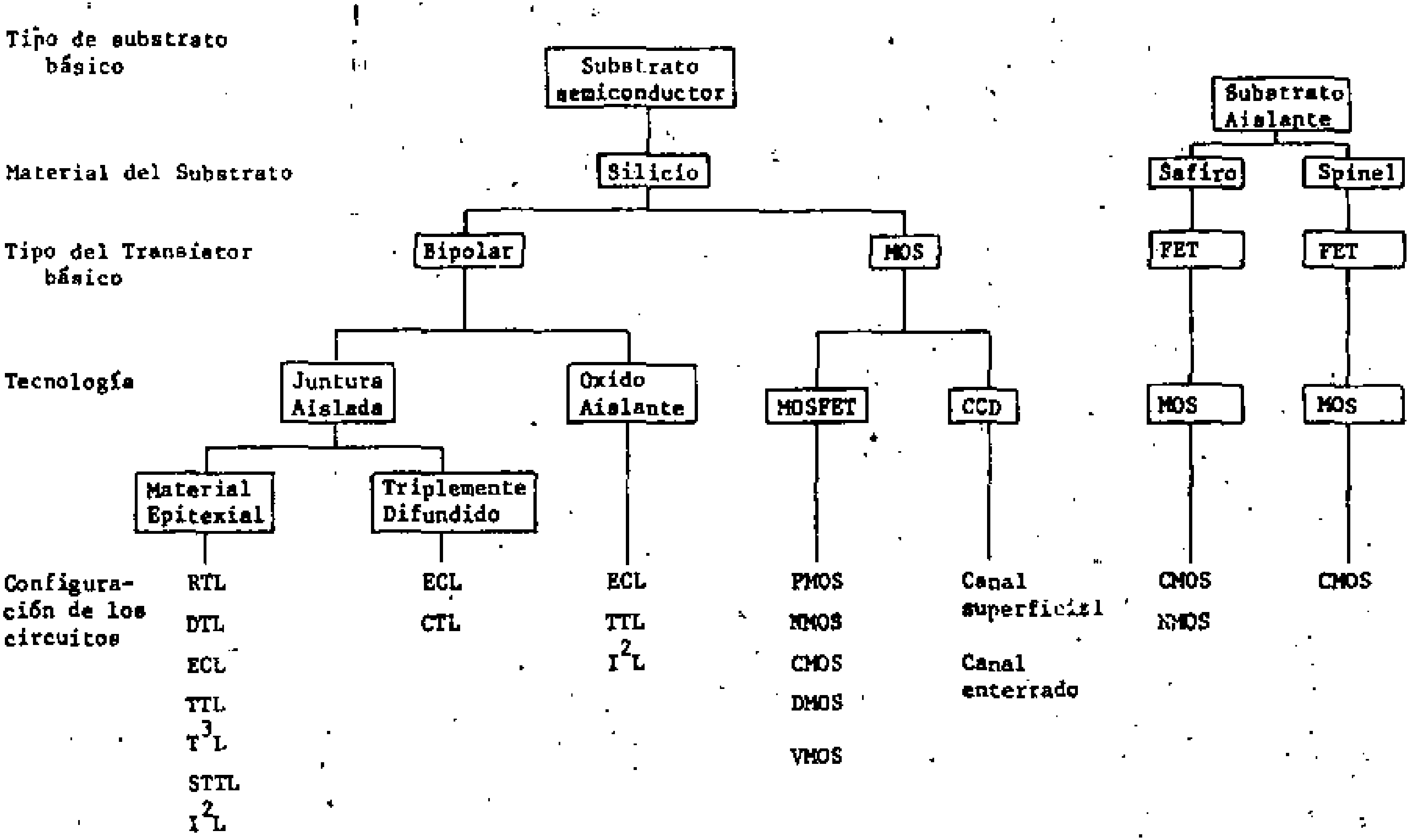


Fig. 1. Tecnologías de semiconductores disponibles actualmente.

En la tabla 1 se presenta una lista de los factores más importantes y de las opciones más comúnes.

1) MOS canal-p. Esta es la tecnología MOS original desarrollada en la segunda mitad de los años 60 y obviamente la tecnología utilizada para el desarrollo del primer procesador monolítico. La Intel 4004 y la 8008, la Fairchild PPS25, La Rockwell PPS4 y PPS 8, la National IMP, y muchos otros productos de la la. generación fueron realizados con un proceso canal-p. La misma se utiliza virtualmente para todos los elementos de proceso usados en las calculadoras.

Por largo tiempo, la tecnología de canal-p MOS ha sido la de batalla y la más estable. Sus principales ventajas son buena respuesta, buenos records, disponibilidad entre otras. Las desventajas iniciales fueron

a) Alto voltaje de umbral (V_t) debido a la orientación del cristal, función de trabajo de la compuerta-aluminio alta, y óxido grueso.

b) Baja velocidad debido a una área grande por compuerta, capacidades grandes y baja movilidad de carriers.

c) Voltajes de alimentación altos que dan por resultados incompatibilidad con los circuitos TTL.

Todas estas desventajas se han superado. Actualmente la tecnología de canal-p tiene bajo V_t y compatibilidad TTL, se ha disminuido la capacitación de traslape y se ha mejorado significativamente su velocidad. Este progreso se logró debido al uso de material <100>, estructuras alineadas de compuertas-silicio, óxido de estaño e implantación de iones.

El microprocesador PALE-16 de National Semiconductor es un excelente ejemplo de lo que se puede hacer con esta tecnología. La principal desventaja remanente del proceso es la limitación en velocidad debido a la baja velocidad de agujeros en dispositivos de canal-p.

2) Canal-n MOS. La desventaja de baja velocidad de los productos de canal-p puede eliminarse usando dispositivos de canal-n MOS. La gran movilidad de electrones comparada con los agujeros en el silicon proporciona una mejor potencial en la velocidad de conmutación en un factor de dos a tres. Por esa razón, todos los microprocesadores de la llamada segunda generación han sido diseñados con canal-n. La intel 8080, Fairchild F-8, Motorla MC800, y otros todos representan productos de segunda generación de gran calidad.

La única desventaja es la sensibilidad a la contaminación que no ha sido eliminada, pero por medio de procesos ultralimpios se ha podido controlar el proceso.

3) MOS complementario (CMOS). La tecnología CMOS se ha considerado como atractiva pero cara. La atracción se basa en las siguientes ventajas

1) Alto factor de forma (igual o mejor que la de canal-n)

2) Extremadamente bajo consumo de potencia (nanowatts en modo estático)

3) Fuente única de voltaje

4) Rango amplio de operación con respecto al voltaje de alimentación y la temperatura ambiente.

5) Alta inmunidad al ruido.

Las desventajas responsables del alto costo son la baja densidad de empaquetamiento comparable con los dispositivos de unicanal MOS y la necesidad de volverse un maestro en el arte de hacer el mismo substrato con canal-n y canal-p para transistores con altos rendimientos. Se han mejorado los procesos de control para la elaboración y se han desarrollado las herramientas para la implantación de iones que han hecho posible hacer dispositivos CMOS con rendimientos comparables a la producción masiva de la tecnología TTL.

Compuerta de Silicio, implantación de iones, y aislamiento de óxido ha decrecido el área por compuerta significativamente comparada con los circuitos CMOS iniciales. Sin embargo, todavía no es posible competir en la densidad de empaque con los de la técnica de unicanal. Pero el avance ha sido significativo para permitir el desarrollo de familias grandes SSI y MSI. Recientemente, se han producido memorias y microprocesadores LSI (tales como COSMAC de RCA y IM6100 de Intersil).

B. Procesos bipolares

El mercado para circuitos digitales bipolares integrados ha sido significativamente mayor que el de la tecnología MOS. Sin embargo, esto ha sido exclusivamente debido a la tremenda cantidad de dispositivos SSI y MSI consumidos por la industria. Hace aproximadamente tres años se introdujeron las memorias bipolares y otros productos LSI. Más recientemente, se ha logrado una acelerada penetración del mercado para LSI con técnicas como la T^2L. Observemos algunos de los contendientes en la técnica bipolar.

1) Lógica Transistor-Transistor (TTL)

El mercado para la tecnología TTL durante 1975 reportó ventas por 260 millones de dólares y para 1980 se estima del orden de 435 millones de dólares. La popularidad de esta tecnología se debe a los siguientes factores.

a) el factor de forma adecuado para la mayoría de aplicaciones

b) muy bajo costo

c) la disponibilidad de más de 300 tipos diferentes

d) un gran número de distribuidores

e) el hecho de que la mayoría de ingenieros de diseño están familiarizados con estos.

Por otro lado, particularmente desde el punto de vista de la tecnología LSI, la tecnología TTL tiene serias desventajas.

La capacidad de empaquetamiento es muy baja, la discipación de potencia es alta, y el proceso es bastante complejo aunque la limitación anterior ha sido menos obvia debido a la tremenda experiencia en esta tecnología. Versiones más recientes que incluyen la tecnología TTL-Schottky ha resuelto algunos de los problemas. La densidad de empaquetamiento ha aumentado de tal suerte que actualmente se tienen del orden de 300 a 400 compuertas por chip. El consumo de potencia se ha reducido de 10 mW a 1-2mW, con cierto aumento en velocidad (4-5 ns en vez de 10 ns) al mismo tiempo. Un buen ejemplo de lo que se puede lograr con el estado del arte de la tecnología TTL son algunos de los varios procesadores orientados a bits y otros productos LSI disponibles ahora. (Fairchild Macrologic, Intel 3001/3002, Memorias monolíticas MM6701, y AMD Am2901). También, el rápido incremento de la lista de fabricantes de memoria RAM bipolar testifican este progreso.

2) Lógica de emisor acoplado (ECL)

Esta tecnología sufre de los mismos problemas que la TTL (potencia y densidad) y, en adición, falta de un aspecto que ha hecho a la TTL exitosa, disponibilidad y popularidad).

Sin embargo la tecnología ECL no tiene rival en cuanto a velocidad se refiere. Hay, actualmente, en el mercado líneas de producción estandar con retardos abajo de 1ns. Otro problema es la alta sensibilidad de la tecnología

ECL a la temperatura.

3) Lógica Integrada de Inyección. (I^2L)

La tecnología I^2L es la nueva y brillante estrella en el firmamento de los semiconductores (LSI-bipolar). Esta tecnología fue desarrollada independientemente por la Phillips en Endhoven Holanda y en IBM en Bueblingen, Alemania (ésta se denomina MTL o merged transistor logic). La razón clave de que la tecnología I^2L sea tan prometedora es el hecho de que parece haber sobrepasado los problemas de los métodos bipolares LSI, propiamente, la baja densidad de empaquetamiento y la alta discipación por compuerta. Con tecnología I^2L se pueden lograr densidades iguales o mejores de las tecnologías MOS (>200 compuertas/mm^2) y discipación de potencia que pueden competir con la CMOS, y al mismo tiempo velocidades bipolares (mejor de 5 ns/compuerta).

La alta densidad de empaquetamiento ha sido el resultado de la eliminación de todo el espacio consumido por resistencias y una suerte de superintegración en donde los transistores n-p-n y p-n-p se forman de tal forma que el área del colector para el transistor p-n-p también funciona como el área de la base del n-p-n y el área de la base del p-n-p está integrada con el área del emisor del n-p-n.

En lo que respecta a la potencia, se ha mejorado en un orden de por lo menos 5 puesto que en vez de usar 5 volts se usa 1 v. También bajo la eliminación de las resistencias se evita el gasto de potencia. Los valores muy bajos de capacidades parásitas y la eficiente inyección de carriers en la región de las bases es otro de los factores. La excelente velocidad se logra de las bajas capacitancias y el no tener problemas de tiempo de almacenamiento. Otra característica es la facilidad de mezclar circuitos digitales y analógicos. El único aspecto negativo es la poca experiencia que se tiene con esta tecnología.

1.1.3. Revisión comparativa

Con este espectro de opciones disponibles, veamos algunas de las debilidades y fuerzas específicas de esas tecnologías para el curso de elementos procesadores LSI monolíticos.

La siguiente es una lista de los parámetros de importancia para procesadores LSI.

a) Area por compuerta

b) Velocidad

c) Potencia por compuerta

d) Costo

e) Capacidad de interface hacia el mundo exterior (fuera del chip)

La tabla 2 lista esos parámetros para las tecnologías líderes p-MOS, n-MOS, CMOS, TTL, ECL y I^2L. Los valores y los rangos registrados son los representativos para las configuraciones LSI actuales.

Otras ilustraciones de naturaleza comparativa se presentan en las figuras 2 y 3. En la figura 2 se presenta el retardo de propagación Vs la potencia y las líneas del producto retardo X potencia se dibuja también. En la figura 3 se trata de cubrir la indicación de comportamiento de complejidad Vs retardo, de lo anterior podemos escribir las siguientes conclusiones.

1) Las tecnologías puede separarse en dos grandes grupos

   - Los procesos MOS densos de bajo factor de forma y los bipolares de baja densidad y alto factor de forma, con la técnica I^2L acercándose claramente en ambas áreas.

2) Por consideraciones de baja-potencia, las tecnologías CMOS y I^2L son las que pueden escogerse.

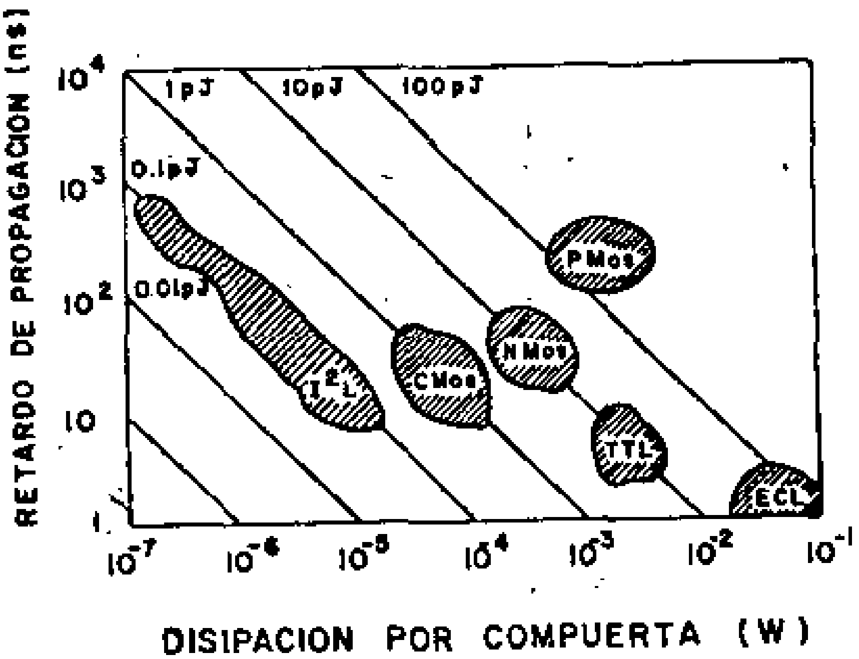


FIG. 2 RETARDO DE PROPAGACION VS DISIPACION PARA LAS PRINCIPALES TECNOLOGIAS LSI.

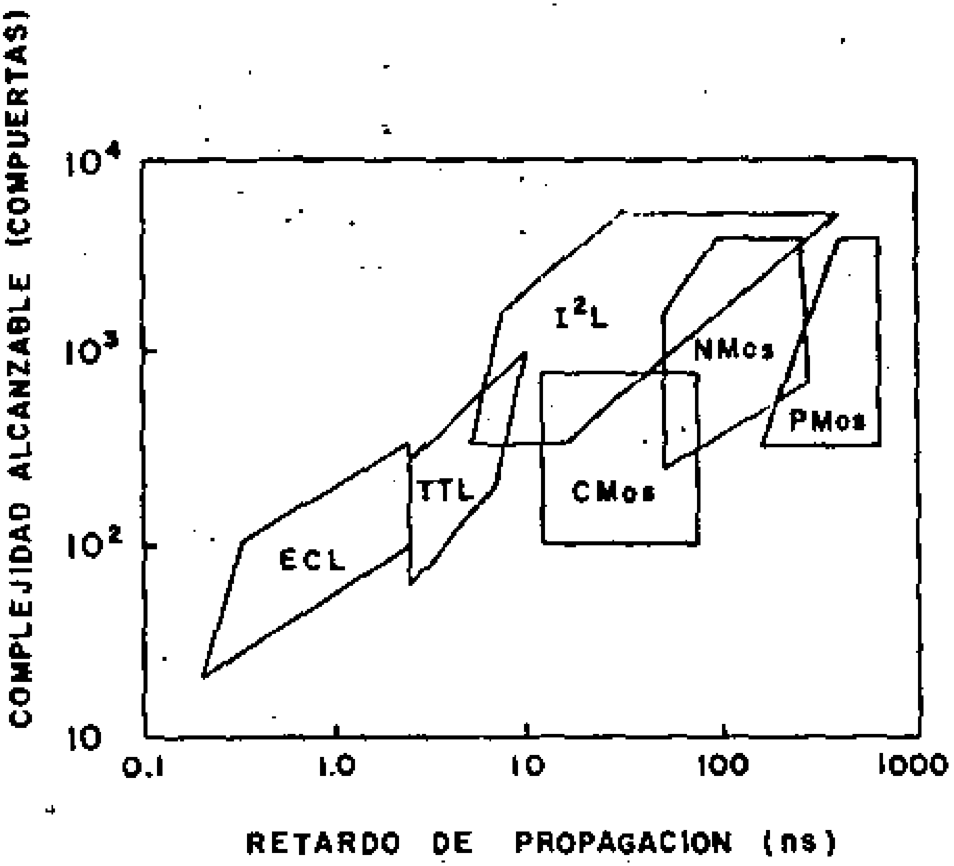


FIG. 3 EL APARENTE COMPROMISO ENTRE COMPLEJIDAD Y RENDIMIENTO. LAS FRONTERAS SON CADA - VEZ MENOS DEFINIDAS.

3) La complejidad de la bipolar no solamente está limitada por la densidad de empaquetamiento pero solo por discipación de potencia.

4) La complejidad de los procesos varían algo, pero las versiones sencillas de cualquier tecnología no resulta necesariamente en el factor de forma requerido o facilidad de aplicación. También, la experiencia con una tecnologí. vs cualquier otra puede poner en evidencia algunas de las diferencias de procesamiento.

5) El costo es una mezcla difícil de definir de área por compuerta y complejidad de proceso (tales como las dadas por el número de etapas de enmascaramiento, el número de difusiones o implantaciones, lo crítico de todas las etapas de procesamiento, y el número total de experiencias con la tecnología particular).

Muchas de esas áreas no necesariamente nos llevan a una interpretación sencilla. Por ejemplo, no obstante que el área por compuerta aparece ser una medida muy directa, esas son algunas complicaciones. Primero, aunque el área interna por compuerta pueda ser pequeña, los requerimientos de bufers y otras interfaces pueden realizarse solamente con gran dificultad. Más aún, el número de fuentes (o referencias) de voltaje y líneas de reloj a ser rotadas a todas las compuertas puede hacer esto imposible para obtener una densidad máxima. Al final de cuentas la cantidad de área activa (el área ocupada por transistores, resistencias, etc) puede ser solamente una porción del total del área requerida por el chip.

Por ejemplo, los chips de los microprocesadores más recientes tienen un área activa de solamente el 50% del area total del chip. El resto lo ocupa el metal de interconexión (los buses de 16 líneas ocupan mucho espacio), trayectorias de bordeo, líneas de potencia, etc.

1.1.4. Conclusiones

Lo siguiente deberá considerarse como un resúmen

a) MOS canal-p. La primer tecnología usada en microprocesadores será utilizada un largo tiempo. Sin embargo, es un proceso básicamente obsoleto y no se usará para nuevos diseños con excepción de calculadoras.

b) MOS canal-n. Esta es la tecnología principal para la gran corriente de los nuevos diseños para aplicaciones de velocidad baja y media en donde un número mínimo absoluto de chips es de gran importancia.

c) CMOS. Esta tiene uso pequeño para procesadores de uno o dos chips.

d) TTL. Su uso se limita a elementos de proceso orientados a la técnica de rabanadas de bits. La tecnología TTL es un método costo-efectivo para construir controladores periféricos, y es un buen acercamiento a usar para emular arquitecturas de máquinas existentes.

e) ECL. Este será probablemente el vehículo mediante el cual se introducirán los microprocesadores a las grandes computadoras. El énfasis se pondría en la complejidad y el comportamiento será probablemente limitado como la TTL.

f) I^2L. La pregunta principal acerca de la tecnología I^2L se centra alrededor de su novedad y a la falta de un buen entendimiento de qué es lo que va a proporcionar. Aparece hasta este punto que I^2L cubre el espectro completo de aplicaciones con la excepción de ECL, y la ECL con consumo de potencia cero cuando no opera (stand by).

Como punto final podemos decir que I^2L es la que tiene más potencial y cubre todas las otras tecnologías en una extensión significativa. En adición, ésta tiene la capacidad de interface directa al mundo real analógico, que es de gran importancia, entre otras, los mercados de automóviles y de consumo.

TABLA 1

Lista de las principales opciones disponibles en la Tecnología MOS

Etapa de proceso	Opciones disponibles
Material inicial	silicio Safiro metal cristalino (spinel)
Orientación del cristal	<100> cubo <111> octaedro
Tipo de dispositivo básico	MOS canal-p MOS canal-n CMOS
Proceso de depósito	Difusión Doble difusión Implantación de iones
Dieléctrico de la compuerta	S_iO_2 $S_{i_3}N_4$ $Al_2\ O_3$
Electrodo de la compuerta	Aluminio Silicio Metal refractario
Control del umbral del campo	Oxido de niquel óxido grueso implantación de iones frenadores de canal óxido undido protección de campo

TABLA 2

Parámetros clave para las tecnologías LSI lideres

tecnología	PMOS	n-MOS	CMOS	TTL	ECL	I^2L
Area/compuerta (mil^2)	8-12	6-8	10-30	20-60	20-50	4-6
Prop retardo/compuerta (ns)	>100	40-100	15-50	3-10	0'5-2	>5
Potencia estática/compuerta (mw)	2-3	0'2-0'5	<0.001	1-3	5-15	<0.2
Producción potencia-vel (pJ)	200	10-50	3	10	10	<1
Número de etapas de enmascaramiento	5	6	7	7	8-9	5-7
Número de difusiones o implantaciones	2	3	4	4	4-5	3-4
Facilidad de interfase	Pobre	Razonable	Razonable	Excelente	Excelente	Buena

1.2 FAMILIAS LOGICAS

En los últimos 25 años han nacido y desaparecido diversas familias lógicas, basadas en técnicas de construcción discreta, monolítica o híbrida. De las tecnologías remanentes, las más populares (DTL,TTL,ECL y CMOS) tiene las siguientes propiedades

- Se dispone de compuertas NAND y NOR
- Se dispone de compuertas AND, OR y OR exclusivo en la mayoría de las series
- Se dispone de flip-flops J1C y D
- Niveles lógicos restaurados a la salida de las componentes
- Se garantiza el número de cargas a la salida bajo las peores condiciones
- Se dispone de contadores, registros, sumadores, decodificadores, selectores de datos y otras funciones de mediana (MSI) y gran (LSI) integración.

En esta sección presentaremos algunos ejemplos de los diversos componentes disponibles en el mercado, desde las más simples hasta las más complejas.

Es muy importante observar la complejidad creciente de los ejemplos. El objetivo es poner de manifiesto el número de componentes e interconexiones logradas en los diversos casos que se presentan.

1.2.1 Familia lógica Serie 54174

La familia lógica 54174 XX es una serie de circuitos integrados realizados con tecnología TTL con velocidad media y alta. La familia incluye un número amplio de funciones presentadas en diversos paquetes. La serie 54 se caracteriza por tener una temperatura de operación, con un rango que va de -55°C a + 125°C. La serie 74 se caracteriza por un rango menor que va de

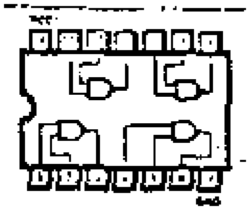
0°C a + 70°C.

La lógica de la serie 54174 se define usando LOGICA POSITIVA, usando la siguiente converción:

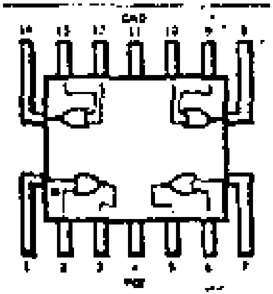
VOLTAJE BAJO = '0' LOGICO

VOLTAJE ALTO = '1' LOGICO

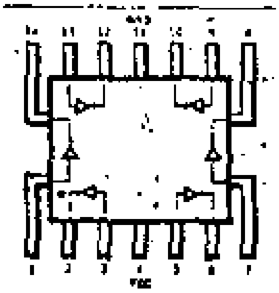
1) SN7400 Compuerta NAND cuadruple con dos entradas positivas



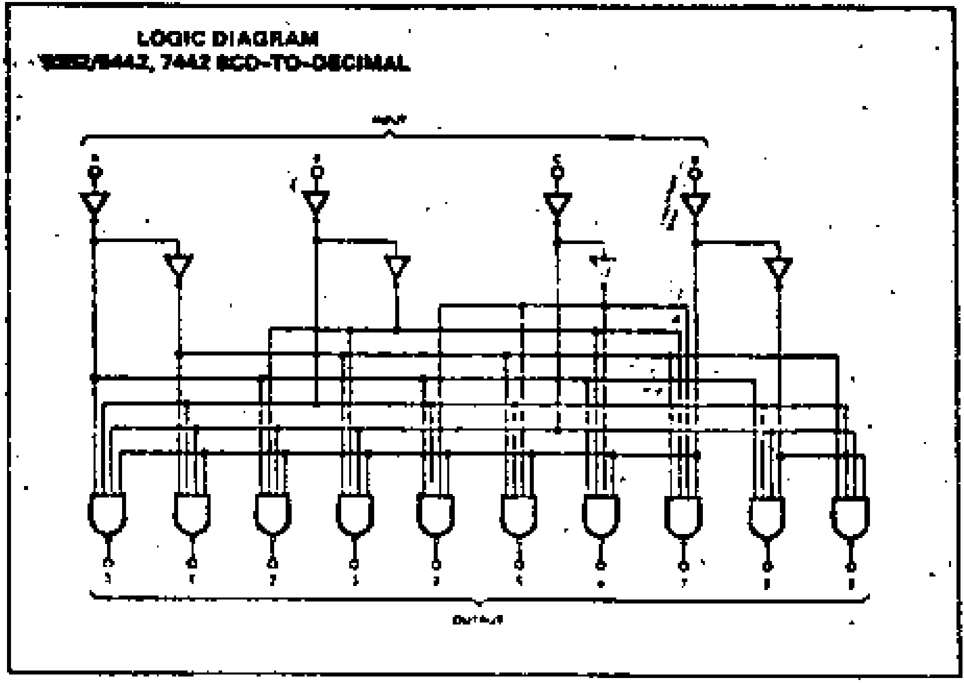
2) SN7402 Compuerta NOR cuadruple con dos entradas positivas

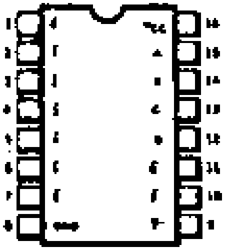


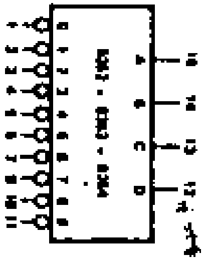
3) SN7404 Inversor sextuple



4) SN7442 Decodificador BCD a Decimal

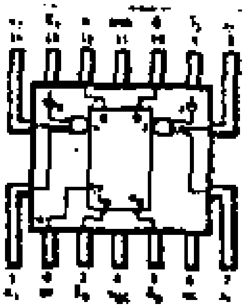






Este es una componente realizada con tecnología TTL MSI (Integración de mediana escala)

5) SN7470 Flip Flop JK



Jn	Kn	Qn+1	Preset	Clear	Q
0	0	Qn	0	0	1
1	0	1	1	0	0
0	1	0	0	1	1
1	1	$\bar{Q}n$	1	1	0

$J = J_1 J_2 J^*$ $K = K_1 K_2 K^*$

El flip-flop SN7470 envía la salida durante la subida del pulso. Esta componente está diseñada especialmente para aplicaciones de mediana y alta velocidad, puede ahorrar energía.

En esta familia se tienen más de 300 funciones repartidas entre la tecnología TTL y TTL Schottky. Se dispone de los siguientes tipos de elementos:

1) Elementos Aritméticos
2) Contadores
3) Selectores/Multiplexores de datos
4) Decodificadores
5) Codificadores
6) Expansores
7) Flip-Flop
8) Compuertas NAND/NOR/AND/OR y Buffers
9) Compuertas inversores AND-OR/AND-OR
10) Latches
11) Memorias
12) Registros

1) Elementos Aritméticos

Los más importantes son las unidades Lógicas-Aritméticas, sumadores y comparadores.

2) Contadores

Son contadores sincronicos de 4 bits, contadores hacia arriba y hacia abajo con modo de control, contadores de décadas, etc.

3) Selectores/Multiplexores de datos

Los hay de 16 a 1, de 8 a 1, de 4 a 1 y de 2 a 1 con los que se puede cubrir cualquier problema de selección/multiplexaje.

4) Decodificadores

Se tienen BCD a Decimal, Exceso 3 a decimal, 4 a 16 líneas, 2 a 4 líneas, BCD a decodificador de siete segmentos.

5) Condificadores

Hay de 10 líneas a 4 líneas y 8 líneas a 3

6) Expansores

Expansores de 4 entradas y de tres entradas

7) Flip-Flops

Tipo JK, JK maestro-esclavo, tipo D y RS.

8)

9) Son los componentes estandar

10) AND y OR, NAND,NOR, inversores, Latches .

11) Memorias

Memorias de solo lectura (ROM), en memoria RAM (acceso aleatorio)

12) Registros

Hay de 4 y 8 bits, carga en paralelo, en serie.

Entrada en serie-salida en paralelo, entrada en paralelo y salida en serie.

Existe, además, una variedad muy grande de funciones realizadas con otras tecnologías, tales como MSI/TTL, ECL, MOS, etc.

1.2.2. Tecnología de Gran escala de Integración (LSI)

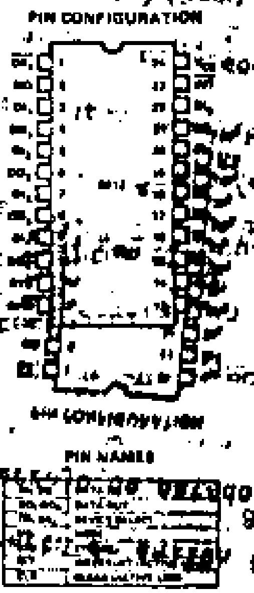
En la actualidad, además de los microprocesadores, existen otros elementos con gran complejidad que sirven para ampliar la capacidad de los microprocesadores y además facilitan la utilización de estos.

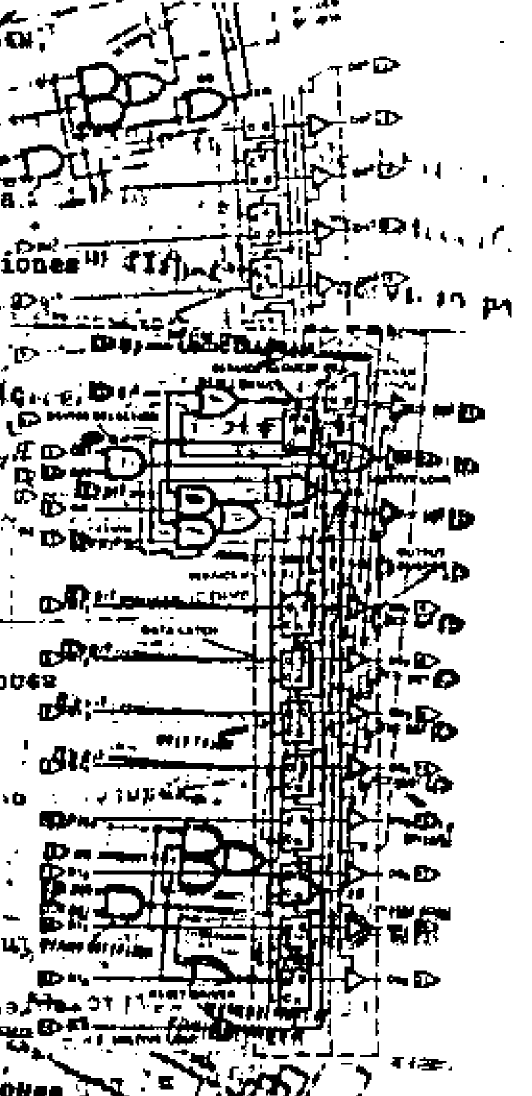
1) Puerto de entrada/salida de 8 bits.

Este puerto de entrada/salida consiste de un cerrojo (Latch) de 8 bits con buffers de salida de 3 estados junto con una sección lógica para selección y control de dispositivos. También cuenta con un flip-flop de servicio para la generación y control de interrupciones de un microprocesador.

Usos de este dispositivo para sistemas de microcomputadoras.

1) Buffer controlado por compuertas
2) Bus-Driver bidireccional
3) Interruptor de un puerto de entrada
4) Interruptor del puerto de instrucciones
5) Puerto de salida (con saludo)
6) Cerrojo de estados





¿Qué es un Microprocesador?

Responder a esta pregunta hubiese sido más fácil hace 2 años que hoy día, ¿Por qué? porque la industria y la tecnología microeléctricas son, quizá, las más cambiantes del mundo, considerando cualquier rama del saber humano. Esto se debe, sin duda, a la amplísima gama de aplicadiones de sus derivados y sub-productos. Es un hecho aceptado, hoy día, que el advenimiento del microprocesador es un acontecimiento de importancia similar al de la utilización de la energía eléctrica, a principios de siglo. Esta afirmación podrá parecer exagerada a algunas personas. Examinemos, sin embargo, el desarrollo histórico de los microprocesadores, como se muestra en la siguiente tabla:

Año	Procesador	Tecnología	Reloj (MHz)	Transistores	Direccionamiento
1972	8008	PMOS	.5	2000	16 K.
1974	8080	PMOS	2	4500	64 K.
1976	Z80	NMOS	4	6000	64 K.
1978(En.)	8086	NMOS (VLSI)	5-8	20000	1 M.
1978(Dic)	Z8000	NMOS (VLSI)	~4	17500	8 M.
1979	68000	NMOS (VLSI)	8	68000	16 M.

En esta tabla observamos sólo los más representativos de los microprocesadores: de INTEL, el 8008, 8080 y 8086; de Zilog el Z80 y Z8000; de MOTOROLA, el M68000.

Observamos el dramático incremento en velocidad y grado de integración. De 1972, año de la aparición del 8008, a la fecha, hemos pasado de un ciclo de reloj de 1/2 MHz a 8 MHz, para un aumentot de 16 veces en velocidad. Asimismo de 2000 transistores, hemos pasado a 68000, para un aumento de 34(!!) veces en la densidad básica, de la unidad. Con los procesadores hoy, podemos direccionar hasta 16 M bytes, vs. sólo 16 K de un 8008; un incremento de 1000 veces!

Pero no sólo en el procesador central existen estas capacidades. Las memorias han sufrido cambios proporcionales. En 1976, la memoria más densa en el mercado era de 4K bits, hoy día, con CCD's y memoria de burbuja magnética, tenemos capacidad de hasta 256 K bits en un 'chip' Un incremento de 64 veces en la capacidad del almacenamiento.

De la misma manera, los controladores, periféricos, etc., se han modificado. El resultado final es que, hoy en día, con cuatro 'chips', tenemos capacidad de cómputo más o menos equivalente a la de una CDC 6600. Como marco de referencia mencionamos a ustedes que en 1970, el documento de CDC que describe a esta máquina empieza con un capítulo que, traducido literalmente, es "Justificación de las Grandes Computadoras".

La conclusión de esta introducción un poco atropellada es la siguiente:

1) Hoy día es posible adquirir capacidad de cómputo a bajo precio en magnitudes que hasta hace una década eran difíciles de justificar por su complejidad.

2) Los costos de cómputo se han reducido dramáticamente. La tendencia es a seguir disminuyendo. Con la integración que aumenta, los costos se abaten.

3) El 'software' es cada vez más sofisticado. Los micro—computadores de hoy aceptan los lenguajes de ayer (COBOL, FORTRAN, BASIC, PL/I, PASCAL, etc.).

4) Los costos bajos y la alta integración propician nuevas arquitecturas en máquinas 'grandes'. Las computadoras del mañana serán arreglos de decenas o cientos de microprocesadores.

Pero, apesar de nuestras 'conclusiones', no hemos aún respondido a la pregunta inicial. ¿Qué es un microprocesador?

De la anterior discusión se desprenden varias características de los microprocesadores:

1) Son pequeños (físicamente).

2) Son baratos (comparativamente).

3) Tienen capacidad de cómputo.

4) Poseen memoria.

¿En resumen, pues, podemos decir que un microprocesador es un computador pequeño y barato? Si y no. En la introducción mencionamos que es hoy más difícil responder a la pregunta que hace algunos años. Esto se debe a que hay, hoy, varios tipos de microprocesadores. ¿Por qué? porque a cambio de reducir el tamaño (y disminuir el consumo de energía) perdemos, también, velocidad. A las 4 características de la lista anterior, hay que agregar una - más:

5) Son "lentos".

¿Qué entendemos por "lentos"? Un micro (procesador) típico actual, tiene un reloj de 2-5 MHz, es decir su ciclo básico de de instrucción es de ~250 ns. En contraste, las máquinas rápidas pueden trabajar a velocidades de ~5 ns. Es decir, 50 veces más - rápido.

Es claro que no en todas los casos (de hecho en muy pocos) necesitamos velocidades de ese orden. Entonces los microsistemas se han subdividido en varios grupos, dependiendo del problema a que están orientados. Básicamente, pues, podemos dividir a los microprocesadores en familias:

a)	Microprocesadores rápidos, "bit slice"	(MSI)
b)	Microprocesadores orientados a bytes	(L,SI)
c)	Microcomputadoras.	(VLSI)
d)	Micro-miniprocesadores	(VLSI)

Para que nuestra definición quede completa, pues, hay que decir que un microprocesador cae dentro de alguna de las anteriores familias.

La definición de un microprocesador es pues, la siguiente:

Un microprocesador es un dispositivo electrónico digital de alta integración, que incorpora todas las características básicas de una computadora convencional; que es de bajo costo, bajo consumo de potencia y que pertenece a alguna de las 4 categorías mencionadas anteriormente.

Para que nuestra definición quede clara, hay que especificar qué entendemos por computador 'convencional'.

Este es un sistema digital que:

1) Tiene medios de entrada.
2) Tiene un almacen, en donde pueden estar instrucciones o datos.
3) Tiene una sección capaz de ejecutar cálculos aritméticos y lógicos.
4) Tiene medios de salida.
5) Tiene una unidad de control, capaz de escoger de entre distintos cursos de acción, dependiendo de los datos.

En este curso nos restringiremos a estudiar microprocesadores del tipo (b) debido a que:

1) Son los más usados.
2) Son los más desarrollados.
3) Son los más útiles en el contexto de nuestro país.
4) Son los más baratos.

Con el objeto de introducir al estudioso al campo de los microprocesadores, sin embargo, mendionaremos brevemente, a un representante de las categorías a, c y d.

Microprocesadores '<u>Bit</u> <u>Slice</u>'.

Este tipo de micros tienen la característica de ser "rebanadas" ("slices") de procesado. Esto significa que cada elemento del procesador está diseñado para un número pequeño de bits (digamos N bits). De esta manera, uniendo M elementos, es posible configurar una computadora de MXN bits.

El ejemplo que presentamos es el procesador serie 3000, de Intel. Este procesador tiene un ancho (o rebanada) de 2 bits por elemento. Para lograr un computador de 16 bits de ancho es necesario, pues, ligar o unir 8 CPU's y sus correspondientes memorias y unidades de control.

La serie 3000 es, además microprogramable. No se debe confundir el término "microprogramable" con el de microprocesador.

Estos conceptos no están ligados en forma alguna. De hecho la mayor parte de los procesadores "grandes" son microprogramados.

En esencia, el microprograma es un conjunto de instrucciones, llamada micropasos, que son los elementos básicos de una instrucción de la máquina.

Por ejemplo, la instrucción:

ADD A,B (suma A = A+B)

en donde A y B son números de punto flotante, consta de una serie de pasos más básicos. Se puede decir que cada instrucción de máquina:

ADD A,B

de un procesador microprogramable, es una llamada al microcódigo - (de hecho, a una rutina de éste).

Un procesador microprogramable, pues, tiene en su interior, una pequeña computadora con memoria ROM, en la cual están los micropasos del acervo de instrucciones de la máquina en cuestión.

A este tipo de programas se les conoce como !firmware', para distinguirlos del 'software' y del 'hardware'.

La serie 3000, como decíamos, es microprogramable. Esto hace que el diseñador defina su propio conjunto de instrucciones.

Además, este micro es bipolar. Esto significa que los tiempos de conmutación (de 0 a 1 y viceversa) son del orden de 30-50 - macrosegundos (es decir 30-50 x 10^{-9} seg.), con ciclos básicos de 150 ns en el procesador central.

Esto hace que este procesador sea rápido y versátil; por - otro lado, es caro (relativamente) e implica que, para cada diseño, el ingeniero debe de elaborar su propio conjunto de instrucciones y su propia arquitectura y, por supuesto, su propio 'software'.

Es posible, sin embargo, emular otro procesador conocido y mejorar su 'thruput' (relación de resultados/seg.) copiando el conjunto de instrucciones de algún procesador comercial.

Sería posible, por ejemplo, diseñar un procesador idéntico a un Z80 con serie 3000. De esta forma tendríamos el 'software' del Z80 y la velocidad de los dispositivos bipolares.

Microprocesadores Orientados a 'Bytes'.

Estos comprenden al 8080, 6800 y Z80 y son el tema fundamental de este curso.

Sus características básicas son las siguientes:

1) CPU en un 'Chip'.
2) Palabras de 8 bits (1 byte).
3) Bajo costo.

4) Amplio acervo de periféricos.

5) 'Software' de alto nivel ya desarrollado (compiladores, intêrpretes, ensambladores, etc.)

6) Pequeño número de integrados para lograr una configuración básica (que cumpla con la definición de computadora).

Puesto que van a ser el tema de las sesiones siguientes, dejaremos su tratamiento para capítulos posteriores, en donde son tratados con gran detalle.

Microcomputadoras.

De lo que se ha venido discutiendo, parece no ser muy obvio que a una familia se le llame microcomputadoras, cuando todas lo son. En realidad, lo que queremos señalar es que, en realidad, los procesadores antes mencionados conforman a una microcomputadora sólo mediante el uso de varios 'Chips'. En esta familia incluímos a aquellos circuitos que incorporan todo lo necesario para tener una computadora en un circuito integrado. Es decir, en un 'Chip' está concentrado el procesador, la memoria de programas y de datos, los puertos de entrada/salida y la unidad de control.

De aquí que hagamos la distinción entre un microprocesador (3000, 6800, etc.) y un microcomputador. Ejemplo de este último es el procesador 8748 de Intel. Esta micro posee puertos, CPU, 64 bytes de RAM y 1K de EPROM, todo en el mismo circuito integrado.

Micro-miniprocesadores.

Lo que deseamos señalar, al acuñar este término es que este tipo de microprocesadores tienen ya, las características de un miniprocesador:

a) Palabras de 16 o más bits de ancho.

b) Amplio espacio de direccionamiento (del orden de Megapalabras).

c) Instrucciones comunmente asociadas a máquinas "grandes".

Ejemplos de ésto son el 8086 de INTEL, el Z8000 de Moztek y el 68000 de Motorola.

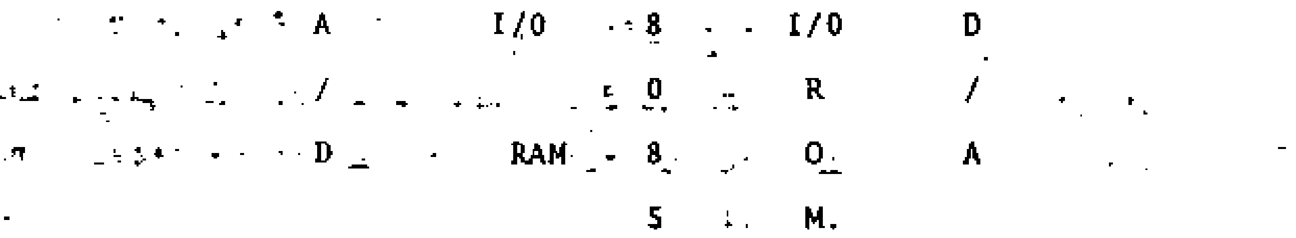
Estas máquinas incluyen en su 'set' de instrucciones aritmética "complicada" (multiplicación y división), direccionamiento indirecto, et.

Perspectivas.

Al desarrollo de los micros hay que asociar el desarrollo de otro tipo de electrónica. En particular, la electrónica de la transducción hace que sea posible atacar problemas del mundo analógico en forma digital.

Por ejemplo, TRW ha desarrollado convertidores A/D (analógica o digital) con velocidad de conversión de 35 mseg. para 8 bits. Compañías como Analog Devices, por mencionar sólo una, tienen convertidores D/A (digital a analógico) de tiempos de conversión de 40 mseg para 8 bits.

Las perspectivas que esto abre son prácticamente ilimitadas En la figura se muestra un sistema de control automático basado en un 8085 y los convertidores arriba mencionados.



Este es un control analógico/analógico logrado con sólo 5 'Chips'. ¿Qué tipo de control? El que el programador desee! Con sólo cambiar el programa interno, el sistema se convierte en otro cualquiera. Los viejos problemas de inestabilidad en controles de máquinas herramientas se eliminan con un motor de pa-

La solución de largas ecuaciones diferenciales simplemente pierde sentido.

Tiempo de respuesta en modo estable:

200 µseg.

Hay, en nuestra opinión dos campos de acción básicos para - los microprocesadores: el campo industrial y el campo de la informática.

En el campo de la industria, es fácil vislumbrar las repercusiones que la siguiente consideración podrá tener:

Cualquier sistema de control puede constar de una computadora para controlar el proceso.

En el campo de la informática, es obvio que la caída de precios en los sistemas de cómputo debe repercutir grandemente. Más aún si se toma en cuenta que no sólo se abaten los precios, sino también aumenta la capacidad de cómputo.

Especular al respecto en esta área es fácil. Algunas predicciones: las computadoras se convierten en artículos de hogar; los sistemas de reconocimiento y síntesis de voz permiten rápidos avances en robótica; las memorias de dico desaparecen para ceder su lugar a CCD's y memorias de burbuja; las arquitecturas de computadoras se orientan a multi-microprocesamientos (pioneros en esta área son Cm* y Tandem, de propósito especial, así como AHR para LISP); en resumen, la necesidad de conocimiento en el área de la microelectrónica y, en particular, de los microprocesadores se presenta como algo inmediato. La tecnología de los microprocesadores está firme y bien establecida.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

Control de entrada/Salida en Serie

Serial I/O Controller

Ing Mario Rodríguez Manzanera

Junio, 1981

SIO - Serial I/O Controler.

Control de entrada/salida en serie.

Introducción.

El desarrollo de microprocesadores ha provocado la implementación de toda una familia de dispositivos de interfase, tanto para conexiones en serie como en paralelo.

En la línea de productos de soporte del procesador Z-80 se encuentra el SIO, un dispositivo programable para controlar la entrada o/y salida de datos en serie entre Z-80 y dispositivos periféricos (ejemplo: CRT, TTY).

En esta plática se presentan las características de conexión y programación del "SIO". (En otras familias USART, UNIVERSAL SINCHRONOUS ASINCHRONOUS RECEIVER TRANSMITER).

Comunicación serie entre microcomputador y periféricos.

En el área de comunicación de datos se integran dos tipos de conocimiento, uno software y el otro hardware. En este último se enlistan las siguientes posibilidades:

a).- Tipo de conexión: full duplex, half duplex, simplex.
b).- Tipo de interfase: voltaje, corriente.
c).- Velocidad de transmisión y recepción.

d).- Número de bits.

e).- Conexión: síncrona, asíncrona.

f).- Niveles de voltaje.

g).- Número de stop bits.

h).- Paridad.

i).- Break generación y detección.

j).- Errores de detección.

k).- Modem?

Y en cuanto al hardware:

a).- Tipo de protocolos (IBM, Bisincrono, HDLC, SDLC).

b).- Manejo de interrupciones.

c).- Modos de direccionamiento.

d).- Programación de dispositivo.

A continuación se muestra la arquitectura de SIO y la disposición física de pins, con su significado y el diagrama de bloques correspondiente.

03

PRELIMINARY

MOSTEK®

Z80 MICROCOMPUTER DEVICES

Technical Manual

MK3884/MK3885 SERIAL I/O CONTROLLER

TABLE OF CONTENTS

LIST OF TABLES & FIGURES

Figure TITLE Page

1.0 INTRODUCTION

The MOSTEK Z80 product line is a complete set of microcomputer components, development systems and support software. The Z80 microcomputer component set includes all of the circuits necessary to build high-performance microcomputer systems with virtually no other logic and a minimum number of low cost standard memory elements.

The Z80-SIO (Serial Input/Output) circuit is a programmable, dual-channel device which provides formatting of data for serial data communication. It is capable of handling asynchronous, synchronous and synchronous bit oriented protocols such as IBM BiSync, HDLC, SDLC and virtually any other serial protocol. It can generate CRC codes in any synchronous mode and can be programmed by the CPU for any traditional asynchronous format.

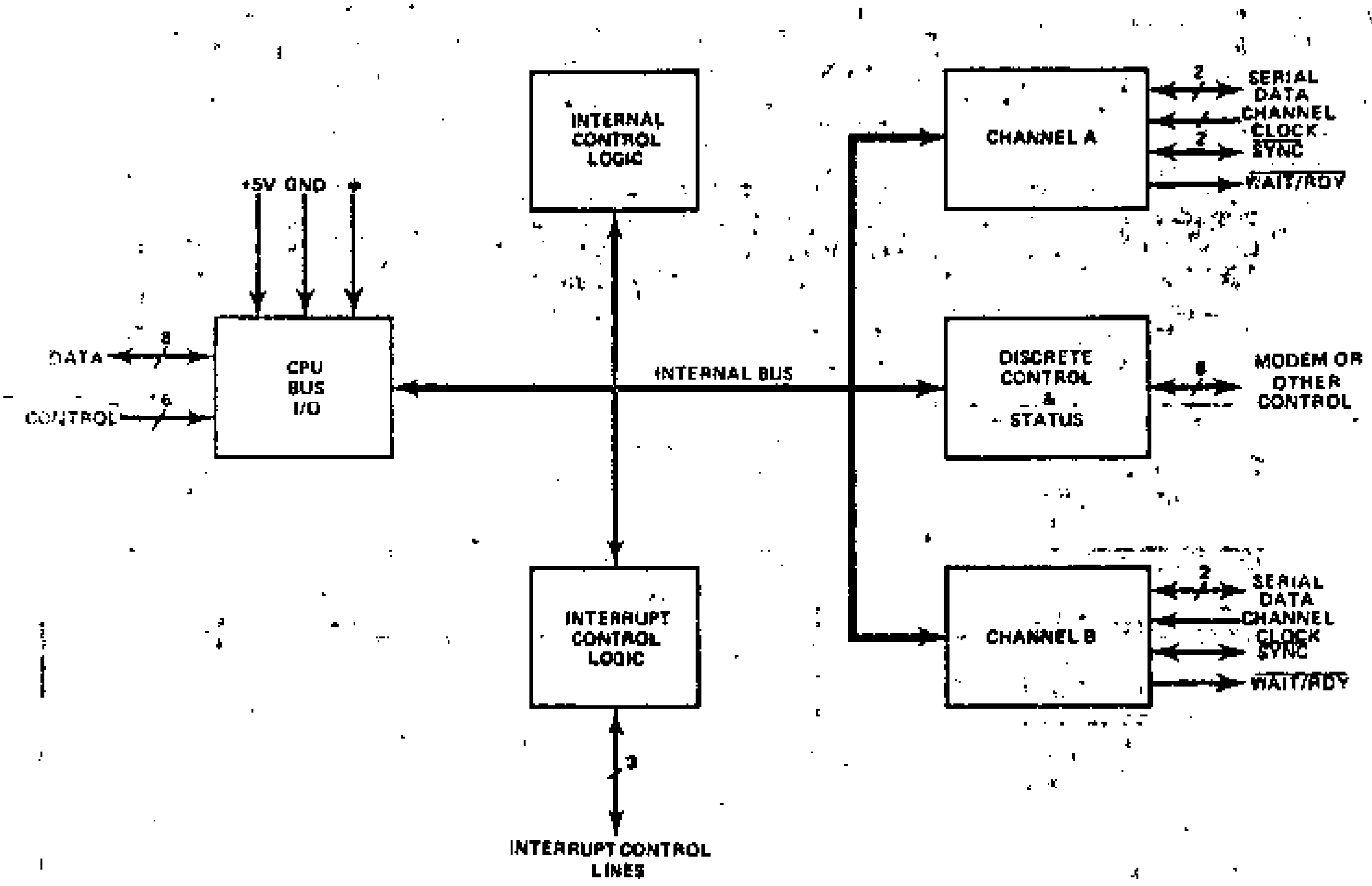
1.1 STRUCTURE

□ N-channel Silicon Gate Depletion Load Technology
□ Forty Pin DIP
□ Single 5 volt power supply
□ Single phase 5 volt clock
□ Two Full Duplex channels

1.2 FEATURES

□ Two independent full duplex channels
□ Data rates -- 0 to 550K bits/second
□ Receiver data registers quadruply buffered; transmitter double buffered.
□ Asynchronous operation
— 5, 6, 7 or 8 bits/character
— 1, 1½ or 2 stop bits
— Even, odd or no parity
— x1, x16, x32 and x64 clock modes
— Break generation and detection
— Parity, Overrun and Framing error detection
□ Binary Synchronous operation
— Internal or external character synchronization
— One or two Sync characters in separate registers
— Automatic Sync character insertion
— CRC generation and checking
□ HDLC or IBM SDLC operation
— Automatic Zero insertion and deletion
— Automatic Flag insertion
— Address field recognition
— I-Field residue handling
— Valid receive messages protected from overrun
— CRC generation and checking
□ Eight modem control inputs and outputs
□ Both CRC-16 and CRC-CCITT (−0 and −1) are implemented
□ Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.
□ All inputs and outputs fully TTL compatible.

SIO BLOCK DIAGRAM
Figure 1.0

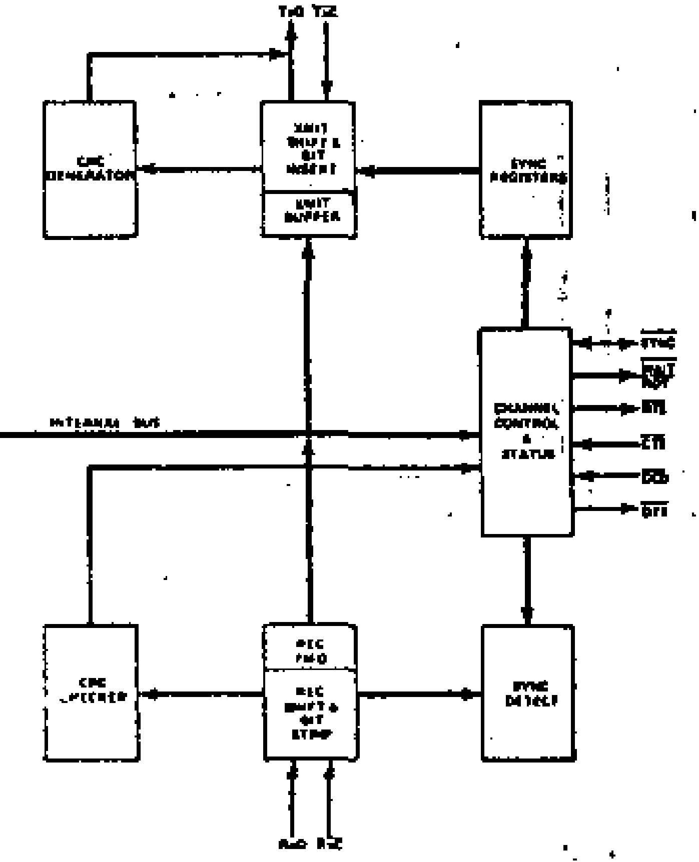


2.0 SIO ARCHITECTURE

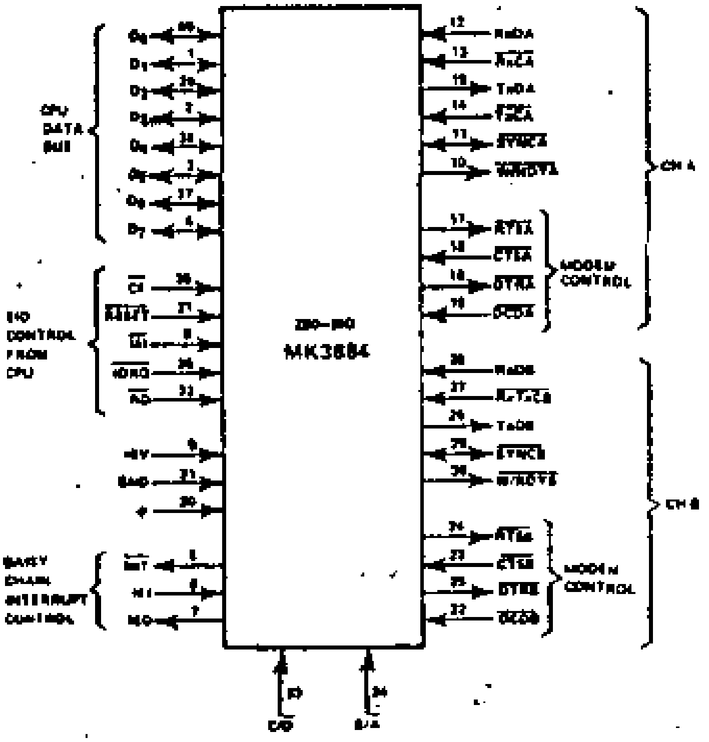
A block diagram of the SIO is shown in Figure 1. The internal structure includes a Z80-CPU bus interface, internal control and interrupt logic and two full duplex channels. The interrupt control logic determines which channel and which device within the channel is the highest priority for purposes of the automatic interrupt vectoring. Priority is fixed with Channel A assigned higher priority than Channel B and the Receiver, Transmitter and External/Status assigned priority in that order within each channel.

The channel logic is shown in block form in Figure 2. Each channel has five 8-bit control registers and three 8-bit status registers. The interrupt vector is written into an 8-bit register in Channel B and may also be read from that channel. The receiver has three 8-bit buffer registers in FIFO arrangement in addition to the 8-bit input shift register. The transmitter has one 8-bit buffer register in addition to the 8-bit output shift register. The CRC generator/checkers are 16-bit shift registers with appropriate internal feedback (programmable) for two different CRC codes.

CHANNEL BLOCK DIAGRAM *
Figure 2.0



SIO PIN OUT
Figure 2.1



*Configuration of Channel B will vary according to bonding option. See Section 2.2.

D0-D7	System Data Bus (bidirectional, tri-state)
B/$\overline{A}$	Channel B or A select (input high is Channel B)
C/$\overline{D}$	Control or Data select (input high is control)
$\overline{CE}$	Chip Enable (input, active low)
$\overline{M1}$	Machine Cycle One Signal from Z80-CPU (input, active low)
$\overline{IORQ}$	Input/Output request from Z80-CPU (input, active low)

Machine Cycle Status from the Z80 CPU (input, active low)

φ — System Clock (input)

$\overline{\text{RESET}}$ — Reset (input, active low) disables both receivers and transmitters. T x DA and T X DB are forced marking. Modem controls are forced high. Control registers must be rewritten after SIO is reset and before any data is transmitted or received. All interrupts are disabled.

IEI — Interrupt Enable In (input, active high)

IEO — Interrupt Enable Out (output, active high) IEI and IEO form a daisy-chain connection for priority interrupt control.

$\overline{\text{INT}}$ — Interrupt Request (output, open drain, active low).

$\overline{\text{WAIT/READY A}}$
WAIT/READY B — Two pins, one for each channel. They may be programmed to serve as ready lines for use with a DMA Controller or they may serve as wait lines to synchronize the Z80-CPU to the SIO data rate.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$ — Clear to Send (2 pins, inputs, active low). When programmed as AUTO ENABLES, these inputs enable the transmitters of their respective channels. If these pins are not programmed as transmitter enables, they may be programmed as general-purpose input pins. These inputs are Schmitt-trigger buffered to allow slow-rise time inputs.

$\overline{\text{DCDA}}$, $\overline{\text{DCDB}}$ — Data Carrier Detect (2 pins, inputs, active low.) These pins are similar to the $\overline{\text{CTS}}$ inputs, except that they are usable as receiver enables rather than transmitter enables.

RxDA, RxDB — Receive Data, (2 pins, inputs, active high.)

TxDA, TxDB — Transmit Data. (2 pins, outputs, active high.)

$\overline{\text{RxCA}}$, $\overline{\text{RxCB}}$ — Receiver clocks (inputs, active low.) (Two pads, one per channel. See note on Bonding Option.) Schmitt-trigger buffered.

$\overline{\text{TxCA}}$, $\overline{\text{TxCB}}$ — Transmitter Clocks (inputs, active low.) (Two pads, one per channel. See note on Bonding Option.) Schmitt-trigger buffered.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$ — Request to Send (2 pins, outputs, active low.) When the RTS bit is set, the $\overline{\text{RTS}}$ pin goes low. When the bit is reset in asynchronous mode, the pin goes high, but only after the transmitter is empty. In synchronous modes, $\overline{\text{RTS}}$ is a simple output which strictly follows the state of the $\overline{\text{RTS}}$ bit.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$ — Data Terminal Ready (2 pins, output, active low.) Pin follows state programmed with DTR bit. (Two pads, one per channel. See note on Bonding Option.)

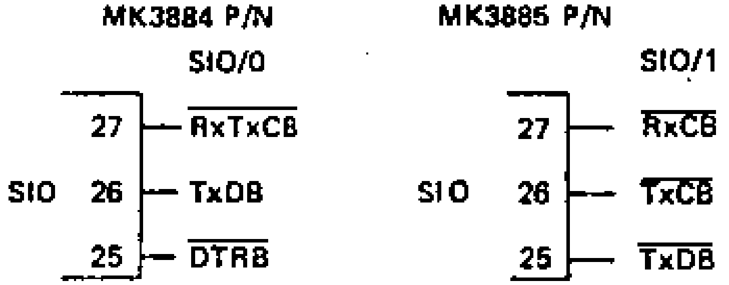
$\overline{\text{SYNCA}}$, $\overline{\text{SYNCB}}$ External Character Synchronization (2 pins, input/output, active low.) If the External Synchronization mode is selected, assembly of characters will begin on the next rising edge of $\overline{\text{R x C}}$. If internal character sync modes are selected, the pins are outputs that are active during part of the clock cycles that a sync character is recognized. The sync condition is not latched, so this pin will be active every time a sync pattern is recognized, regardless of character boundaries. In asynchronous modes, these pins are simple inputs to the HUNT/SYNC bits in Status Register 0 and may be used for any input function desired. However, if EXTERNAL/STATUS interrupts are enabled in the asynchronus mode, then SYNC should not be left floating as this could cause spurious interrupts to occur.

NOTE: When used as an external synchronization pin, it must not become active for three system clock cycles after the previous rising edge of $\overline{\text{R x C}}$. This requirement normally can be met by allowing $\overline{\text{SYNC}}$ to change only on the falling edge of $\overline{\text{R x C}}$.

2.2 NOTE ON BONDING OPTION:

Due to package constraints, there are only two pins available for the three signals, $\overline{\text{T x CB}}$, $\overline{\text{R x CB}}$ and $\overline{\text{DTRB}}$. They are normally bonded so that $\overline{\text{T x CB}}$ and $\overline{\text{R x CB}}$ are one pin, and $\overline{\text{R x T x CB}}$ and $\overline{\text{DTRB}}$ is an available output. If there is a requirement for different clock rates or phases for $\overline{\text{R x CB}}$ and $\overline{\text{T X CB}}$, they may be bonded independently by sacrificing $\overline{\text{DTRB}}$. See Figure 2.2.

ONDING OPTION
igure 2.2



Operation of the SIO is determined by the contents of the control registers. These must be programmed before any operations can be performed by the SIO. Some commands and modes may be changed during operation. The device status registers can be read at any time.

3.1 ASYNCHRONOUS MODES

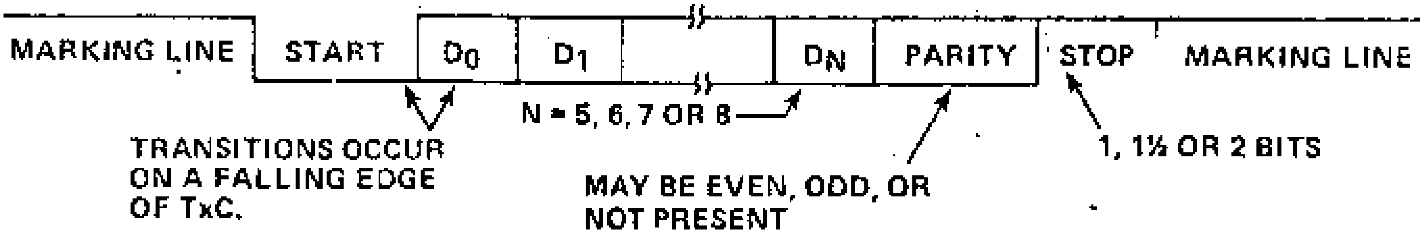
The receiver ports are quadruply buffered, i.e. there are three storage registers in addition to the input shift register. This allows additional time for the CPU to service an interrupt at the beginning of a block of high-speed data transfer. The error flags are also quadruply buffered and are loaded at the same time as the character. The RECEIVER OVERRUN and PARITY ERROR flags are not reset unless an ERROR RESET Command (Command 6) is issued. END OF FRAME and CRC/FRAMING errors always reflect the state of the character currently in the buffer and are not reset by ERROR RESET. Thus, when the error status is read, it will reflect an error in the current word in the receive buffer in addition to any parity or overrun errors received since the last ERROR RESET Command. In order to keep correspondence between the stat of the error buffer and the contents of the receive registers, the status register should be read before the data (see exception). This is easily accomplished if the vectored interrupts are used since a special interrupt vector is generated for errors or end of frame.

If the status is read after the data is read, the error data for the next data word will also be included if it has been stacked in the buffer. If operations are being performed rapidly enough so that the next character will not yet be received, then the status register will remain valid. The exception occurs when the RECEIVE INTERRUPT ON FIRST CHARACTER ONLY mode is selected. A special interrupt in this mode will hold error data and the character itself (even if read from the buffer) until the ERROR RESET, Command is issued. This prevents further data from becoming available in the receiver until the Reset is issued.

If the INTERRUPT ON EVERY CHARACTER mode is selected, the interrupt vector will be different if error states exist in the status register. If receiver overrun should occur despite the quadruple buffering, the most recent character received will be loaded. The character preceding it will be lost. When the character which has been written over other characters is read, the OVERFLOW bit will be set and the SPECIAL RECEIVE CONDITION vector returned if STATUS AFFECTS VECTOR is enabled.

It is possible to use the SIO in a polled environment. This requires monitoring of the RECEIVE CHARACTER AVAILABLE bit to know when to read a character. This bit is reset automatically when the receive buffers are all empty. The TRANSMIT BUFFER EMPTY bit is high whenever the transmit buffer is empty. In polled operation, it should be checked before writing data into the transmitter to prevent overwriting of data.

ASYNCHRONOUS FORMAT
Figure 3.0



TRANSMISSION

A data character sent by the SIO will be assembled as follows in asynchronous modes:

Idle state (no characters being sent) is a marking line (high) unless a break has been programmed in the control register, in which case, the line will remain spacing until the SEND BREAK command has been removed or the chip is reset.

Transmission cannot begin unless the TRANSMIT ENABLE bit is set. If the AUTO ENABLES bit Is selected, then $\overline{CTS}$ must be low as well. If the 5 bits/character mode is selected, then unused bits (D_5, D_6, and D_7) must be zero in each data byte written into the SIO.

RECEIVING

Asynchronous reception will begin when the RECEIVER ENABLE bit is set. If the AUTO ENABLES bit is selected, the $\overline{DCD}$ must be low as well: A low (spacing) condition on R x D indicates a start bit. If the low persists for ½ bit time, the start bit is assumed to be valid and the data input is then sampled at mid-bit time until the entire character is assembled. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line. If the X1 clock mode is selected, bit synchronization must be accomplished externally.

3.2 SYNCHRONOUS MODES

The various synchronous modes all require a x1 clock for transmission and reception. Data is sampled on the rising edge of $\overline{RxC}$. Transmitter data transitions occur on the falling edge of $\overline{TxC}$.

In all cases, the receiver is in a hunt mode after a reset (internal or external). The hunt can begin only when the receiver is enabled. Only when character synchronization has been achieved can data transfer begin. If there is a loss of character synchronization, the hunt mode can be re-entered by writing a control word with the ENTER HUNT MODE bit set.

The differences in operation of the monosync, bisync and external sync modes are only in the manner in which initial synchronization is achieved. Note: The mode of operation must be selected before the sync characters are loaded, since the registers are used differently in the various modes.

MONOSYNC; (8-BIT SYNC MODE)

Matching of a single sync character, programmed into Write register 7, implies character synchronization, which enables data transfer.

BISYNC: (16-BIT SYNC MODE)

Matching of two adjacent sync characters programmed in Write Registers 6 and 7 implies character synchronization. In both monosync and bisync modes, the $\overline{SYNC}$ pin will be active (low) any time the sync character sequence is detected and will remain low for the clock cycle in which it is detected.

EXTERNAL SYNC MODE

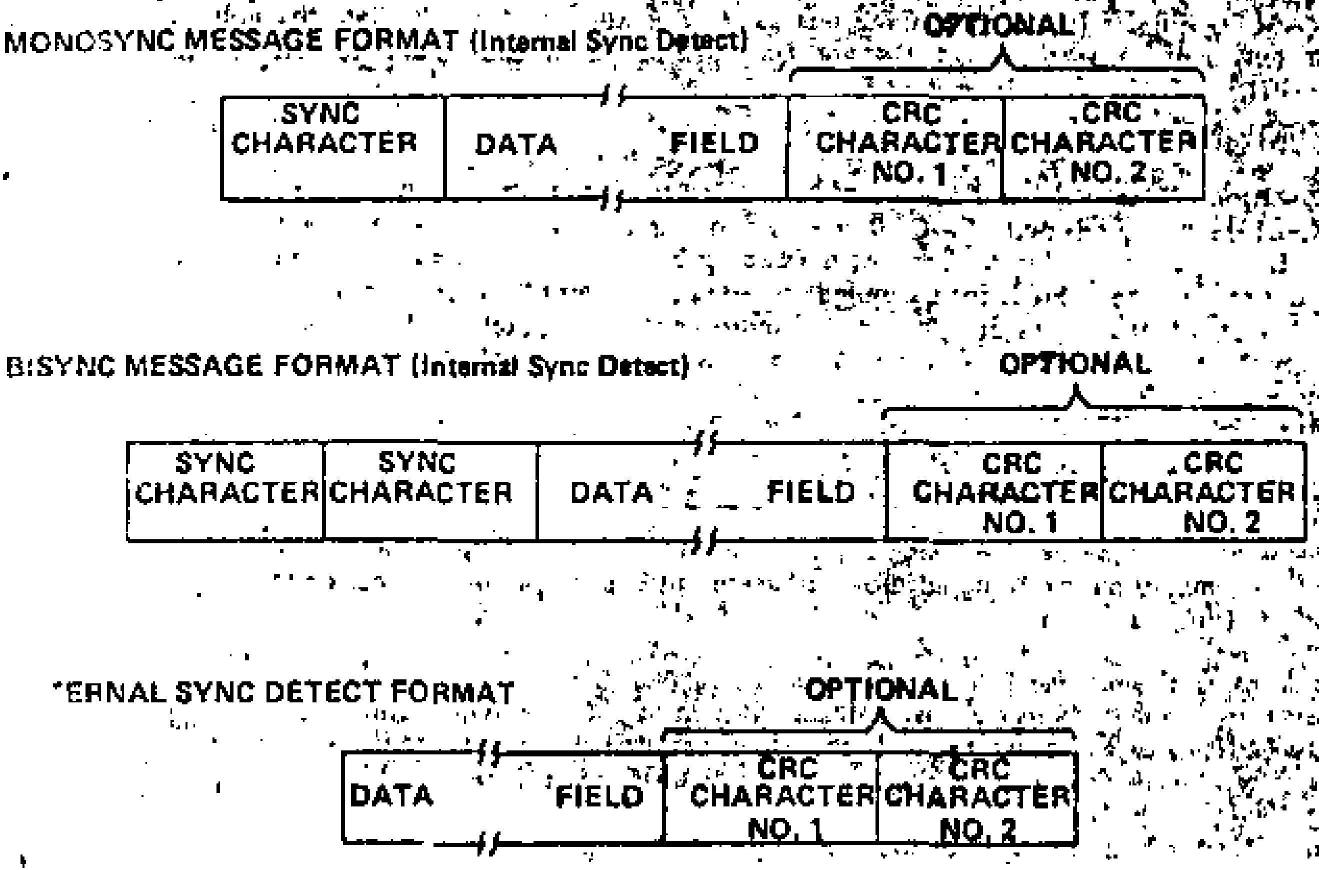
In this mode, character assembly begins on the first rising edge of $\overline{RxC}$ after the $\overline{SYNC}$ pin becomes active (low). It should be held active for at least three complete clock cycles.

In Monosync, Bisync and External sync modes, assembly will continue until the SIO is reset (either internally or with the Reset pin) or until the receiver is disabled (by command or with the DCD pin in the AUTO ENABLES mode) or until the CPU sets the ENTER HUNT MODE bit.

CC 14 [illegible] has been achieved [illegible] Monosync, Bisync, and External Sync modes are very similar. Any differences will be noted in the following which is meant to apply to all three modes.

SYNCHRONOUS FORMATS

Figure 3.1



MONOSYNC MESSAGE FORMAT (Internal Sync Detect)

BISYNC MESSAGE FORMAT (Internal Sync Detect)

EXTERNAL SYNC DETECT FORMAT

Synchronous Modes (Except SDLC) Transmission:

A.[1] Default state (after a Reset or transmitter not enabled) is a marking Line. Break may be programmed to generate a spacing line, which begins as soon as programmed, regardless of the contents of the send register. With the transmitter enabled, and after modes have been selected, default is continuous transmission of the 8 or 16 bit sync character.

B. Several Interrupt modes are possible:

1. Transmit Interrupts enabled – every time that the transmit buffer becomes empty, an interrupt will be generated if the TRANSMIT INTERRUPT ENABLE bit is set. The interrupt may be satisfied by either writing another character into the transmitter or by resetting the TRANSMITTER INTERRUPT PENDING bit with the RESET TRANSMITTER INTERRUPT PENDING command (Command 5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there will be no further transmitter interrupts, as it is the buffer becoming empty that causes the interrupt. When another character is written, the the transmitter can again become empty and interrupt again.

2. External/Status Interrupts enabled – If the EXTERNAL/STATUS INTERRUPT ENABLE bit is set, Transmitter conditions such as starting to send CRC characters, starting to send Sync characters, $\overline{DCD}$, $\overline{SYNC}$, and $\overline{CTS}$ changing state cause interrupts, which have a unique vector if STATUS AFFECTS VECTOR mode is selected.

3. All interrupts may be disabled for operation in a polled mode or to prevent interrupts at inappropriate times in a program's execution.

C. If CRC is not enabled, sync characters will automatically be inserted when the transmitter has no data to send. An interrupt is generated only after the first automatically inserted sync character has been loaded. If CRC is enabled, the first time the transmitter has no data to send, the 16-bit CRC is automatically sent, followed by sync characters. While sending CRC, the SENDING CRC/SYNC bit is set and the TRANSMIT BUFFER EMPTY bit indicates full. CRC is not calculated on the automatically inserted sync characters, but it will be calculated on any sync character sent as data unless the CRC generator is disabled when that character is loaded to the transmit shift register from the transmit buffer. When the CRC has been sent, the TRANSMIT BUFFER EMPTY bit goes high and an interrupt is generated to indicate that another message can begin. Control of the CRC generator may proceed as follows:

The CRC generator should be reset by issuing the RESET TRANSMIT CRC GENERATOR Command, before any data is loaded. After CRC and the entire transmitter is enabled, data may be loaded. Before CRC is to be sent (but after the first data has been loaded), the SENDING CRC/SYNC bit must be reset with the RESET SENDING CRC/SYNC Command.

Because sending of the CRC is inhibited when the SENDING CRC/SYNC bit is set, the SIO can be used to automatically insert fill characters within messages instead of automatically sending the CRC. CRC is not calculated on syncs automatically inserted and when the end of the message is reached, the bit can be reset thus allowing the CRC to be sent.

D. If the transmitter is disabled while a character is being sent, that character (whether Data, CRC or SYNC) will be sent as normal but will be followed by a marking line rather than CRC or sync characters. A character in the buffer when the transmitter is disabled will remain in the buffer. However, a programmed break will be effective as soon as it is written into the control register. Characters being transmitted, if any, will be lost.

E. In all modes, characters are sent low-order bits first, i.e., D_0 before D_1, etc. for as many bits as are programmed. This requires right-hand justification of data to be transmitted if word length is less than 8 bits. If word length is 5 bits or less, the special technique described in the TRANSMIT BITS/CHAR section must be used for the data format.

Synchronous Modes (Except SDLC) Reception:

A. After programming the mode and sync characters (in that order), the receiver may be enabled. It will then be in the HUNT MODE and will stay in that mode until:

1. A match is made with a single sync character (monosync mode) or

2. A match is made with a dual sync character (BiSync mode) or

3. The external $\overline{\text{SYNC}}$ pin is forced low. In cases (1) and (2) the external $\overline{\text{SYNC}}$ pin is an output which indicates that character synchronization has been achieved. In case (3) it is an input.

B. Character assembly begins after sync has been achieved. Four interrupt modes are possible.

1. NO INTERRUPTS ENABLED – for a purely polled operation or for "off line" conditions.

2. INTERRUPT ON FIRST CHARACTER ONLY. This mode would normally be used to start a software polling loop or a block transfer instruction using the WAIT/READY output to synchronize the CPU to the incoming data rate. It could also be used with a DMA device. In this mode, the SIO will interrupt on the first character and thereafter will only interrupt if errors are detected. The mode is reset with the RESET RECEIVE INTERRUPT ON FIRST CHARACTER command (Command 4).
   The first character received after this command is issued will also cause an interrupt. If External/Status interrupts are enabled, they may interrupt at any time. Parity errors do not cause interrupts in this mode, but End-of-Frame (SDLC Mode) and receiver overrun do cause interrupts.

3. INTERRUPT ON EVERY CHARACTER – whenever the receiver buffer has a character an interrupt is generated. Error and special conditions generate a special vector if the STATUS AFFECTS VECTOR mode is selected. A parity error may optionally not generate the special vector.

C. CRC checking generation may be used in the synchronous modes.

1. Calculation of the CRC on a particular character begins 8 bit times after the word has been transferred to the receive buffer. If CRC is enabled before the next character is transferred to the receive buffer, CRC will be calculated on the character. If CRC is disabled before the time of the next transfer, calculation will proceed on the word in progress, but the word just transferred to the buffer will not be included. This allows starting and stopping CRC checking on the various characters employed in BiSync.

2. The CRC may be enabled and disabled as many times as necessary for a given calculation.

3. CRC Codes are selected during the mode selection process. Either the CRC-16 polynominal $X^{16} + X^{15} + X^2 + 1$ or the SDLC polynomial $X^{16} + X^{12} + X^5 + 1$ may be used. In all except SDLC mode, the CRC calculator and checker are reset to all 0's. Transmitter and receiver must use the same polynomial.

4. In Monosync, Bisync and External Sync modes, the CRC/FRAMING ERROR bit contains the result of the comparison of the CRC checker to "all zeros" 16 bit times after the character has been loaded from the receive shift register to the buffer. The comparison is made with each load and is valid only as long as the character remains in the buffer. If time increases down the page, then the following holds:

   Character "A" loaded into the buffer

   Character "B" loaded into the buffer . . .
   If CRC is disabled before "C" is in the buffer it will not be calculated on "B".

   Character "C" loaded into buffer . . .
   After "C" is loaded the CRC/FRAMING ERROR bit shows the result of the comparison thru Character "A"

   Character "D" loaded into buffer . . .
   After "D" is in buffer, the CRC/FRAMING ERROR bit shows the result of the comparison thru Character "B".

   Because of the serial operation of the CRC calculation, the receiver clock ($\overline{RxC}$) must go through 16 cycles after the CRC character has been loaded into the receive buffer (20 cycles after the last bit is at the SIO RxD pin) before the CRC calculation is complete.

TRANSMISSION SDLC/HDLC MESSAGE FORMAT
Figure 3.2

FLAG 01111110	ADDRESS 8 BITS	DATA "I"	FIELD	CRC No. 1	CRC No. 2	FLAG 01111110

SDLC MODE TRANSMISSION:

A. Normally, the CRC generator should be reset (with the RESET TRANSMIT CRC GENERATOR command) before a data block is transmitted. Reset may occur any time after the CRC of the previous message has been sent. During the time that CRC is being sent the SENDING CRC/SYNC bit will be set, the TRANS BUFFER EMPTY bit will not be set. After the CRC has been sent the TRANS BUFFER EMPTY bit is set which will cause an interrupt signifying that the CRC has been sent, if transmit interrupts are enabled.

B. The idle device state (if the transmitter is enabled) is continuous flags being transmitted. If the transmitter is not enabled, a marking line is sent (idle line state).

C. An abort sequence may be sent by issuing the SEND ABORT command (Command 1). This causes at least 8 but less than 14 one's to be sent before the line reverts to continuous flags. Any data being transmitted and any data in the transmit buffer will be lost.

D. One to 8 bits per character may be sent. See the Register Description of Write Register 5, Transmit Bits/Char. for an explanation of how this is accomplished. Since the number of bits/character may be changed "on the fly", this feature may be used to fill a data field with any number of bits. When used in conjunction with the Receiver Residue Codes, the SIO may receive a message of any number of bits length and retransmit it exactly as received with no previous information about the character structure of the I-field (if any). A change in the number of bits/character will not affect the character in the process of being shifted out. Characters will be sent with the number of bits programmed at the time that the character is loaded from the buffer to the transmitter.

E. As in other synchronous modes, the two byte CRC sequence will be sent automatically when the transmitter has no more data to send, i.e. when there is no character in the transmit buffer and the transmit shift register is empty. When the CRC sending begins, the SENDING CRC/SYNC bit is set and a status change interrupt is generated if external/status interrupts are enabled. This may be used as a transmitter underrun indication. After the CRC has been sent, the line reverts to continuous flags, without shared zeros, i.e. . . .
0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 . . .

18 Control of the CRC generator may proceed as follows:

0. Set up necessary mode (only at initial power on)
1. Reset CRC generator
2. Write first 2 bytes of data (i.e. address and or control bytes)
3. Reset SENDING CYC/SYNC bit
4. Write rest of data
5. After data is complete, CRC & flags will be sent automatically, and this sequence can repeat from 1.

F. Extra zeros are automatically inserted in the data stream where required to fulfill the requirement of 5 ones maximum in a row, except for flags or aborts.

G. When SDLC mode is selected, Reset of the CRC generator is actually a preset to all 1's. The SDLC CRC code must be selected.

RECEPTION SDLC/HDLC MESSAGE FORMAT

Figure 3.3

FLAG 01111110	ADDRESS 8 BITS	DATA "I"	FIELD	CRC NO. 1	CRC NO. 2	FLAG 01111110

SDLC OPERATION, RECEIVER

A. Data transfer begins with the first non-flag character received after at least one flag (01111110) has been received if ADDRESS SEARCH MODE has not been enabled. If ADDRESS SEARCH MODE is enabled, then a flag followed by either the programmed address or the global address (1111111) is required before data transfer will begin.

1. If interrupts are disabled, the presence of characters in the receive buffer can be detected by observing the RECEIVE CHARACTER AVAILABLE bit in Read Register 0.
2. If the INTERRUPT ON FIRST CHARACTER ONLY mode has been selected, this would normally be used to initiate a block transfer. If the length of the message is unknown, the SPECIAL RECEIVE CONDITION (End of Frame) interrupt may be used to exit the instruction of software loop. The RESET INTERRUPT ON FIRST CHARACTER command (Command 4) must be issued before an interrupt for a following block's first character can be operated.
3. Flags are not transferred. The extra zeros inserted in transmission are automatically deleted.
4. Aborts are detected as 7 or more one's and cause a status interrupt (if enabled) with the BREAK/ABORT bit set in Read Register 0. After the RESET EXTERNAL/STATUS INTERRUPT command (Command 2) has been issued, a second interrupt will occur when the coninuous one's condition has been cleared.

B. In SDLC mode, control of the receive CRC checker is automatic. It is reset by the leading flag and CRC is calculated up to the final flag. The CRC/FRAMING ERROR bit indicates the result of the CRC check and is located in Read register 1. If the CRC/FRAMING ERROR bit is not set, then the CRC indicates a valid message. A special check sequence is used for the SDLC check because of the preset to all one's. The final check must be
0001110100001111.

C. Character length may be changed "on the fly." If address and control bytes are processed as 8-bit characters, the receiver may be switched to a smaller character length during the time that the first information character is being assembled. This change must be made quickly enough so that it is effective before the number of bits specified have been assembled, i.e., if the change is to be from the 8-bit control to a 7-bit information field character length, the change must be made before the first 7 bits of the I-field have been assembled.

D. If address search mode is not used, or if messages have multi-byte addresses, an unwanted message need not be completely read by the CPU. Once the determination has been made that the message is not needed, writing the ENTER HUNT MODE bit will suspend reception until another message headed by a flag has been received.

E. When the trailing flag is received, an interrupt with a special vector is generated (if enabled). This signals that the byte with the END OF FRAME bit set has been received. In addition to the results of the CRC check. Read Register 1 has 3 bits of Residue Code valid at this time. For those cases in which the number of bits in the I-field is not an integral multiple of the character length used, these bits indicate the boundary between the CRC check bits and the I-field bits. For a detailed description of the meaning of these bits, see the description of the Residue Codes in Read Register 1.

F. Parity checking may be used on data in the information field only if 5-7 bit characters are used and only if a half-duplex protocol is being used. (There are no separate controls for parity on the receiver and transmitter so parity cannot, for example, be simultaneously disabled for transmitting an 8-bit address and enabled for receiving a 5-bit I-field character).

20

)9

4.1 GENERAL

The Z80-SIO is a multi-function peripheral component specifically designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic role is that of a serial to parallel, parallel to serial converter/controller but within that role it is configured by systems software programming so that its function or "personality" can be optimized for a given serial data communications application.

To program the Z80-SIO the systems software issues a series of commands that initialize the basic mode of operation desired and other commands to qualify conditions within the mode selected i.e. Stop Bits, Bits/Char, Sync Char etc. The command structure of the Z80-SIO is designed to take advantage of the powerful Z80 BLOCK I/O instructions to simplify programming, minimize overhead and optimize CPU interaction activities.

Each of the two channels of the Z80-SIO contain command registers that must be programmed via system software prior to functional operation. The channel select input (B/$\overline{A}$) and the control/data input (C/$\overline{D}$) are the command structure addressing controls, normally controlled by the address bus of the Z80 CPU.

C/$\overline{D}$	B/$\overline{A}$	FUNCTION
0	0	Channel A Data
0	1	Channel B Data
1	0	Channel A Commands/Status
1	1	Channel B Commands/Status

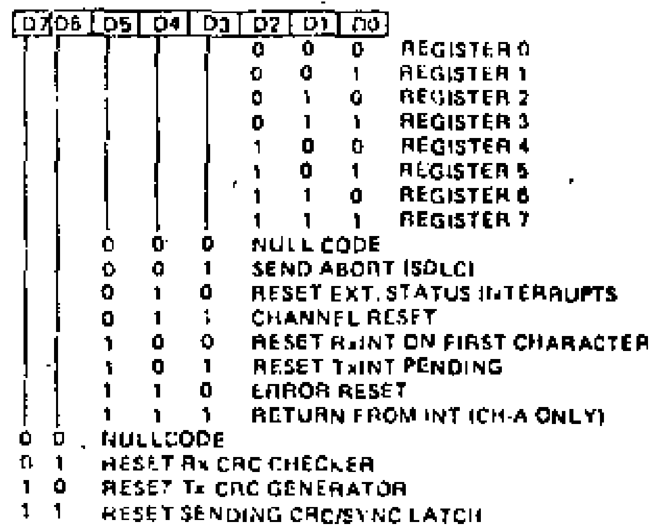
4.2 WRITE REGISTERS

The Z80-SIO contains eight (8) registers that are programmed (written into) by the system software to configure the functional personality of each channel. All Write Registers, with the exception of Write Register 0, require two bytes to be properly programmed. The first byte contains 3 bits which point to the selected register (D0-D2) the second byte is the actual control word that is being written that register to configure the SIO.

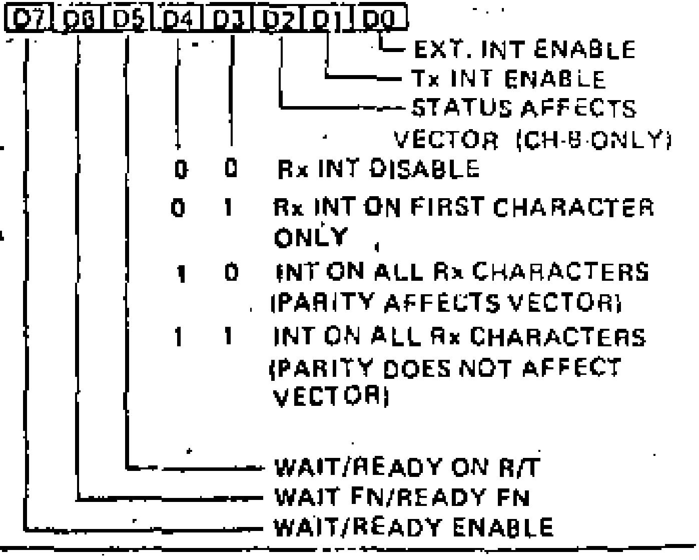
Write Register 0 is a special case. RESET (either internal command or external input) will initialize the SIO to Write Register 0. All basic commands (CMD0-CMD2) and CRC controls (CRC0, CRC1) can be accessed with a single byte using Write Register 0.

Contained in the first byte of any Write Register access are the basic commands (CMD0-CMD2) and the CRC controls (CRC0, CRC1) so that maximum system control and flexibility is maintained.

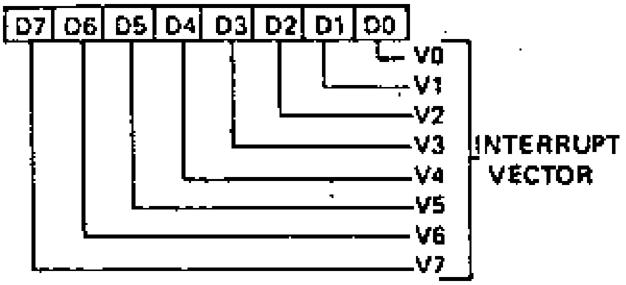
WRITE REGISTER 0 *



WRITE REGISTER 1

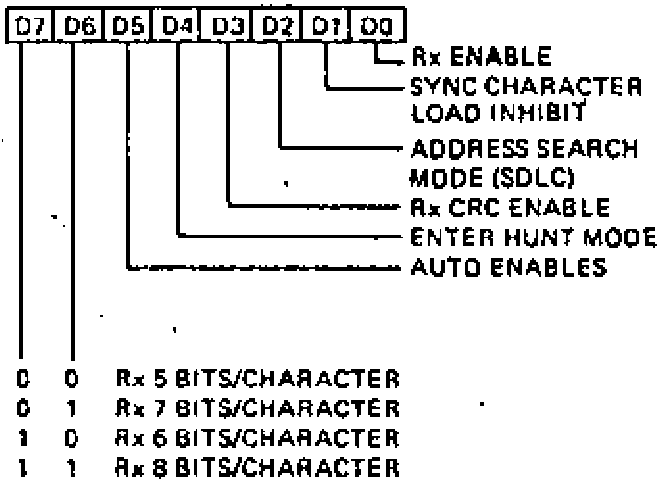

D7 D6 D5 D4 D3 D2 D1 D0
EXT. INT ENABLE
Tx INT ENABLE
STATUS AFFECTS VECTOR (CH-B ONLY)
0 0 Rx INT DISABLE
0 1 Rx INT ON FIRST CHARACTER ONLY
1 0 INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR)
1 1 INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR)
WAIT/READY ON R/T
WAIT FN/READY FN
WAIT/READY ENABLE

WRITE REGISTER 2*

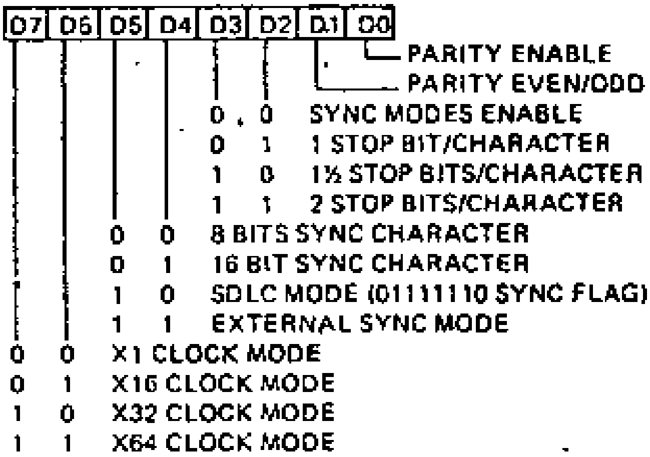


*CAN ONLY BE WRITTEN INTO CHANNEL B

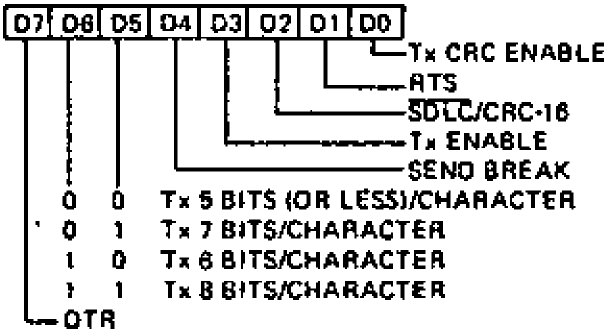
WRITE REGISTER 3



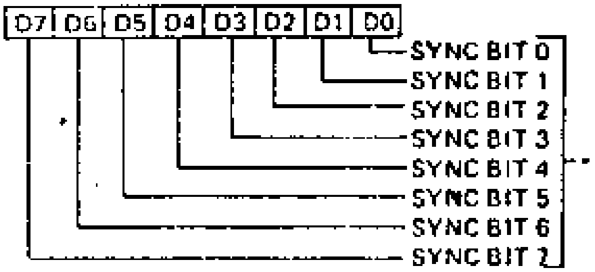
WRITE REGISTER 4



WRITE REGISTER 5

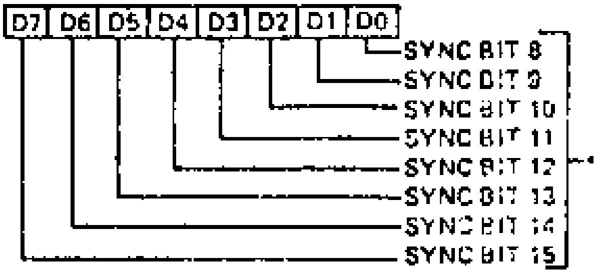


WRITE REGISTER 6



*ALSO SDLC ADDRESS FIELD

WRITE REGISTER 7



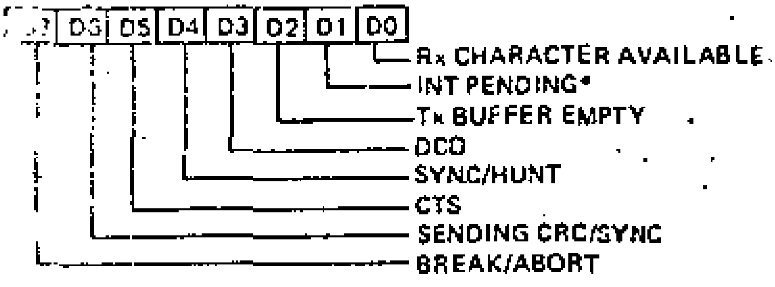
*FOR SDLC IT MUST BE PROGRAMMED
TO "01111110" FOR FLAG RECOGNITION

[illegible]

The Z80-SIO contains three (3) registers that can be read to obtain the status of each channel. Status information includes error conditions, interrupt vector, and standard communication interface protocol signals. To read the contents of a selected Read Register the system software must first write out to the SIO the byte containing pointer information (D0-D2) in exactly the same manner as a Write Register operation. Then by issuing a READ operation the contents of the addressed Read/Status Register can be read by the Z80-CPU.

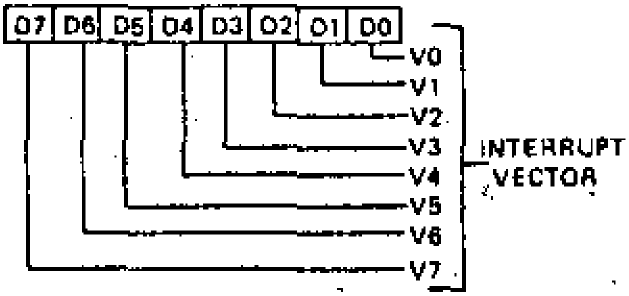
The real power in this type of command structure is that the programmer has complete freedom after pointing to the selected register of either Reading or Writing to initialize or test that register. By designing software to initialize the Z80-SIO in a modular, structured fashion, the programmer can use the powerful Z80 BLOCK I/O instructions to significantly simplify and speed his software development and debug.

READ REGISTER 0

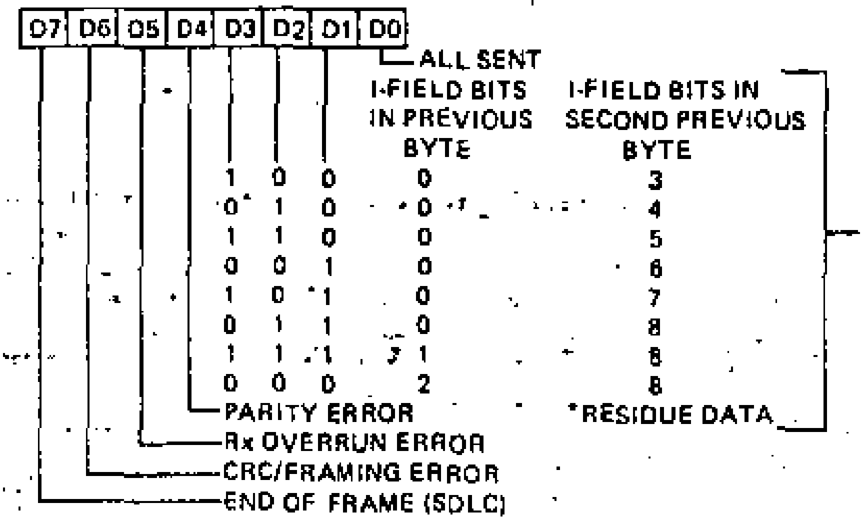


*CAN ONLY BE READ BY CHANNEL A

READ REGISTER 2 (Channel B Only)



READ REGISTER 1



D3	D2	D1	I-FIELD BITS IN PREVIOUS BYTE	I-FIELD BITS IN SECOND PREVIOUS BYTE
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

*RESIDUE DATA

4.4 REGISTER DESCRIPTION

Each channel contains the following control registers, addressed as commands (not data):

Write Register 0, a command register:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PNT 2	PNT 1	PNT 0

PNT$_0$ – PNT$_2$ (D$_0$-D$_2$)

These are pointer bits which tell the SIO into which register the following byte is to be written. The first byte written into each channel after a reset (either by command or with the external reset pin) will go to write register 0. The byte following a read or write to any register (not register 0) will be to register 0.

CMD$_0$ to CMD$_2$ (D$_3$-D$_5$)

These are commands:

Command	CMD$_2$	CMD$_1$	CMD$_0$	
0	0	0	0	Null Command (no affect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Reset Receive Interrupt on First Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A only)

COMMAND 0 (The NULL command) has no affect. It's normal use is to do nothing while setting the pointers for a following byte.

COMMAND 1 (SEND ABORT) is used only with the SDLC mode to generate a sequence of 8 to 13 ones.

COMMAND 2 (RESET EXTERNAL/STATUS INTERRUPTS). After an external or status interrupt (indicating a change on a modem line or a break condition, for example) the status bits of Read Register 0 are latched. This command reenables them and allows interrupts to occur. The latching allows capture of short pulses on the inputs until such time as the CPU can read the change.

COMMAND 3 (CHANNEL RESET). This command performs the same operation as an external reset, but only on a single channel. The Channel A Reset also resets the interrupt prioritization logic. All control registers must be rewritten after this command. After this command is written, four extra system (Φ) clock cycles should be allowed for the SIO reset time before any additional commands or controls are written into that channel of the SIO.

COMMAND 4 (RESET RECEIVE INTERRUPT ON FIRST RECEIVE CHARACTER.) If the INTERRUPT ONLY ON FIRST RECEIVE CHARACTER mode of operation is programmed, it needs to be reactivated after each complete message is received, in preparation for the next message.

COMMAND 5 (RESET TRANSMITTER INTERRUPT PENDING.) The transmitter will interrupt when it becomes empty if the ENABLE TRANSMIT INTERRUPT mode is selected. In those cases when there are no additional characters to be sent, issuing this command will prevent further transmitter interrupts (i.e. until after the next character has been loaded into the transmitter).

COMMAND 6 (ERROR RESET, LATCHES.) Parity and overrun errors are latched in Read Register 1 until reset with this command. This allows errors occurring in block transfers to be examined only at the end of the block.

[illegible]

[illegible] 24

RETURN FROM INTERRUPT.) This command (which must be issued in Channel A) is interpreted by the SIO in exactly the same way as it would interpret an RETI Command on the data bus, i.e. it would reset the Interrupt Under Service latch of the internal device (receiver, transmitter, etc.) under service and thus, by means of the daisy chain, allow lower priority devices to interrupt. The internal daisy chain may be used even in systems with no external daisy chain and no RETI Command by use of this command.

CRC RESET CODE 0 (D_6) and CRC RESET CODE 1 (D_7)

Together, these bits specify three reset modes.

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no affect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset SENDING CRC/SYNC latch

WRITE REGISTER 1 contains the control bits for the various interrupt and WAIT/READY modes.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Wait/ Ready Enable	ReadyFN/ WaitFN	W/Ready On R/T	Receive Interrupt Mode 1	Receive Interrupt Mode 0	Status Affects Vector	Trans Interrupt Enable	Ext Interrupts Enable

EXT INT ENABLE (D_0)

External Interrupt Enable, allows interrupts to occur as a result of transitions on the $\overline{DCD}$, $\overline{CTS}$ or $\overline{SYNC}$ lines or as a result of a Break Condition or the beginning of sending CRC or sync characters. $\overline{DCD}$, $\overline{CTS}$, or $\overline{SYNC}$ if not used, should be pulled up to V_{CC} to prevent spurious interrupts from occuring.

TRANS INT ENABLE (D_1)

Transmitter Interrupt Enable. If enabled, interrupts will occur whenever the transmitter buffer becomes empty.

STATUS AFFECTS VECTOR (D_2) (Channel B only)

If this mode is selected, the vector returned from an interrupt acknowledge cycle will be variable according to the following:

	V_3	V_2	V_1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch. B Receive Character Available
	0	1	1	Ch B Special Receive Condition
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition

If this bit is 0, the fixed vector programmed in the vector register is returned.

REC INT MODE 0 (D_3), REC INT MODE 1 (D_4)

Receive Interrupt Mode 0 and Receive Interrupt Mode 1 together specify the various character available conditions:

MODE	D_4 REC INT MODE 1	D_3 REC INT MODE 0	
0	0	0	Receiver interrupts disabled
1	0	1	Receive interrupt on first character only
2	1	0	Interrupt on all Receive Characters-Parity affects Vector
3	1	1	Interrupt on all Receive Characters-Parity error does not affect Vector.

W/READY on R/T (D_5)

When the W/Ready line is enabled, this bit selects whether it will be active when the receiver is empty (bit=1) or when the transmit buffer is full (bit =0).

READY FN/WAIT FN (D_6)

When used with the CPU as a Wait line, this bit should be programmed "0". When used with a DMA as a Ready line, it must be programmed "1". The ready function can occur any time, regardless of whether the SIO is addressed or not. The Wait function is active only if the CPU attempts to read SIO data that has not yet been received, as would frequently occur if block transfer instructions are used with the SIO, or tries to write data while the transmit buffer is still full.

Also, as a Wait function, the output is open drain and occurs from the negative edge of Φ. As a Ready function, it is actively driven high and occurs from the positive edge of Φ.

WAIT/READY ENABL (D_7)

The Wait/Ready pin will remain high (Ready mode) or floating (Wait mode) until this bit is programmed to one.

WRITE REGISTER 2 (Channel B only)

Write Register 2 is the interrupt vector register and it exists only in Channel B. V_4-V_7 and V_0 are always returned exactly as written. V_1-V_3 are returned as written if the "Status Affects Vector". Control bit is "0".

WRITE REGISTER 3

Write register 3 contains control bits for some of the receiver logic.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
RCVR Bits/ Char 0	RCVR Bits/ Char 1	Auto Enables	Enter Hunt Mode	RECVR CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

RECEIVER ENABLE (D_0)

A "1" programmed here allows receiver operations to begin.

Sync characters preceding a message will not be loaded into the receiver buffers if this option is selected. The CRC calculation is not stopped by the sync character being stripped.

ADDRESS SEARCH MODE (D_2)

If the SDLC mode is selected, this mode will cause messages with addresses not matching the programmed address or the global (11111111) address to be rejected, i.e., no interrupts occur unless an address match occurs if this mode is selected.

RECVR CRC ENABLE (D_3)

Receiver CRC Enable. If this bit is set, a calculation of CRC begins (or restarts) at the start of the last character transferred from the receive register to the buffer stack regardless of the number of characters in the stack.

ENTER HUNT MODE (D_4)

If character synchronization is lost for any reason, or if in SDLC mode, it is determined that the contents of an incoming message are not needed, Hunt mode may be reentered by writing a "1" to this bit.

AUTO ENABLES (D_5)

If this mode is selected, the $\overline{DCD}$ and $\overline{CTS}$ inputs are receiver and transmitter enables, respectively. If the mode is not selected, $\overline{DCD}$ and $\overline{CTS}$ are only inputs to their corresponding bits in Read Register 0.

RCVR BITS/CHAR 1 (D_6), RCVR BITS/CHAR 0 (D_7)

These bits together determine the number of serial receive bits that will be assembled to form a character.

These bits may be changed during the time that a character is being assembled, if it is done before the number of bits currently programmed is reached.

D7 Receiver Bits/Character 1	D_6 Receiver Bits/Character 0	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

WRITE REGISTER 4

Write Register 4 contains control bits affecting both the receiver and transmitter.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate 1	Clock Rate 0	Sync Modes 1	Sync Modes 0	Stop Bits 1	Stop Bits 0	Parity Even/$\overline{\text{Odd}}$	Parity

PARITY (D_0)

If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data.

PARITY EVEN/$\overline{\text{ODD}}$ (D_1)

If parity is specified, this bit determines whether it is sent or checked as even or odd parity.

STOP BITS 0 (D_2), STOP BITS 1 (D_3)

These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit.

The special (00) mode is used to signify that a synchronous mode is to be selected.

D_3 Stop Bits 1	D_2 Stop Bits 0	
0	0	Sync Modes
0	1	1 Stop Bit Per Character
1	0	1½ Stop Bits Per Character
1	1	2 Stop Bits Per Character

SYNC MODES 0 (D_4), SYNC MODES (D_5)

These select the various options for character synchronization:

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC Mode (01111110 sync pattern)
1	1	External Sync Mode

CLOCK RATE 0 (D_6), CLOCK RATE 1 (D_7)

Specifies the multiplier between clock and data rates. For synchronous modes X1 must be specified. Any rate may be specified for the asynchronous modes. The same multiplier is used for both the receiver and transmitter.

In all modes, the system clock (Φ) must be at least 4.5 X the data rate. If the X1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate X 1 = Clock Rate
0	1	Data Rate X16 = Clock Rate
1	0	Data Rate X32 = Clock Rate
1	1	Data Rate X64 = Clock Rate

WRITE REGISTER 5

Write Register 5 contains mostly control bits affecting the transmitter.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Transmit Bits/ Char 0	Transmit Bits/ Char 1	Send Break	Transmit Enable	$\overline{\text{SDLC}}$/ CRC16	RTS	Transmit CRC Enable

TRANSMIT CRC ENABLE (D_0)

This bit determines whether CRC is to be calculated on any particular send character. If set at the time of loading the character from the transmit buffer to the transmit shift register, CRC will be calculated on the character. CRC will not be automatically sent unless this bit is set when the transmitter is completely empty.

28 2

Request to Send is the control bit for the $\overline{RTS}$ pin. When the $\overline{RTS}$ bit is set, the $\overline{RTS}$ goes active (low). When the bit is reset (to 0), the $\overline{RTS}$ pin will go inactive (high) only after the transmitter is empty.

$\overline{SDLC}$/CRC16 (D_2)

This bit selects the CRC code used by both the transmitter and the receiver. When reset, the SDLC polynomial $X^{16} + X^{12} + X^5 + 1$ is used. (In SDLC mode, the registers are preset to "all 1's" and a special check sequence is used.) When set, the CRC-16 polynomial $X^{16} + X^{15} + X^2 + 1$ is used, and the CRC registers are reset to "all 0's".

TRANSMIT ENABLE (D_3)

Data will not be transmitted and the TxD pin will be held marking (high) until this bit is set. Data or Sync characters in the process of being transmitted will be completely sent if the transmit enable bit is reset after transmission has started. CRC characters will not be completely sent if the transmitter is disabled during the sending of a CRC character.

SEND BREAK (D_4)

When set, this bit directly forces the TxD pin spacing, regardless of any data being transmitted. When reset, the TxD pin is released.

TRANSMIT BITS/CHAR 0 (D_5), TRANSMIT BITS/CHAR 1 (D_6)

These bits together control the number of bits that will be sent from each byte transferred to the transmit buffer.

D_6	D_6	
Transmit Bits/ Character 1	Transmit Bits/ Character 0	Bits/Character
0	0	5 or less
0	1	7
1	0	6
1	1	8

Bits to be sent are assumed to be right justified. Low order bits (D_0) are sent first. The "5 or less" mode allows transmission of 1 to 5 bits in a character.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one bit
1	1	1	0	0	0	D	D	Sends two bits
1	1	0	0	0	D	D	D	Sends three bits
1	0	0	0	D	D	D	D	Sends four bits
0	0	0	D	D	D	D	D	Sends five bits

D=DATA BIT

DTR (D_7)

Data Terminal Ready is the control bit for the $\overline{DTR}$ pin. When set, $\overline{DTR}$ is active (low). When reset (0) DTR is inactive (high).

WRITE REGISTER 6

This register contains the first 8 bits of a BiSync sequence. It must be programmed with the check address (if used) in SDLC mode, and must contain the sync character in the 8-bit sync mode. It is not used in the external sync mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SYN7	SYN6	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0	MONO OR BI SYNC MODE
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	SDLC MODE

WRITE REGISTER 7

This register contains the second byte of a 16-bit synchronization sequence, or the 8-bit sync character. For SDLC mode, it must be programmed to 01111110 . It is not used in the external sync mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SYN15	SYN14	SYN13	SYN12	SYN11	SYN10	SYN9	SYN8	BI SYNC MODE
0	1	1	1	1	1	1	0	SDLC MODE

READ REGISTER 0

This is the register read if the register pointers are (000).

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break/ Abort	Sending CRC/ Syncs	CTS	Sync/ Hunt	DCD	Transmit Buffer Empty	Interrupt Pending	Receive Character Available

RECEIVE CHARACTER AVAILABLE (D_0)

This bit is set when at least one character is available in the receive buffers.

INTERRUPT PENDING (D_1) (Channel A only)

Any interrupt condition present in the entire SIO will cause this bit to be set, but it is present only in Channel A and is always 0 in Channel B.

TRANSMIT BUFFER EMPTY (D_2)

The Transmit Buffer Empty bit is set whenever the transmit buffer is empty, except when a CRC character is being sent in a synchronous mode.

DCD (D_3)

Shows the state of the $\overline{\text{DCD}}$ pin at the time of the last change of any of the five External/ Status bits. (DCD, CTS, SYNC/HUNT, BREAK/ABORT or SENDING CRC/SYNC.) To get the current state of the DCD pin, this bit must be read immediately following a RESET EXTERNAL/STATUS INTERRUPT command. (Command 2.)

SYNC/HUNT (D_4)

In asynchronous modes, this bit is similar to the $\overline{\text{DCD}}$ and the $\overline{\text{CTS}}$ bits, except that it shows the state of the $\overline{\text{SYNC}}$ pin. In synchronous modes, this bit is reset when character synchronization is achieved and is set by writing the ENTER HUNT MODE bit. Unlike the external pin, the bit remains reset until set by the ENTER HUNT MODE bit.

CTS (D_5)

This bit is similar to the $\overline{\text{DCD}}$ bit, except that it shows the state of the $\overline{\text{CTS}}$ pin.

In asynchronous modes, this bit is set when a "break" is detected. After the inputs have been re-enabled (by the RESET EXTERNAL/STATUS INTERRUPTS command, Command 2), the bit will be reset when the break stops. If EXTERNAL STATUS interrupts are enabled, these changes of state cause interrupts. In SDLC mode, this bit is set by the detection of an abort sequence (7 or more 1's). It is not used in other synchronous modes.

SENDING CRC/SYNCS (D_6)

In synchronous modes, CRC is automatically sent when the transmitter is empty for the first time in a message. Interrupts are generated (if enabled) when this bit is set, but not when reset. If this bit is set and the TRANSMIT BUFFER EMPTY bit is not set, then the CRC character is being sent. TRANSMIT BUFFER EMPTY and SENDING CRC/SYNC both set imply that SYNC characters are being sent.

READ REGISTER 1

This register is read when the register pointers are (001). The pointers automatically reset to (000) after a read from this register.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End Of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

ALL SENT (D_0)

In asynchronous modes, this bit is set when all characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. It is always set in synchronous modes.

RESIDUE CODE 0 (D_1) – RESIDUE CODE 2 (D_3)

These three bits indicate the length of the I-field in the SDLC mode in those cases where the I-field is not an integral multiple of the character length used. Only on the transfer on which the END OF FRAME (SDLC) bit is set do these codes have meaning.

For a receiver setting of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field In Previous Byte	I-Field In Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

I-field bits are right-justified in all cases.

If a receive character length different from eight bits is used for the I-field, a table similar to the above may be constructed for each different character length. For no residue, i.e., the last character boundary coincides with the boundary of the I-Field and CRC Field, the Residue Code will always be:

Residue Code 2	Residue Code 1	Residue Code 0
0	1	1

PARITY ERROR (D_4)

When parity is enabled, this bit is set for those characters whose parity does not match the sense programmed. The bit is latched so that once an error occurs, the bit remains set until the ERROR RESET COMMAND, Command 6, is given.

RECEIVER OVERRUN ERROR (D_5)

This indicates that more than four characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the ERROR RESET COMMAND, Command 6. If STATUS AFFECTS VECTOR bit is enabled, the character that has been overrun will interrupt with the SPECIAL RECEIVE CONDITION vector.

CRC/FRAMING ERROR (D_6)

If a framing error occurs (in asynchronous modes), this bit is set (and not latched) only for the character on which it occurred. Detection of a framing error adds an additional ½ bit time to the character time so that the framing error will not also be interpreted as a new start bit. In synchronous modes, this bit indicates the result of comparing the CRC checker to the appropriate check value.

END OF FRAME (SDLC) (D_7)

In SDLC mode, this bit indicates that a valid ending flag has been received and that the CRC error and residue codes are valid.

READ REGISTER 2 (Channel B Only)

This register contains the interrupt vector as written into Write Register 2 if the STATUS AFFECTS VECTOR control bit is not set. If that control bit is set, it contains the interrupt vector as it would be returned were an interrupt from the SIO to be processed exactly at the time of the read. If no interrupts are pending, $V_3 = 0$, $V_2 = 1$, $V_1 = 1$ and other bits are as programmed. The register may be read only through Channel B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3 *	V_2 *	V_1 *	V_0

*V_1, V_2, and V_3 are varible if STATUS AFFECTS VECTOR mode is enabled

4.5 Z80-SIO COMMAND STRUCTURE

Reg.	Control			DATA BITS							
[illegible]	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	0	0	0
	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	0	0	0
	1	0	1	Break/Abort	SENDING CRC/SYNC	CTS	SYNC/HUNT	DCD	TxBuffer EMPTY	INT Pending (CH A Only)	RxChar Avail
1	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	0	0	1
	1	1	0	Wait/RDY EN	Wait FN/RDY FN	Wait/RDY on R/T	RxINT Mode 1	RxINT Mode 0	Status Effects V (CH B Only)	TxINT EN	EXT INT EN
	1	0	1	End of Frame SDLC	CRC Framing Error	RxOVRN Error	Parity Error	Res. Code 2	Res. Code 1	Res. Code 0	All Sent

CHB ONLY

[illegible]	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	0	1	0
	1	1	0	V7	V6	V5	V4	V3	V2	V1	V0
	1	0	1	V7	V6	V5	V4	V3	V2	V1	V0
[illegible]	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	0	1	1
	1	1	0	RxBits/Char 1	RxBits/Char 0	Auto Enables	EnterHuntMode	RxCRC EN	AddressSearchMd	SyncChar LD INH	RxEN
[illegible]	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	1	0	0
	1	1	0	Clock Rate 1	Clock Rate 0	Sync Mode 1	Sync Mode 0	Stop Bits 1	Stop Bits 0	Parity Even/$\overline{\text{Odd}}$	Parity
6	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	1	0	1
	1	1	0	DTR	TxBits/Char 1	TxBits/Char 0	Send BREAK	TxEN	$\overline{\text{SDLC}}$/CRC 16	RTS	TxCRC EN
6	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	1	1	0
	1	1	0	SYNC/SDLC 7	SYNC/SDLC 6	SYNC/SDLC 5	SYNC/SDLC 4	SYNC/SDLC 3	SYNC/SDLC 2	SYNC/SDLC 1	SYNC/SDLC 0
7	1	1	0	CRC 1	CRC 0	CMD 2	CMD 1	CMD 0	1	1	1
	1	1	0	SYNC/SDLC 15	SYNC/SDLC 14	SYNC/SDLC 13	SYNC/SDLC 12	SYNC/SDLC 11	SYNC/SDLC 10	SYNC/SDLC 9	SYNC/SDLC8

4.6 PROGRAMMING EXAMPLE

A typical start-up routine following an internal or external reset, would be as follows:

B/$\overline{\text{A}}$	C/$\overline{\text{D}}$	$\overline{\text{RD}}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	COMMENTS
1	1	1	0	0	0	0	0	0	1	0	Pointer set to Register 2B
1	1	1	V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0	Interrupt Vector loaded
1	1	1	0	0	0	0	0	1	0	0	Pointer set to Write Register 4B
1	1	1	0	1	X	X	0	1	1	1	Even parity, 1 stop bit, X16 clock asynchronous mode selected
1	1	1	0	0	0	0	0	1	0	1	Pointer set to Write Register 5B
1	1	1	0	0	1	0	1	0	1	0	7 bits/transmit character, transmitter
1	1	1	0	0	0	0	0	0	1	1	Pointer set to Write Register 3B
1	1	1	0	1	1	0	0	0	0	1	7 bits/receive character, DCD and CTS enable Receiver and Transmitter, Receive enabled
1	1	1	0	0	0	0	0	0	0	1	Pointer set to Register 1B
1	1	1	0	0	0	1	0	1	1	1	Interrupt on every character, status affect Vector external/status interrupts enabled

Channel B is now setup to send and receive asynchronous data.

tup for Channel A follows:

0	1	1	0	0	0	0	0	1	0	0	Pointer set to Write Register 4A
0	1	1	0	0	1	0	0	0	0	0	SDLC mode and X1 clock selected, no parity

Programming Example

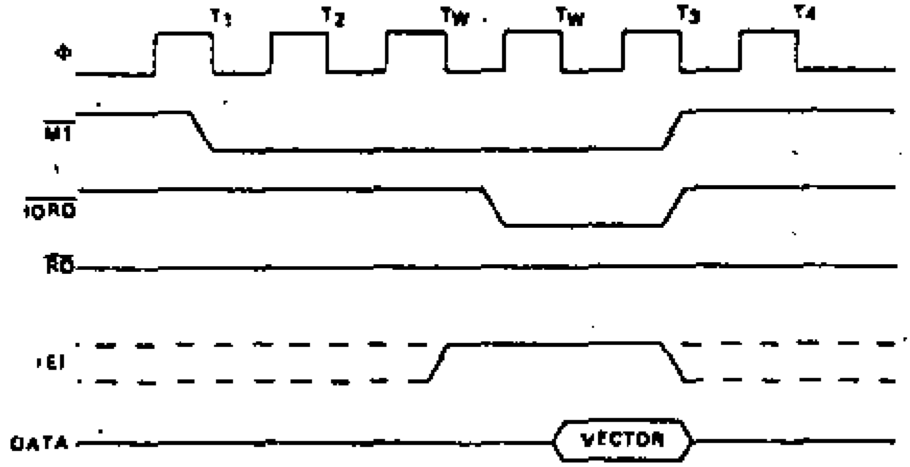
B/$\overline{A}$	C/$\overline{D}$	$\overline{RD}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	COMMENTS
0	1	1	0	1	0	0	0	1	1	0	Pointer set to Write Register 6A, Reset Receive CRC Checker
0	1	1	AD_7	AD_6	AD_5	AD_4	AD_3	AD_2	AD_1	AD_0	SDLC message address entered
0	1	1	1	0	0	0	0	1	1	1	Pointer set to Write Register 7A, Reset mit CRC generator
0	1	1	0	1	1	1	1	1	1	0	SDLC Flag entered
0	1	1	0	0	0	0	0	0	0	1	Pointer set to Register 1A
0	1	1	0	0	0	1	0	1	1	1	Interrupt every character, status affects vector, external/status interrupts enabled
0	1	1	0	0	0	1	0	1	0	1	Pointer set, to Write Register 5A, Reset External/Status Interrupts
0	1	1	1	1	1	0	1	0	0	0	SDLC CRC Code selected, 8 bits/transmit character, CRC and transmitter enabled
0	1	1	0	0	0	0	0	0	1	1	Pointer set to Write Register 3A
0	1	1	1	1	1	0	1	1	0	1	8 bits/receive character, DCD and CTS enable receiver and transmitter, receiver is enabled, SIO searches for programmed address.

Channel A is now programmed for SDLC transfers.

0	0	1	D	D	D	D	D	D	D	D	Address byte to be sent by Ch. A
0	0	1	D	D	D	D	D	D	D	D	Address or control byte to be sent by Ch.
0	1	1	1	1	1	0	0	0	0	0	Reset SENDING CRC/SYNC pointer to register 0, so CRC can be automatically sent at end of message.

INTERRUPT ACKNOWLEDGE CYCLE

Sometime after an interrupt is requested by the SIO, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$.) During this time, the interrupt logic of the SIO will determine the highest priority function which is requesting an interrupt. To insure that the daisy chain enable lines stabilize, channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active (low). If the SIO is the highest priority device requesting an interrupt, the SIO will place the appropriate interrupt vector on the data bus when $\overline{IORQ}$ goes active.

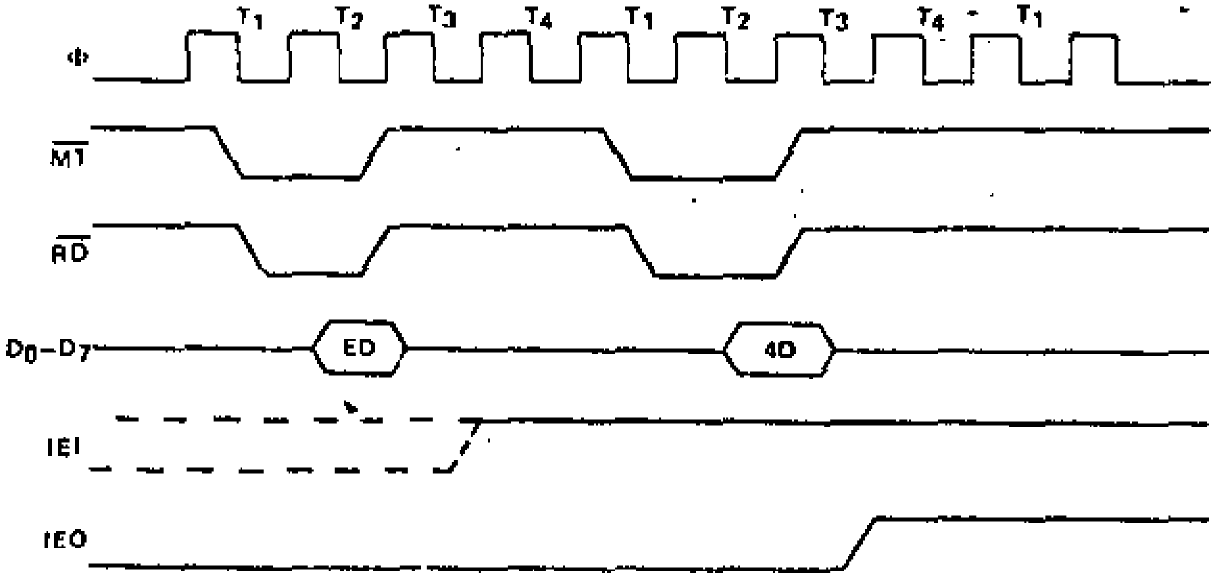


RETURN FROM INTERRUPT CYCLE

If a Z80 peripheral device has no interrupt pending and is not under service, then its IEO=IEI. If it has an interrupt under service (i.e. it has already interrupted and received an interrupt acknowledge) then its IEO is always low, inhibiting lower priority chips from interrupting. If it has an interrupt pending which has not yet been acknowledged, IEO will be low unless an "ED" is decoded as the first byte of a two byte opcode. In this case, IEO will go high until the next opcode byte is decoded, whereupon it will again go low. If the second byte of the opcode was a "4D" then the opcode was an RETI instruction.

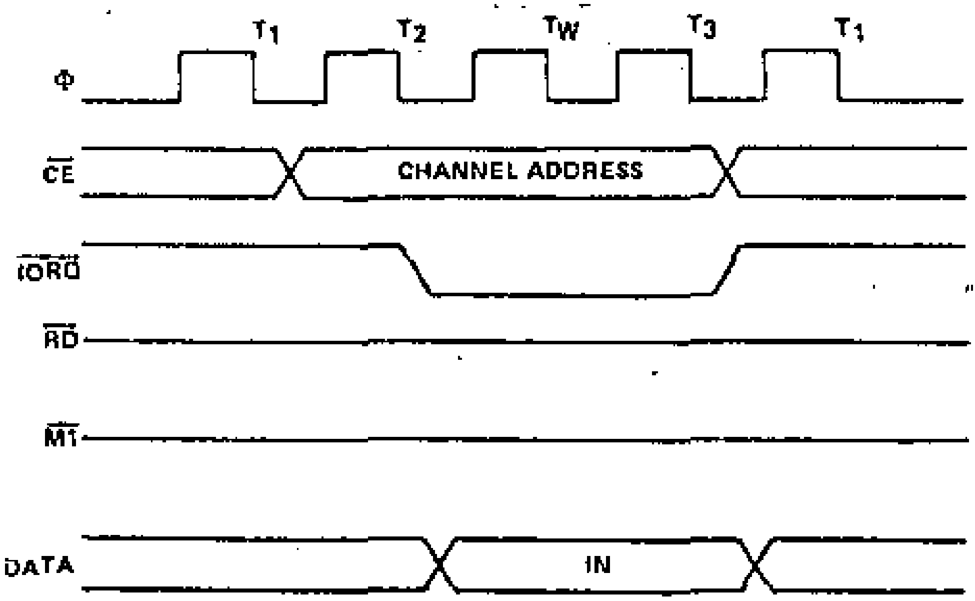
After an "ED" opcode is decoded, only the peripheral device which has interrupted and is currently under service will have its IEI high and its IEO low. This device is the highest priority device in the daisy chain which has received an interrupt acknowledge. All other peripherals have IEI=IEO. If the next opcode byte decoded is "4D", this peripheral device will reset its " interrupt under service" condition.

Wait cycles are allowed in the $\overline{M1}$ cycles.



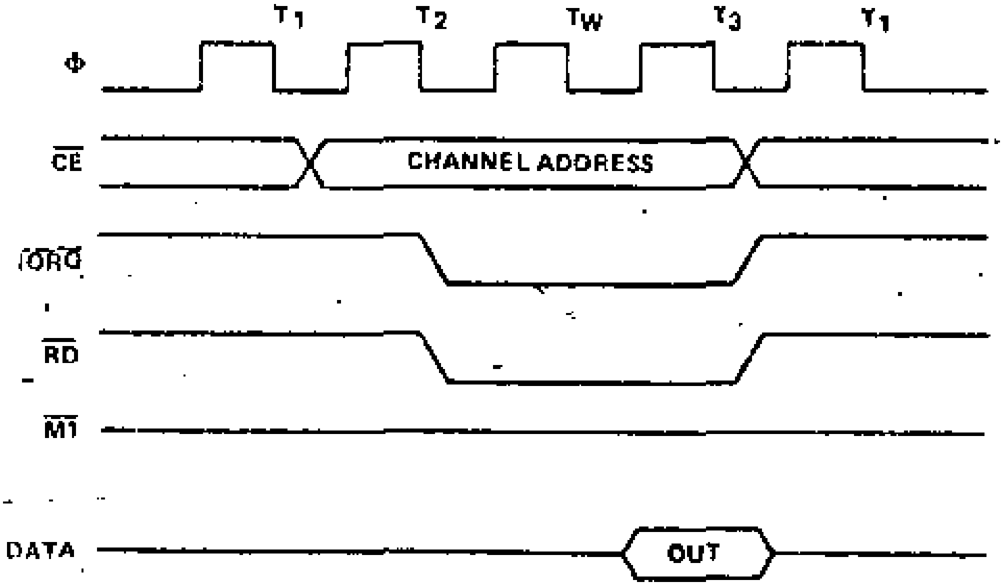
WRITE CYCLE

Illustrated here is the timing associated with a data or control byte being written into the SIO. Z80 output instructions satisfy this timing.



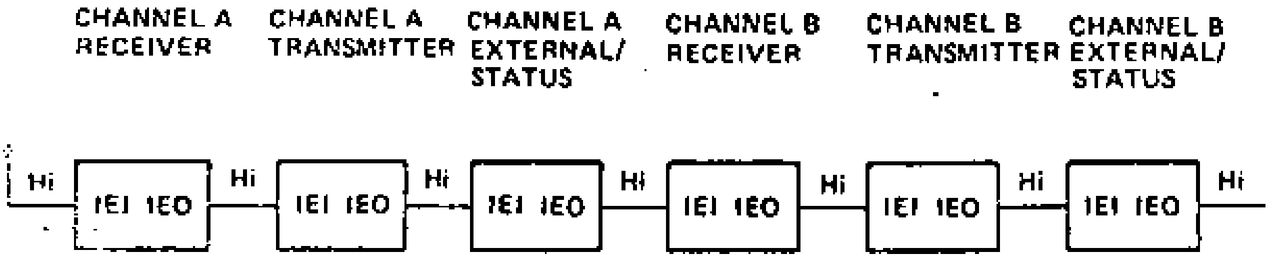
READ CYCLE

The timing associated with reading data or a status register within the SIO is illustrated here. Z80 Input instructions satisfy this timing.

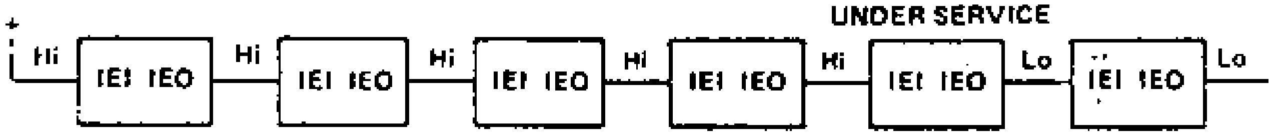


26 The following illustrates a typical nested interrupt sequence which may occur in the SIO. In a system with several peripheral chips, the other chips may be included in the daisy chain with either higher or lower priority than the SIO channels.

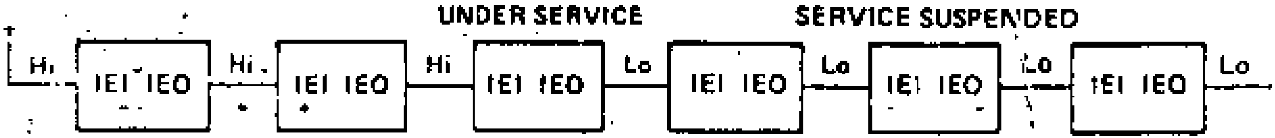
In this sequence, the transmitter of Channel B interrupts and is granted service. While it is being serviced, an external/status interrupt from Channel A occurs and is granted service. The service routine for the Channel A interrupt is completed and either the RETI instruction is executed or the RETI command is written into the SIO to indicated to Channel A that the external/status interrupt routine is complete. At this time, the service routine for the Channel B transmitter is resumed. When this routine is completed, another RETI instruction is executed to complete the service.



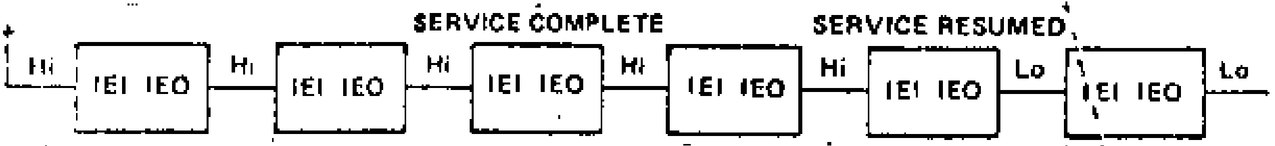
1. Priority Interrupt Daisy Chain before any interrupt occurs.



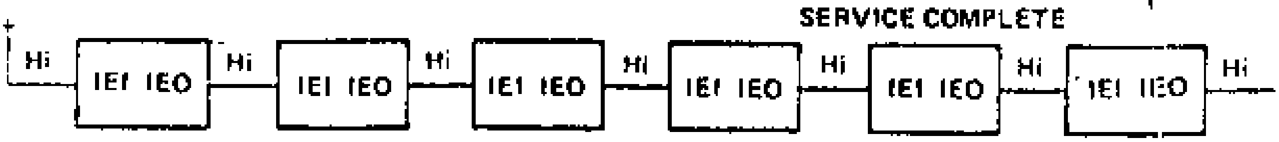
2. Channel B's transmitter interrupts and is acknowledged.



3. External/Status of Channel A interrupts suspending service of Channel B transmitter.



4. Channel A External/Status routine complete, RETI issued, Channel B transmitter service resumed.



5. Channel B transmitter's service routine complete, second RETI issued.

7.0 ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to GND	−0.3V to +7V
Operating Temperature (Ambient) T_A	0°C to 70°C
Storage Temperature - Ceramic (Ambient)	−65°C to +150°C
Storage Temperature — Plastic (Ambient)	−55°C to +125°C
Power Dissipation	1.5W

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

7.1 D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified.

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		.40	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.2		V_{CC}	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = 250μA
V_{CC}	Power Supply Current			140	mA	t_c = 400 nsec
I_{LI}	Input Leakage Current			10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT} = 0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{CC}$

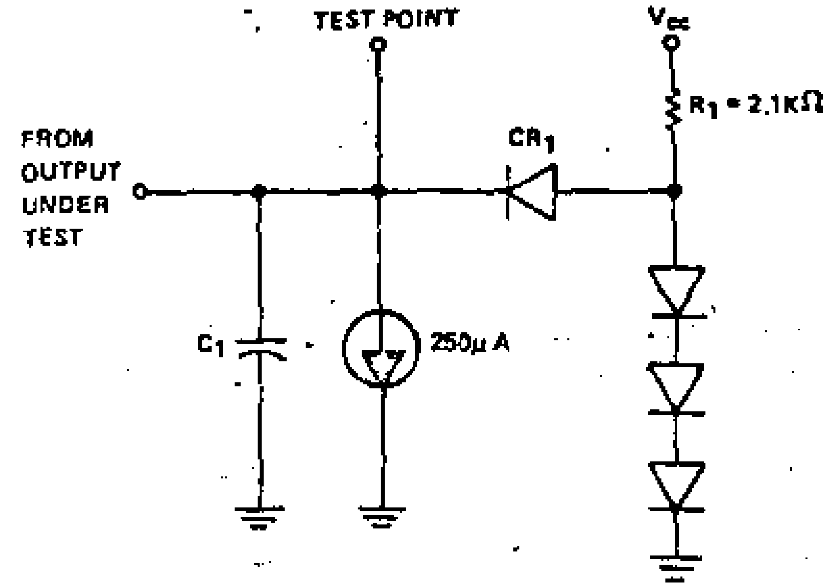
7.2 CAPACITANCE

T_A = 25°C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_{ϕ}	Clock Capacitance	35	pF	Unmeasured Pins
C_{IN}	Input Capacitance	5	pF	Returned to Ground
C_{OUT}	Output Capacitance	10	pF	

[illegible] 39

7.7 LOAD CIRCUIT FOR OUTPUT



A. C. TIMING DIAGRAM

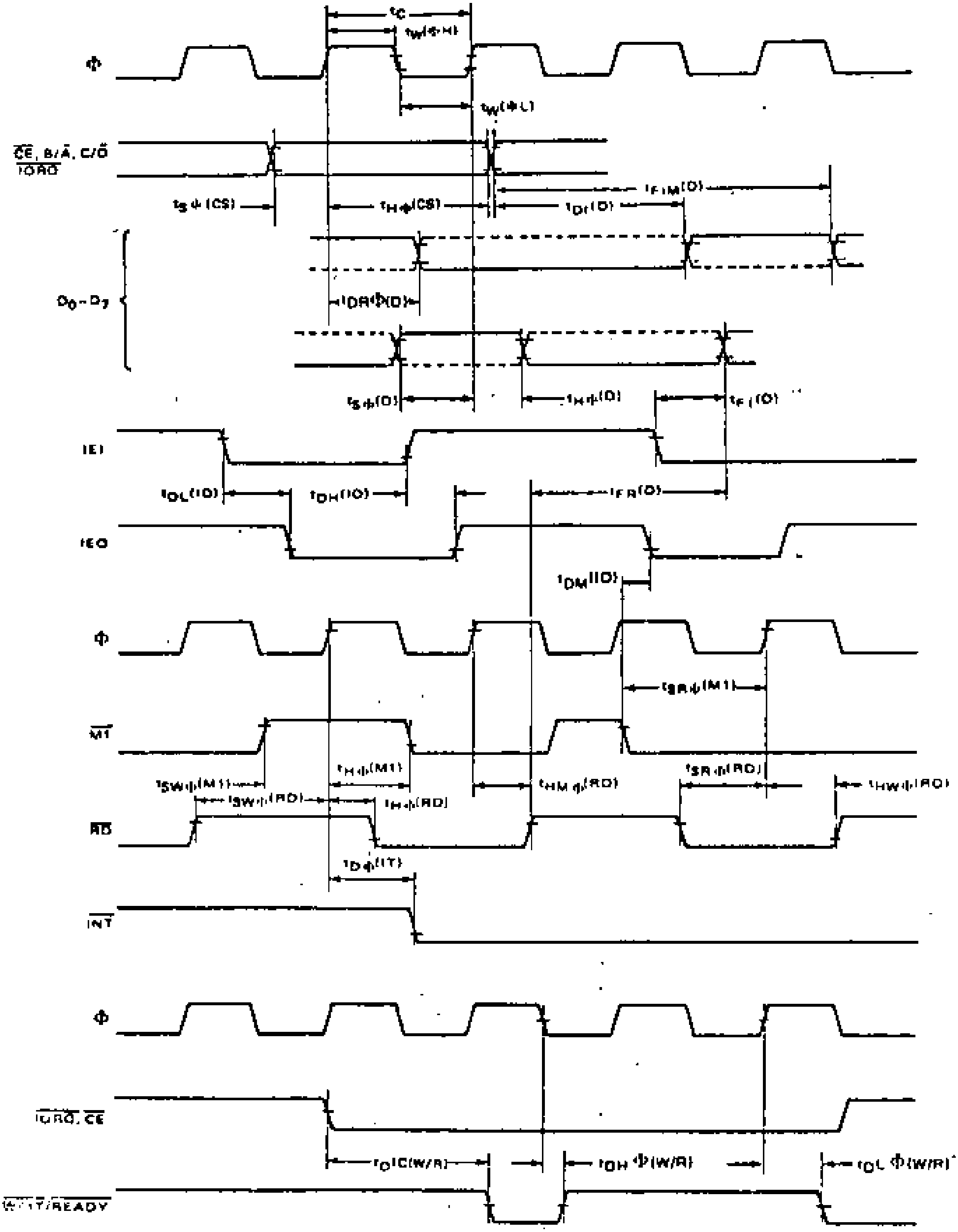
Figure 7.4

PRELIMINARY

Timing measurements are made at the following voltages, unless otherwise specified:

Only for timing measurements

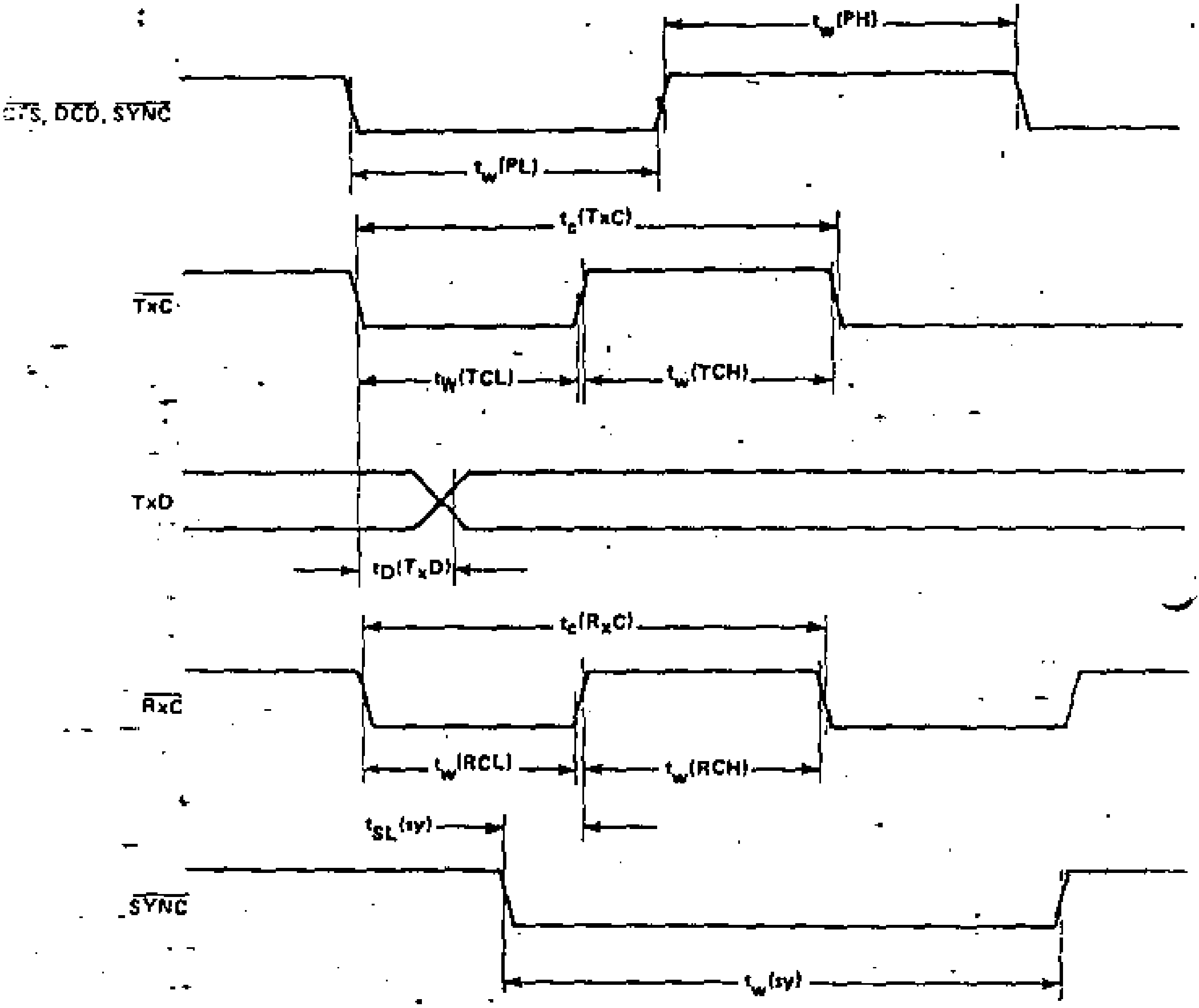
CLOCK	4.2V		.8V
OUTPUT	2.0V		.8V
INPUT	2.0V		.8V
FLOAT	ΔV	=	±0.5V



[illegible] 39

A. C. TIMING DIAGRAM (Cont'd.)

Figure 7.4



A. C. CHARACTERISTICS

PRELIMINARY

Figure 7.5

T_s = 0°C to 70 T_A = 0°C to 70°C, V_{cc} = +5V ±5%, unless otherwise noted

Signal	Symbol	Parameter	Min	Max	Unit	Comments
Φ	$t_c(\Phi)$	Clock Period	400		nsec	
	$t_w(\Phi H)$	Clock Pulse Width, Clock High	170	2000	nsec	
	$t_w(\Phi L)$	Clock Pulse Width, Clock Low	170	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times	-0-	30	nsec	
CE, B/A	$t_H(\Phi CS)$	Control signal hold time from Rising Edge of Φ	-0-		nsec	NOTE 1
C/D, IORQ	$t_S(\Phi CS)$	Control Signal setup time from Rising Edge of Φ	180		nsec	
D0-D7	$t_{DR}(\Phi D)$	Data Output Delay from Rising Edge of Φ during Read Cycle		480	nsec	
	$t_{S}\Phi(D)$	Data Setup Time to Rising Edge of Φ during Write Cycle or M1 Cycle	50		nsec	
	$t_H\Phi(D)$	Data Hold Time from Rising Edge of Φ during Write Cycle or M1 Cycle	-0-		nsec	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of IORQ during INTA Cycle		340	nsec	
	$t_{FI}(D)$	Delay to Floating Bus from Rising Edge of IORQ during INTA Cycle		230	nsec	
	$t_{FR}(D)$	Delay to Floating Bus from Rising Edge of RD during Read Cycle		230	nsec	
	$t_{FI}(D)$	Delay to Floating Bus from Falling Edge of IEI during INTA Cycle		230	nsec	
IEO	$t_{DL}(IO)$	IEO Delay Time from Falling Edge of IEI		200	nsec	
	$t_{DH}(IO)$	IEO Delay Time from Rising Edge of IEI		200	nsec	
	$t_{DM}(IO)$	IEO Delay Time from Falling Edge of M1 (when interrupt occurs just prior to M1)		300	nsec	
M1	$t_{SW}\Phi(M1)$	M1 Setup Time to Rising Edge of Φ during Read or Write Cycle	210		nsec	
	$t_{SR}\Phi(M1)$	M1 Setup Time to Rising Edge of Φ during INTA or M1 Cycle	210		nsec	
	$t_H\Phi(M1)$	M1 Hold Time From Rising Edge of Φ	0		nsec	
RD	$t_{SW}\Phi(RD)$	RD Setup Time to Rising Edge of Φ during Write or INTA Cycle	240		nsec	
	$t_H\Phi(RD)$	RD Hold Time from Rising Edge of Φ during INTA Cycle	0		nsec	
	$t_{SR}\Phi(RD)$	RD Setup Time to Rising Edge of Φ during Read or M1 Cycle	240		nsec	
	$t_{HW}\Phi(RD)$	RD Hold Time from Rising Edge of Φ during Write Cycle	0		nsec	
	$t_{HM}\Phi(RD)$	RD Hold Time from Rising Edge of Φ during M1 Cycle	0		nsec	
INT	$t_{DRx}(IT)$	INT Delay Time from center of Receive Data Bit	10	13	Φ Periods	
	$t_{DTx}(IT)$	INT Delay Time from center of Transmit Data Bit	5	9	Φ Periods	
	$t_D\Phi(IT)$	INT Delay Time from Rising Edge of Φ		200	nsec	
WAIT/ READY	$t_{DIC}(W,R)$	WAIT/READY Delay Time from IORQ or CE in WAIT Mode		180	nsec	
	$t_{DH}\Phi(W,R)$	WAIT/READY Delay Time from Falling Edge of Φ, WAIT/READY in WAIT Mode		150	nsec	
	$t_{DRx}(W,R)$	WAIT/READY Delay Time from center of Receive Data Bit, Ready Mode	10	13	Φ Periods	
	$t_{DTx}(W,R)$	WAIT/READY Delay Time from Center of Transmit Data Bit, Ready Mode	5	8	Φ Periods	
	$t_{DL}\Phi(W,R)$	WAIT/READY Delay from Rising Edge of Φ, WAIT/READY, (i.e.) Ready Mode		120	nsec	
CTSA, CTSB DCDA, DCDB SYNCA, SYNCB	$t_w(PH)$	Minimum High Pulse Width for latching state into register and generating interrupt	200		nsec	
	$t_w(PL)$	Minimum Low Pulse Width for latching state into register and generating interrupt	200		nsec	
SYNCA,SYNCB	$t_{DL}(SY)$	Sync Pulse Delay Time from Center of Receive Data Bit, Output	4	7	Φ Periods	
	$t_S(SY)$	Sync Pulse Setup Time to Rising Edge of RxC, External Sync	100		nsec	
	$t_w(SY)$	Sync Pulse Width to Start Character Assembly	1		RxC Period	
TxCA,TxCB	$t_c(TxC)$	Transmit Clock Period	400	∞	nsec	NOTE 2
	$t_w(TCH)$	Transmit Clock Pulse Width, Clock High	180	∞	nsec	
	$t_w(TCL)$	Transmit Clock Pulse Width, Clock Low	180	∞	nsec	
TxDA,TxDB	$t_D(TxD)$	TxD Output Delay from Falling Edge of TxC (1x Clock Mode)		400	nsec	
RxCA,RxCB	$t_c(RxC)$	Receive Clock Period	400	∞	nsec	NOTE 3
	$t_w(RCH)$	Receive Clock Pulse Width, Clock High	180	∞	nsec	
	$t_w(RCL)$	Receive Clock Pulse Width, Clock Low	180	∞	nsec	

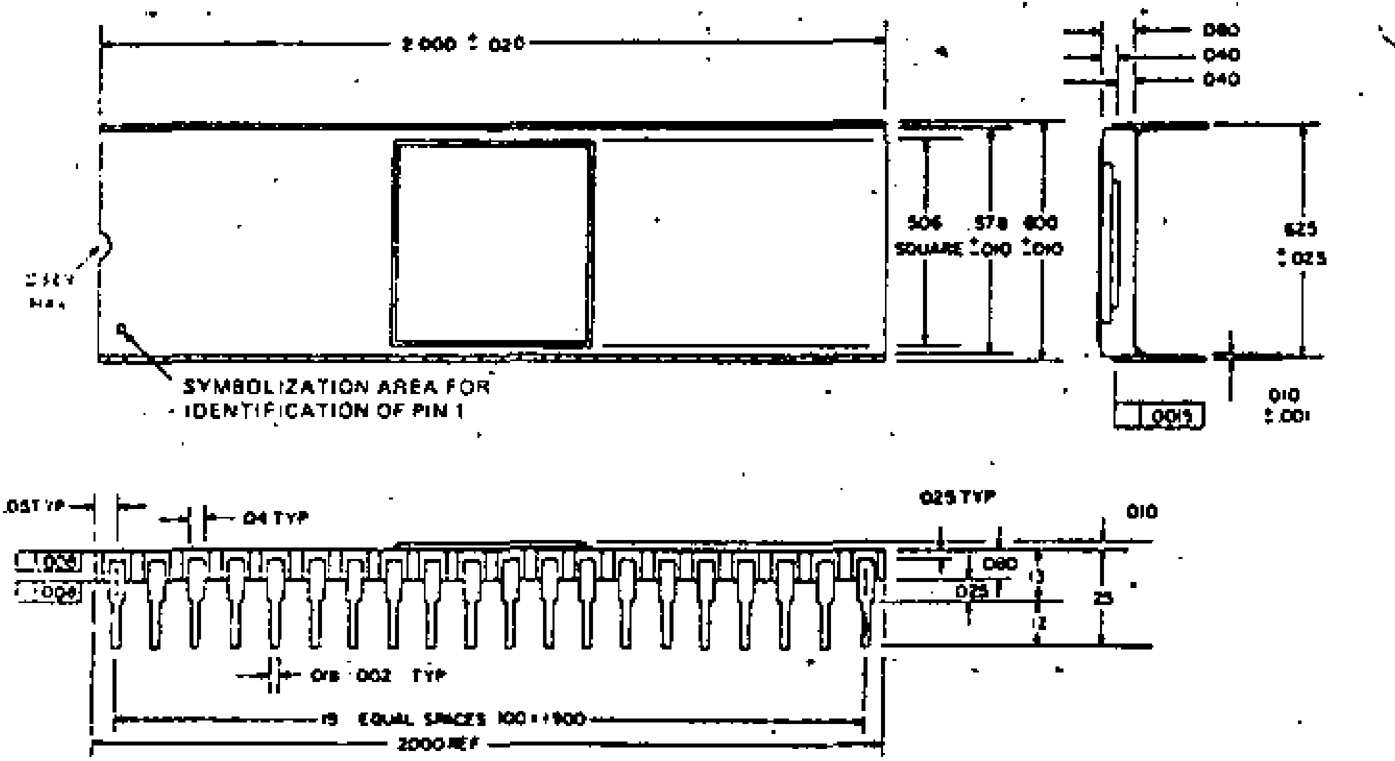
NOTE 1. If WAIT is to be used CE, IORQ, C/D and M1 must be valid for as long as WAIT condition is to persist.

NOTE 2. In all modes, maximum data rate must be less than $\frac{1}{5}$ of system clock Φ rate

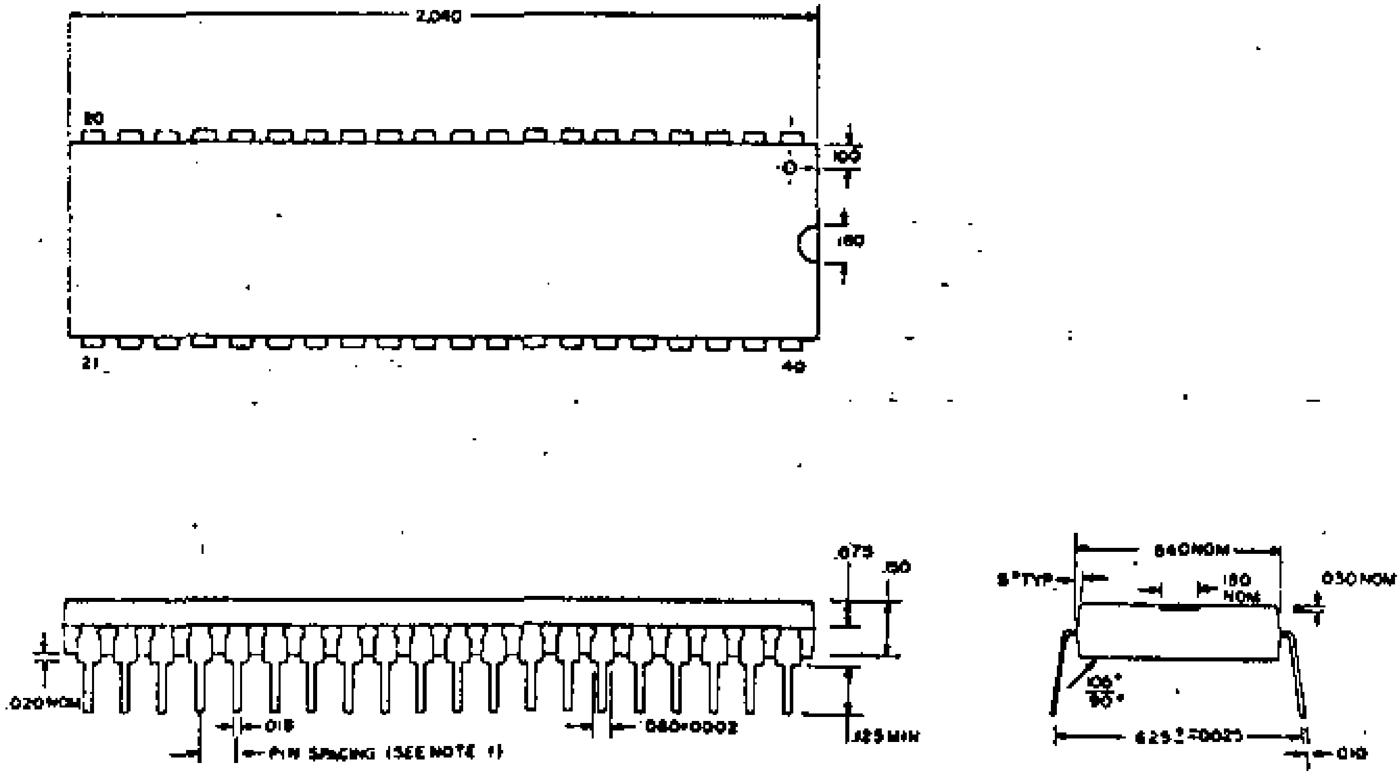
NOTE 3: The RESET signal must be active a minimum of one complete Φ cycle

8.0 PACKAGE CON[illegible]

PACKAGE DESCRIPTION — 40 Pin Dual-In-Line Ceramic Package



PACKAGE DESCRIPTION — 40-Pin Dual-In-Line Plac Plastic Package



NOTES:

1. The true position pin spacing is 0.100 between centerlines. Each pin centerline is located within ± 0.010 of its true longitudinal position relative to pins 1 and 40.

9.0 ORDERING INFORMATION

PART NO.		PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3884N	Z80 - SIO/0	Plastic	2.5MHz	0°C to + 70°C
MK3884P	Z80 - SIO/0	Ceramic	2.5MHz	0°C to + 70°C
MK3884N-10	Z80 - SIO/0	Plastic	2.5MHz	-40°C to +85°C
MK3884P-10	Z80 - SIO/0	Ceramic	2.5MHz	-40°C to +85°C
MK3884N-4	Z80A - SIO/0	Plastic	4MHz	0°C to +70°C
MK3884P-4	Z80A - SIO/0	Ceramic	4MHz	0°C to +70°C
MK3885N	Z80 - SIO/1	Plastic	2.5MHz	0°C to +70°C
MK3885P	Z80 - SIO/1	Ceramic	2.5MHz	0°C to +70°C
MK3885N-10	Z80 - SIO/1	Plastic	2.5MHz	-40°C to +85°C
MK3885P-10	Z80 - SIO/1	Ceramic	2.5MHz	-40°C to +85°C
MK3885N-4	Z80A - SIO/1	Plastic	4MHz	0° C to +70°C
MK3885P-4	Z80A - SIO/1	Ceramic	4MHz	0°C to +70°C

NOTE: See Section 2.2 for explanation of the differences between the MK3884 and MK3885

MICROPROCESADORES: TEORIA Y APLICACIONES

LENGUAJE ENSAMBLADOR

Ing Mario Rodríguez Manzanera

Julio, 1981

1

INDICE

I. INTRODUCCION

Muchos diseñadores deciden incorporar los microprocesadores a sus proyectos debido a su reducida complejidad funcional y bajo precio. Sin embargo, los mayores problemas vendrán durante la programación del microprocesador, por lo que no es raro ver proyectos realizados alrededor de un microprocesador de 30 dólares y que ha requerido inversiones de hasta 70,000 dólares de software.

Para desarrollar un sistema que incluya microprocesador se necesitará por lo menos una persona del área de electrónica y otra del - área de programación. Una razón fundamental es que habrá que decidir la distribución de funciones entre hardware y software. De una forma general, cuantas más funciones se hagan descansar sobre el - software, más flexibilidad de uso y posibilidades de cambio en su diseño tendrá el sistema, tanto en la fase de desarrollo como durante su funcionamiento. Es razonable pasar funciones a hardware - cuando el microprocesador sea incapaz de realizarlas, bien por su estructura interna o por su limitada capacidad de proceso.

El más elemental ingrediente de software y a veces el único con el que tiene que enfrentarse el diseñador de un sistema que incluye - un microprocesador, es el lenguaje de programación. Un microprocesador realiza las acciones que le especifica un programa. Un programa está formado por una secuencia de instrucciones. Una instrucción es una secuencia de bits que tiene un significado para la unidad de control del microprocesador.

El conjunto de instrucciones válidas para un microprocesador es lo que se denomina su lenguaje de máquina. Programar en lenguaje de máquina supone escribir secuencias de números en binario que son directamente descodificables por los circuitos de la unidad de control o interpretables por los microprogramas de la memoria de control. Son varias las dificultades que se plantean al programar directamente en lenguaje de máquina, entre otras tenemos:

1. Los códigos de operación son difíciles de recordar en binario. La codificación es lenta y difícil por los números en binario.

2. Las direcciones de los operandos de las instrucciones son difíciles de recordar. Muchas instrucciones contienen direcciones relativas y a la hora de corregir presentan gran dificultad especialmente cuando hay que insertar o suprimir instrucciones.

3. Si se considera que el programa no funcionará a la primera prueba es difícil seguir las ejecuciones de prueba a través de direcciones en binario.

4. El lenguaje de máquina es el que produce mayor grado de incompatibilidad entre programas. Un programa escrito en lenguaje de máquina sólo puede ser trasladado a otro microprocesador igual al primero.

5. Al cabo de un cierto tiempo un programa en lenguaje de máquina es imposible de entender hasta por su propio autor.

La programación en lenguaje de máquina puede sistematizárse y mejorarse utilizando una metodología adecuada.

El análisis previo, la confección de diagramas de flujo, el escribir previamente el programa en algún lenguaje simbólico, el confeccionar tablas de símbolos y el empleo del sistema octal o hexadecimal pueden constituir una buena ayuda.

La automatización de estas ayudas a la programación se concreta en la utilización de los lenguajes ensambladores. Se debe hacer notar que con el nombre de ensamblador se conocen dos cosas muy distintas. Se llama ensamblador,a un lenguaje simbólico en que se pueden escribir programas para un microprocesador y también recibe el mismo nombre el programa traductor.encargado de convertir los programas escritos en lenguaje simbólico en programas objeto en lenguaje de máquina. Son tres las grandes ayudas que proporciona el ensamblador al programador: le permite utilizar nemónicos para designar operaciones; nombres para designar direcciones y para especificar datos (constantes).

La distancia que separa los lenguajes de alto nivel del ensamblador es mucho mayor que la que separa a éste del lenguaje de máquina. Al pasar a programar en lenguaje de alto nivel pasamos a manejar no ya nuestro microprocesador, con su estructura de registros, acumuladores, stacks ypuertos sino un procesador de estructura distinta, concebido no para ser realizado físicamente, sino para adaptarse a la solución de problemas planteados.

Es necesario programas-traductores bastante complejos, llamados compiladores, para generar programas en código de máquina a partir de sentencias en lenguaje de alto nivel.
En la elección de un microprocesador, casi más importante que su arquitectura o velocidad de trabajo es la de un buen ensamblador y/o compilador. El presente trabajo da una idea de las facilidades que a nivel de microprocesadores existen, principalmente en lo referente a lenguajes de alto nivel.

II. LENGUAJE ENSAMBLADOR

El lenguaje ensamblador es una herramienta de software básica. Hay dos tipos generales de programas ensambladores y están disponibles para microcomputadores: cross-assembler y self-assembler. Cualquier microprocesador ahora producido tiene uno o más programas crosss-assembler. Comunmente el fabricante de chips escribe un cross-assembler antes de que el microprocesador esté físicamente disponible. Cross-assembler son populares porque muchas microcomputadoras no fueron configuradas para manejar convenientemente operaciones de ensamblador, mientras que las computadoras grandes equipadas con más impresoras adecuadas y más espacio de memoria, ofrecen al programador muchas conveniencias. El cross-assembler está preparado para funcionar sobre otra máquina, miniordenador y ordenador, así como cross-assembler, el self-assembler está escrito con un sistema de computación definido en mente.

La operación de un self-assembler es altamente dependiente del equipo de entrada y salida que rodea al microprocesador. Este sistema de dependencia específica puede algunas veces causar problemas.

Varias de las características de ensambladores son potencialmente importantes pero los usuarios de microcomputadoras, por ejemplo, ensambladores relocalizables permiten que las localidades de memoria del programa de lenguaje de máquina sean transparentes para el usuario. Algunos ensambladores-conocidos como ensambladores absolutos-siempre comienzan a almacenar el programa de lenguaje de máqui na en la misma localidad fija de memoria; otros ofrecen varias alternativas empezando localizaciones y permiten algunas limitaciones de ligado de programas segmentados en diferentes localizaciones.

Una característica de ensamblado condicional está disponible con - algunos programas de ensamblador. Esto permite al usuario decidir cual de las varias secciones del programa será ensamblada y a encontrar el orden más eficiente para ensamblar.

La capacidad de macros en un programa ensamblador permite al usuario usar una sola instrucción del lenguaje ensamblador para llamar a una secuencia específica de instrucciones del lenguaje de máquina. Esta puede ser una muy fuerte herramienta cuando ciertas secuencias son repetidas durante un mismo programa.

III. LENGUAJES DE ALTO NIVEL

La utilización de un lenguaje de alto nivel reduce los costos de programación, incrementa la habilidad del software producido y simplifica el mantenimiento y documentación de los programas si lo comparamos con la utilización de lenguajes de bajo nivel (máquina o ensamblador). Como contrapartida, la utilización de lenguajes de alto nivel supone la utilización de volúmenes de memoria que son desde un 10 a un 100% mayores que los que necesitaría un programa equivalente en ensamblador.

Algunas ventajas innegables de los lenguajes de alto nivel son:

1. Fiabilidad de los programas. Los lenguajes escritos en algún lenguaje de alto nivel son mucho más compactos. Se entiende mucho más fácilmente, lo que hace cada sentencia o grupo de sentencias.

2. Rapidez de puesta a punto. La velocidad de codificación para un programador viene a ser de unas 100 instrucciones por día (contando tiempos de preparación, depuración, etc.). Esta velocidad es independiente del lenguaje. Como un mismo programa escrito en un lenguaje de alto nivel puede tener diez veces menos líneas que uno en ensamblador, el aumento de velocidad es considerable.

3. La vida media de los lenguajes del alto nivel sobre los ensambladores es otra de las grandes ventajas. Mientras el lenguaje ensamblador cambia para cada arquitectura de microprocesador, el lenguaje de alto nivel es independiente de estos cambios.

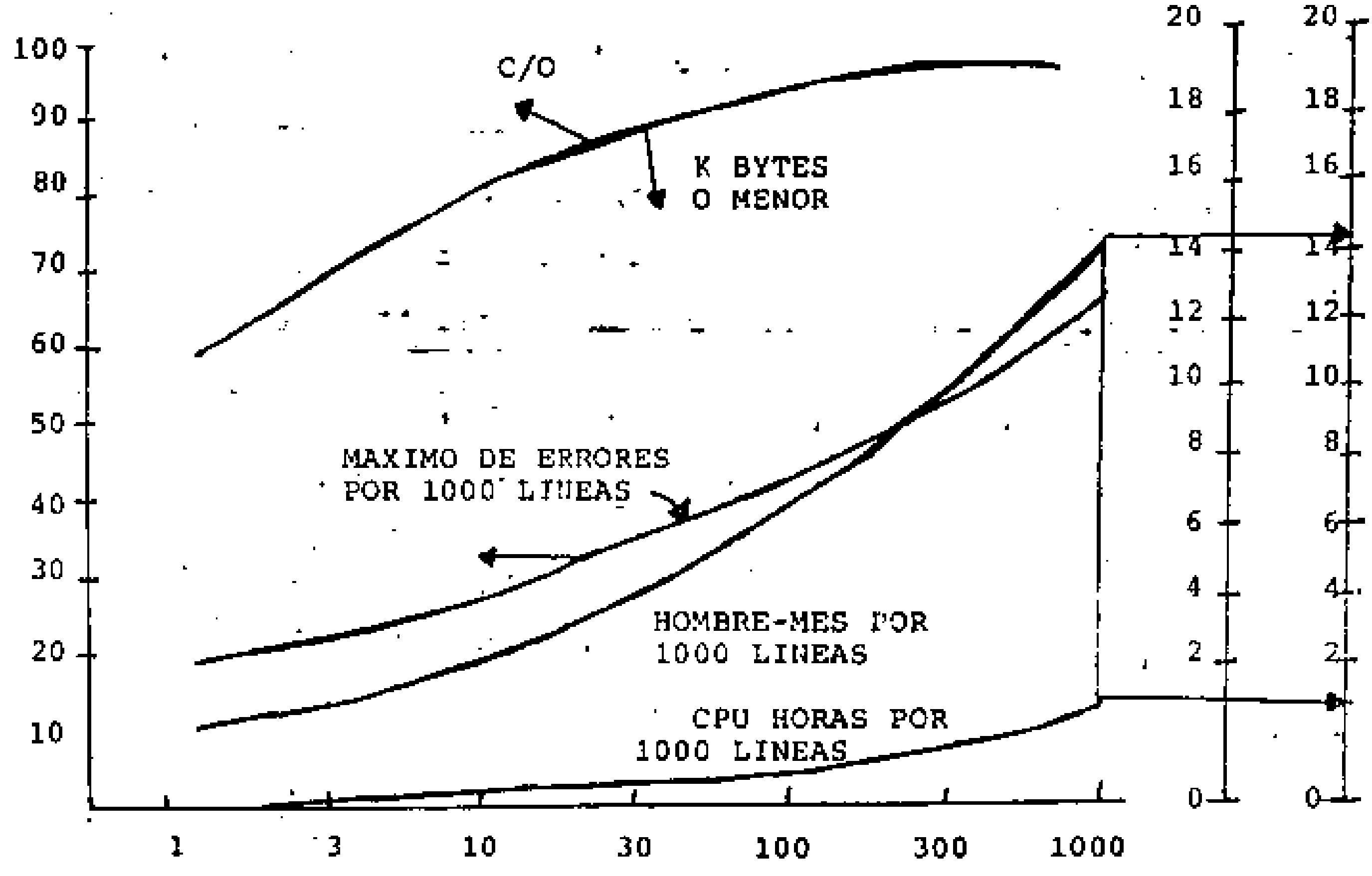
Desarrollando el compilador adecuado se pueden tener todos los programas escritos en este lenguaje, adaptados a cualquier nuevo microprocesador. La independencia de los programas respecto al microprocesador utilizado para una aplicación no es sólo algo deseable, sinoque es una necesidad si uno no quiere verse atrapado por el desarrollo de software. Actualmente, el desarrollo de nueva programación es mucho más costosa y lenta que la adopción de un nuevo microprocesador, cuando de la adopción del microprocesador puede depender la permanencia en el mercado. Si hay que partir de cero a cada cambio, el resultado puede ser desastroso. Si por el contrario, hemos adoptado un lenguaje de programación de alto nivel y estándar, podremos aprovechar no sólo la propia experiencia sino la de otros grupos que ya se hayan enfrentado con experiencias similares.

De los tres puntos anteriores no debe inferirse la inutilidad del lenguaje ensamblador. Un microprocesador puede no contar con compiladores. Un lenguaje de alto nivel puede no proporcionar acceso a ciertas características deseadas del microprocesador. En estos casos hay que utilizar forzosamente el lenguaje ensamblador. La no utilización del compilador puede también venir dictada por medidas económicas. El problemas es el esceso de memoria que para un programa dado ocupa el código generado por compilador. Si se está proyectando un sistema del que se van a vender pocas unidades, interesa rebajar el costo de programación que es fijo y muy superior al de la memoria. Si se van aproducir miles de sistemas puede resultar rentable usar ensamblador, con lo que se podrá ahorrar una cantidad considerable de memoria. Hay un punto para un cierto número de sistemas fabricados, en el que se equilibran ambos costos. 10

Dado que el precio de la memoria bajará en los próximos años (64KB por 500 dólares en 1976 y 256KB por 300 dólares en 1980), este punto no cesará de desplazarse a favor de la utilización de lenguajes de alto nivel.

En conclusión, en un sistema que incluye un microprocesador, gran parte del diseño y por lo tanto de los costos va ligada a la programación y depende del lenguaje de programación utilizado.

La figura siguiente muestra que la mayoría de los problemas de software derivan de programas muy largos - aquellos que sobrepasan los 64K bytes de código fuente - los cuales representan apenas el cuatro por ciento del total de la programación. La IBM ha encontrado



que la capacidad del hombre en programación y el tiempo de CPU por 1000 líneas de código ensamblador (incluyendo líneas de comentarios) crece exponencialmente cercano con el tamaño del programa. Si la extensión del programa es largo se esperará un promedio de error creciente e inconsistencias del programa.

Otra parte del problema es la profusión de lenguajes de programación, muchos de los cuales han sido llamados solución universal a su arribo. En realidad hay poca esperanza para esos lenguajes. La explicación es que no son eficientes, dentro de un común denominador aún y cuando realizarán la transformada rápida de Fourier, generación de reportes y diseño simple de fórmulas. Lo mejor que se puede hacer es utilizar el lenguaje correcto en la aplicación correcta. Basic es muy bueno para programas rápidos de 20 líneas y Cobol sigue siendo el mejor para reportes financieros. Fortran, el lenguaje técnico más favorecido está ganando nuevos adeptos con el estándar de la ANSI-1978 que incluye muchas de las mejores características del Pascal. Sin embargo, el lenguaje ensamblador sigue siendo el más rápido según lo muestra la comparación de la Figura 1 para la ejecución de un programa de matemáticas.

El PL11, goza de una muy buena cantidad de usuarios y aún sin haber ganado la popularidad de Basic, Fortran y Cobol.

Si no se está familiarizado con APL, uno se puede sorprender de como con un simple golpe de tecla puede cumplir, especialmente en procesamiento de arreglos, funciones complejas.

Pero un teclado especial con letras griegas y la mucha potencia de el lenguaje plantea problemas: si no se usa APL todo el tiempo, se gastará más tiempo buscando en el manual las funciones complejas, que escribiendo un programa largo en Fortran.

LENGUAJE	TIEMPO RELATIVO DE EJECUCION	VENDEDOR	CLASIFICACION DE USUARIO (ESC. 4)		
			FACILIDAD DE USO	EFICIENCIA	SATISFAC-CION TOTAL
APL	-	IBM	3.4	2.6	3.4
BASIC	3-5	DEC	2.5	2.2	2.4
COBOL	5-10	BURROUGHS	3.5	3.1	3.3
		DEC	3.7	2.2	3.5
		HONEYWELL	3.2	3.0	3.2
		IBM (AUGE)	3.4	3.1	3.3
FORTRAN	2-3	DEC	3.2	3.2	3.1
		IBM (AUGE)	3.3	3.0	3.3
		ONA SYSTEMS	3.7	3.6	3.8
PASCAL	-	UCSD	3.2	2.8	3.0
PL/1	-	IMB	3.5	3.6	3.4

FIGURA 1

Un muestreo reciente de programas ofrecidos para microcomputadoras en su mayor parte paquetes de utilería - revelan que un 37% están - escritos en lenguaje de máquina, 37% en lenguaje ensamblador, 10% - en Basic, 7% en Fortran y 3% en Pascal. El resto está escrito en - lenguajes muy diversos (Figura 2).

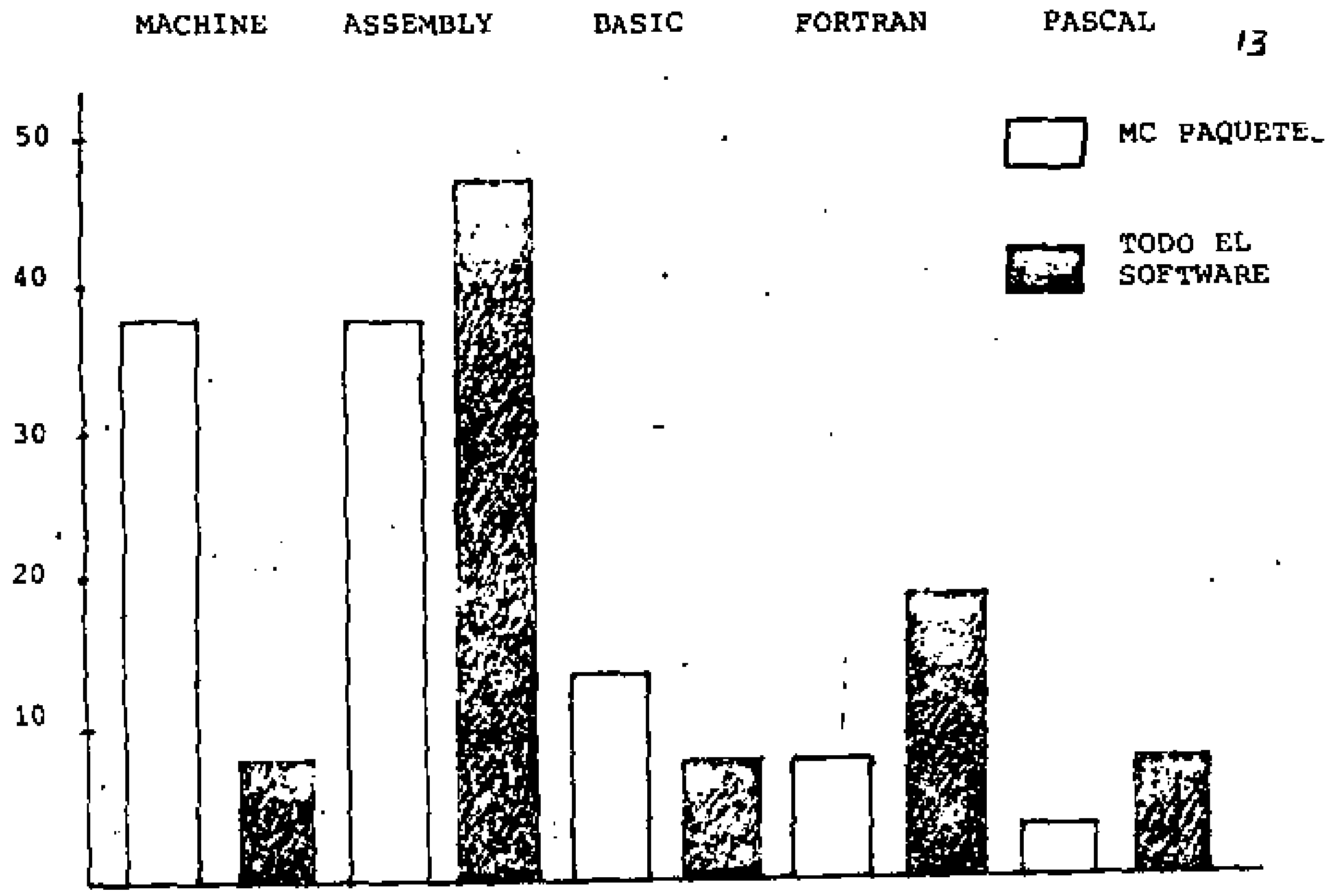
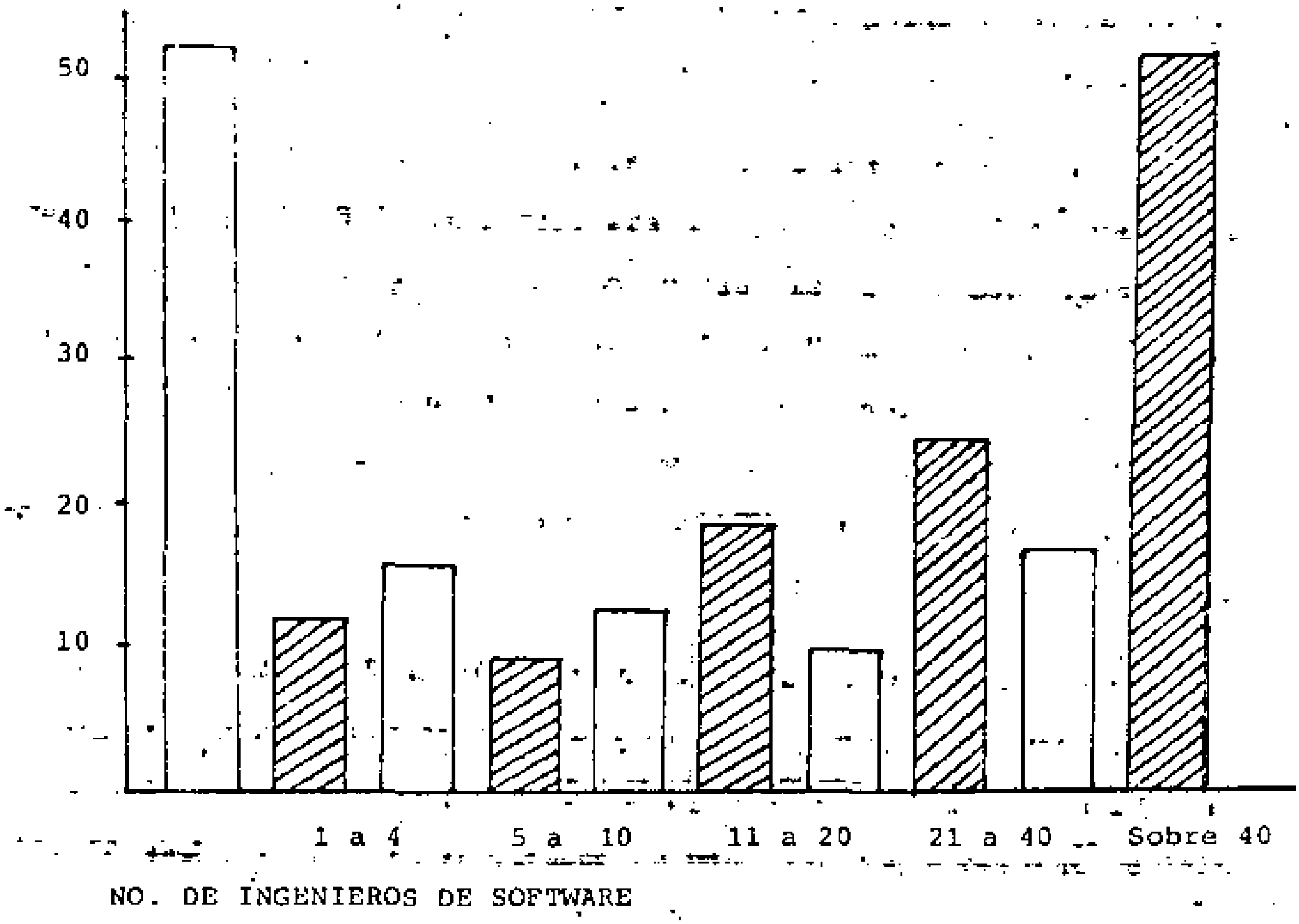


FIGURA 2.

Para todas las computadoras (incluyendo las microcomputadoras), la mezcla de lenguajes de diseño de sistemas es bastante diferente. Ensamblador está aûn más adelantado que Fortran (ensamblador 46% - y Fortran 15%), mientras que código de máquina, Pascal y Basic hacen cerca de un 7% cada uno.

En aplicaciones de software, incluyendo micros y computadoras, el lenguaje ensamblador sigue siendo principal (40%) seguido por Basic (30%), Fortran (8%) y un amplio rango de otros lenguajes.

Electronics Design realizó una encuesta con objeto de saber en donde se encontraban los ingenieros de software. Más del 50% de las - compañías investigadas empleaban menos de 5 ingenieros en software y el 15% más de 40. Figura 3.



□ % de compañías

▨ % de ingenieros

Esta gráfica es producto de entrevistas que Electronics Design hizo a diferentes compañías.

IV. LENGUAJES DE ALTO NIVEL EN EL MERCADO

Toda una serie de empresas de software ofrecen los lenguajes de alto nivel de siempre (Fortran, Cobol, Algol y Basic) con ciertas restricciones, para los microprocesadores más populares. Estos lenguajes tienen el inconveniente de que fueron creados en un ambiente totalmente ajeno a las necesidades de los microprocesadores.

Primero Intel y más tarde Motorola, han presentado lenguajes más apropiados a las necesidades de sus clientes. El PL/M, aunque creado inicialmente por Intel para su serie 8000 existe ahora para otros microprocesadores como Motorola 6800, por ejemplo. En 1975 Motorola presentó, el MPL, lenguaje de alto nivel para 6800. Ofrece como posibilidades peculiares el fácil manejo a nivel de bit, la posibilidad de definir estructuras de datos parecidos al Cobol y de incluir sentencias en lenguaje ensamblador.

La generación de código de máquina funciona en 2 pasos. En un primer paso, el compilador genera un programa en ensamblador, que luego es ensamblado obteniendo el código objeto. La posibilidad de conocer el código generado y por lo tanto de poderlo retocar tiene como contrapartida un tiempo de compilación mayor.

Plessey tiene el Pl-microp, una extensión del Algol y Mits dotó a su Altair 8800 de un super-basic. Toda esta evolución en el área de software hizo que para finales de 1975, Texas Instruments para su nuevo microprocesador, preparará compiladores de Cobol, Fortran y Basic.

Según Electronics Design de Marzo de 1979 los paquetes de utilería, ofrecidos frecuentemente en floppy disks, no pueden proveer mucha utilidad si no se está familiarizado con ellos. Usualmente contienen muchas de las siguientes herramientas de software:

- Editores. Ayuda a introducir y modificar el programa fuente.
- Compiladores y ensambladores. Traducen el programa fuente a código de máquina. Si se desea diseñar sistemas operativos, encuentre un traductor que imprima o muestre (display) el código fuente y objeto paso a paso. Hay usualmente cuatro campos, uno para la etiqueta, código de operación, operando y comentarios. Durante la depuración se desea frecuentemente un desensamblador que retraduzca el código objeto a un lenguaje fuente y la habilidad para mostrar mapas de almacenamiento y contenidos de registro.
- Macroensambladores, Ensambla grupos de instrucciones llamados por nombre, y trabaja mucho como subrutinas o funciones en lenguajes de alto nivel.
- Cargadores. Transfieren el código objeto de algún medio externo (cinta o disco) dentro de RAM. Los ligadores cargadores pueden, adicionalmente eslabonar juntos diferentes módulos de programa, y una característica de relocalización permite cargar dentro de bloques de dirección diferente, tal como fueron asignados por el traductor. Obviamente, en este caso, cargador y traductor deben ser capaces de comunicarse.
- Intérpretes. Los intérpretes son una herramienta muy popular con los microprocesadores. Permiten la utilización de gramáticas asociadas a compiladores (Basic, APL, Lisp, etc.), pero difieren de éstos en que procesan el programa fuente sin generar

código de máquina "interpretando" las instrucciones del programa fuente cada vez que este último se ejecuta. Esto impone restricciones en el tamaño de la memoría durante ejecución (ya que el intérprete debe estar en memoría durante la ejecución de cualquier programa) y en la velocidad de ejecución (ya que un intérprete es, en realidad, un programa que hace lo que otro le impone). Sin embargo, un intérprete puede ser pequeño (hay versiones de Basic en 3K bytes) y ofrece facilidades de depuración - que resultan atractivas para programadores bisoños.

- Depuradores. Permiten examinar y cambiar contenidos de memoria y empezar o parar la ejecución en una predeterminada localización o condición (breakpoint). Otros depuradores auxiliares incluyen trazo y pruebas de flujo, los cuales muestran cuando - cierto contenido de dirección cambia y cómo el control es transferido dentro de el programa.
- Rutinas de Graficación. Convierte los datos de salida en formas analógicas. Ellos van desde un pequeño programa dibuja una gráfica simple sobre un TTY a sistemas complejos que producen despliegues tridimensionales de múltiples colores, complementados con letreros en diferentes tamaños de caracteres.
- Manejo de Base de Datos. Es importante para sistemas que tienen que operar sobre un gran cantidad de datos, siempre a través de las manipulaciones que deberán ser muy simples. Las facilidades auxiliares de prueba de computadora, control de inventarios y - sistemas de reservación correrán a paso de tortuga sin el acceso directo a memoria, rutinas de búsqueda y manipulación de cadenas.

- Librerías. Son similares a los sistemas DBM, pero ellos alimentan programas más bien que datos. Esta utilería es bastante nueva, así que sólo está disponible para compradores principales.
- Sistemas Operativos. Incluyen varias de las utilerías discutidas, hardlers para I/O, rutinas de comunicación, cargadores de bootstrap y programas similares que hacen todo el trabajo hardware fácil.
- Cross-software. Permite desarrollar programas para una computadora específica, usualmente una microcomputadora, sobre una computadora huesped, la cual es usualmente una mini o tiempo compartido del centro principal, pero también puede ser un sistema desarrollado con base en microprocesadores.
- Simuladores. Son programas cross que hacen que una computadora huesped trabaje como la target machine así que se puede probar el desarrollo del software huesped antes de que el sistema esté disponible.

La Tabla 1 muestra qué paquetes ofrecen algunos vendedores.

	ASSEMBLER	MACRO ASSEMBLER	INTERPRETER	COMPILER	LINKING LOADER	EDITOR	DEBUGGER	MONITOR	SIMULATOR	DATA BASE MANAGEMENT	OPERATING SYSTEMS	CROSS SOFTWARE	APLIC PACKAGES	MISCELLANEOUS
American Microsystems	•	•	•	•		•	•	•	•					
The Boston Systems Office		•						•	•			•		
Compassmicrosystems	•			•		•	•	•	•					
Control Data									•					
Cromemco	•	•	•	•		•	•	•	•	•	•			•
Data General		•		•							•		•	•
Digital Equipment Corp.											•		•	•
Digital Research (*)		•					•				•			•
Forth (*)				•							•			•
Heunkon		•	•	•				•			•			
Hudge Taylor & Associates						•					•		•	•
Ihsas Manufacturing											•			•
Innova Systems														•
Innovatek Microsystems														•
Intersit											•		•	•
Micro Information Systems (*)						•				•				

	ASSEMBLER	MACRO ASSEMBLER	INTERPRETER	COMPILER	LINKING LOADER	EDITOR	DEBUGGER	MONITOR	SIMULATOR	DATA BASE MANAGEMENT	OPERATING SYSTEMS	CROSS SOFTWARE	APALIC PACKAGES	MISCELLANEOUS
Microtec	•													
Monolithic Systems								•			•			
Mostek		•					•				•	•	•	•
Motorola Microsystems	•	•	•	•	•	•	•	•	•		•	•	•	•
National Software Exchange (*)										•	•	•	•	•
PRS Corp. (*)							•	•		•				•
RCA Soled-State Division		•				•					•	•		•
Rockwell International	•		•	•								•		
Ryan-Farland (*)				•										•
Signetics MOS UP Dio	•			•			•		•	•	•			•
Software Dynamics (*)	•					•	•				•		•	•
Technical Systems Consult.	•	•	•	•	•	•	•				•		•	•
Texas Instruments	•						•				•			•
Wintek	•		•	•	•				•			•		
Woodley Associates (*)														•
Wyle Raboratories		•				•							•	

(*) Indica firmas de software.

Tabla 1

Los sistemas operàtivos de multilenguaje se están volviendo populares. En el lenguaje del sistema Zilog's PLZ se pueden mezclar lenguajes de ensamblador y de alto nivel y el procesador Technology's PTO permite mezclar varios lenguajes de alto nivel en un solo programa. Mientras se usa lo existente en software en una mayor concurrencia, se espera más Sistemas "mix-and-match" para mantener abajo el costo de software.

Otro medio barato para el mismo fin son los "canned"(enlatados) pro grams. El archivo de software regularmente se paga el solo en un - mes siempre por los compradores principales y más tarde los pequeños compradores pueden obtener su sistema de cualquier tienda de - computador y adaptarlo a los paquetes de software de sus búsquedas.

APLICACIONES DE LENGUAJES DE ALTO NIVEL EN TIEMPO REAL

Se ha desarrollado una versión en tiempo real del lenguaje de programación Basic para uso de adquisición de datos y aplicaciones de control. El intérprete está basado sobre el Lowrence Livermore Laboratories Basic con la adición de comandos al medio ambiente de tiempo real. Este intérprete ha sido adaptado a varios sistemas de microcomputadoras basados en el Intel 8080, incluyendo un sistema con un bus de 16 bit de I/O.

El set de instrucciones extendido de este Basic de tiempo real fue definido después de examinar varias versiones de microcomputadoras de Basic similar.

El intérprete principal requiere de 4.25k bytes para si mismo y de 1.75k bytes adicionales para los

TABLAS DE DATOS COMPARATIVOS

MICROCOMPUTADORAS Y MICROPROCESADORES

ABREVIATURAS

DMA	-	Acceso directo a memoria.
INSTR	-	Instrucciones
K	-	Kilo
K-B	-	Keyboard
MUX	-	Multiplexor
O/S	-	Operating System
POR'	-	Bus paralelo
R-T	-	Tiempo real
T/S	-	Time-share
UNLIM	-	Unlimited
WD	-	Palabra

FABRICANTE	TIPO	CAPACIDAD DE DIREC.	PUERTOS DE ENTRADA	PUERTOS DE SALIDA	PERIFERICOS	SOFTWARE
						stock: diskette dos systems
Essex International	SX-200 (PMOS)	J-K byte 41 instr.	2 lines	1 line	Dedicate a applications	Assembler, Fortran: decimal adjust arithmetic
General Instruments AEG, Germany, SGS Italy	8000 (PMOS) 3-chips	48 reg's 8-b 48 instr.	3 líneas 8-b	3 líneas 8-b	Todas las opciones	BCD arithmetic 48x1 scratch pad. Assembler on simulator on Fortran
Mostek	3870 NMOS-si-gate	64-K 70 instr.	4 líneas 8-b/port	4 líneas 8-b/port	Todas las opciones	Binario timer with programmable prescaler. Emulation and prototyping are PROM based

Programas de usuario. El paquete de punto flotante requiere de 1.75 K bytes. El micropac tiene un monitor y un área de trabajo del monitor la cual ocupa localidades de memoria de la 0000-1020 hexadecimal.

El paquete de punto flotante arranca de la 1100 (Hex) el intérprete principal a la 1800 (Hex.), el comienzo de los programas de usuario son a partir de la 2900 (Hex.) y el final de la memoria a 2FFF.

El set de instrucciones en tiempo real incluye, entre otras, las siguientes:

STM	Pone tiempo de día (segundos)
GTM	Toma tiempo de día (segundos)
TON	Arrance el programa al tiempo X
DOT	Saca datos (16 bits)
DIN	Mete datos (16 bits)
ATO	Entrada de señal analógica a digital
DTA	Salida de señal digital a analógica

Multiporogramación en tiempo real teniendo como software lenguaje - Pascal para sistemas pequeños.

Un usuario de un microprocesador Pascal puede desarrollar software para aplicaciones de propósito general usando una de dos computadoras huesped; el usuario simple FS990 floppy-disk-based minicomputer, o el multi-usuario OS990 hard-disk based minicomputer. Los microprocesadores Pascal comprenden una variedad de herramientas de software de soporte, incluyendo un editor iterativo inteligente para preparación de fuente, un compilador que genera código de intérprete, un generador de código que suple al código objeto primero, y un intérprete de depuración iterativo.

BIBLIOGRAFIA

ELECTRONICS DESIGN. MARCH 1979

ELECTRONICS. JUNE 1979

AN ADAPTATION OF BASIC FOR REAL-TIME MICROPROCESSOR APPLICATIONS

IEEE SPECTRUM. OCTOBER 1974

ELECTRONICS. JUL. 1979

FABRICANTE	TIPO	CAPACIDAD DE DIREC.	PUERTOS DE ENTRADA	PUERTOS DE SALIDA	PERIFERICOS	SOFTWARE
IBM	5100 portable computer (NMOS)	16K a 64K byte	14-key-K-B tape cartridge	CRT, tape CRAM-TU Printer	CRT, TU, K-B Tape, printer	Basic O/S, APL (Resident) APL y Basic.
Dec (Western DIG MCP-1600 NMOS)	LSL-111 PDP-11-03	32K 400 custom instr.	I/O: 33 líneas Por: DMA 16: L MUX	I/O: DMA 833-K, Bus WD/S	CRT (UT-50) Paper-tape	Basic, Fortran IV, Diagnostics, O/S. T/S Macro II, etc.
RCA	CDP-1801 (CMOS)	64K 59 Instr.	8 ports 250 líneas	8/256 8-b/ports	CRT, TT4, Paper tape cassette	Assembler (non resident s-package CDP, 185900.
Intel Intellec 4-AMD	4004 (DMOS)	4-K 46 Instr.	16 4-b/port	32 4-b/port	Paper tape k-b printer	Sin lenguajes de alto nivel
Intel Model MDS	8080 (NMOS)	64K 78 Instr.	44 4-b/port	44 4-b/port	Todas las opciones excepto tape	All options external mem. nonrelocatable PL/M
Intersic (69001 Tawgfn)	IM-6100 (CMOS)	32-k byte 60 instr.	64/256 12-b/port	64/256 12-b/port	Todas las opciones	All options are available externally
ME ASSOCIATES (JOLT)	6502 (NMOS)	64K 139 instr.	2/MUX unlim. 8-b/port	2 8-b/port	Todas las opciones excepto tape y disk	Todas las opciones excepto lenguajes de alto nivel
Mostek	5065 (NMOS)	32-K 51 instr.	-a	-a	Requiere-interface	Interfase controller

FABRICANTE	TIPO	CAPACIDAD DE DIREC.	PUERTOS DE ENTRADA	PUERTOS DE SALIDA	PERIFERICOS	SOFTWARE
Fairchild	3850 (NMOS)	64-K 70 instr.	2/252 8-b/port	2/252 8-b/port	Todas las opciones	Debug, text edit, applications programs
General Instruments Corp.	CP-1600 A (NMOS)	64-K 87 instr.	2/unlim. 16-b/port	2/unlim. 16-b/port	Todas las opciones excepto disco cassette	O/S, programas de aplicación, no macro y alto nivel
Motorola (Exorciser)	MC-6800 (NMOS) Si-gate	64-K byte 72 instr.	0/104 8-b/port	0/104 8-b/port	Todas las opciones (no k-b)	Debug, macro Todas las opciones, PL/M
National Semiconductor (PACE)	IMP-8/16 (NMOS)	64-K 387 instr.	0/unlim. MUX 8/16-0 port	0/unlim. 8/14-b port	Todas las opciones y tape cartridge	Self-assembler debug, time share, etc. Alto nivel SLN/PL
National Semiconductor	SC/MP (PMOS)	1024 byte	0/unlim.	0/unlim.	Special K-B/display or custom	Pointer addressing auto-index Hex software for economy calculator K-B/ display
Plessey	Miproc-16 (bipolar)	64-K 82 instr.	32/256 16-b/port	Unlim. with ext (MUX)	Todas las opciones. No cassette tape	Todas las opciones except macro assembler No. O/S
Intel, on-chip MC (MCS-48)	8048/8748 Si-gate NMOS	8-K 90 instr.	I/O resident 17 lines	I/O resident 17 lines	Floppy disk CRT/K-B chips	MDS based assembler, editor, monitor (MEXA)
Rockwell International Type-I System	PPS-8 (PMOS)	32-K byte 109 instr.	4/30 8-b/port	4/30 8-b/port	Todas las opciones excepto car-	No resident memory. All other options.

FABRICANTE	TIPO	CAPACIDAD DE DIREC.	PUERTOS DE ENTRADA	PUERTOS DE SALIDA	PERIFERICOS	SOFTWARE
					tridge	APS assemulator and system analysis module
Scientific Microsystems (MC-SIM)	SMS-300 (bipolar)	8-K as requi-red	1/512 8-b/port	1/512 8-b/port	K-B paper tape	Assembler resident or nonresident. Relocatable macro applications programs
Texas Instruments (990/4)	TMS-9900 (NMOS Si-gate)	32-K 69 instr.	1/256 16-b/port	1/256 16-b/port	Todas las opciones excepto cartridge	I/O memory merged or separate Portable ANSI Fortran assembler. All options
Texas Instruments	SBP-9440 (32L-16 bit)	32-K 69 instr.	1/256 16-b/port	1/256 16-b/port	Same	Mismo software Airborne navigational processor, guidance system
Texas Instruments	SBP-0400 A (I2L-4 bit) bipolar	I2L-MM como requerido 512 instr.	Data-in 12-b	Data-out 12-b	User's peripherals	Assembler on PROMS according to user's needs
Signetics (2650-PC 1000)	2650 (NMOS depletion mode)	32-K 77 instr.	2 8-b/port	2 8-b/port	Todas las opciones	Fortran resident. Assembler absolute memory-debug O/S (16/32-b)

FABRICANTE	TIPO	CAPACIDAD DE DIREC.	PUERTOS DE ENTRADA	PUERTOS DE SALIDA	PERIFERICOS	SOFTWARE
NEC (Japan) (PDA-80)	MPD-8080D (NMOS)	64-K 78 instr.	1/1, Z80 8-b/port	1/1, Z80 8-b/port	Todas las opciones No k-b	Todas làs opciones. Memory not relocatable. No imking loader
Electronic Arrays	EA 9002 (NMOS)	As required 46 instr.	Standard I/O	Standard I/O	Controller applications	All options. Fortran emulator editor.
Adavanced micro devices (rally architecture)	AM 2900 (Schottky TTL-LS1)	As required user definid	2905/6/7 type bus tranceiver	2905/6/7	All options	AMD ASM micro program. Assembler: user's own software
Zilog	Z-80 (NMOS)	64-K 128 instr.	Option 44 8-b/port	Option 44 8-b/port	Todas las opciones	Software compatible to 8080A R-T disk O/S, files of any sizes macro-assembler, Basic, PL/2, 16-bit BCD add, subtract
MOS Tech, Inc.	MCS-66021 6612 (NMOS si-gate)	2-K 57 instr.	I/O port and address bus or I/O port	Simplifica I/O bus serial port	Appliance controller	Not compatible to 6800 software. More instruction with single-byte of code. Decimal mode
Micro Nova Data General	MNova (MNOS si-gate)	1-K word	47 line I/O controller	47 line I/O controller	Todas las opciones	R-T O/S Basic Extensive library, multiply/divide

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

SOFTWARE

M en C Angel Kuri Morales

Julio, 1981

MICROPROCESADOR Z80:UN PASO ADELANTE

El procesador Z80 surgió como una respuesta de mercado a la enorme popularidad del 8080 de INTEL. En su afán competitivo, la compañía fabricante (Zilog) obtuvo un producto excepcional, que es aún actual, a pesar de los nuevos procesadores de 16 Bits.

Veáse la siguiente Tabla:

NO. DE INSTRUCCIONES	6800	8080	Z80
Registros de 8 bits	72	78	158
Registros de 16 bits	2	7	14
Registros de índice	1	0	2
Modos de dirección	8	7	10
Dirección de E/S	0	256	256
Bits de bandera	6	5	6
Voltaje requerido	+5V	+5V,-5,+12V	+5V
Velocidad de reloj máxima	1MHZ	2MHZ	4MHZ
Compatible con TTL	SI	NO	SI
Modos de interrupción	2	1	4
Refrescamiento de memoria automático	NO	NO	SI
Velocidad relativa en ejecución	5.86	5.46	1.0

5.3 INSTRUCTION OP CODES

This section describes each of the Z-80 instructions and provides tables listing the OP codes for every instruction. In each of these tables the OP codes in bold type are identical to those offered in the 8080A CPU. Also shown is the assembly language mnemonic that is used for each instruction. All instruction OP codes are listed in hexadecimal notation. Single byte OP codes require two hex characters while double byte OP codes require four hex characters. The conversion from hex to binary is repeated here for convenience.

Hex		Binary		Decimal	Hex		Binary		Decimal
0	=	0000	=	0	8	=	1000	=	8
1	=	0001	=	1	9	=	1001	=	9
2	=	0010	=	2	A	=	1010	=	10
3	=	0011	=	3	B	=	1011	=	11
4	=	0100	=	4	C	=	1100	=	12
5	=	0101	=	5	D	=	1101	=	13
6	=	0110	=	6	E	=	1110	=	14
7	=	0111	=	7	F	=	1111	=	15

Z-80 instruction mnemonics consist of an OP code and zero, one or two operands. Instructions in which the operand is implied have no operand. Instructions which have only one logical operand or those in which one operand is invariant (such as the Logical OR Instruction) are represented by a one operand mnemonic. Instructions which may have two varying operands are represented by two operand mnemonics.

LOAD AND EXCHANGE

Table 5.3-1 defines the OP code for all of the 8-bit load instructions implemented in the Z-80 CPU. Also shown in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row while the destination is specified by the left hand column. For example, load register C from register B uses the OP code 48H. In all of the tables the OP code is specified in hexadecimal notation and the 48H (=0100 1000 binary) code is fetched by the CPU from the external memory during M1 time, decoded and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination followed by the source (LD DEST.,SOURCE). Note that several combinations of addressing modes are possible. For example, the source may use register addressing and the destination may be register indirect; such as load the memory location pointed to by register HL with the contents of register D. The OP code for this operation would be 72. The mnemonic for this load instruction would be as follows:

LD (HL),D

The parentheses around the HL means that the contents of HL are used as a pointer to a memory location. In all Z-80 load instruction mnemonics the destination is always listed first, with the source following. The Z-80 assembly language has been defined for ease of programming. Every instruction is self documenting and programs written in Z-80 language are easy to maintain.

Note in table 5.3-1 that some load OP codes that are available in the Z-80 use two bytes. This is an efficient method of memory utilization since 8, 16, 24 or 32 bit instructions are implemented in the Z-80. Thus often utilized instructions such as arithmetic or logical operations are only 8-bits which results in better memory utilization than is achieved with fixed instruction sizes such as 16-bits.

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory with the third byte being the displacement d. For example a load register E with the operand pointed to by IX with an offset of +8 would be written:

LD E, (IX + 8)

The instruction sequence for this in memory would be:

Address A	DD	OP Code
A+1	5E	
A+2	08	Displacement operand

The two extended addressing instructions are also three byte instructions. For example the instruction to load the accumulator with the operand in memory location 6F32H would be written:

LD A, (6F 32H)

and its instruction sequence would be:

Address A	3A	OP Code
A+1	32	low order address
A+2	6F	high order address

Notice that the low order portion of the address is always the first operand.

The load immediate instructions for the general purpose 8-bit registers are two-byte instructions. The instruction load register H with the value 36H would be written:

LD H, 36H

and its sequence would be:

Address A	26	OP Code
A+1	36	Operand

Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example:

LD (IX - 15), 21H

would appear as:

Address A	DD	OP Code
A+1	36	
A+2	F1	displacement (-15 in signed two's complement)
A+3	21	operand to load

Notice that with any indexed addressing the displacement always follows directly after the OP code.

Table 5.3-2 specifies the 16-bit load operations. This table is very similar to the previous one. Notice that the extended addressing capability covers all register pairs. Also notice that register indirect operations specifying the stack pointer are the PUSH and POP instructions. The mnemonic for these instructions is "PUSH" and "POP." These differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack respectively. For example the instruction:

PUSH AF

Is a single byte instruction with the OP code of F5H. When this instruction is executed the following sequence is generated:

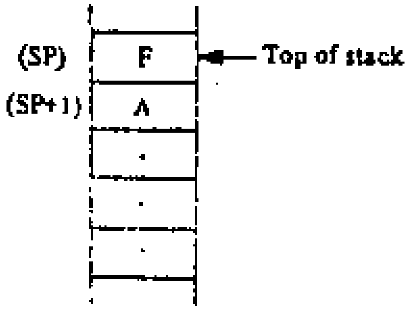
Decrement SP

LD (SP), A

Decrement SP

LD (SP), F

Thus the external stack now appears as follows:



		SOURCE															
		IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR	IMME.
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
DESTINATION REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXED	(IX + d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY + d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)			32 n n													
IMPLIED	I			ED 47													
	R			ED 4F													

8 BIT LOAD GROUP

'LD'

TABLE 5.3-1

05

The POP instruction is the exact reverse of a PUSH. Notice that all PUSH and POP instructions utilize a 16-bit operand and the high order byte is always pushed first and popped last. That is a:

```
PUSH BC   is PUSH B then C
PUSH DE   is PUSH D then E
PUSH HL   is PUSH H then L
POP  HL   is POP  L then H
```

The instruction using extended immediate addressing for the source obviously requires 2 bytes of data following the OP code. For example:

LD DE, 0659H

will be:

Address A	11	OP Code
A+1	59	Low order operand to register E
A+2	06	High order operand to register D

In all extended immediate or extended addressing modes, the low order byte always appears first after the OP code.

Table 5.3-3 lists the 16-bit exchange instructions implemented in the Z-80. OP code 08H allows the programmer to switch between the two pairs of accumulator flag registers while D9H allows the programmer to switch between the duplicate set of six general purpose registers. These OP codes are only one byte in length to absolutely minimize the time necessary to perform the exchange so that the duplicate banks can be used to effect very fast interrupt response times.

BLOCK TRANSFER AND SEARCH

Table 5.3-4 lists the extremely powerful block transfer instructions. All of these instructions operate with three registers.

HL points to the source location.

DE points to the destination location.

BC is a byte counter.

After the programmer has initialized these three registers, any of these four instructions may be used. The LDI (Load and Increment) instruction moves one byte from the location pointed to by HL to the location pointed to by DE. Register pairs HL and DE are then automatically incremented and are ready to point to the following locations. The byte counter (register pair BC) is also decremented at this time. This instruction is valuable when blocks of data must be moved but other types of processing are required between each move. The LDIR (Load, increment and repeat) instruction is an extension of the LDI instruction. The same load and increment operation is repeated until the byte counter reaches the count of zero. Thus, this single instruction can move any block of data from one location to any other.

Note that since 16-bit registers are used, the size of the block can be up to 64K bytes (1K = 1024) long and it can be moved from any location in memory to any other location. Furthermore the blocks can be overlapping since there are absolutely no constraints on the data that is used in the three register pairs.

The LDD and LDDR instructions are very similar to the LDI and LDIR. The only difference is that register pairs HL and DE are decremented after every move so that a block transfer starts from the highest address of the designated block rather than the lowest.

		SOURCE									
		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
DESTINATION REGISTER	AF										F1
	BC								01 n n	ED 4B n n	C1
	DE								11 n n	ED 5B n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
PUSH INSTRUCTIONS → REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

↑ POP INSTRUCTIONS

NOTE: The Push & Pop Instructions adjust the SP after every execution

16 BIT LOAD GROUP
'LD'
'PUSH' AND 'POP'
TABLE 5.3-2

		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

EXCHANGES
'EX' AND 'EXX'
TABLE 5.3-3

			SOURCE	
			REG. INDIR.	
			(HL)	
DESTINATION	REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE) ← (HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE) ← (HL) Inc HL & DE, Dec BC, Repeat until BC = 0
			ED A8	'LDD' – Load (DE) ← (HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE) ← (HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Reg	HL	points to source
Reg	DE	points to destination
Reg	BC	is byte counter

BLOCK TRANSFER GROUP
TABLE 5.3-4

Table 5.3-5 specifies the OP codes for the four block search instructions. The first, CPI (compare and increment) compares the data in the accumulator, with the contents of the memory location pointed to by register HL. The result of the compare is stored in one of the flag bits (see section 6.0 for a detailed explanation of the flag operations) and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

The instruction CPIR is merely an extension of the CPI instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. Thus, this single instruction can search the entire memory for any 8-bit character.

The CPD (Compare and Decrement) and CPDR (Compare, Decrement and Repeat) are similar instructions, their only difference being that they decrement HL after every compare so that they search the memory in the opposite direction. (The search is started at the highest location in the memory block).

It should be emphasized again that these block transfer and compare instructions are extremely powerful in string manipulation applications.

ARITHMETIC AND LOGICAL

Table 5.3-6 lists all of the 8-bit arithmetic operations that can be performed with the accumulator, also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, except INC and DEC, the specified 8-bit operation is performed between the data in the accumulator and the source data specified in the table. The result of the operation is placed in the accumulator with the exception of compare (CP) that leaves the accumulator unaffected. All of these operations affect the flag register as a result of the specified operation. (Section 6.0 provides all of the details on how the flags are affected by any instruction type). INC and DEC instructions specify a register or a memory location as both source and destination of the result. When the source operand is addressed using the index registers the displacement must follow directly. With immediate addressing the actual operand will follow directly. For example the instruction:

AND 07H

would appear as:

Address A	E6	OP Code
A+1	07	Operand

SEARCH LOCATION	
REG. INDIR.	
(HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory
to be compared with accumulator
contents
BC is byte counter

BLOCK SEARCH GROUP
TABLE 5.3-5

Assuming that the accumulator contained the value F3H the result of 03H would be placed in the accumulator:

Acc before operation	1111 0011 = F3H
Operand	0000 0111 = 07H
Result to Acc	0000 0011 = 03H

The Add instruction (ADD) performs a binary add between the data in the source location and the data in the accumulator. The subtract (SUB) does a binary subtraction. When the add with carry is specified (ADC) or the subtract with carry (SBC), then the carry flag is also added or subtracted respectively. The flags and decimal adjust instruction (DAA) in the Z-80 (fully described in section 6.0) allow arithmetic operations for:

multiprecision packed BCD numbers

multiprecision signed or unsigned binary numbers

multiprecision two's complement signed numbers

Other instructions in this group are logical and (AND), logical or (OR), exclusive or (XOR) and compare (CP).

There are five general purpose arithmetic instructions that operate on the accumulator or carry flag. These five are listed in table 5.3-7. The decimal adjust instruction can adjust for subtraction as well as addition, thus making BCD arithmetic operations simple. Note that to allow for this operation the flag N is used. This flag is set if the last arithmetic operation was a subtract. The negate accumulator (NEG) instruction forms the two's complement of the number in the accumulator. Finally notice that a reset carry instruction is not included in the Z-80 since this operation can be easily achieved through other instructions such as a logical AND of the accumulator with itself.

Table 5.3-8 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions including add with carry and subtract with carry. ADC and SBC affect all of the flags. These two groups simplify address calculation operations or other 16-bit arithmetic operations.

09

	SOURCE										
	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

8 BIT ARITHMETIC AND LOGIC
TABLE 5.3-6

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

GENERAL PURPOSE AF OPERATIONS
TABLE 5.3-7

		SOURCE					
DESTINATION		BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC'		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

16 BIT ARITHMETIC
TABLE 5.3–8

ROTATE AND SHIFT

A major capability of the Z-80 is its ability to rotate or shift data in the accumulator, any general purpose register, or any memory location. All of the rotate and shift OP codes are shown in table 5.3-9. Also included in the Z-80 are arithmetic and logical shift operations. These operations are useful in an extremely wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the accumulator to be rotated with the two digits in a memory location pointed to by register pair HL. (See figure 5.3-9). These instructions allow for efficient BCD arithmetic

BIT MANIPULATION

The ability to set, reset and test individual bits in a register or memory location is needed in almost every program. These bits may be flags in a general purpose software routine, indications of external control conditions or data packed into memory locations to make memory utilization more efficient.

The Z-80 has the ability to set, reset or test any bit in the accumulator, any general purpose register or any memory location with a single instruction. Table 5.3-10 lists the 240 instructions that are available for this purpose. Register addressing can specify the accumulator or any general purpose register on which the operation is to be performed. Register indirect and indexed addressing are available to operate on external memory locations. Bit test operations set the zero flag (Z) if the tested bit is a zero. (Refer to section 6.0 for further explanation of flag operation).

JUMP, CALL AND RETURN

Figure 5.3-11 lists all of the jump, call and return instructions implemented in the Z-80 CPU. A jump is a branch in a program where the program counter is loaded with the 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative or Register Indirect). Notice that the jump group has several different conditions that can be specified to be met before the jump will be made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the flag register. (Refer to section 6.0 for details on the flag register). The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (two to specify the 16-bit address) with the low order address byte first followed by the high order address byte.

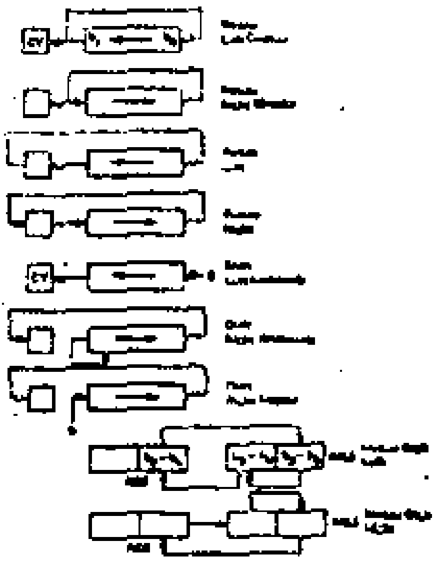
//

11

Source and Destination

TYPE OF ROTATE OR SHIFT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]								[illegible]		
[illegible]								[illegible]		

	A
RLCA	[illegible]
RRCA	[illegible]
RLA	[illegible]
RRA	[illegible]



ROTATES AND SHIFTS
TABLE 5.3–9

For example an unconditional Jump to memory location 3E32H would be:

Address A	C3	OP Code
A+1	32	Low order address
A+2	3E	High order address

The relative jump instruction uses only two bytes, the second byte is a signed two's complement displacement form the existing PC. This displacement can be in the range of +129 to -126 and is measured from the address of the instruction OP code.

Three types of register indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the PC. This capability allows for program jumps to be a function of previous calculations.

A call is a special form of a jump where the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the PC to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instructions have been included in the Z-80 family of components. The return from interrupt instruction (RETI) and the return from non maskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the OP code C9H. The difference is that (RETI) can be used at the end of an interrupt routine and all Z-80 peripheral chips will recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction coupled with the Z-80 peripheral devices implementation simplifies the normal return from nested interrupt. Without this feature the following software sequence would be necessary to inform the interrupting device that the interrupt routine is completed:

	BIT	REGISTER ADDRESSING							REG. INDIR.	INDEXED	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST BIT	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

BIT MANIPULATION GROUP
TABLE 5.3-10

Disable Interrupt	— prevent interrupt before routine is exited.
LD A, n OUT n, A	— notify peripheral that service routine is complete
Enable Interrupt	
Return	

This seven byte sequence can be replaced with the two byte RETI instruction in the Z-80. This is important since interrupt service time often must be minimized.

To facilitate program loop control the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. A simple example of its use might be:

Address	Instruction	Comments
N, N + 1	LD B, 7	; set B register to count of 7
N + 2 to N + 9	(Perform a sequence of instructions)	; loop to be performed 7 times
N + 10, N + 11	DJNZ -8	; to jump from N + 12 to N + 2
N + 12	(Next Instruction)	

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE—CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

JUMP, CALL and RETURN GROUP
TABLE 5.3-11

Table 5.3-12 lists the eight OP codes for the restart instruction. This instruction is a single byte call to any of the eight addresses listed. The simple mnemonic for these eight calls is also shown. The value of this instruction is that frequently used routines can be called with this instruction to minimize memory usage.

CALL ADDRESS	OP CODE	
0000_H	C7	'RST 0'
0008_H	CF	'RST 8'
0010_H	D7	'RST 16'
0018_H	DF	'RST 24'
0020_H	E7	'RST 32'
0028_H	EF	'RST 40'
0030_H	F7	'RST 48'
0038_H	FF	'RST 56'

RESTART GROUP
TABLE 5.3–12

INPUT/OUTPUT

The Z-80 has an extensive set of Input and Output instructions as shown in table 5.3-13 and table 5.3-14. The addressing of the input or output device can be either absolute or register indirect, using the C register. Notice that in the register indirect addressing mode data can be transferred between the I/O devices and any of the internal registers. In addition eight block transfer instructions have been implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands) while register B is used as a byte counter. Register C holds the address of the port for which the input or output command is desired. Since register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

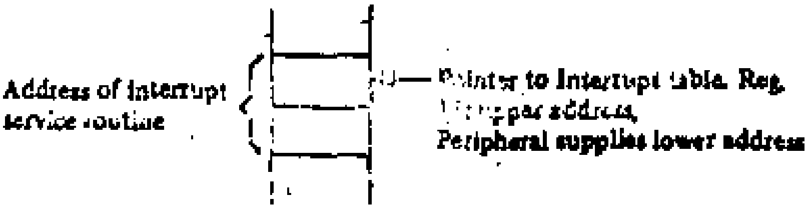
In the instructions IN A, n and OUT n, A the I/O device address n appears in the lower half of the address bus (A_0-A_7) while the accumulator content is transferred in the upper half of the address bus. In all register indirect input output instructions, including block I/O transfers the content of register C is transferred to the lower half of the address bus (device address) while the content of register B is transferred to the upper half of the address bus.

				IMMED.	REG. INDIR.	
				(n)	(C)	
INPUT DESTINATION	INPUT 'IN'	REG ADDRESSING	A	DB n	ED 78	
			B		ED 40	
			C		ED 48	
			D		ED 50	
			E		ED 58	
			H		ED 60	
			L		ED 68	
	'INI' — INPUT & Inc HL, Dec B	REG. INDIR.	(HL)		ED A2	BLOCK INPUT COMMANDS
	'INIR'— INP, Inc HL, Dec B, REPEAT IF B≠0				ED B2	
	'IND'— INPUT & Dec HL, Dec B				ED AA	
	'INDR'— INPUT, Dec HL, Dec B, REPEAT IF B≠0				ED BA	

INPUT GROUP
TABLE 5.3—13

CPU CONTROL GROUP

The final table, table 5.3-15 illustrates the six general purpose CPU control instructions. The NOP is a do nothing instruction. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU into any of the three available interrupt response modes as follows. If mode zero is set the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode where the CPU automatically executes a restart (RST) to location 0038H so that no external hardware is required. (The old PC content is pushed onto the stack). Mode 2 is the most powerful in that it allows for an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address where the upper 8-bits are the content of register I and the lower 8-bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table where the address of the service routine is located. The CPU automatically obtains the starting address and performs a CALL to this address.


Address of interrupt service routine
Pointer to Interrupt table. Reg. I supplies upper address,
Peripheral supplies lower address

			SOURCE							
			REGISTER							REG. IND.
			A	B	C	D	E	H	L	(HL)
'OUT'	IMMED.	(n)	D3 n							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB

BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

OUTPUT GROUP
TABLE 5.3–14

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

MISCELLANEOUS CPU CONTROL
TABLE 5.3–15

6.0 FLAGS

Each of the two Z-80 CPU Flag registers contains six bits of information which are set or reset by various CPU operations. Four of these bits are testable; that is, they are used as conditions for jump, call or return instructions. For example a jump may be desired only if a specific bit in the flag register is set. The four testable flag bits are:

1) Carry Flag (C) – This flag is the carry from the highest order bit of the accumulator. For example, the carry flag will be set during an add instruction where a carry from the highest bit of the accumulator is generated. This flag is also set if a borrow is generated during a subtraction instruction. The shift and rotate instructions also affect this bit.

2) Zero Flag (Z) – This flag is set if the result of the operation loaded a zero into the accumulator. Otherwise it is reset.

3) Sign Flag (S) – This flag is intended to be used with signed numbers and it is set if the result of the operation was negative. Since bit 7 (MSB) represents the sign of the number (A negative number has a 1 in bit 7), this flag stores the state of bit 7 in the accumulator.

4) Parity/Overflow Flag (P/V) – This dual purpose flag indicates the parity of the result in the accumulator when logical operations are performed (such as AND A, B) and it represents overflow when signed two's complement arithmetic operations are performed. The Z-80 overflow flag indicates that the two's complement number in the accumulator is in error since it has exceeded the maximum possible (+127) or is less than the minimum possible (-128) number than can be represented in two's complement notation. For example consider adding:

```
      +120 =    0111 1000
      +105 =    0110 1001
      ---------------------
         C = 0  1110 0001  =  -95 (wrong) Overflow has occured
```

Here the result is incorrect. Overflow has occurred and yet there is no carry to indicate an error. For this case the overflow flag would be set. Also consider the addition of two negative numbers:

```
        -5 =    1111 1011
       -16 =    1111 0000
      ---------------------
         C = 1  1110 1011  =  -21 correct
```

Notice that the answer is correct but the carry is set so that this flag can not be used as an overflow indicator. In this case the overflow would not be set.

For logical operations (AND, OR, XOR) this flag is set if the parity of the result is even and is reset if it is odd.

There are also two non-testable bits in the flag register. Both of these are used for BCD arithmetic. They are:

1) Half carry (H) – This is the BCD carry or borrow result from the least significant four bits of operation. When using the DAA (Decimal Adjust instruction) this flag is used to correct the result of a previous packed decimal add or subtract.

2) Subtract Flag (N) – Since the algorithm for correcting BCD operations is different for addition or subtraction, this flag is used to specify what type of instruction was executed last so that the DAA operation will be correct for either addition or subtraction.

The Flag register can be accessed by the programmer and its format is as follows:

S	Z	X	H	X	P/V	N	C

X means flag is indeterminate.

Table 6.0-1 lists how each flag bit is affected by various CPU instructions. In this table a '•' indicates that the instruction does not change the flag, an 'X' means that the flag goes to an indeterminate state, a '0' means that it is reset, a '1' means that it is set and the symbol '‡' indicates that it is set or reset according to the previous discussion. Note that any instruction not appearing in this table does not affect any of the flags.

Table 6.0-1 includes a few special cases that must be described for clarity. Notice that the block search instruction sets the Z flag if the last compare operation indicated a match between the source and the accumulator data. Also, the parity flag is set if the byte counter (register pair BC) is not equal to zero. This same use of the parity flag is made with the block move instructions. Another special case is during block input or output instructions, here the Z flag is used to indicate the state of register B which is used as a byte counter. Notice that when the I/O block transfer is complete, the zero flag will be reset to a zero (i.e. B=0) while in the case of a block move command the parity flag is reset when the operation is complete. A final case is when the refresh or I register is loaded into the accumulator, the interrupt enable flip flop is loaded into the parity flag so that the complete state of the CPU can be saved at any time.

Instruction	C	Z	P/V	S	N	H	Comments
ADD A, s; ADC A,s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s; SBC A, s; CP s; NEG	↕	↕	V	↕	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s; XOR s	0	↕	P	↕	0	0	And set's different flags
INC s	•	↕	V	↕	0	↕	8-bit increment
DEC m	•	↕	V	↕	1	↕	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA; RLCA, RRA, RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m; RLC m; RR m; RRC m SLA m; SRA m; SRL m	↕	↕	P	↕	0	0	Rotate and shift locations
RLD, RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r,(C)	•	↕	P	↕	0	0	Input register indirect
INI; IND; OUTI; OUTD	•	↕	X	X	1	X	Block input and output
INIR; INDR; OTIR, OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI, LDD	•	X	↕	X	0	0	Block transfer instructions
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	X	1	X	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The state of bit b of location s is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

The following notation is used in this table:

Symbol	Operation
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from into bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

SUMMARY OF FLAG OPERATION
TABLE 8.0-1

7.0 SUMMARY OF OP CODES AND EXECUTION TIMES

The following section gives a summary of the Z-80 instructions set. The instructions are logically arranged into groups as shown on tables 7.0-1 through 7.0-11. Each table shows the assembly language mnemonic OP code, the actual OP code, the symbolic operation, the content of the flag register following the execution of each instruction, the number of bytes required for each instruction as well as the number of memory cycles and the total number of T states (external clock periods) required for the fetching and execution of each instruction. Care has been taken to make each table self-explanatory without requiring any cross reference with the test or other tables.

GC 23 47.

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments	
		Flags											
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4	r, r'	Reg.
LD r, n	r ← n	•	•	•	•	•	•	00 r 110	2	2	7	000	B
								← n →				001	C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101	3	5	19	011	E
								01 r 110				100	H
								← d →				101	L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101	3	5	19	111	A
								01 r 110					
								← d →					
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7		
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101	3	5	19		
								01 110 r					
								← d →					
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101	3	5	19		
								01 110 r					
								← d →					
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110	2	3	10		
								← n →					
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7		
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7		
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010	3	4	13		
								← n →					
								← n →					
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7		
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7		
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010	3	4	13		
								← n →					
								← n →					
LD A, I	A ← I	•	↕	IFF	↕	0	0	11 101 101	2	2	9		
								01 010 111					
LD A, R	A ← R	•	↕	IFF	↕	0	0	11 101 101	2	2	9		
								01 011 111					
LD I, A	I ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 000 111					
LD R, A	R ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 001 111					

Notes: r, r' means any of the registers A, B, C, D, E, H, L

IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,

↕ = flag is affected according to the result of the operation

8-BIT LOAD GROUP
TABLE 7.0-1

24
22

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	•	•	•	•	00 dd0 001 ← n → ← n →	3	3	10	dd Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	•	•	•	•	•	•	11 011 101 00 100 001 ← n → ← n →	4	4	14	
LD IY, nn	IY ← nn	•	•	•	•	•	•	11 111 101 00 100 001 ← n → ← n →	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	•	•	•	•	00 101 010 ← n → ← n →	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	•	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	•	•	•	•	11 011 101 00 101 010 ← n → ← n →	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	•	•	•	•	11 111 101 00 101 010 ← n → ← n →	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	•	•	•	•	00 100 010 ← n → ← n →	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	•	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	•	•	•	•	11 011 101 00 100 010 ← n → ← n →	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	•	•	•	•	11 111 101 00 100 010 ← n → ← n →	4	6	20	
LD SP, HL	SP ← HL	•	•	•	•	•	•	11 111 001	1	1	6	
LD SP, IX	SP ← IX	•	•	•	•	•	•	11 011 101 11 111 001	2	2	10	
LD SP, IY	SP ← IY	•	•	•	•	•	•	11 111 101 11 111 001	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	•	•	•	•	11 qq0 101	1	3	11	qq Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	•	•	•	•	11 011 101 11 100 101	2	4	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	•	•	•	•	11 111 101 11 100 101	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	•	•	•	•	11 qq0 001	1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	•	•	•	•	11 011 101 11 100 001	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	•	•	•	•	11 111 101 11 100 001	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively
E.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ flag is affected according to the result of the operation.

16-BIT LOAD GROUP
TABLE 7.0-2

22

CG- 25

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix} \leftrightarrow \begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	IX_H ↔ (SP+1) IX_L ↔ (SP)	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	IY_H ↔ (SP+1) IY_L ↔ (SP)	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	① ‡	•	0	0	11 101 101 10 100 000	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	0	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	① ‡	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	0	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A − (HL) HL ← HL+1 BC ← BC-1	•	② ‡	① ‡	‡	1	‡	11 101 101 10 100 001	2	4	16	
CPIR	A − (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	② ‡	① ‡	‡	1	‡	11 101 101 10 110 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A − (HL) HL ← HL-1 BC ← BC-1	•	② ‡	① ‡	‡	1	‡	11 101 101 10 101 001	2	4	16	
CPDR	A − (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	② ‡	① ‡	‡	1	‡	11 101 101 10 111 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1.
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

TABLE 7.0-3

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	V	↕	0	↕	10 $\boxed{000}$ r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A ← A + n	↕	↕	V	↕	0	↕	11 $\boxed{000}$ 110 ← n →	2	2	7	
ADD A, (HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10 $\boxed{000}$ 110	1	2	7	
ADD A, (IX+d)	A←A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 $\boxed{000}$ 110 ← d →	3	5	19	
ADD A, (IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 $\boxed{000}$ 110 ← d →	3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	V	↕	0	↕	$\boxed{001}$				s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction
SUB s	A ← A − s	↕	↕	V	↕	1	↕	$\boxed{010}$				
SBC A, s	A ← A − s − CY	↕	↕	V	↕	1	↕	$\boxed{011}$				
AND s	A ← A ∧ s	0	↕	P	↕	0	1	$\boxed{100}$				The indicated bits replace the 000 in the ADD set above
OR s	A ← A ∨ s	0	↕	P	↕	0	0	$\boxed{110}$				
XOR s	A ← A ⊕ s	0	↕	P	↕	0	0	$\boxed{101}$				
CP s	A − s	↕	↕	V	↕	1	↕	$\boxed{111}$				
INC r	r ← r + 1	•	↕	V	↕	0	↕	00 r $\boxed{100}$	1	1	4	
INC (HL)	(HL) ← (HL) + 1	•	↕	V	↕	0	↕	00 110 $\boxed{100}$	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d) + 1	•	↕	V	↕	0	↕	11 011 101 00 110 $\boxed{100}$ ← d →	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 $\boxed{100}$ ← d →	3	6	23	
DEC m	m ← m − 1	•	↕	V	↕	1	↕	$\boxed{101}$				m is any of r, (HL), (IX+d), (IY+d) as shown for INC. Same format and states as INC. Replace 100 with 101 in OP code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

8-BIT ARITHMETIC AND LOGICAL GROUP
TABLE 7.0-4

CC- 27 25

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	↕	↕	P	↕	•	↕	00 100 111	1	1	4	Decimal adjust accumulator
CPL	A ← $\bar{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	A ← 0 − A	↕	↕	V	↕	1	↕	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	CY ← $\overline{CY}$	↕	•	•	•	0	X	00 111 111	1	1	4	Complement carry flag
SCF	CY ← 1	1	•	•	•	0	0	00 110 111	1	1	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	1	1	4	
DI	IFF ← 0	•	•	•	•	•	•	11 110 011	1	1	4	
EI	IFF ← 1	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notes: IFF indicates the interrupt enable flip flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

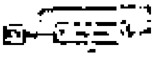
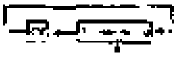
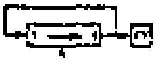
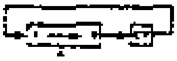
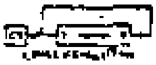
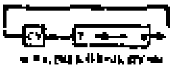
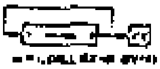
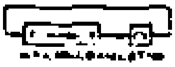
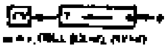
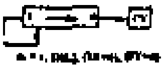
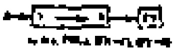
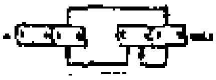
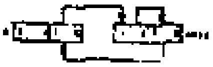
GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS
TABLE 7.0-5

Mnemonic	Symbolic Operation	Flags						Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments
		C	Z	P/V	S	N	H	76 543 210				
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss Reg. 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL ← HL+ss+CY	↕	↕	V	↕	0	X	11 101 101 01 ss1 010	2	4	15	
SBC HL, ss	HL ← HL-ss-CY	↕	↕	V	↕	1	X	11 101 101 01 ss0 010	2	4	15	
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101 00 pp1 001	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	↕	•	•	•	0	X	11 111 101 00 rr1 001	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101 00 100 011	2	2	10	
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101 00 100 011	2	2	10	
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101 00 101 011	2	2	10	
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101 00 101 011	2	2	10	

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

16-BIT ARITHMETIC GROUP
TABLE 7.0-6

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		↕	•	•	•	0	0	00 000 111	1	1	4	Rotate left circular accumulator
RLA		↕	•	•	•	0	0	00 010 111	1	1	4	Rotate left accumulator
RRCA		↕	•	•	•	0	0	00 001 111	1	1	4	Rotate right circular accumulator
RRA		↕	•	•	•	0	0	00 011 111	1	1	4	Rotate right accumulator
RLC r		↕	↕	P	↕	0	0	11 001 011 00 000 r	2	2	8	Rotate left circular register r
RLC (HL)		↕	↕	P	↕	0	0	11 001 011 00 000 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		↕	↕	P	↕	0	0	11 011 101 11 001 011 ← d → 00 000 110	4	6	23	
RLC (IY+d)		↕	↕	P	↕	0	0	11 111 101 11 001 011 ← d → 00 000 110	4	6	23	
RL m		↕	↕	P	↕	0	0	010				Instruction format and states are as shown for RLC,m. To form new OP-code replace 000 of RLC,m with shown code
RRC m		↕	↕	P	↕	0	0	001				
RR m		↕	↕	P	↕	0	0	011				
SLA m		↕	↕	P	↕	0	0	100				
SRA m		↕	↕	P	↕	0	0	101				
SRL m		↕	↕	P	↕	0	0	111				
RLD		•	↕	P	↕	0	0	11 101 101 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected.
RRD		•	↕	P	↕	0	0	11 101 101 01 100 111	2	5	18	

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

ROTATE AND SHIFT GROUP
TABLE 7.3-7

Mnemonic	Symbolic Operation	Flags						Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments
		C	Z	P/V	S	N	H	76 543 210				
BIT b, r	$Z \leftarrow \overline{r_b}$	•	‡	X	X	0	1	11 001 011 01 b r	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	•	‡	X	X	0	1	11 001 011 01 b 110	2	3	12	
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)_b}$	•	‡	X	X	0	1	11 011 101 11 001 011 ← d → 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)_b}$	•	‡	X	X	0	1	11 111 101 11 001 011 ← d → 01 b 110	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 $\boxed{11}$ b r	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 $\boxed{11}$ b 110	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 ← d → $\boxed{11}$ b 110	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 ← d → $\boxed{11}$ b 110	4	6	23	
RES b, m	$s_b \leftarrow 0$ m ≡ r, (HL), (IX+d), (IY+d)							$\boxed{10}$				To form new OP-code replace $\boxed{11}$ of SET b,s with $\boxed{10}$. Flags and time states for SET instruction.

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) or location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

BIT SET, RESET AND TEST GROUP
TABLE 7.0-8

29

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011 ← n → ← n →	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	•	•	•	•	11 cc 010 ← n → ← n →	3	3	10	cc Condition 000 NZ non zero 001 Z zero 010 NC non carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000 ← e-2 →	2	3	12	
JR C, e	If C = 0, continue	•	•	•	•	•	•	00 111 000 ← e-2 →	2	2	7	If condition not met
	If C = 1, PC ← PC+e								2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	•	•	•	•	00 110 000 ← e-2 →	2	2	7	If condition not met
	If C = 0, PC ← PC + e								2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000 ← e-2 →	2	2	7	If condition not met
	If Z = 1, PC ← PC + e								2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	•	•	•	•	00 100 000 ← e-2 →	2	2	7	If condition not met
	If Z = 0, PC ← PC + e								2	3	12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101 11 101 001	2	2	8	
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101 11 101 001	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	•	•	•	•	00 010 000 ← e-2 →	2	2	8	If B = 0
	If B ≠ 0, PC ← PC + e								2	3	13	If B ≠ 0

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <-126, 129>.

e-2 in the op-code provides an effective address of pc + e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ‡ = flag is affected according to the result of the operation.

JUMP GROUP
TABLE 7.0-9

32
3

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	(SP-1)←PC_H (SP-2)←PC_L PC←nn	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3	3	10	If cc is false
									3	5	17	If cc is true
RET	PC_L←(SP) PC_H←(SP+1)	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	1	1	5	If cc is false
									1	3	11	If cc is true
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	(SP-1)←PC_H (SP-2)←PC_L PC_H←0 PC_L←P	•	•	•	•	•	•	11 t 111	1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
↕ = flag is affected according to the result of the operation.

CALL AND RETURN GROUP
TABLE 7.0-10

20

31

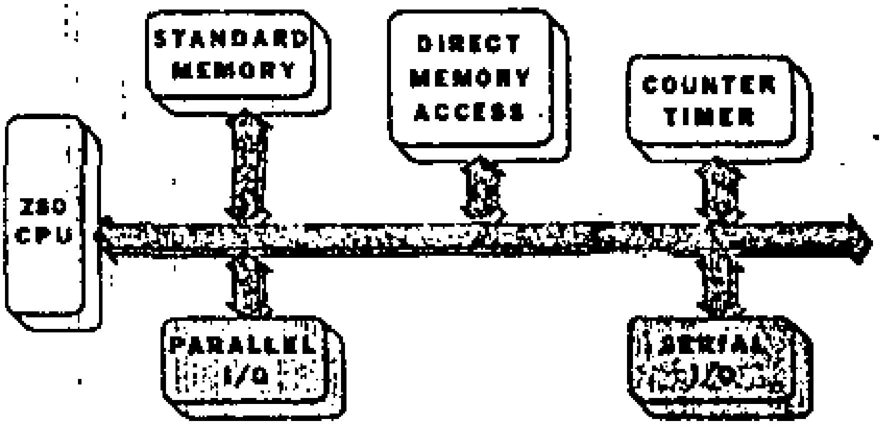
Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) If r = 110 only the flags will be affected	•	↕	P	↕	0	0	11 101 101 01 r 000	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	•	↕ ①	X	X	1	X	11 101 101 10 100 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 110 010	2	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
									2	4 (If B = 0)	16	
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	•	↕ ①	X	X	1	X	11 101 101 10 101 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 111 010	2	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
									2	4 (If B = 0)	16	
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	•	↕ ①	X	X	1	X	11 101 101 10 100 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 110 011	2	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
									2	4 (If B = 0)	16	
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	•	↕ ①	X	X	1	X	11 101 101 10 101 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 111 011	2	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
									2	4 (If B = 0)	16	

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

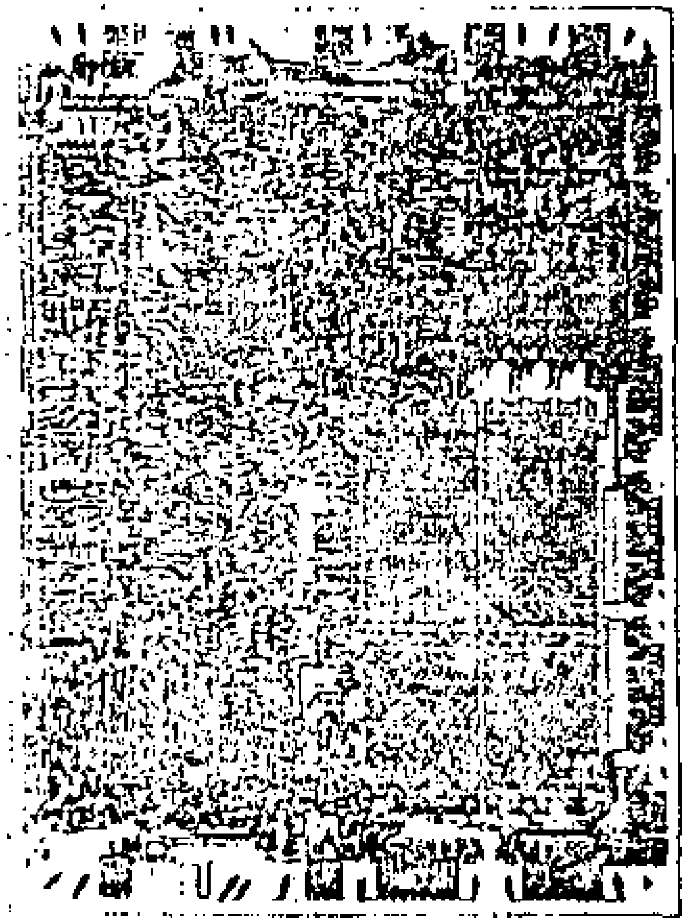
Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

INPUT AND OUTPUT GROUP
TABLE 7.0-11

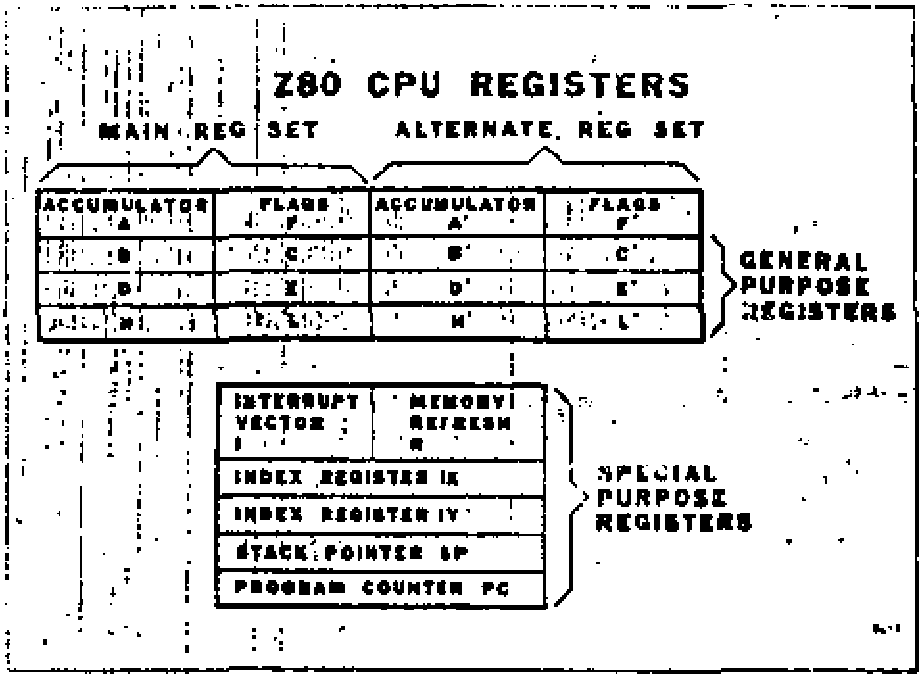
MOSTEK Z80 FAMILY



NOTES:



NOTES:



Z80 CPU REGISTERS

MAIN REG SET		ALTERNATE REG SET	
ACCUMULATOR A	FLAGS F	ACCUMULATOR A'	FLAGS F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

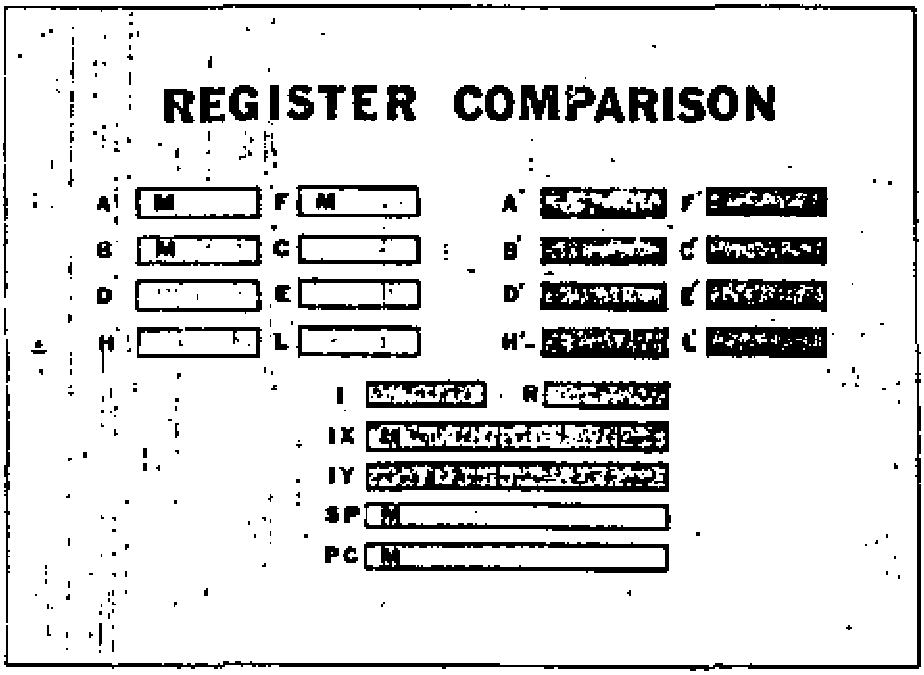
GENERAL PURPOSE REGISTERS

INTERRUPT VECTOR I	MEMORY REFRESH R
INDEX REGISTER IX	
INDEX REGISTER IY	
STACK POINTER SP	
PROGRAM COUNTER PC	

SPECIAL PURPOSE REGISTERS

NOTES:

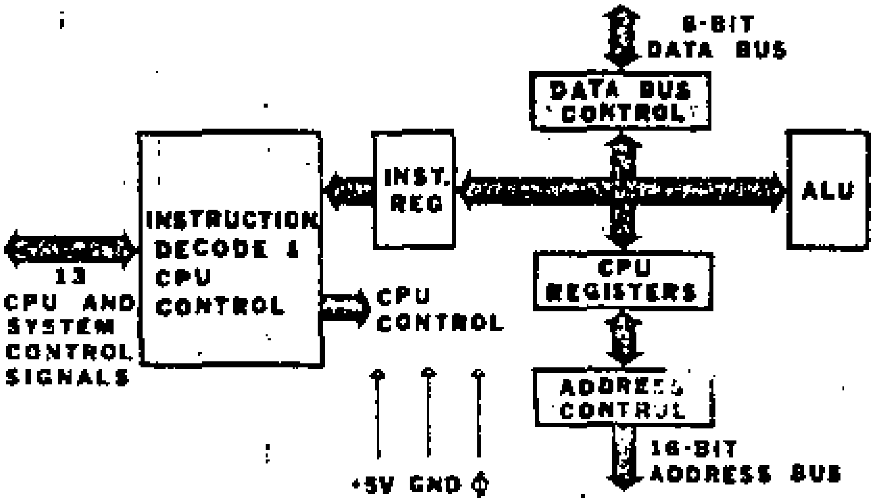
□ More than twice the registers of the 8080A allows faster interrupt service routines as well as more flexibility in programming.



REGISTER COMPARISON

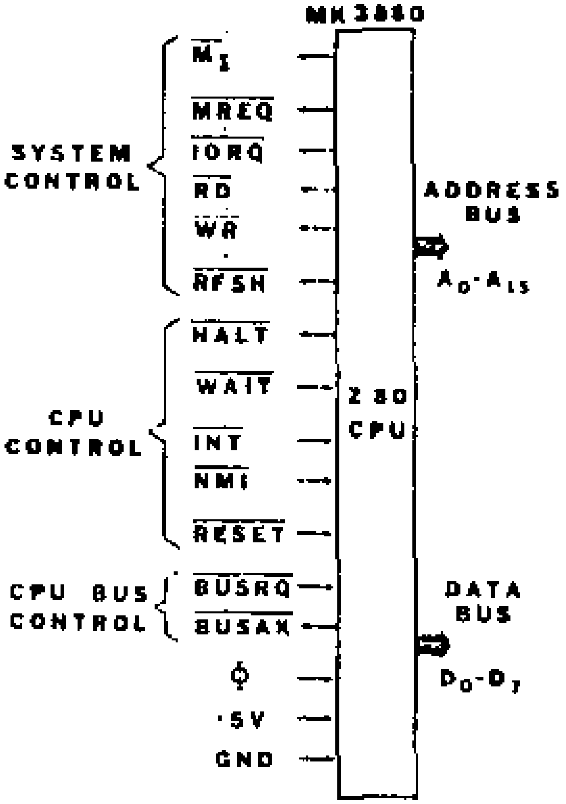
NOTES:

Z80 CPU BLOCK DIAGRAM



NOTES:

MOSTEK Z80 CPU

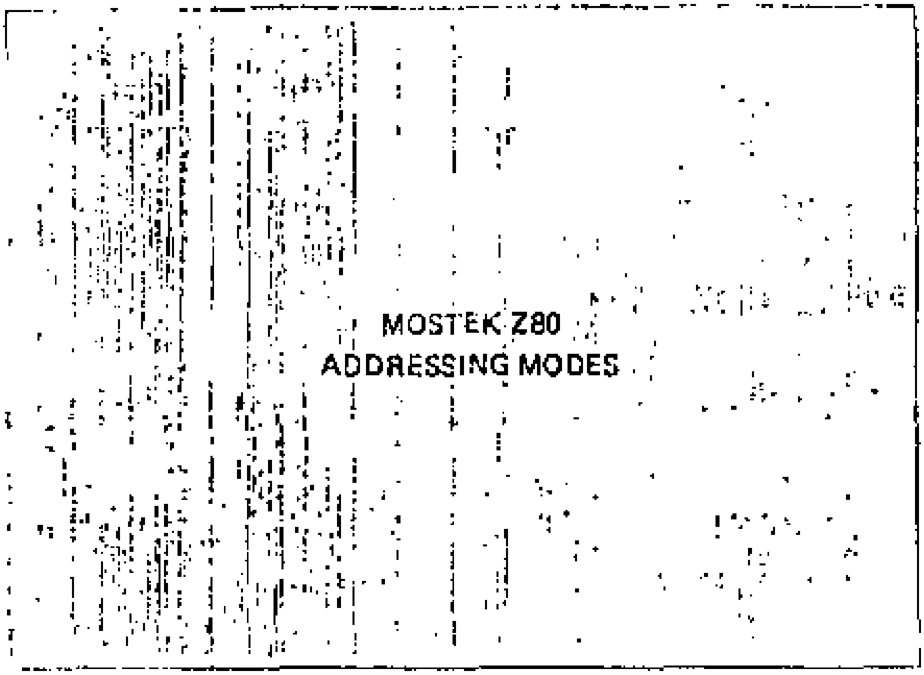


(1) All status signals are provided at the pins of the Z80 thereby eliminating the external status latch

(2) All signals except the single phase clock are TTL level compatible.

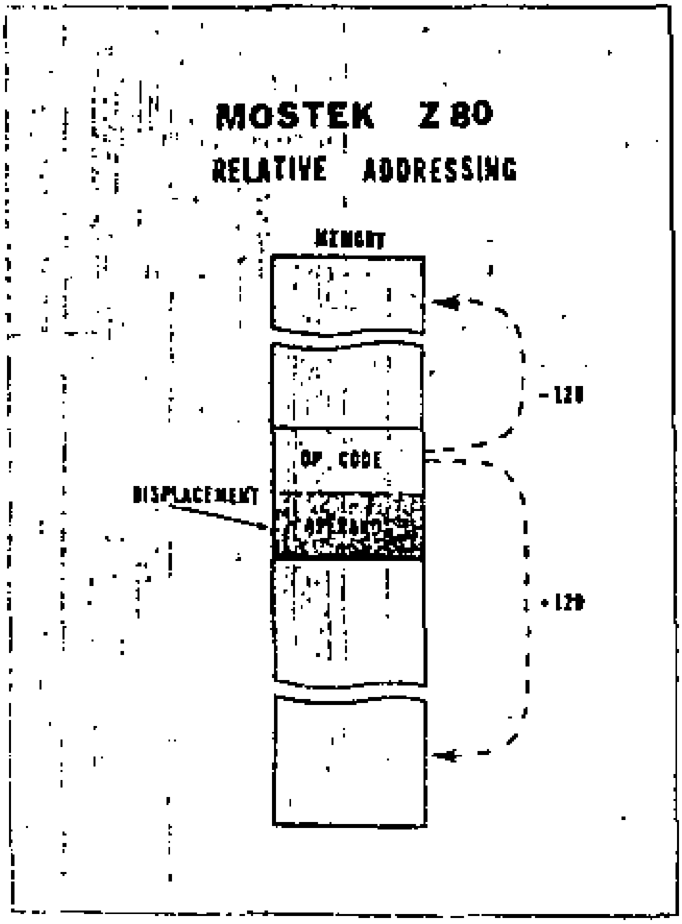
NOTES:

00. 37



MOSTEK Z80
ADDRESSING MODES

NOTES:

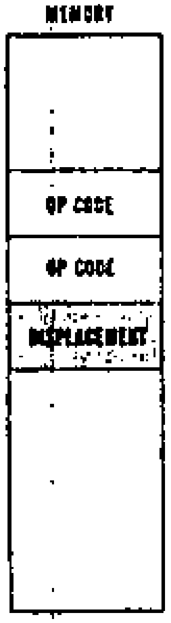


MOSTEK Z 80
RELATIVE ADDRESSING

□ The Relative Addressing Mode of the Z80 produces shorter programs because of the availability of two byte jumps rather than the normal three byte jumps.

NOTES:

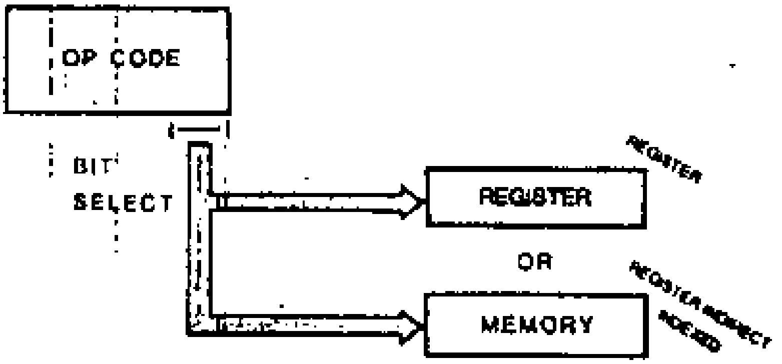
MOSTEK Z80
INDEXED ADDRESSING



(1) In addition to providing two additional 16 bit memory pointers the indexed addressing mode provides for efficient handling of blocks of data. The second index register allows efficient handling of two tables of data simultaneously.

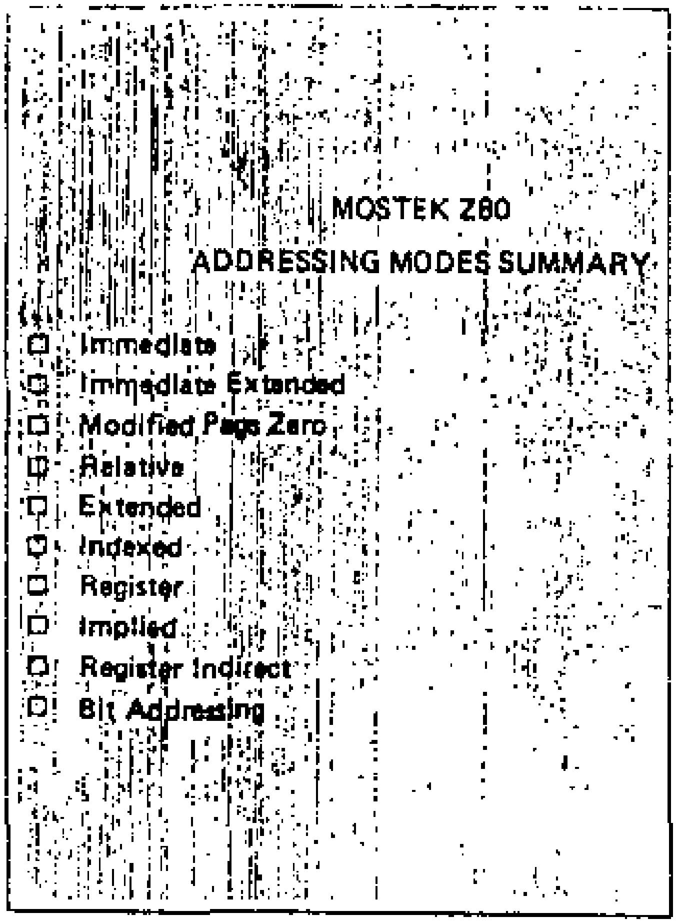
NOTES:

MOSTEK Z80
BIT ADDRESSING



The Bit Addressing Mode allows specifying an individual data bit anywhere in memory or in any CPU register. This avoids having to put data in the accumulator to mask out bits in order to modify or test an individual bit.

NOTES:

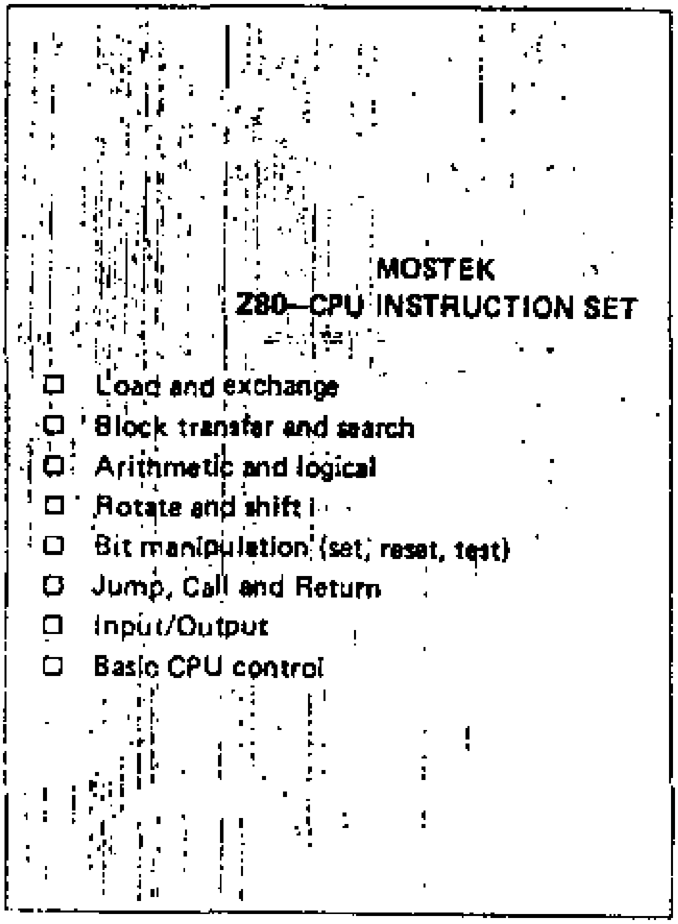


MOSTEK Z80

ADDRESSING MODES SUMMARY

- □ Immediate
- □ Immediate Extended
- □ Modified Page Zero
- □ Relative
- □ Extended
- □ Indexed
- □ Register
- □ Implied
- □ Register Indirect
- □ Bit Addressing

NOTES:



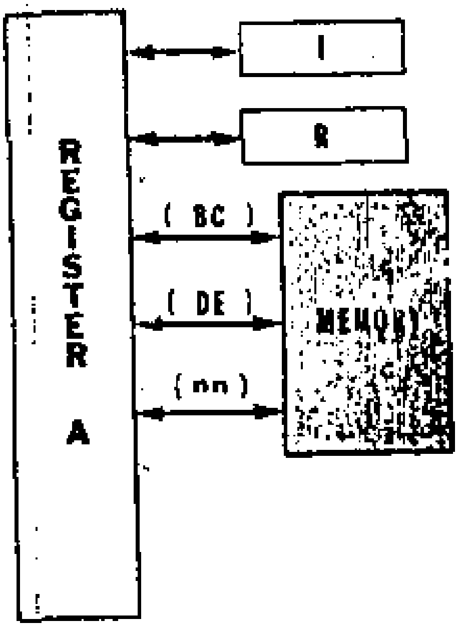
MOSTEK

Z80—CPU INSTRUCTION SET

- □ Load and exchange
- □ Block transfer and search
- □ Arithmetic and logical
- □ Rotate and shift
- □ Bit manipulation (set, reset, test)
- □ Jump, Call and Return
- □ Input/Output
- □ Basic CPU control

NOTES:

8-BIT LOAD GROUP



REGISTER A

I

R

(BC)

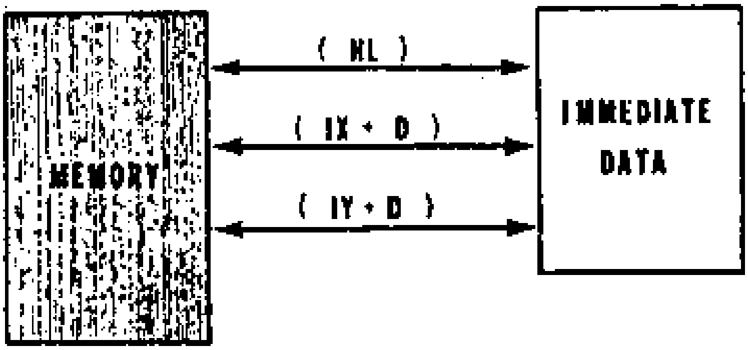
(DE)

(nn)

MEMORY

NOTES:

8-BIT LOAD GROUP



MEMORY

(HL)

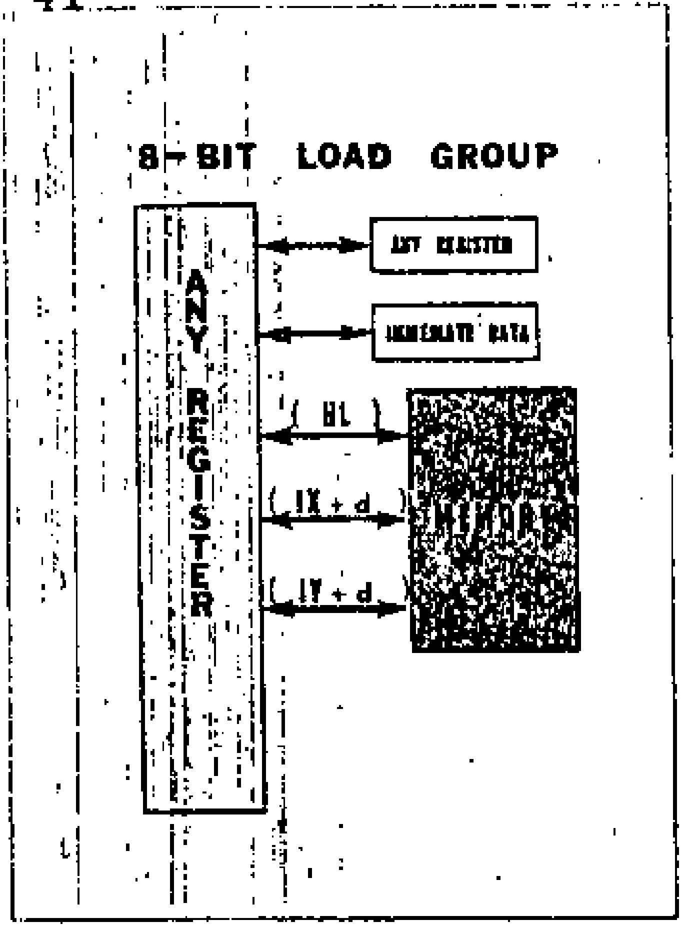
(IX + D)

(IY + D)

IMMEDIATE DATA

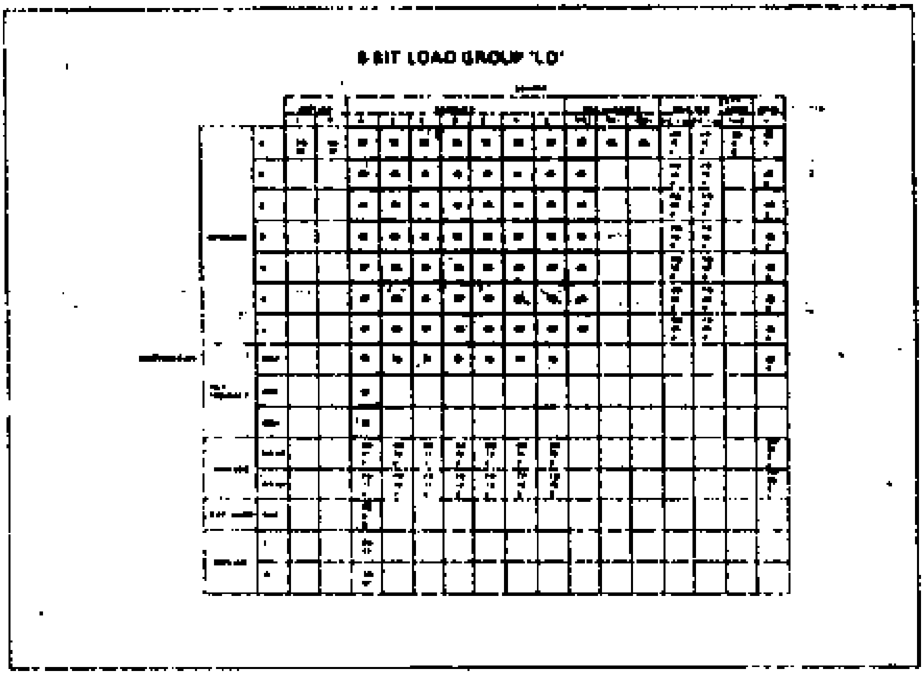
NOTES:

The index registers provide two additional memory pointers to transfer data between memory and the CPU's registers.



8-BIT LOAD GROUP

NOTES:

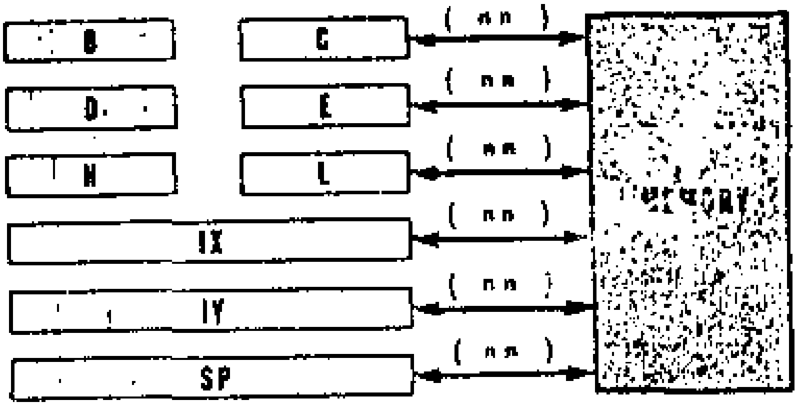


8 BIT LOAD GROUP 'LD'

NOTES:

NOTES:

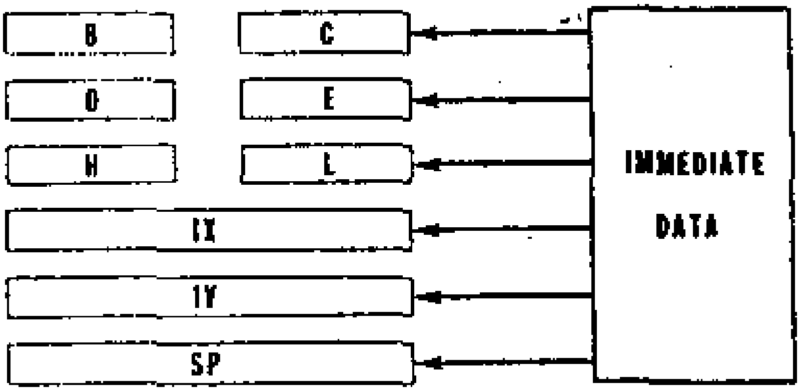
16—BIT LOAD GROUP



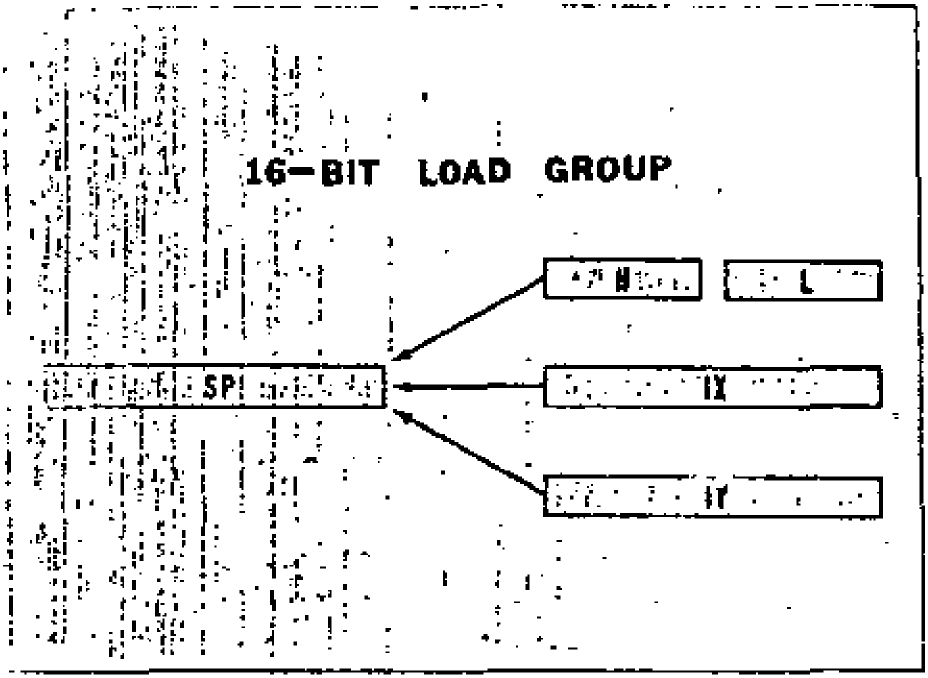
The extended addressing mode applies to all 16 bit registers instead of only the HL register pair as in the 8080A. This allows more flexibility in accessing 16 bit operands for subsequent processing by the Z80 16 bit instructions.

NOTES:

16—BIT LOAD GROUP



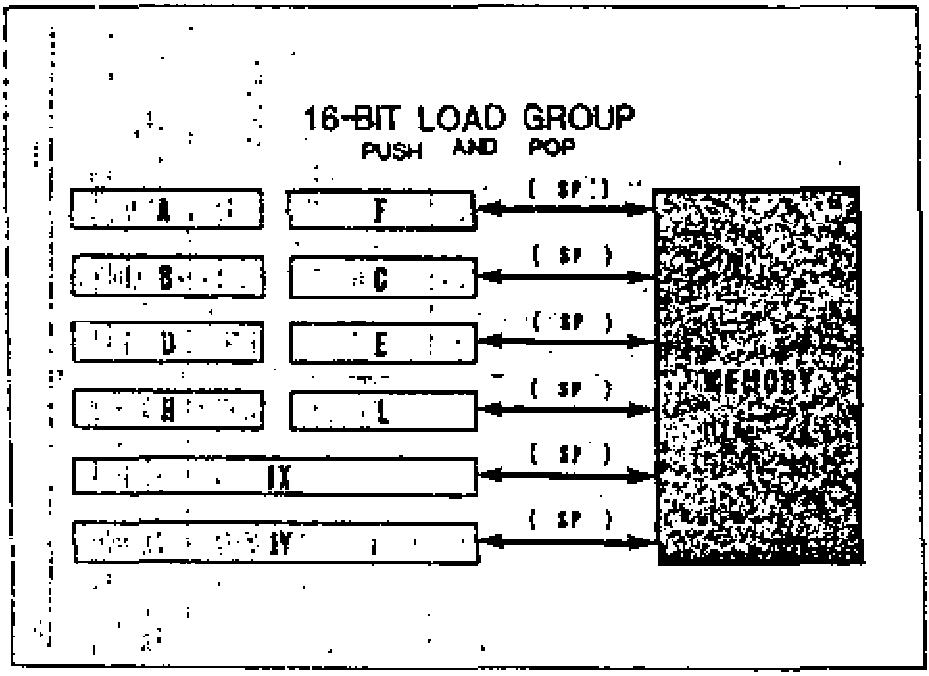
16—BIT LOAD GROUP



NOTES:

16-BIT LOAD GROUP

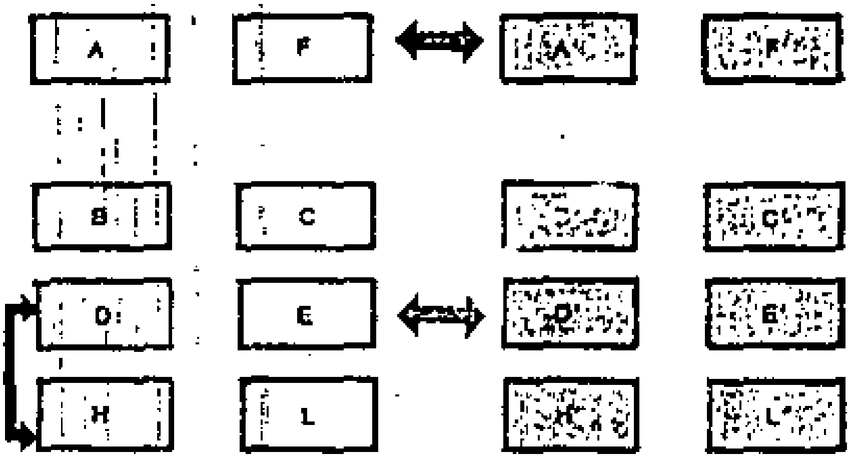
PUSH AND POP



NOTES:

□ All 16 bit registers can be PUSHed on or POPed off of the Stack thus allowing several tasks to use the same registers (re-entrent programs).

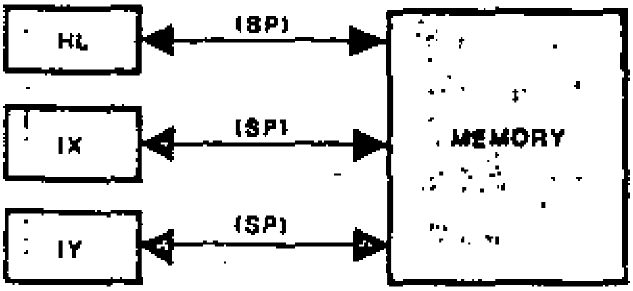
EXCHANGE GROUP



NOTES:

- The use of the exchange instructions in the Z80 can provide very fast interrupt service routines.

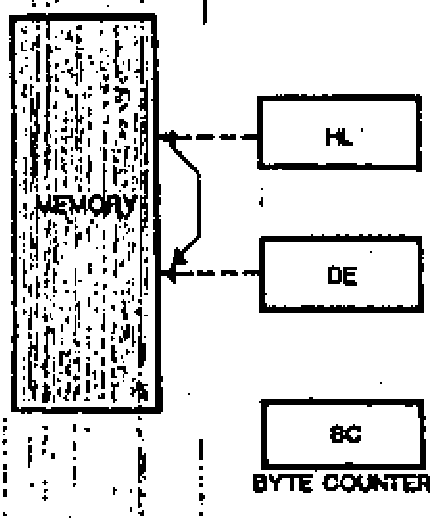
EXCHANGE GROUP



NOTES:

45 4/3

BLOCK TRANSFER GROUP



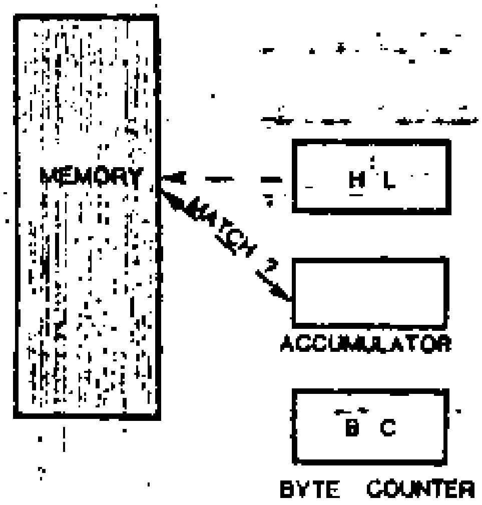
1. INC HL & DE. DEC BC
2. INC HL & DE. DEC BC. REPEAT UNTIL BC = 0
3. DEC HL & DE. DEC BC
4. DEC HL & DE. DEC BC. REPEAT UNTIL BC = 0

□ These instructions are very useful in most terminal applications where data movement is a large task for the microprocessor.

□ These single instructions in the Z80 are equivalent to subroutines in second generation architectures and can move data at a rate of 8.4 μ sec/Byte.

NOTES:

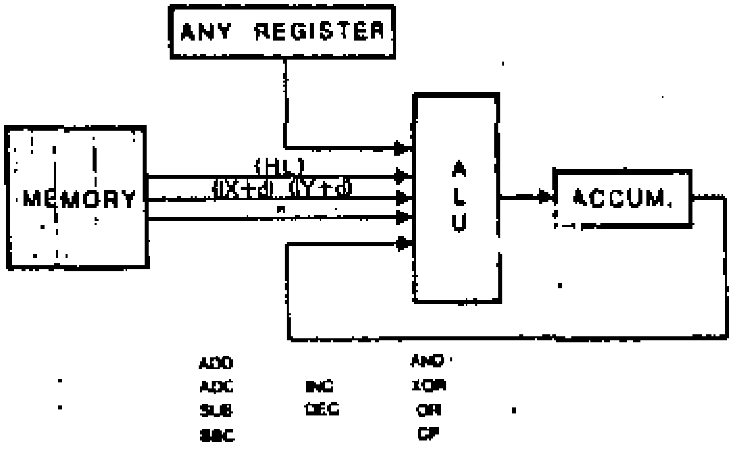
BLOCK SEARCH GROUP



1. INC HL.DEC BC
2. INC HL.DEC BC.REPEAT UNTIL 'MATCH' OR BC=0
3. DEC HL.DEC BC
4. DEC HL.DEC BC.REPEAT UNTIL 'MATCH' OR BC=0

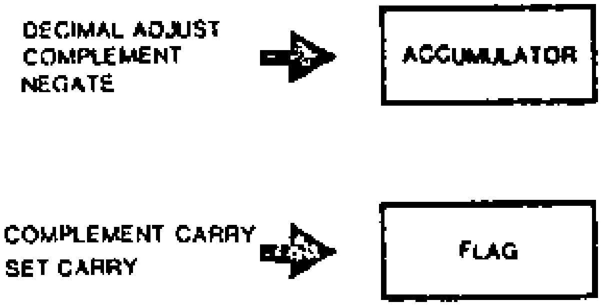
NOTES:

8 BIT ARITHMETIC AND LOGIC



NOTES:

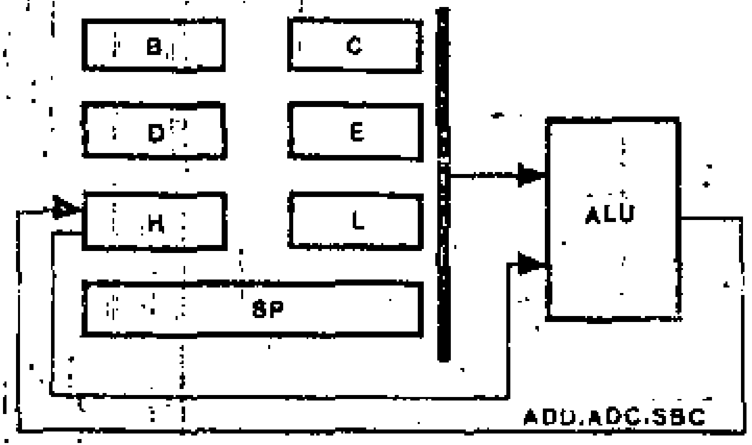
GENERAL PURPOSE AF OPERATIONS



NOTES:

47 4/5

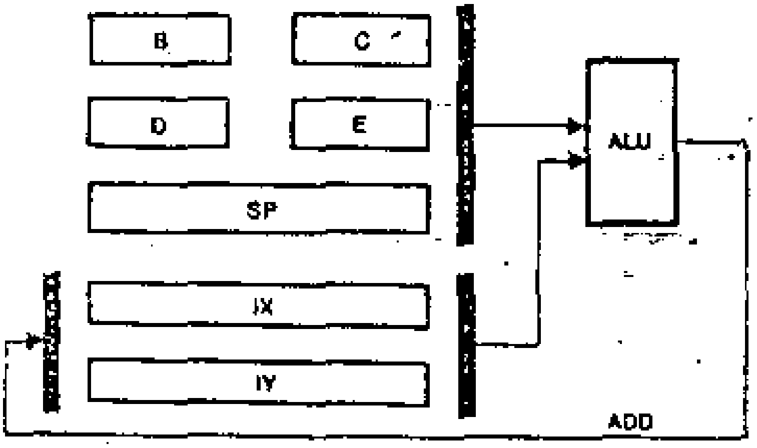
16 BIT ARITHMETIC



NOTES:

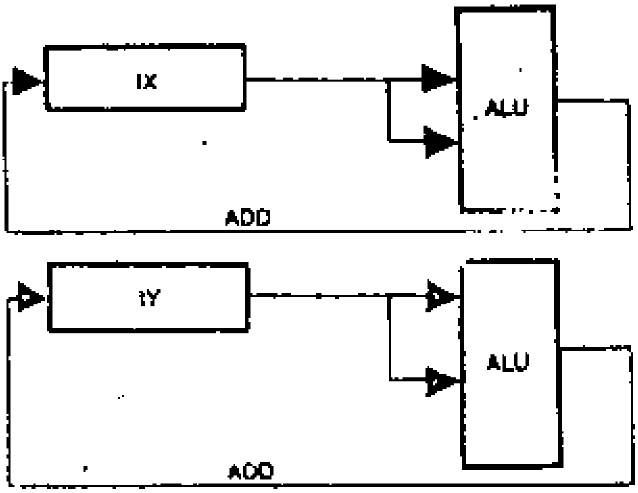
□ The addition of the ADC and SBC instructions in the Z80 allows more efficient multi-precision math routines by using the HL register pair as a 16 bit accumulator.

16-BIT ARITHMETIC



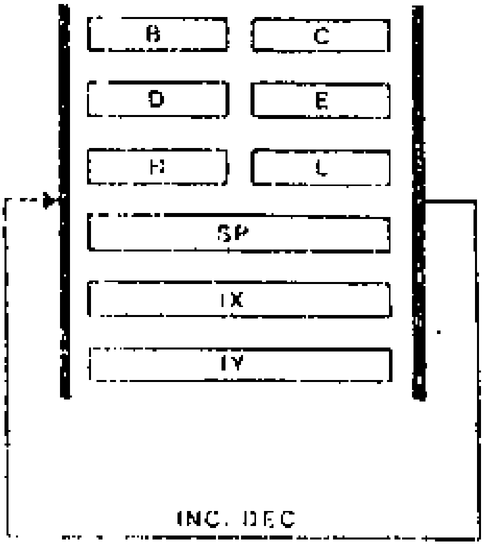
NOTES:

16-BIT ARITHMETIC



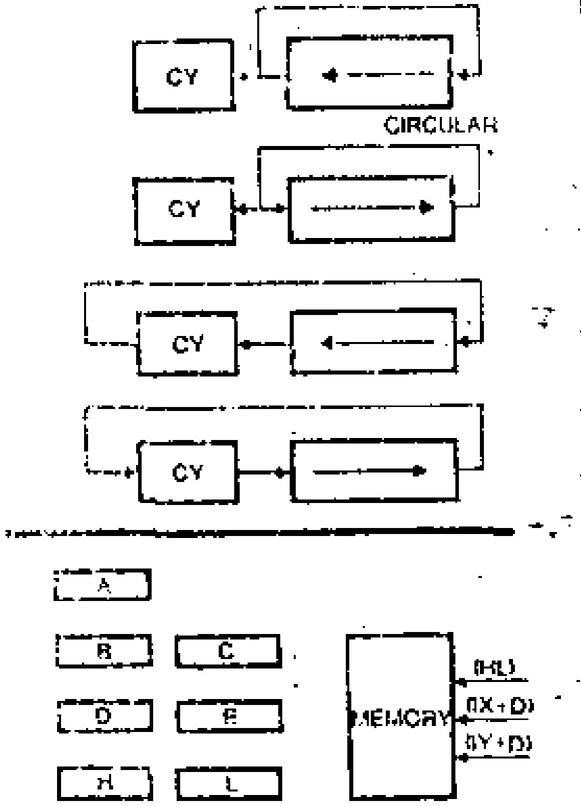
NOTES:

16-BIT ARITHMETIC



NOTES:

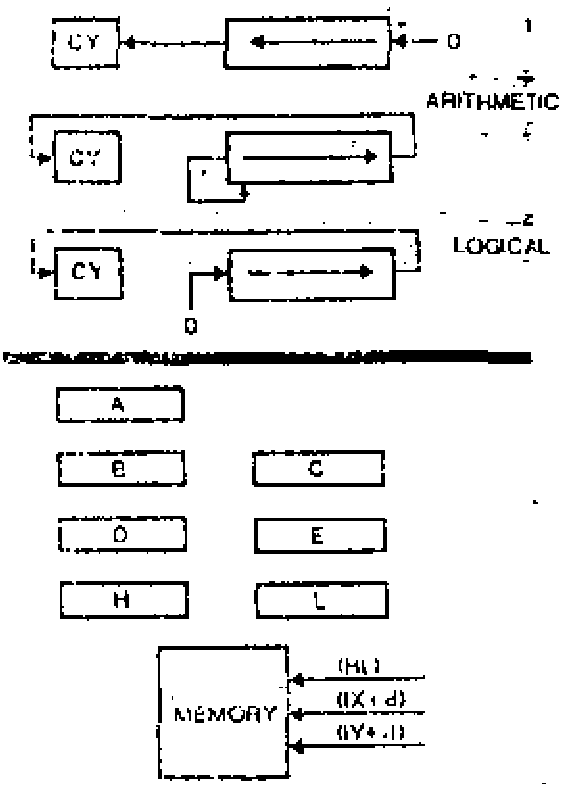
ROTATES



[illegible] Z80 Rotate and Shift instructions operate on all memory locations as well as all internal registers.

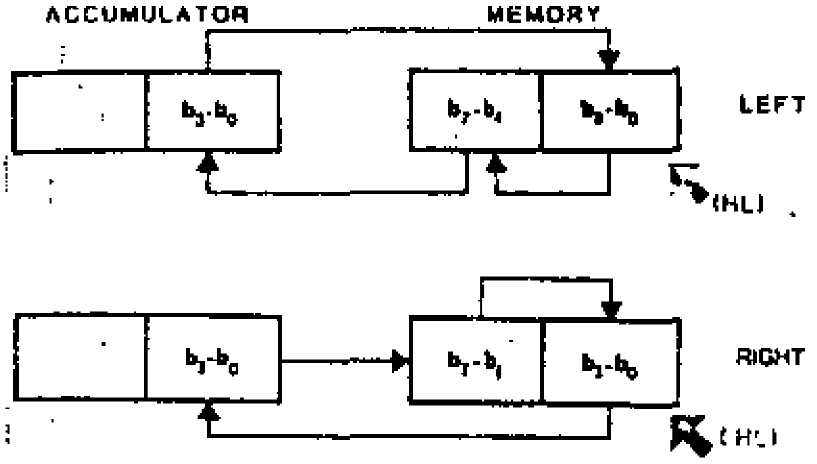
NOTES:

SHIFTS



NOTES:

ROTATE DIGIT



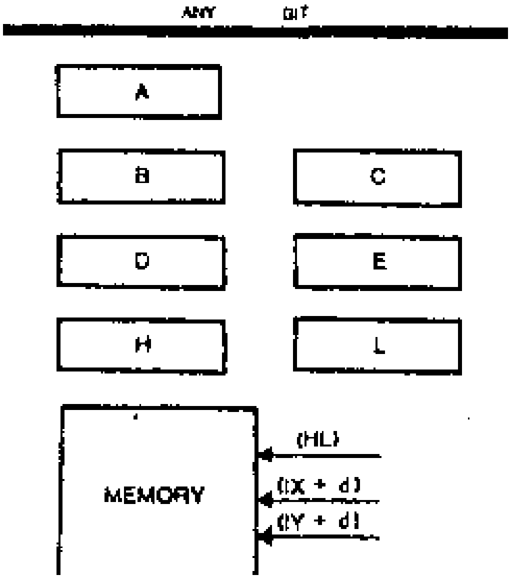
NOTES:

BIT MANIPULATION

TEST

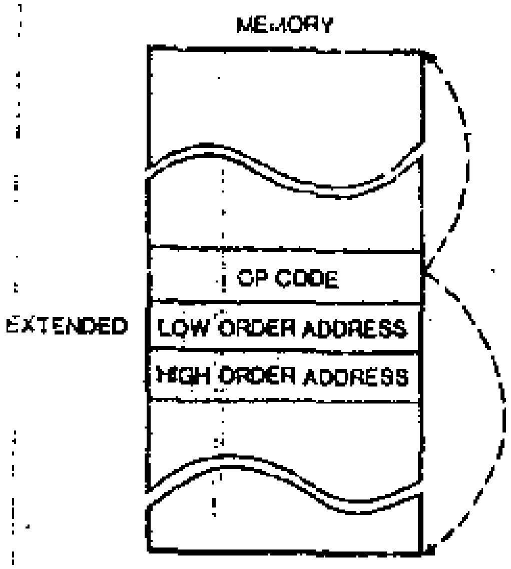
RESET

SET



NOTES:

JUMP



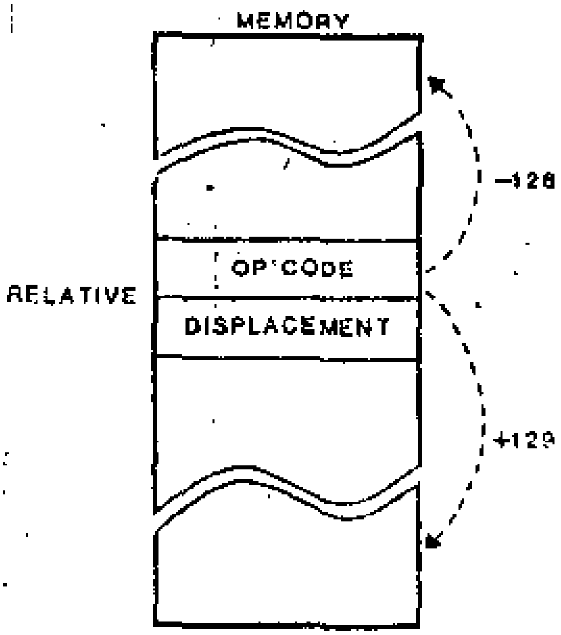
CONDITIONAL

CARRY
ZERO
SIGN
PARITY/OVERFLOW

UNCONDITIONAL

NOTES:

JUMP



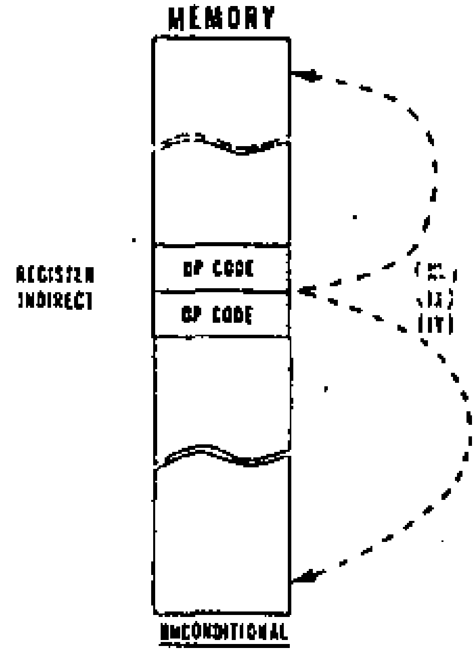
CONDITIONAL

CARRY
ZERO
DEC B JUMP IF NON ZERO

UNCONDITIONAL

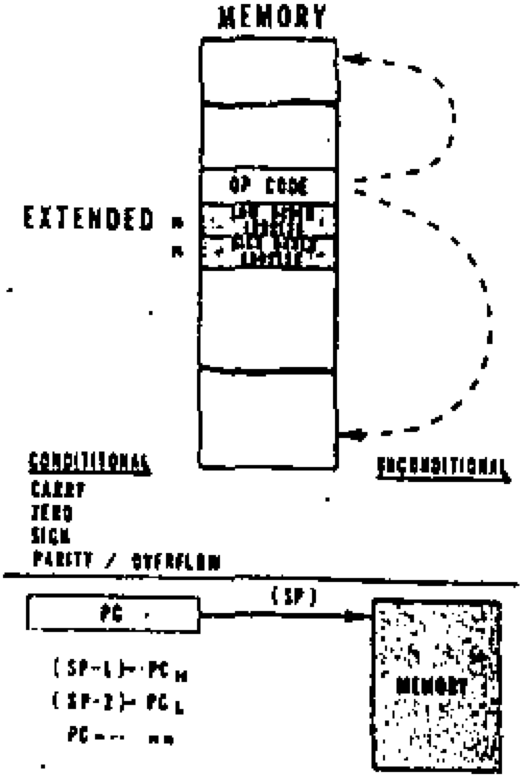
NOTES:

JUMP



NOTES:

CALL



$(SP-1) \leftarrow PC_H$

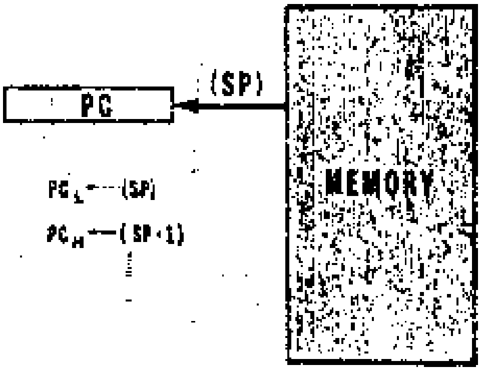
$(SP-2) \leftarrow PC_L$

$PC \leftarrow nn$

NOTES:

NOTES:

RETURN



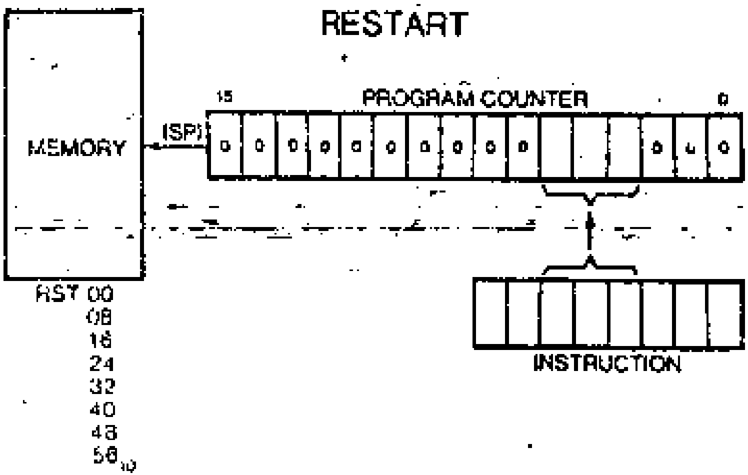
$PC_L \leftarrow (SP)$

$PC_H \leftarrow (SP+1)$

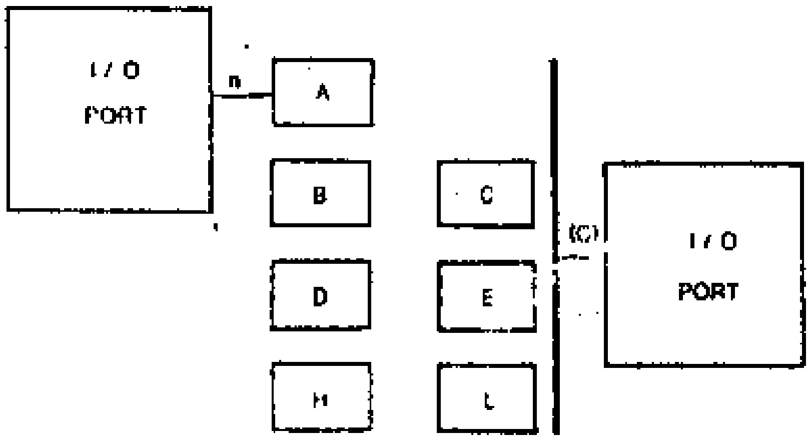
CONDITIONAL	UNCONDITIONAL
CARRY	
ZERO	
SIGN	RETURN FROM INTERRUPT
PARITY / OVERFLOW	RETURN FROM NON-MASKABLE INTERRUPT

NOTES:

RESTART



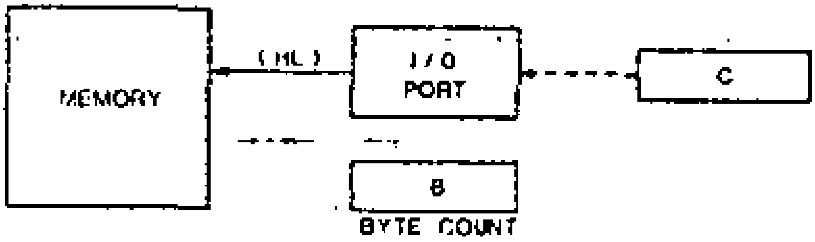
INPUT GROUP



NOTES:

The Z80 can input data from any port to any eight bit CPU register thus eliminating the bottleneck in the Accumulator.

BLOCK INPUT

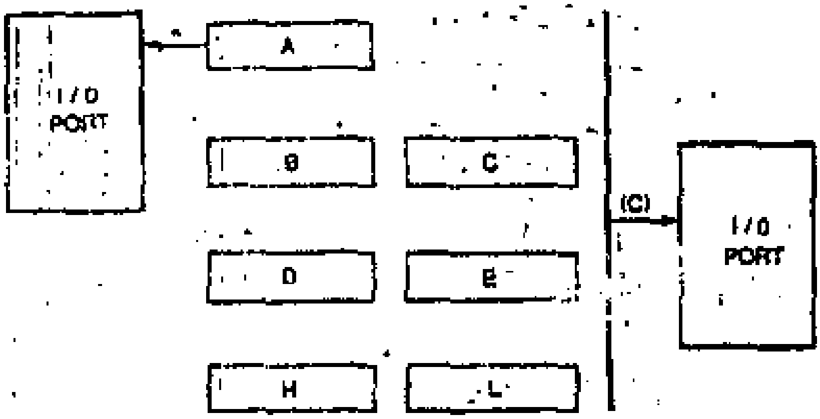


1. INC HL, DEC B
2. INC HL, DEC B, REPEAT IF B ≠ O
3. DEC HL, DEC B
4. DEC HL DEC B REPEAT IF B ≠ O

NOTES:

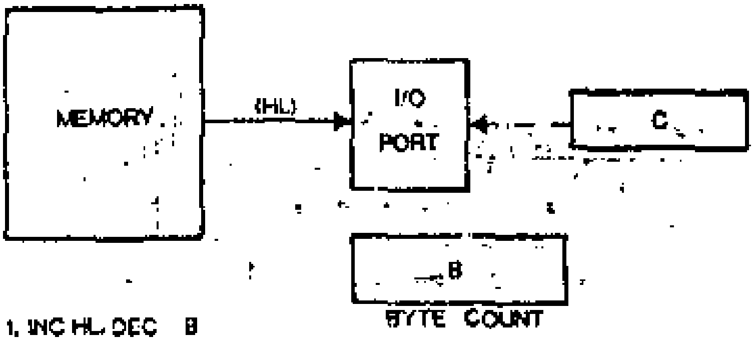
The Block Input or Output instructions can be thought of as a "Software" DMA. Data is moved from a peripheral to memory at a rate of 8 µs/Byte.

OUTPUT GROUP



NOTES:

BLOCK OUTPUT

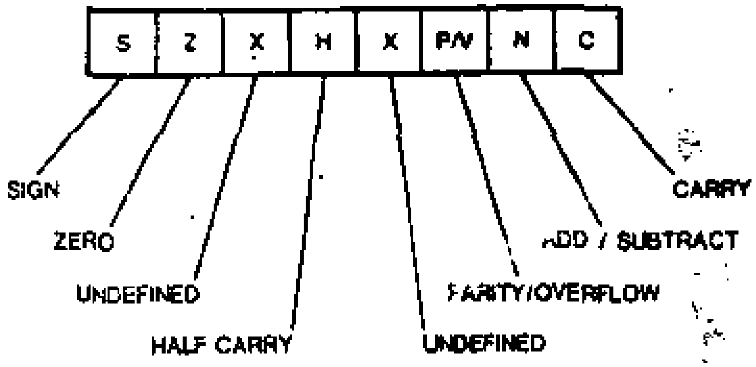


1. INC HL, DEC B
2. INC HL, DEC B, REPEAT IF B ≠ 0
3. DEC HL, DEC B
4. DEC HL, DEC B, REPEAT IF B ≠ 0

NOTES:

NOTES:

Z80 FLAG REGISTER



NOTES:

HOW TO SAVE MEMORY BYTES IN YOUR 8080 PROGRAM WITH THE Z80

□ Use Block Instructions for auto updating of memory pointers.

□ Exchange instructions speed up Interrupt Service Routines with fewer Bytes.

□ Index Register provides additional memory pointers plus indexed mode of addressing.

□ New 16 bit Arithmetic instructions perform multi-precision math with fewer bytes.

□ Rotates and Shifts can be done on all registers as well as external memory to avoid the Accumulator bottleneck.

□ Bit Addressing saves masking out of bits in the Accumulator.

□ Relative Addressing saves Bytes, in short loops and allows relocatable programs.

□ In and Out instructions using an Indirect Port Address allows one I/O Routine to service multiple peripherals.

52

NOTES:

BENCHMARK SUMMARY

Benchmark *	Execution Speed Relative to Z80		Memory Bytes Required Relative to Z80	
	8080A	6800	8080A	6800
Triple Precision Binary Multiply	2.0	1.4	1.3	1.4
Move A Block Of Data	2.3	4.8	1.5	1.7
Search A Memory Block For A Substring	2.2	1.9	1.3	1.3
Interrupt Driven I/O	2.4	1.7	2.1	1.2

*See MOSTEK Z80 Comparison Report For Details

NOTES:

Z80 INTERRUPT MODES

□ Mode 0 – Jam Next Instruction On the Data Bus like The 8080A

□ Mode 1 – Automatic Restart to Hex 38

□ Mode 2 – Fetch Eight Bit Vector From Interrupting Device

Combine With I Register To Form 16 Bit Table Pointer

Get Service Routine starting address From Table

□ Non Maskable – Automatic Restart to Hex 66

□ Mode 2 is used by the Z80 Peripheral devices to form a powerful interrupt structure while eliminating the need for an external interrupt controller.

SOFTWARE

EN

MICROCOMPUTADORAS

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

SOFTWARE EN MICROCOMPUTADORAS

M en C Angel Kuri Morales

Julio, 1981

Ensamblador:

La mayoría de los microprocesadores que se encuentran en el mercado tienen como soporte de programación programas que facilitan al usuario la tarea de introducir y ejecutar sus programas, entre éstos se tiene a monitores, editores, ensambladores, traductores, intérpretes y compiladores.

El propósito de esta plática será el conocer la actividad del programa llamado monitor y la del ensamblador.

Monitor:

Un monitor nos permite por lo regular introducir código a memoria, desplegarlo en algún tipo de display, cambiarlo, aumentarlo o reducirlo, copiarlo de una sección de memoria a otra, observar el contenido de los registros del sistema, agregar puntos de ruptura (break points) para monitorear al programa y por último ejecutarlo a partir de una dirección dada por el usuario.

Este programa (por lo regular pequeño) recibe de un teletipo o pantalla comandos de un carácter, y consta de algunas rutinas de utileria para apoyar en cierto modo al usuario. Por ejemplo: conversión de códigos ASCII-BCD, BCD-ASCII.

La entrada comunmente se realiza introduciendo en ASCII el programa en código de máquina, representación hexadecimal, las rutinas de apoyo traducen el ASCII al hexadecimal correspondiente (representación interna), ocupando 4 bits de cada carácter. Una vez terminada la inserción del programa, éste puede ejecutarse. Esta labor es compleja ya que el usuario cada vez introduce todo el código del programa en forma numérica, lo cual incrementa la posibilidad de falla.

Ensamblador:

El programa ensamblador surge de la necesidad de facilitar la interfase entre hombre-máquina. Varios son sus propósitos pero quizás el más importante es el de facilitar la escritura y chequeo del flujo de un programa, ya que la entrada al ensamblador se realiza utilizando mnemónicos fácilmente reconocibles y memorizables por el usuario en lugar de los códigos de máquina correspondientes.

Varias son las rutinas que integran al programa ensamblador, la primera denominada scanner lee una línea de código revisando formato previamente establecido. (Libre o fijo) pudiendo discriminar códigos válidos de etiquetas.

Una etiqueta simplemente representa el nombre de una dirección, a la que, durante el desarrollo del programa se hará mención.

La segunda rutina tiene como finalidad la formación del código correspondiente al mnemónico de entrada dejando libres las localidades que requieran posteriormente ser resueltas.

5-3 WRITING TO THE DISPLAY

The following program causes the character "8" to move from right to left across the display.

```
2000   3E 00               LD     A,00H
2002   D3 88               OUT    (88H),A     ;ACTIVATE ALL SEGMENTS
2004   3E 01               LD     A,01H
2006   D3 8C        LOOP:  OUT    (8CH),A     ;SELECT FIRST DIGIT
2008   CD 4F  06           CALL   D20MS
200B   CD 4F  06           CALL   D20MS
200E   CD 4F  06           CALL   D20MS
2011   CD 4F  06           CALL   D20MS
2014   CD 4F  06           CALL   D20MS       ;DELAY APPROX 100MS
2017   07                  RLCA               ;ROTATE TO NEXT DIGIT
2018   18 EC               JR     LOOP-$      ;LOOP BACK
```

Enter the program and Execute address 2000. Use MON key to return to ZBUG. This is not a good program to Single Step through, as the Single Step routines use the display which destroys the "8" character loaded in the first two instructions.

5-4 INTERRUPT DRIVEN DELAY

The following program uses Channel zero of the Z80-CTC to interrupt after a fixed delay, rather than the software timing loop (D20MS) used in the previous example. This program uses the Z80 Mode 2 interrupts in which the interrupting device sends in a vector during the Interrupt Acknowledge [illegible] se apoya en tablas de información, la primera [illegible] códigos de traducción de mnemónico a código de máquina, la segunda (tabla de etiquetas) lleva el recuento de las direcciones a las que se ha asignado un nombre especial o sea el nombre etiqueta.

En general la mayoría de los ensambladores leen varias veces (2 veces) el programa escrito en mnemónicos para completar aquellas localidades que en pasos anteriores no pudieron ser resueltos. A esta actividad la bibliografía se refiere como número de pasadas del ensamblador.
A continuación se muestra la salida del ensamblador habiendo ensamblado correctamente un programa cualquiera.
En los listados anteriores observamos la salida del programa ensamblador, en ella la primer columna nos indica la dirección física en donde fueron ensamblados los códigos de máquina de la segunda columna, los que ha su vez corresponden a los mnemónicos utilizados por el usuario, en la misma línea del programa.

Errores:

Otra rutina del programa ensamblador a la cual puede llamarse en cualquier instante es la rutina de error la cual contabiliza los errores del usuario haciéndocelo saber al final. Son errores: la inclusión de mnemónicos no establecidos, la definición de etiquetas fuera de la sintaxis aceptada, los saltos o llamadas a subrutina a etiquetas inexistentes, etc.

A continuación se analizarán los siguientes programas con la finalidad de introducirnos a la programación con los mnemónicos del sistema Z-80, estos programas serán explicados en el curso.

Accumulator is rotated left once each time the breakpoint is encountered. Reset using S1 to clear CTC.

```
;INTERRUPT DRIVEN DELAY
                      ORG  2000H
2000 3E 21            LD   A,21H
2002 ED 47            LD   I,A       ;INITIALIZE I=21
2004 31 00 23         LD   SP,2300H ;INITIALIZE STACK POINTER
2007 3E 00            LD   A,00H
2009 D3 84            LD   (84H),A   ;CTC VECTOR
200B 3E A5            LD   A,0A5H
200D D3 84            OUT  (84H),A   ;CONFIGURE CTC-SEE PAGE 26
                                     ;OF MICRO REFERENCE MANUAL
200F 3E FF            LD   A,0FFH
2011 D3 84            OUT  (84H),A
2013 3E 01            LD   A,01H
2015 ED 5E            IM   2
2017 FB         LOOP: EI
2018 76               HALT
2019 C3 17 20         JP   LOOP
;TABLE OF INTERRUPT SERVICE ROUTINES
2100 00 22
;INTERRUPT SERVICE ROUTINE
2200 FB               EI             ;ENABLE INTERRUPTS
2201 07               RLCA           ;ROTATE ACCUMULATOR
2202 ED 4D            RETI           ;RETURN FROM INTERRUPT CLEAR
                                     ;CTC
```

5-5 GENERATE AN INTERRUPT FROM THE PIO

Place a 10K ohm pull-up resistor from $\overline{\text{ASTRB}}$ (marked near wire wrap area) to +5 volts. Enter the following program that initializes the PIO to accept an interrupt on the A Port. Set a BREAKPOINT at 2200 and Execute from 2000. The display will go dark as the Z80-CPU has executed the HALT instruction. Take a clip lead and momentarily touch $\overline{\text{ASTRB}}$ to GND, which will cause a PIO interrupt and the breakpoint should be displayed. For more details on the Z80-PIO operation, see the MOSTEK or Zilog PIO Technical Manual.

```
;PIO INTERRUPT TEST
     2000 3E 21   LD    A,21H
     2002 ED 47   LD    I,A        ;INITIALIZE I
     2004 3E 00   LD    A,00
     2006 D3 82   OUT   (82H),A    ;VECTOR
     2008 3E 4F   LD    A,4FH      ;SET UP PIO FOR INPUT
     200A D3 82   OUT   (82H),A    ;MODE - SEE PAGE 25 OF MI-
     200C 3E 87   LD    A,87H      ;CRO REF MANUAL MODE TWO
     200E D3 82   OUT   (82H),A    ;INTERRUPTS
     2010 ED 5E   IM    2
     2012 FB      EI
     2013 76      HALT

     ;INTERRUPT SERVICE ROUTINE TABLE
     2100 00 22
```

```
;INTERRUPT SERVICE ROUTINE
2200 FB       EI
2201 ED 4D    RETI              ;RETURN AND CLEAR PIO
```

5-6 USING THE Z80 STARTER KIT AS AN EPROM PROGRAMMER

Due to limitations in the memory allotted for the ZBUG Monitor, (it had to fit in a 2K byte ROM) some restrictions were placed on the operation of the EPROM programmer. These restrictions are fully documented in Section 3-13 and should not hinder the average user of the Z80 STARTER KIT.

Should a user desire to program the entire contents of a 2758 (1024 bytes) or a 2716 (2048 bytes) EPROM, the above mentioned restrictions need to be removed. The following two example programs give the user the capability of copying any block of memory to any other block of memory, plus allowing any RAM location to be the source of data for the EPROM programmer. These programs can be put on cassette tape to be loaded into RAM when needed, or they can be put into EPROM in the PROM1 socket for on-line use.

The first program is a copy utility based on the Z80 block move instruction. It can be used to copy a block of data from any memory location (source data) to any new memory location (destination data). It can also be used to copy the data from an EPROM in either socket PROM1 or PROM2 into RAM for minor modification before re-programming. To use the following program initialize the Z80 registers using the REG EXAM key as follows:

Because the 2758/2716 EPROMs can be programmed a block or section at a time, the capability is provided by the initialization of 23C2H and 23C3H to start the programming at any address in the EPROM. This feature can be used to program a full 2K byte EPROM with less than 1K of RAM in the standard kit by programming the EPROM a section at a time.

```
                          ;PROGRAM TO MOVE ANY RAM BLOCK TO ANY
                          ;STARTING ADDRESS IN EPROM
2000 3E 01     C12:  LD    A,01H
2002 32 DA 23        LD    (PRFLG),A  ;SET PROM PROG FLG
2005 E5              PUSH HL          ;BYTE COUNT IN HL
2006 C1              POP   BC         ;SAVE IT
2007 E5              PUSH HL
2008 3A C0 23        LD    A,(23C0H)  ;SOURCE DATA
200B 67              LD    H,A
200C 3A C1 23        LD    A,(23C1H)
200F 6F              LD    L,A
2010 3A C2 23        LD    A,(23C2H)  ;DESTINATION DATA
2013 57              LD    D,A
2014 3A C3 23        LD    A,(23C3H)
2017 5F              LD    E,A
2018 3E 25     C12A; LD    A,25H      ;CTC FOR 26 MS
201A D3 86           OUT   (86H),A    ;ZC/TO, NO INTR
201C 3E CB           LD    A,203D
201E D3 86           OUT   (86H),A    ;TIME CONST
201F 3E 80           LD    A,80H      ;CLEAR DISPLAY, SET
```

```
2022 D3 8C        OUT   (DIGLH),A ;PROM PROG EN = 1
2024 ED A0        LDI             ;WAIT STATE INSERTED
2026 3E 00        LD    A,00H     ;UNTIL CTC TIMES TWICE
2028 D3 8C        OUT   (DIGLH),A ;CLEAR PROM PROG EN
202A 3E 03        LD    A,03H     ;RESET CTC2
202C D3 86        OUT   (86),A
202E EA 18 20     JP    PE,C12A   ;LOOP BACK IF BC-1 NE 0
2031 C1           POP   BC        ;RESTORE BYTE COUNT
2032 3A C0 23     LD    A,(23C0H)
2035 67           LD    H,A
2036 3A C1 23     LD    A,(23C1H) ;SOURCE DATA
2039 6F           LD    L,A
203A 3A C2 23     LD    A,(23C2H)
203D 57           LD    D,A
203E 3A C3 23     LD    A,(23C3H) ;DEST DATA
2041 5F           LD    E,A
2042 C3 04 06     JP    CCS12B    ;USE ROM CODE
```

TRIPLE PRECISION BINARY MULTIPLY — Z80 LISTING 78

PROGRAM LABEL	INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
1. MLTPLY:	LD	BC,00H	;CLEAR PARTIAL SUM	3	4.0
2.	LD	DE,00H	;BUFFER IN ALTER–	3	4.0
3.	LD	HL,00H	;NATE REGISTER SET.	3	4.0
4.	EXX			1	1.6
5.	LD	BC,1800H	;LOAD B WITH 24	3	4.0
6. MLOOP:	LD	A,(APART+2)		3	2.8
7.	AND	01H	;RETAUB LSB	2	2.8
8.	CP	C	;COMPARE WITH PREVIOUS	1	1.6
9.	LD	C,A	; LSB & SWAP OLD/NEW	1	1.6
10.	JR	Z,SHIFT	;DOES CURRENT=PREVIOUS?	2	4.8/2.8
11.	OR	A		1	1.6
12.	LD	HL,(BPART+2)		3	4.0
13.	JR	Z,ADD	; ADD IF LSB IS 0	2	4.8/2.8
14.	EXX		; SUBTRACT MAND FROM	1	1.6

TRIPLE PRECISION BINARY MULTIPLY — Z80 LISTING (CONTINUED) 79

PROGRAM LABEL	INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
15.	LD	A,D	;PARTIAL SUM	1	1.6
16.	SUB	(HL)	;SUB. LS BYTE	1	2.8
17.	LD	D,A	;SAVE IT IN D	1	1.6
18.	DEC	HL	;DECREMENT B PART	1	2.4
19.	LD	A,C		1	1.6
20.	SBC	(HL)	;SUB. MIDDLE BYTES	1	2.8
21.	LD	C,A	;SAVE IT IN C	1	1.6
22.	DEC	HL		1	2.4
23.	LD	A,B		1	1.6
24.	SBC	(HL)	;SUB. MS BYTE	1	2.8
25.	LD	B, A	;SAVE IT IN B	1	1.6
26.	JR	SHIFTO	;GO TO SHIFT	1	4.8
27. ADD:	EXX		;ADD MAND TO PARTIAL	1	1.6
28.	LD	A,D	;SUM	1	1.6
29.	ADD	(HL)	;ADD LS BYTE	1	2.8
30.	LD	D,A	;SAVE RESULTS IN D	1	1.6
31.	DEC	HL		1	2.4

PROGRAM LABEL		INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
32.		LD	A,C	;LOAD MIDDLE BYTE FROM C	1	1.6
33.		ADC	(HL)	;SAVE IT IN C	1	2.8
34.		LD	C,A		1	1.6
35.		DEC	HL		1	2.4
36.		LD	A,B		1	1.6
37.		ADC	(HL)	;ADD MS BYTE	1	2.8
38.		LD	B,A	;SAVE IT IN B	1	1.6
39.	SHIFTO:	EXX		;SWAP REGISTERS	1	1.6
40.	SHIFT:	LD	HL, APART	;LOAD A PART POINTER	3	4.0
41.		SRA	(HL)		1	2.4
42.		INC	HL		1	2.4
43.		RR	(HL)		2	6.0
44.		INC	HL		1	2.4
45.		RR	(HL)		2	6.0
46.		EXX		;SWAP & SHIFT PARTIAL SUM	1	1.6
47.		SRA	B			3.2

PROGRAM LABEL		INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
48.		RR	C		2	3.2
49.		RR	D		2	3.2
50.		RR	E		2	3.2
51.		RR	H		2	3.2
52.		RR	L		2	3.2
53.		EXX			1	1.6
54.		DJNZ	MLOOP	;REPEAT UNLESS DONE	2	5.2/3.2
55.	DONE:	EXX			1	1.6
56.		LD	HL,P	;SAVE RESULT IN MEMORY	3	4.0
57.		LD	(HL),B	;STARTING AT P	1	2.8
58.		INC	HL		1	2.4
59.		LD	(HL),C		1	2.8
60.		INC	HL		1	2.4
61.		LD	(HL),D		1	2.8
62.		INC	HL		1	2.4
63.		LD	(HL),E		1	2.8

PROGRAM LABEL	INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
64.	INC	HL		1	2.4
65.	LD	(HL),H		1	2.8
66.	INC	HL		1	2.4
67.	LD	(HL),L		1	2.8

SiSTEMAS
INTEGRADOS
EN INGENIERIA

SEARCH A MEMORY BLOCK FOR A SUBSTRING – Z80 LISTING 90

PROGRAM LABEL	INSTRUCTION MNEMONICS	OPERAND FIELD	COMMENTS	BYTES	TIME IN µSEC
1. SEARCH:	LD	HL, BP	;INITIALIZE BLOCK POINTER	3	4.0
2.	LD	IX, SSP	;INITIALIZE SUBSTRING POINTER	4	5.6
3.	LD	D,SSL	;INITIALIZE SUBSTRING LENGTH	2	2.8
4.	LD	BC,BLNGT	;INITIALIZE BLOCK LENGTH	3	4.0
5. LOOP1:	LD	(SVBP),HL	;SAVE BLOCK POINTER	3	6.4
6. P2	LD	A,(IX+0)	;GET SS BYTE	3	7.6
7.	CPI		;COMPARE WITH BLOCK VALUE	2	6.4
8.	JR	Z,MATCH–$	;(HL) = A MATCH FOUND	2	2.8
9.	JP	PE,LOOP1	;NO MATCH CHECK NEXT BYTE	3	4.0
10.	JR	ENDBLK–$	;NO MATCH END OF BLOCK	2	4.8
11. CONT:	EX	AF,AF'	;RESTORE STATUS FROM CPI	1	1.6
12.	JR	NZ,ENDBLK–$	;BLOCK END HIT BEFORE SS END	2	2.8
13.	INC	IX	;INCREMENT SUBSTRING POINTER	2	4.0
14.	JR	P2–$	;CHECK NEXT BYTE	2	4.8
15. ENDBLK:	LD	HL,0000H	;NO MATCH RETURN WITH HL= 0	3	4.0
16.	JR	DONE –$		2	4.8
17. MATCH:	EX	AF,AF'	;SAVE STATUS FROM CPI	1	1.6
18.	DEC	D	;DECREMENT SS LENGTH	1	1.6
19.	JR	NZ,CONT–$	;IF NOT ZERO CONTINUE MATCHING	2	4.8
20.	LD	HL,(SVBP)	;MATCH RETURN HL=SVBP	3	6.4
21. DONE:					

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

SISTEMAS INTEGRADOS EN INGENIERIA

M en C Angel Kuri Morales

Julio, 1981

SISTEMAS INTEGRADOS EN INGENIERIA

Frecuentemente, el diseñador de un sistema debe de enfrentarse a la alternativa de diseñar desde el nivel básico de "chips" o a base de sistemas integrados que cumplan una función específica. Esta última alternativa se presenta con mayor facilidad en el área de los microprocesadores porque el "software" permite adaptar con facilidad un grupo de periféricos a las necesidades de un problema específico. Los problemas de control industrial; manejo de dispositivos electro-mecánicos, acústicos y opteros; comunicación de datos y otros afines, no son más que el condicionamiento y transformación de señales. Si ésto se puede lograr en forma digital (como casi siempre es posible hacerlo) un microprocesador es la herramienta más adecuada. En este sentido hay que considerar los siguientes puntos:

1) Velocidad de muestreo.
2) Complejidad del proceso.

En base a ésto, es posible decidir si es factible captar y procesar la información a un paso suficiente. Para ello hay que considerar el tipo de convertidos y el microprocesador a utilizar. Hoy día, el convertidor más veloz de señales analógico a digitales utiliza cerca de 50 nanosegundos para obtener 8 bits. Esto nos permite muestrear a 20 MHZ. Sin embargo, un proceso de operación sobre este tipo de muestras requeriría de un procesador con un ciclo de máquina de 50 picosegundos (si suponemos mil operaciones por muestra). Sólo los procesadores más sofisticados (y ningún micro) son capaces de alcanzar tales velocidades. El estado del arte nos permite manejar anchos de banda reales de hasta 20 KHZ. (Ver 2920).

La mayor parte de las aplicaciones (o al menos una parte importante del espectro de aplicaciones) no requieren de un ancho de banda tan amplio. El procesador Z80, con 4MHZ en su ciclo básico - abarca un ancho de banda que alcanza hasta 40 mil muestras/seg: (50 operaciones/muestra) para un ancho de banda efectivo de 20 KHZ. Si consideramos 200 operaciones/muestra, el ancho de banda - baja hasta 5KHZ.

Una vez establecido el tipo de proceso, es necesario seleccionar el tipo de sub-sistemas a usar y la forma de programar el procesador en cuestión. En el texto se mencionarán dispositivos de conversión que permiten obtener hasta 180,000 muestras/seg. A máxima velocidad, el Z80 puede ejecutar cerca de 10 instrucciones/muestra.

ESTANDARES EN MICROSISTEMAS

En la actualidad, cada fábricante de microcomputadoras utiliza - sus propios canales de comunicación interna (o buses). Por razones históricas, sin embargo, dos de ellos sobresalen por su amplica definición y utilización:

1) Multibus
2) S-100

El primero es usado por INTEL, NATIONAL y algunos otros fabricantes. El canal S-100, sin embargo, ha sido adoptado por una gran cantidad de fabricantes y es en él en el que nos enfocaremos. En las figuras 1 a 4 se muestran las características del canal S-100 Para nosotros, el interés de este estándar radica en el hecho de que existen al menos las siguientes tarjetas compatibles con dicho canal:

a) Sistema de procesador central.

b) Sistema de memoria (4K) RAM.

c) Sistema de memoria (16K) RAM.

d) Sistema de memoria (64K) RAM.

e) Sistema de memoria (8K) ROM.

f) Sistema programador de ROM's.

g) Sistema de puertos paralelos.

h) Sistema de puertos eléctricamente aislados (opto-acoplados y con relés).

i) Sistema de gráficas a monitor.

j) Sistema de conversión a video.

k) Sistema de interfaz a grabadora (cassette).

l) Sistema de interfaz a disco flexible.

m) Sistema de conversión A/D y D/A.

n) Sistema de puertos serie y paralelo.

o) Sistema de secuenciadores a base de interrupciones.

p) Sistema de interfaz a impresora.

q) Sistema de cómputo completo (CPU, RAM, ROM, puertos).

Estas tarjetas o sub-sistemas son producidos por, al menos, 10 fabricantes (lo que garantiza su permanencia) y fluctúan en costos entre 200 y 300 dólares aproximadamente. De esta manera, producir un dispositivo que muestree una señal analógica, esté aislado de altos voltajes y accione uno o varios motores de paso, implica seleccionar los módulos adecuados (en este caso a, b, h, m y o; o bien q, h, m y o), fabricar la fuente de poder correspondiente y programar el sistema.

Ante la imposibilidad de examinar cada subsistema, hemos seleccionado un ejemplo de diseño que utiliza las siguientes tarjetas:

1) CPU (a).

2) RAM (c).

3) Interfaz a disco (l).

4) Interaz serial/paralelo (m).

5) Secuenciados (o).

6) Interfaz a video (j).

El sistema diseñado trabaja, simultáneamente, con una terminal interactiva, un par de discos flexibles, una impresora y 4 monitores de video.

El enlace entre los monitores y el microprocesador se logró a través de otro tipo de interfaz: el RS-232. Esta interfaz es muy importante en el área de los microprocesadores ya que es virtualmente universal. La asignación y funciones de los 'pins' de este tipo de interfaz se muestran en las figuras 5 a 8.

LA INTEGRACION DE UN DISEÑO

Si tomamos en cuenta que existen tarjetas que se ajustan a estándares internacionales; que existen suficientes de ellas para facilitar el trabajo de desarrollo y que es factible producir solamente, entonces, algunos módulos especializados, veremos la conveniencia de aprovechar al máximo el diseño modular.

El diseño modular es característico del diseño usando microprocesadores y se puede resumir en los siguientes pasos:

1) Identificación del problema.

2) Selección de los módulos (o sub-sistemas adecuados).

3) Diseño de alguno (s) que no existan en el mercado (normalmente a nivel de alambrado o 'wire-wrap').

4) Selección del lenguaje de programación.

5) Programación del sistema.

6) Pruebas en un simulador/emulador.

7) Modificaciones al sistema (Pasa al punto 6).

8) Implantación física.

9) Pruebas y ajustes finales.

Como se mencionó anteriormente, la mejor forma de ilustrar estas etapas es usando un ejemplo de diseño. Este ejemplo utilizó módulos estandar, interfaces y adaptaciones y será descrito en función de los 9 puntos señalados.

1. Identificación del problema.- Se requiere de un sistema que permita captar información alfanumérica (nombres y números). Esta información corresponde a personas físicas a las que se asigna un número. Cada número se clasifica en alguno de cuatro grupos*. Los números deben desplegarse en monitores alejados del punto de captura y deben de actualizarse al mismo tiempo que se capturan. Algunos de ellos deben ser eliminados selectivamente y deberán - desaparecer de los monitores.

2. Selección de los módulos.- Se seleccionaron los siguientes - módulos para el diseño:

a) Módulo procesador Z80.

b) Dos módulos de memoria de 16K cada uno.

c) Un módulo de interfaz a disco.

d) Un módulo de interfaz a consola.

e) Dos módulos para generación de los ciclos de reloj e interrup ción, con dos puertos serie cada uno.

f) Cuatro módulos de conversión de RS232 a RS170 (video compuesto).

3. Diseño adicional.- Se utilizaron monitores comerciales que fueron modificados para enlazarse con RS170.

4. Se seleccionó el lenguaje ensamblador ya que era necesario tener acceso a los dispositivos del sistema selectivamente (lo que elimina casi todos los compiladores) y debería ser altamente eficiente (por lo que es preferible a un intérprete).

5. Programación del sistema.- El sistema fue programado en su parte de captación (indicada por un * y en su parte electrónica).

6. Pruebas en un simulador.- Las pruebas se efectuaron exhaustivamente, pero sin probar aún el proceso de interrupciones.

7. Modificaciones al sistema.

8. Implantación física.- Se enlazó el sistema con los monitores vía cable coaxial.

9. Pruebas y ajustes.- Se detectaron algunos problemas de sincronía y acoplamiento que fueron corregidos.

Sistema de Despliegue en Tiempo Real Utilizando un Microprocesador.

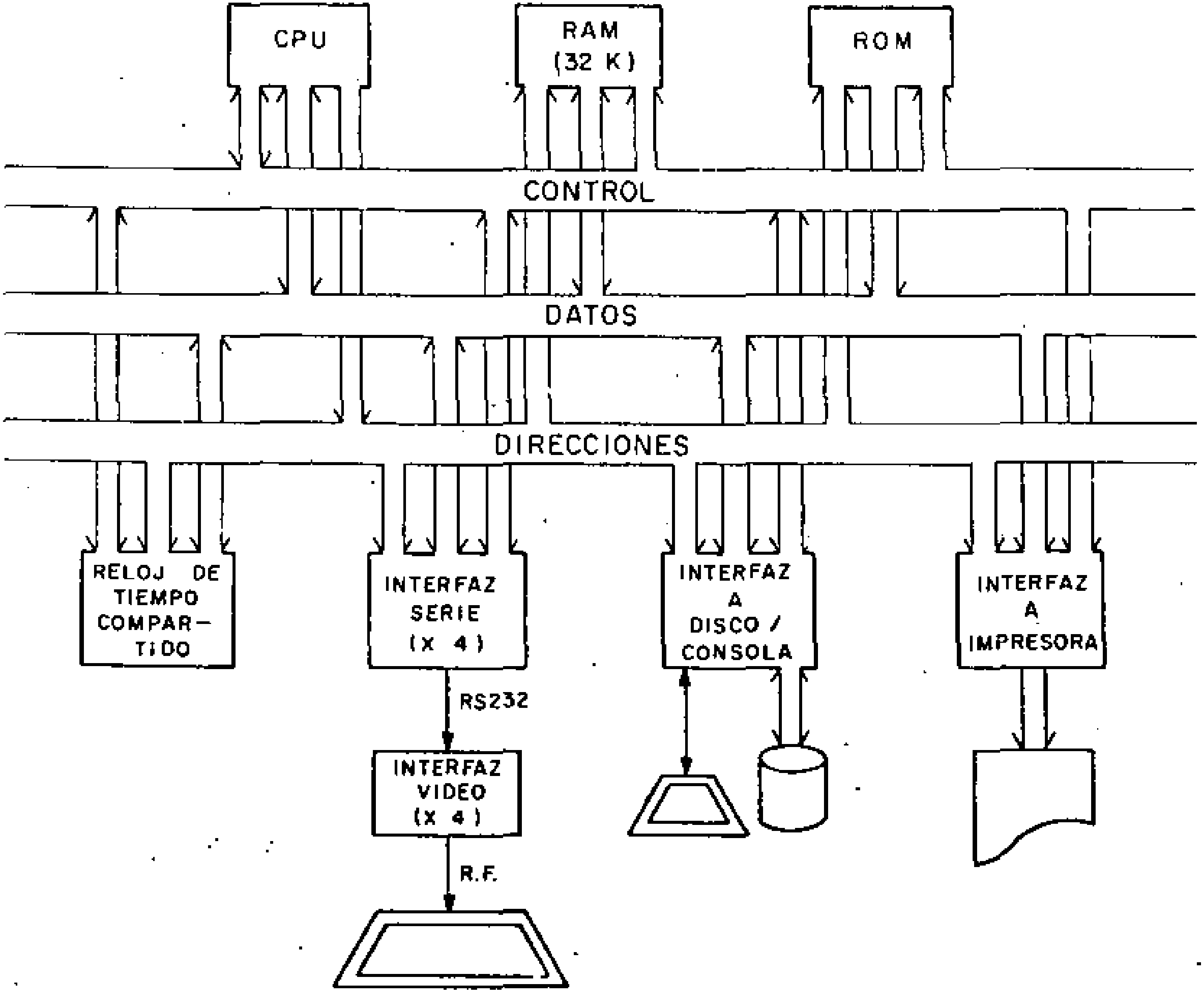
Introducción.

La misión básica de cualquier computadora -micro, mini o maxi- consiste en manejar información, bien sea en forma numérica, alfabética o alguna otra (imágenes, sonidos, etc.). La computadora solo puede manejar está información utilizando transductores, de modo que los dígitos, sonidos, imágenes, etc. puedan representarse digitalmente y en forma codificada. Por este simple proceso de codificación (y su subsecuente decodificación) es posible analizar, reproducir, amplificar, distribuir, etc., etc. alguna de las informaciones ya mencionadas. En el caso que nos ocupa, la información a manejar es de índole numérica (números de documentos) y se desea convertirla en información visual (desplegándola en monitores). Esta información debe ser manejada, adicionalmente, de suerte que pueda ser clasificada, recuperada y exhibida (o desplegada) en medios impresos y visuales en tiempo real. Es decir, los datos son captados, almacenados, procesados, impresos, clasificados y desplegados simultáneamente. Para ello se utilizan técnicas de interrupción en 'round-robin'. Además, es preciso que la respuesta del sistema no se degrade en ningún momento. Esto se logró usando un sistema de 'stacks' múltiples y procesamiento en paralelo, como se describe a continuación.

1. Definición del Problema.

En una dependencia oficial, se manejan documentos que son identificados con un número de secuencia. Estos documentos se deben captar y los números correspondientes deben ser desplegados en monitores de forma tal que el público pueda, fácilmente, localizar el número del documento. Para ello, los documentos (y sus números correspondientes) se distribuyen en 4 grupos y se asigna un monitor a cada grupo. Adicionalmente en cada monitor, los números deben aparecer en orden creciente. Cada monitor debe poder desplegar hasta 500 números. Los números deberán ser lo suficientemente grandes para ser de fácil observación. La captación de los documentos debe poder hacerse simultáneamente con el despliegue en los monitores, y la impresión de ciertos resultados parciales (así como la captación) deben tener escaza o nula influencia en el tiempo de respuesta del sistema.

La captación de los documentos deberá estar sujeta a normas de validación y a restricciones de seguridad, de manera que el sistema no se dañe por fallas de energía u otras eventuales. Finalmente, la computadora deberá encontrarse en un lugar físicamente alejado de la sala de despliegue. No se desea, sin embargo, agregar sistemas de transmisión de datos (tales como módems) para mantener el costo del sistema en niveles bajos.


CPU
RAM (32 K)
ROM
CONTROL
DATOS
DIRECCIONES
RELOJ DE TIEMPO COMPARTIDO
INTERFAZ SERIE (X 4)
RS232
INTERFAZ VIDEO (X 4)
R.F.
INTERFAZ A DISCO / CONSOLA
INTERFAZ A IMPRESORA

2. Solución Propuesta.

Para cubrir las necesidades anteriores señaladas se propuso el siguiente equipo básico:

1). Un microprocesador Z80.
2). 32 k bytes de memoria real.
3). Un sistema de diskettes de 500 k bytes.
4). Interfaz a disco.
5). Interfaz a impresora.
6). Interfaz a consola.
7). 4 interfaces para puertos serie.
8). Reloj en tiempo real.
9). 4 interfaces RS-232 a R.F.
10). Una consola.
11). Una impresora.
12). 4 monitores.

Para este equipo, por compuesto, fue necesario desarrollar el 'software' consecuente, que se divide en 4 sistemas básicos.

1). Sistema de manejo de discos y periféricos estándar.
2). Sistema de tiempo compartido.
3). Sistema de captación.
4). Sistema de reportes.

Con excepción de los monitores, todo el sistema se encuentra contenido en un chasis central, con bus S-100 y tarjetas con reguladores independientes. La fuente general del sistema provee de voltajes no regulados. El enlace entre el procesador central y los monitores se efectúa a través de cable coaxial de 50 ohms. A continuación se hace una breve descripción tanto del 'hardware' como del 'software'. Nos detendremos, sin embargo, en los puntos que, creemos son de mayor interés.

3. Hardware.

a). Procesador. El CPU trabaja a 4 MHZ y es, básicamente, una versión mejorada del 8080 de INTEL. El autor ha tenido oportunidad de comparar ambos sistemas y estima que el Z80 del orden de 50% más eficiente que el 8080.

b). Memoria La memoria es dinámica y se encuentra contenida en 2 tarjetas de 16 k bytes cada una.

c). Interfaz a diskette. Esta interfaz, a diferencia de otros equipos, no utiliza DMA para hacer transferencias a y de disco. Esto degrada al sistema un poco en relación con otros equivalentes que sí usan DMA. Sin embargo, la degradación es muy pequeña debido a que el CPU poseee instrucciones de máquina que permiten también hasta 1.6 megabits/seg. (200 k bytes/seg.). Puesto que la velocidad de transferencia de un diskette es de 250 k bytes/seg., la degradación es de un 20%, relativa a un sistema que pudiese transferir, en DMA un ciclo de.4 MHZ.

d). Interfaz a Consola e Impresora. Las interfaces a consola y a impresora son, simplemente, dos puertos serie y paralelo, respectivamente. (Con circuitería adicional para lograr los niveles estándar RS-232).

e). Reloj en Tiempo Real. El circuito de reloj en tiempo real es un dispositivo programable, con un ciclo máximo de tiempo de 16.32 milisegundos. Este reloj, además, tiene la característica de poder causar una interrupción con la cuenta terminal.

Como es conocido para el Z80, la parte más significativa de la localidad de interrupción está dada por el contenido del registro del procesador denominado "Registro I". La parte menos significativa es definida por el dispositivo que causa la interrupción (en este caso el reloj). En esta aplicación,

$$I = 12_H \quad (0001 \quad 0010_B)$$

y la parte dada por el reloj corresponde a

$$R = 60_H \quad (0110 \quad 0000_B)$$

de forma que la interrupción nos lleva a la localidad

$$<I> <R> = 1260_H$$

En esta localidad se encuentra un apuntador a la rutina de interrupción.

El proceso de concatenación y llamada a la localidad de servicio de la interrupción son dados por el 'hardware' del procesador, automáticamente.

f). Interfaz RS-232 a R.F. En esta aplicación, los monitores deben ser lo suficientemente grandes como para facilitar la observación por parte del usuario. Se especificaron, pues, pantallas de más de 19 pulgadas. Como se sabe, las terminales de video estándar no alcanza estas dimensiones. Debido a ello, fue necesario adaptar un monitor comercial y proveerle de la lógica necesaria para que éste actuara como terminal de despliegue. Esto implica que, al sistema del tubo de rayos catódicos se le anexen:

1). Un generador de caracteres.

2). Memoria suficiente para almacenar la página de despliegue.

3). Un generador de R.F. tal que la información almacenada se concentre en una señal de video susceptible de ser "interpretada" por el monitor.

4). Lógica de control.

El circuito que se seleccionó, hace uso de un microprocesador 3870 de Motorola para lograr todas las funciones previas al punto (3) antes mencionado. Esto permite tener, en un circuito, toda la electrónica necesaria para, de hecho, implementar una terminal de video. Por otra parte, el circuito es compatible con el bus estándar del computador (S-100), lo cual permite conservar la modularidad del sistema.

La tarjeta, pues, recibe información serial RS-232 a 300 bauds y convierte esta inforamción en caracteres que almacena en una página de memoria, y los cuales envían en una señal de video compuesto al monitor de despliegue.

Debe hacerse notar que la información que se despliega es diferente para cada uno de los monitores y que transmite a 3C0 bauds impone serias restricciones al sistema. (Para escribir 4 líneas se pierden cerca de 8 segundos) sin embargo, por razones de tiempo de implantación (el sistema, en su totalidad se implementó en 60 días) se tuvo que trabajar con estos sistemas lentos. La solución que se le dió al problema de velocidad se discute más adelante.

Un diagrama a bloques de la interfaz a video se muestra en la figura 1.

4. Software.

a). Sistema de Manejo de Discos y Periféricos. El sistema operativo se utiliza para manejar archivos tanto en discos como aquellos que corresponden a las terminales (de hecho, cualquier dispositivo de E/S, sea disco, impresora o algún otro puede ser tratado como un archivo. Así se maneja en lenguajes como COBOL, por ejemplo). Se usan las siguientes rutinas.

1). Crea archivo.

2). Selecciona disco.

3). Cambia de nombre a archivo.

4). Lee de consola.

5). Escribe en consola.

6). Escribe en impresora.

7). Abre un archivo.

8). Cierra un archivo.

9). Lee de un archivo.

10). Escribe en un archivo.

11). Borra un archivo.

El enlace al sistema operativo sigue las convenciones del sistema CP/M, que se desarrolló para el 8080 y que se ha convertido en un estándar. (Digital Research).

b). Sistema de Tiempo Compartido. Este sistema funciona generando interrupciones que se reconocen aproximadamente cada 2 segundos. La información correspondiente a los números que se despliegan en pantalla está divida en 4 áreas y es examinada cíclicamente con cada interrupción. Con cada ciclo de 'round-robin', se escribe una línea (64 caracteres) si se cumple alguna de las siguientes condiciones:

1). Hay información nueva (algún número fue dado de alta o de baja).

2). La cantidad de números en memoria excede la capacidad de despliegue del monitor, (cada monitor despliega 16 líneas de 64 caracteres cada una).

Los números se presentan en orden numérico ascendente y se actualizan con cada alta o baja. Es decir, cuando se da de alta un documento, el número se inserta en el lugar correspondiente y recorre a todos los subsecuentes un lugar a la derecha. En el caso de una baja, el número correspondiente se elimina y todos los números subsecuentes se recorren un lugar a la izquierda. La información se "sortea" durante la captación, de manera que este módulo solamente despliega información ya actualizada.

Como se mencionó anteriormente, la velocidad de transmisión es de 20 caracteres/seg. Esto implica que se requieren de un poco más de dos segundos para desplegar 10 números, como se ilustra en la figura 2. Si suponemos que los cuatro monitores requieren de servicio, veremos que el sistema de captación sólo obtendrá el control de procesador marginalmente, ya que las interrupciones se suceden cada 2 segundos. Esto ocasiona que el tiempo de respueta se hiciera intolerantemente grande. Dado que los monitoes pueden desplegar (en 'scroll') hasta 500 números, no es difícil suponer que al menos 2 de ellos están en actividad constante y el tiempo de respuesta sea, en promedio inaceptable. Por ello, se adoptó el siguiente método.

c). Sistema de Múltiples Stacks. Como es conocido, cuando se acepta una interrupción, es indispensable guardar el 'status' del sistema previo a la interrupción. Esto implica, normalmente, el uso del 'stack' para este almacenamiento. En operación estándar los pasos que se siguen para el despliegue son los siguientes:

1). Se acepta la interrupción.

2). Se guarda el 'status' en el 'stack'.

3). Se toma una línea de 64 caracteres de memoria.

4). Si se cumple alguna de las condiciones antes señaladas se "escribe" la línea en el momento en turno.

5). Se recupera el 'status' del 'stack'.

6). Se prosigue en donde había entrado la interrupción.

Eventualmente, el punto 4 se lleva, en ejecución, 64 x 1/30 segundos. (Los 2 segundos que se habían mencionado).

Para evitar una pérdida de tiempo tan marcada, se toman las siguientes medidas:

1). Se ejecutan los puntos 1 al 4. Sin embargo, cuando se escribe un carácter (el n-ésimo de 64) se guarda este 'status' en un nuevo 'stack'.

2). Se habilitan interrupciones por una condición del UART que indica "carácter transmitido".

3). Se regresa al 'stack' original.

4). Se reinstaura el 'status' previo a la interrupción.

Cuando (después de 33 300 μ segundos) el UART interrumpe por carácter transmitido (TBE), se inicia el sistema a partir del punto 1. Después de 64 interrupciones, se regresa al modo de reloj (en donde el timer interrumpe). El diagrama de flujo se muestra en la figura 3.

Con este sistema, se dispone de la diferencia entre el tiempo de transmisión y el de proceso de E/S para proceso del programa principal. Considerando cerca de 3 µs. por instrucción, en 33 300 µ segs. podemos ejecutar cerca de 11 000 instrucciones. (Como referencia, todo el programa es de cerca de 1 500 instrucciones de AZM80).

El diagrama del proceso se muestra en la figura 3.

d). Sistema de Captación. La captura de datos se logra en paralelo con el sistema de despliegue en tiempo real como ya se señaló. Es esta la parte en la que un número se da de alta o de baja. Además el módulo de captación incluye una serie de opciones adicionales que, en conjunto con altas y bajas son ocho, a saber:

1). L (lave. Actualización del pase al sistema.

2). M (ensaje. Despliegue de textos en los monitores.

3). D (espliegue. Vaciado de los contenidos de archivo.

4). G (uarda. Copia la imagen en despliegue de disco.

5). T (rae. Copia la imagen de disco a memoria.

6). A (lta. Toma un documento y lo almacena en un archivo histórico y tiene, además, 4 opciones:

i). Despliega el nûmero en el momento indicado.

ii). Guarda el documento en disco, sin despliegue.

iii). Toma un documento de disco y despliega su nûmero en el momento correspondiente.

io). Toma todos los documentos de disco y despliega sus nûmeros en el momento correspondiente.

7). B (aja. Cancela un nûmero y lo incluye en un archivo histórico, con las siguientes opciones:

i). Elimina el nûmero del monitor correspondiente.

ii). Elimina el nûmero de disco.

8). F (in. Cierra archivos, deshabilita reloj e interrupciones y entrega control al sistema operativo.

La opción se le indica al sistema tecleando, simplemente, la primera letra. A continuación haremos una brevísima descripción de alguna de las opciones.

- La opción Llave permite al usuario modificar la clave de entrada al sistema. Ningún usuario puede hacer uso del sistema si no proporciona esta clave autorizada.
- La opción 2 permite enviar textos a los monitores

deteniendo, momentáneamente, el despliegue de los números en los monitores.

- La opción 3 permite verificar el contenido del archivo creado por la opción ii de la función de altas.
- La opción 4 tiene por objeto poder eliminar una imagen del contenido de cada uno de los monitores. Esto es necesario cuando se desee interrumpir temporalmente el despliegue sin tener que re-cargar todos los documentos.
- Como complemento a la opción anterior, la opción 5 permite restaurar la condición de la memoria para re-iniciar el despliegue.
- Las opciones 6, 7 y 8 se explican en su descripción. Hay, sólo, que hacer notar que toda alta y toda baja quedan registradas en archivos que se manejarán posterioremente.

e). Sistema de Reportes. A partir de las altas y bajas es posible recopilar la siguiente información:

1). Documentos que están en disco pero que aún no se han dado de alta.

2). Documentos que están en despliegue.

3). Documentos que han causado baja porque han sido procesados.

4). Documentos que han causado baja por cancelación.

5). Estadísticas de aquellos documentos que están pen dientes de proceso, es decir, aquellos que están en despliegue.

5. Conclusiones.

El sistema antes bosquejado, fue desarrollado teniendo en mente las siguientes metas básicas:

1). Bajo costo.
2). Corto tiempo de implementación.
3). Confiabilidad.
4). Alta eficiencia.

Para lograr alcanzar los cuatro puntos, simultáneamente, fue necesario conjuntar equipo y programas que estuvieran fuertemente ligados. Por esta razón, todo el sistema fue escrito en código de máquina utilizando el ensamblador residente. Unicamente se escribió un segmento (el correspondiente al punto 5 del sistema de reportes) en un lenguaje de alto nivel (COBOL). Como resultado se lograron eficiencia y economía en el código. Todo el 'software' (incluyendo tablas y textos) exceptuando la parte de reportes, requiere de 5 k bytes. Esta última (que funciona "off-line") requiere de 12 k bytes de los cuales las primeras 4 opciones ocupan 3 k y la última (escrita en COBOL) requiere de 9 k.

De la experiencia adquirida se puede constatar algo tan trillado ya en la industria electrónica actual: el uso de mi croprocesadores y de circuitos asociados acelera y abarata enormemente los desarrollos híbridos (en los cuáles 'software' y - 'hardware' son ambos importantes). Hay que enfatizar, sobre todo, que la selección de equipo que se ajuste a estándares de amplia aceptación es un punto clave para la viabilidad de sistemas como el que aquí se reporta. En nuestro caso esos estándares empiezan con el procesador (el compilador COBOL genera código 8080 que es un subconjunto del Z80), pasando por el canal del computador (el BUS S-100 permitió conseguir el reloj en tiempo real e interfaces sin necesidad de desarrollarlas) y terminando con el 'software' (el compilador COBOL es ANSI, así como - otros lenguajes; por otra parte usar CP/M permitió usar este - compilador en particular). Es relativamente fácil apreciar que el tiempo de desarrollo en otras circunstancias hubiera multiplicado el tiempo y sobre todo el esfuerzo y costo del sistema. Es por experiencias como esta que se puede explicar la reticencia justificada a adoptar máquinas nuevas que son incompatibles con otras.

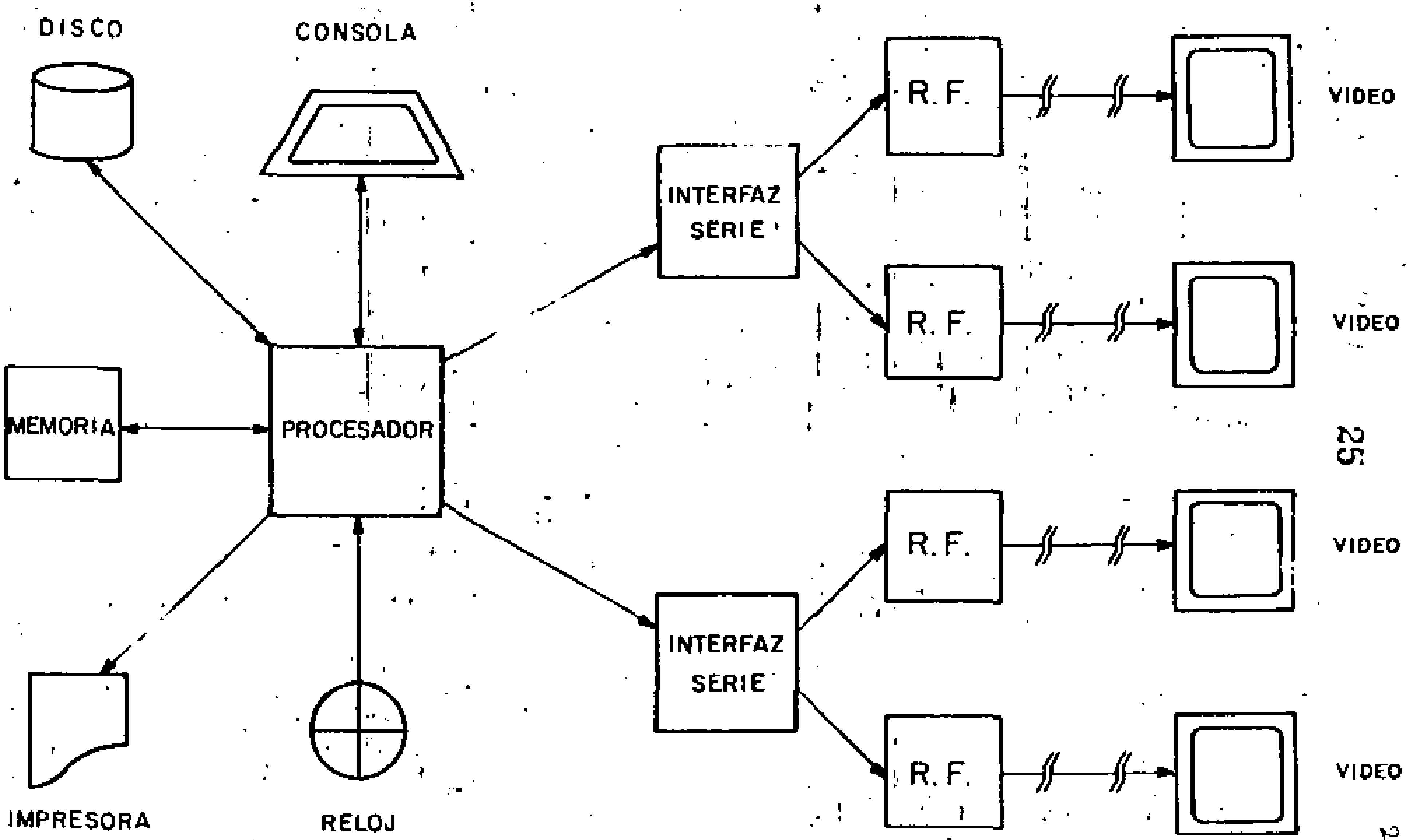
Figure 3 Summary Of Register Formats For TU-ART, Each Device

OFFSET	FUNCTION	D7	D6	D5	D4	D3	D2	D1	DØ	REF. PAGE
0	IN STATUS	TBE	RDA	IPG	SBD	FBD	SRV	ORE	FME	6
0	OUT STATUS	STOP	96ØØ	48ØØ	24ØØ	12ØØ	3ØØ	15Ø	11Ø	7
1	IN SERIAL	MSB	Direction of shift					——→	LSB	7
1	OUT SERIAL	MSB	Direction of shift					——→	LSB	7
2	OUT COMMAND	. . .	. . .	TB5	HBD	INE	RS7	BRK	RES	7-3
3	IN INT ADDR	1	1	14*	12*	1Ø*	1	1	1	8
3	OUT INT MASK	T5/PI7	T4	TBE	RDA	T3	SENS	T2	T1	8
4	IN PARALLEL	MSB							LSB	8
4	OUT PARALLEL	MSB							LSB	8
5-9	OUT Timer 1-5	MSB	(Delay=count x 64 μsec, HBD=Ø) (Delay=count x 8 μsec, HBD=1)							9

14	12	10	Source of Interrupt
0	0	0	Timer 1
0	0	1	Timer 2
0	1	0	SENS
0	1	1	Timer 3
1	0	0	RDA
1	0	1	TBE
1	1	0	Timer 4
1	1	1	Timer 5/PI7

Table 1 Z80 (Mode 2) Response

Priority	TU-ART's (Hex) Z-80 INTA Response								Source of Interrupt
	D7	D6	D5	D4	D3	D2	D1	DØ	
15 (Highest)	Set By Device A Adr. A7	Set By, Device A Adr. A6	Set By Device A Adr. A5	Ø	Ø	Ø	Ø	Ø	Device A, Timer 1
14				Ø	Ø	Ø	1	Ø	Device A, Timer 2
13				Ø	Ø	1	Ø	Ø	Device A, SENSA
12				Ø	Ø	1	1	Ø	Device A, Timer 3
11				Ø	1	Ø	Ø	Ø	Device A, RDA
10				Ø	1	Ø	1	Ø	Device A, TBE
9				Ø	1	1	Ø	Ø	Device A, Timer 4
8				Ø	1	1	1	Ø	Device A, Timer 5 (PI7)
7				1	Ø	Ø	Ø	Ø	Device B, Timer 1
6				1	Ø	Ø	1	Ø	Device B, Timer 2
5				1	Ø	1	Ø	Ø	Device B, SENSB
4				1	Ø	1	1	Ø	Device B, Timer 3
3				1	1	Ø	Ø	Ø	Device B, RDA
2				1	1	Ø	1	Ø	Device B, TBE
1				1	1	1	Ø	Ø	Device B, Timer 4
Ø (Lowest)				1	1	1	1	Ø	Device B, Timer 5 (PI7)


DISCO
CONSOLA
MEMORIA
PROCESADOR
IMPRESORA
RELOJ
INTERFAZ SERIE
INTERFAZ SERIE
R.F.
R.F.
R.F.
R.F.
VIDEO
VIDEO
VIDEO
VIDEO

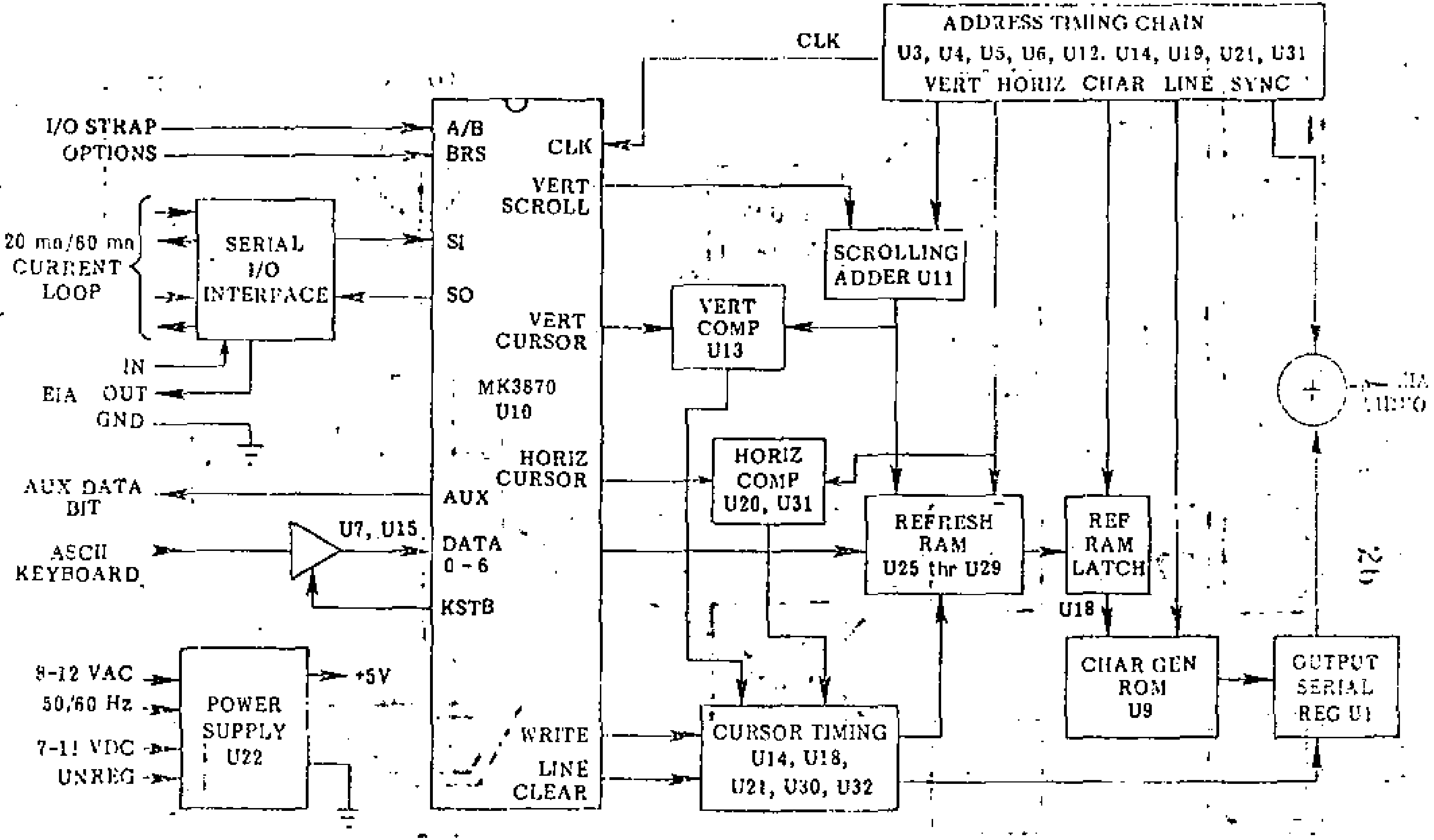
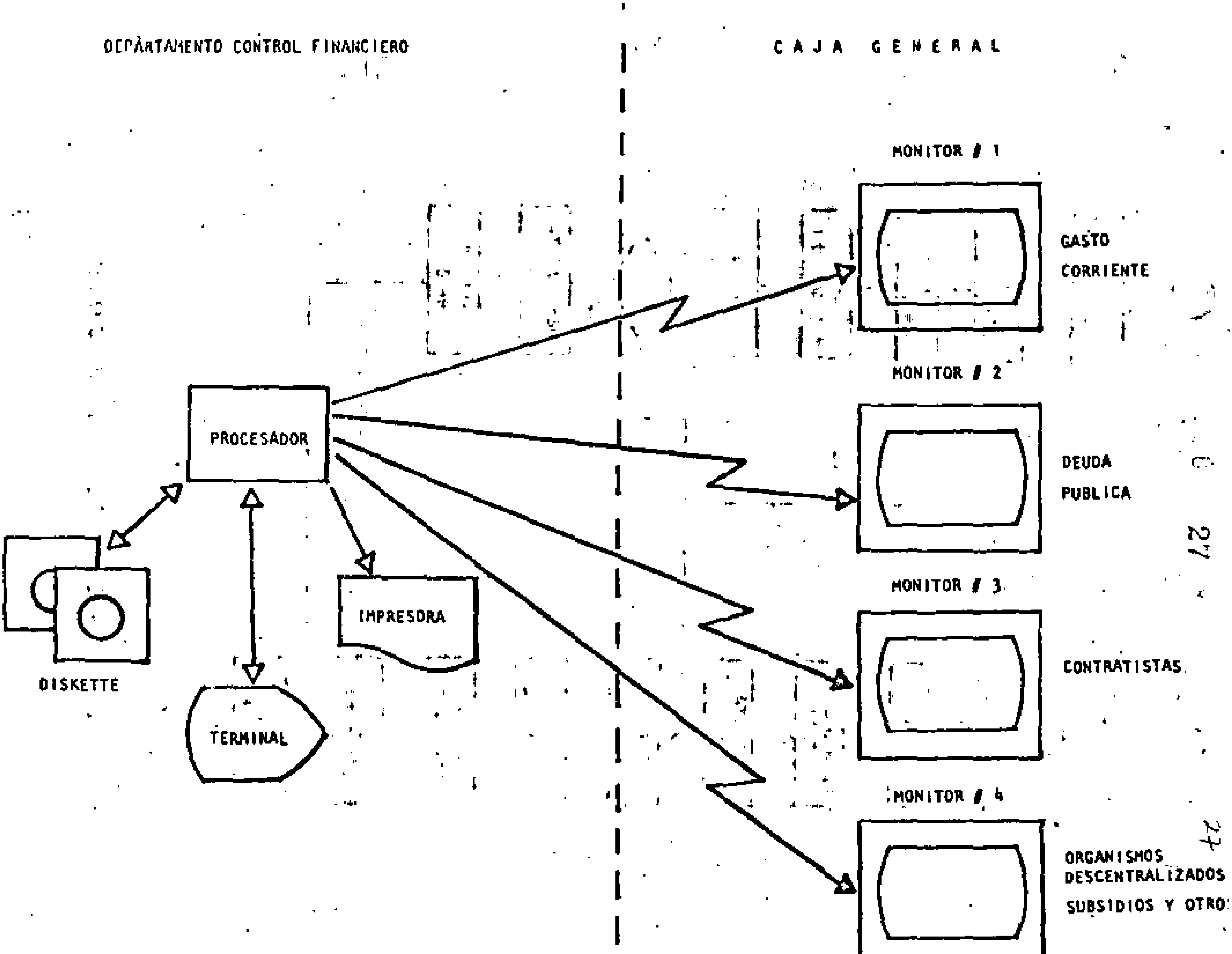


DIAGRAMA A BLOQUES DE LA INTERFAZ DE VIDEO

FIGURA 1.


DEPARTAMENTO CONTROL FINANCIERO
CAJA GENERAL
MONITOR # 1
GASTO CORRIENTE
MONITOR # 2
DEUDA PUBLICA
MONITOR # 3
CONTRATISTAS
MONITOR # 4
ORGANISMOS DESCENTRALIZADOS
SUBSIDIOS Y OTRO
PROCESADOR
DISKETTE
TERMINAL
IMPRESORA

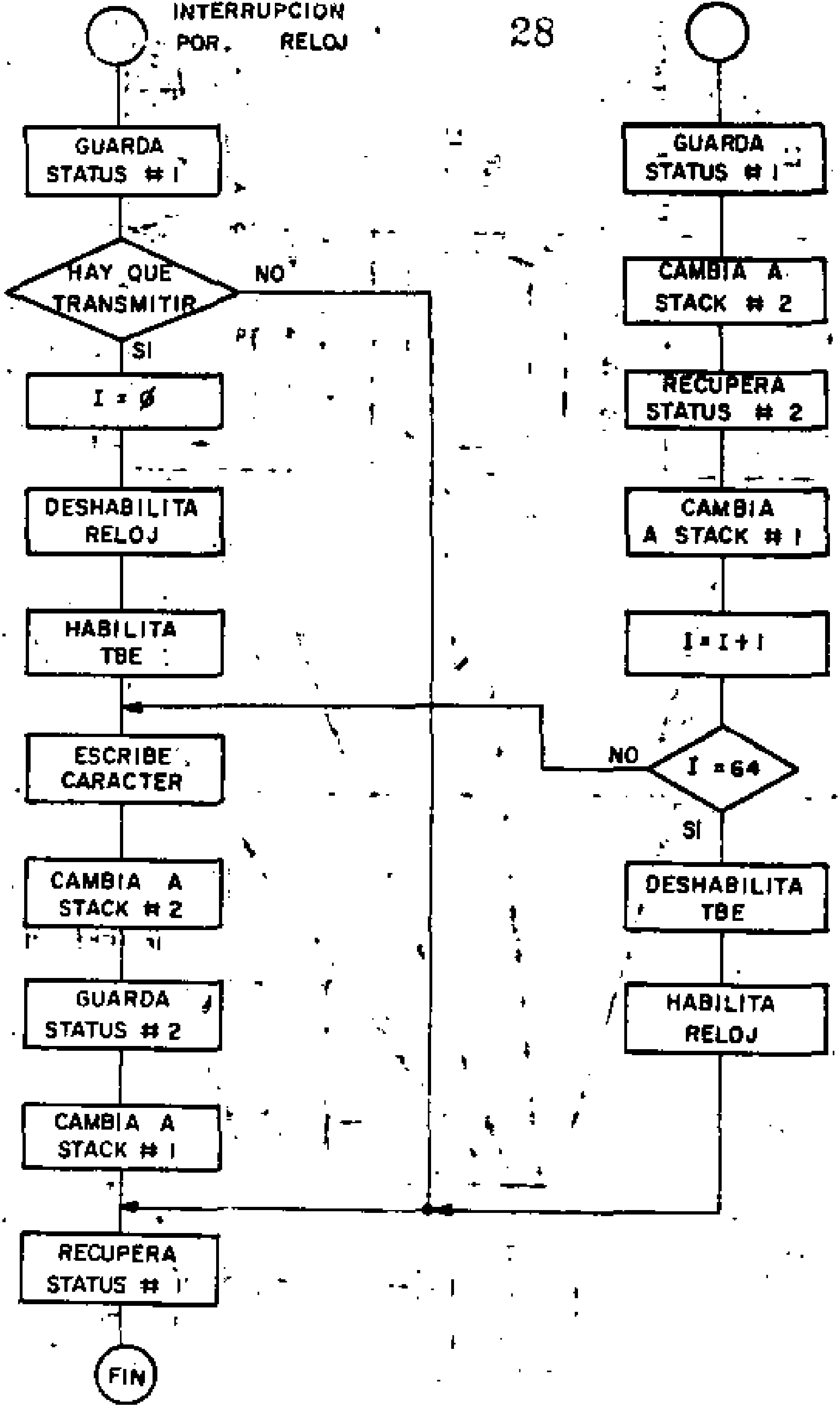
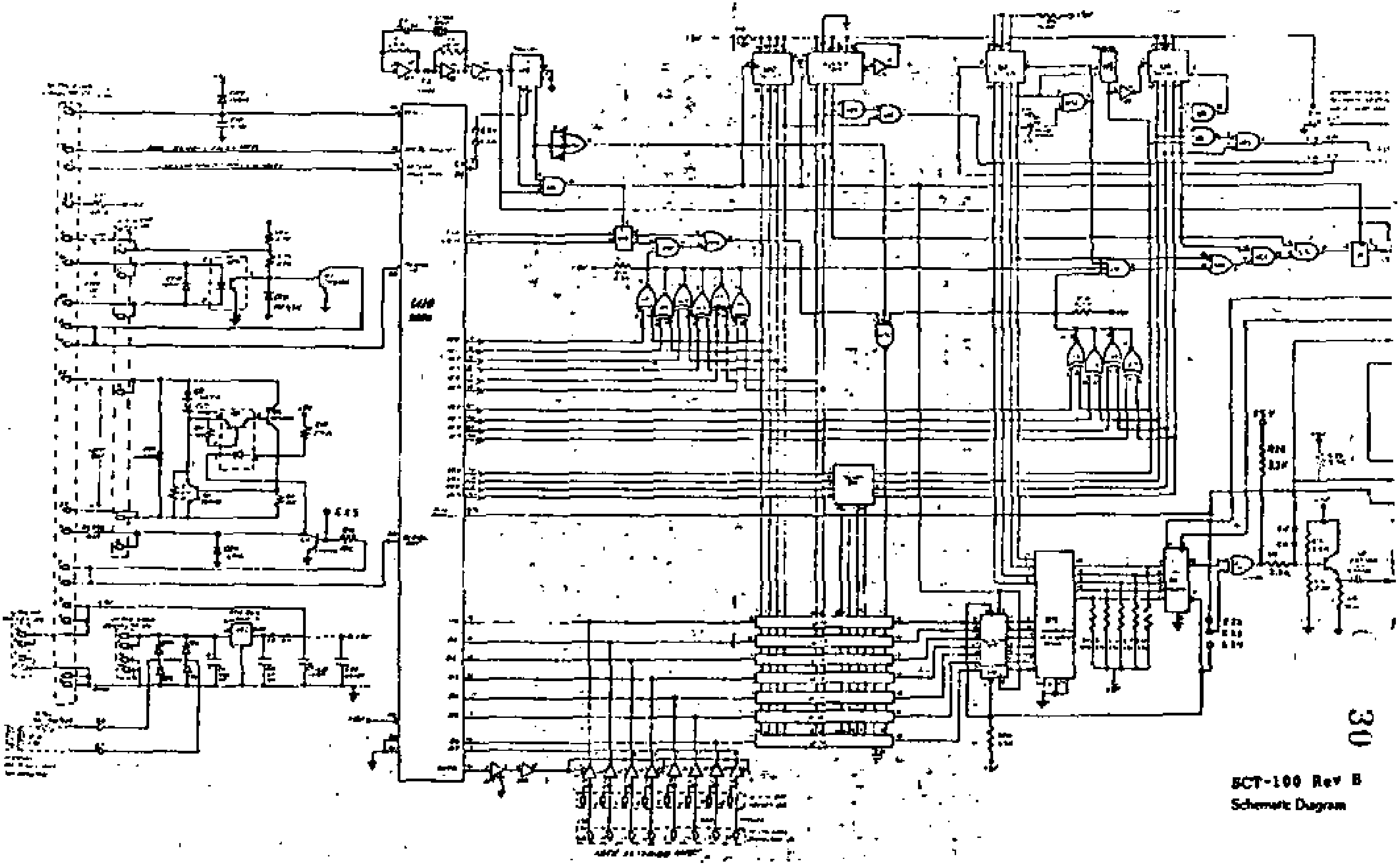


FIG. 3 SISTEMA DE INTERRUPCIONES.

```
00000 00001 00002 00003 00004   00006 00007 00008 00011 00012
00013 00014 00015 00016 00101   00292 01010 01091 01466 01928
01991 02020 03030 04040 05050   06060 07070 08080 08808 09090
12287 12300 12345 12400 12500   12635 14141 15151 16161 17171
17687 18181 18827 19191 19829   20001 20002 20003 20004 20005
20202 21212 22222 23009 23232   23456 24242 25252 26262 26627
27272 27872 28282 29292 30000   30001 30002 30004 30007 30010
30100 30303 31313 32000 32323   34343 35353 36363 36737 37200
37373 38383 39393 40001 40002   40003 40006 40020 40022 40404
44444 45555 46477 48484 49494   50022 50505 51515 52525 53124
53535 54545 56565 57575 58585   59595 60606 61616 61772 62626
62727 62730 63636 63663 63666   64642 64646 64666 64730 64747
64774 65656 65666 65757 65777   66366 66777 67676 68686 69696
70704 70707 71772 71818 71871   72651 72672 72777 73883 74010
74621 74663 74664 74701 74710   74740 74742 74747 74773 74774
74840 74883 74884 75757 75775   75785 75880 75885 76232 76767
76818 76876 77377 78309 78738   78787 79797 81881 81889 82828
82911 82999 83883 83993 84884   84994 85775 85810 85858 85995
87282 87464 87889 88388 89201   90021 90165 90829 91029 92092
92998 93003 98092 98273 98393   98567 98757 98822 98889 99999
```



SCT-100 Rev B
Schematic Diagram

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

BASIC DE CONTROL

Julio, 1981

CHAPTER 3

Elements of the Control Basic Language

3.1 Numbers and Constants

In Control Basic all numbers are integers and must be less than 32767. Numbers are stored internally as 16-bit two's complement (low byte-high byte). For programming and input/output, numbers are expressed in decimal, hexadecimal, or the ASCII character code. Hexadecimal numbers are denoted by the character "%" followed by up to 4 hexadecimal digits. For example, %21 and 33 are both stored internally as ØØ1ØØØØ1 ØØØØØØØØ; %FFF4 and -12 are both stored internally as 1111Ø1ØØ 11111111. An ASCII code (7-bit) is denoted by a colon (:) followed by an ASCII character. For example, :A has the value 65 (or 41H) and is stored internally either as one byte, Ø1ØØØØØ1, or as two bytes, Ø1ØØØØØ1 ØØØØØØØØ, depending on how it is used.

3.2 Variables

There are 52 variables denoted by letters A through Z and AØ through ZØ. There is also an array @(I). The dimension of this array (i.e., the range of the index I) is set automatically to make use of all the memory space that is allocated but left unused by the text of the current program (i.e., Ø through SIZE/2; see SIZE function below. Also see section 8.1.1 or 8.2.1 on memory allocation.). Each variable and each element of the array @(I) uses two bytes of memory stored low byte-high byte.

The same memory space that is used to store the array @(I) can also be accessed, one byte at a time, through &(J) (or as strings through $(J) where J runs from Ø through SIZE. That is, &(Ø) is the low byte of @(Ø), &(1) is the high byte of @(Ø), &(2) is the low byte of @(1), &(3) is the high byte of @(1), etc. Strings of up to 132 characters each can be stored in the same memory area and are accessed through $(J). The length N of the string $(J) is stored in &(J); the 7-bit ASCII characters of the string are stored in &(J+1) through &(J+N). Thus, $(J+N+1) can be another string.

The multiple naming of the same memory space is intentional. This not only gives the programmer the freedom to allocate available memory between arrays and strings but also is very convenient for packing and unpacking data, and for string manipulation.

3.3 Functions

Control Basic has 10 built-in functions, which are described below. The values returned by these functions will always be printed in decimal unless the user requests the answer in hex. This may be done by means of the format controls of the PRINT command, section 5.4.1. In the following x and y denote variable or constant expressions.

ABS(x) gives the absolute value of x.

RND(x) gives a random number between 1 and x (inclusive).

SIZE gives the number of bytes allocated to, but left unused by the current program in the text area. See section 8.1.1 or 8.2.1 for a discussion of memory allocation.

SGN(x) returns a 1 if x is positive (or zero) or a -1 if x is negative.

AND(x,y) gives the 16-bit Boolean AND of x and y.

OR(x,y) gives the 16-bit Boolean inclusive OR of x and y.

XOR(x,y) gives the 16-bit Boolean exclusive OR of x and y. Note that XOR(x,%FFFF) gives the one's complement of x, and -x gives the two's complement of x. (The result z of this operation must be in the range $-32767 \le z \le 32767$. Hence there is no one's complement of 7FFFH nor two's complement of 8000H.)

GET(x) gives the 8-bit contents (1 byte) of memory location x.

IN(x) gives the 1-byte value input from port x, where Ø<=x<=255.

LOC gives the absolute address of @(Ø), &(Ø), and $(Ø).

3.4 Arithmetic and Compare Operators

Control Basic allows the following operators in variable and constant expressions:

/ divide.

* multiply, or logical AND.

- subtract, or two's complement.

+ add, or logical OR.

> greater than (compare).

< less than (compare).

= equal to (compare).

not equal to (compare).

>= greater than or equal to (compare).

<= less than or equal to (compare).

The operations +, -, *, and / result in a value between -32767 and 32767. Results outside this range will cause an error, and Control Basic will print "HOW?". All compare operators result in a 1 if true and a Ø if false (not true).

In the Logical context, a Ø is False and any non-zero value is True. Thus, the add operator (+) is also used as logical OR, and the multiply operator (*) is also used as logical AND. For logical complement, one may use Ø=. For example:

```
A+B  can mean  A OR B.
A*B  can mean  A AND B.
Ø=A  can mean  NOT A.
```

4

Note that if one wants the 16-bit-by-bit Boolean AND, one should use the function AND(A,B) instead of the logical AND just described. The same applies for Boolean OR and XOR.

3.5 Expressions

Expressions are formed from numerical elements with arithmetic or compare operators between them. Numerical elements include constants (decimal, hex, or ASCII), variables, two-byte array elements @(I), one-byte array elements &(J), functions, and other expressions in parentheses. A "+" or a "-" sign can also be used as a unary operator. The hierarchy for evaluating expressions is as follows: functions are evaluated first, then parentheses (from the innermost), then multiplication and division, then addition and subtraction, then the compare operations. Cascaded * and / or cascaded + and - are evaluated from left to right. Compare operators cannot be cascaded. Several commented examples of both legal and illegal expressions follow:

1+2*3	has the value 7, not 9.
2>1+3	is false and has the value 0 (> is used as a compare operator and 2 is not greater than 1+3).
2*5/3	has the value 3, while 2/3*5 is 0 due to truncation.
A+-3	is invalid, but A>-3 is allowed.
A<X<B	is invalid; (A<X) * (X<B) is the proper way of testing whether X is between A and B.
(A>B)*X + (A=B)*Y	is equal to X if A>B, is equal to Y if A=B, and is equal to 0 if A<B.
(&(8)>=:A)*(&(8)<=:Z)*2 + (&(8)>=:0)*(&(8)<=:9)	has a value of 2 if the character &(8) is a letter, a value of 1 if it is a digit, and a value of 0 otherwise.

CHAPTER 4

Control Basic Syntax

4.1 Control Basic General Syntax

There are a number of special definitions used throughout this manual as well as some general rules of syntax which should be followed when entering and editing Control Basic programs. The definitions of important terms follow:

command -
Applies to Control Basic keywords which may be used in either direct commands or statements.

direct command -
Commands which may be typed directly from the console in response to the Control Basic prompt.

statement -
A numbered line in a Control Basic program which is composed of one or several commands.

multiple-command statement -
A numbered line in a Control Basic program which is composed of several commands (on the same line) separated by semi-colons.

function -
One of ten control structures intrinsic to Control Basic which may be used in expressions. (Note that commands may not be used in expressions.)

current program -
The program which may be entered or edited directly without first LOADing it; the program which prints on the console when "LIST" is typed.

text area -
The area of memory reserved for storage of the current program and the arrays.

6

There are also a number of rules of syntax to be followed when entering and editing Control Basic programs. These are summarized below:

1. A Control Basic programs consists of one or more numbered statements. The statement number must be an integer between 1 and 32767, inclusive.

2. Multiple-command statements (see above) must use semi-colons (;) to separate the commands. There are three exceptions to this: GOTO, STOP, and RETURN cannot be followed by a semicolon or other commands; they must be the last command on any given line.

3. When the direct command "RUN" is issued, the statement with the lowest line number is executed first, followed by the one with the next lowest line number, etc. The statements GOTO, RUN, STOP, GOSUB, and RETURN can alter this sequence, however, by causing control to branch to a specific line number (see Chapter 5).

4. Execution of the commands within a multiple-command statement is from left to right. The IF command is a special case in that if the condition tested is false, all commands to the right of that point will be skipped over and execution will continue with the next line.

5. Spaces (blanks) may be used or omitted freely but for the following exceptions: constants, and command and function keywords may <u>not</u> contain embedded blanks. They may be abbreviated, however; see section 4.2.

6. Execution of a running program or listing of any type of output may be aborted by pressing the Control-C key (usually the two keys, CTRL and C, depressed simultaneously) on the console input device.

7. Throughout this manual, portions of example programs which are underlined signify that those expressions are user-typed.

4.2 Abbreviations and Summary of Commands

Control Basic command and function keywords may be abbreviated if desired. Since the Interpreter stores programs as the actual ASCII characters which make up the statements and this requires one byte per character (see Memory Allocation sections), the abbreviated forms of commands greatly reduce their memory storage requirements. This is important if the program is to be stored in PROM for future use (see SAVE and EPROM commands). For example, the following command line:

```
10 FOR I=1 TO 1000 STEP 2 ; PRINT I , ; NEXT I
```

would perform exactly the same function if abbreviated to the form:

```
10 F.I=1TO1000S.2;P.I,;NE.I
```

This second line requires only about half the bytes for storage as the first line. However, since this second line is not very legible, it is recommended that a programmer develop a program using the full-word commands, and only translate it to the abbreviated form just prior to programming it into PROM.

The abbreviations are formed by truncating several of the trailing letters and replacing them with a period (.). For example, "P.", "PR.", "PRI.", and "PRIN." all stand for the word "PRINT". Command keywords cannot be truncated to a shorter length than the minimum abbreviations given below or Control Basic will print "WHAT?". Also note that the word "LET" in a LET statement may be omitted altogether. There are three columns in the table below: the first shows all Control Basic commands, the second shows only those which may be used in numbered statements (these may also be used as direct commands), and the third column shows the ten Control Basic functions. The shortest keyword abbreviations are:

66. 8

Direct Commands	Statements	Functions
* AUTORUN=AUTORUN	* AUTORUN=AUTORUN	A.=ABS
C.=CALL	C.=CALL	AND=AND
E.=EPROM		G.=GET
F.=FOR	F.=FOR	IN=IN
G.=GOTO	G.=GOTO	L.=LOC
GOS.=GOSUB	GOS.GOSUB	OR=OR
IF=IF	IF=IF	R.=RND
IN.=INPUT	IN.=INPUT	S.=SIZE
=LET (implied)	=LET (implied)	SGN=SGN
L.=LIST		X.=XOR
LO.=LOAD		
LOCK=LOCK		
NEW=NEW		
NE.=NEXT	NE.=NEXT	
NU.=NULL		
O.=OUT	O.=OUT	
P.=PRINT	P.=PRINT	
PUT=PUT	PUT=PUT	
Q.=QUIT		
* R.=RDOS		
R.=RETURN	R.=RETURN	
REM=REM	REM=REM	
RUN=RUN	RUN=RUN	
S.=SAVE		
S.=STEP	S.=STEP	
ST.=STOP	ST.=STOP	
TO=TO	TO=TO	
W.=WIDTH		

* These commands are found in model MCB-216 only.

As defined in section 4.1, a command is part of a statement, which, in turn, is part of a program. However, when a command is typed at the console input device without a line number preceding it, it is a direct command. Direct commands are not stored by Control Basic but are executed immediately.

The commands listed in the second column above may be used both as statements of stored programs and as direct commands. Those not listed in column two but listed in column one may be used as direct commands only. There are three groups of special cases:

1. When used as direct commands, GOSUB, INPUT, and RUN cannot be followed by semi-colons and other commands.

2. AUTORUN, NEXT, RETURN, REM, STEP, STOP, and TO are meaningless as direct commands. However, NEXT, STEP, and TO may be used within a direct FOR command all typed on one line as shown in this example:

```
        FOR I=0 TO 255 STEP 4; PRINT #%,,I,; NEXT I
```

3. When used in statements, GOTO, RETURN, and STOP must be the last command on the line (cannot be followed by semi-colons and other commands).

4.3 Memory Organization of Control Basic

The text of the current program is stored in memory one ASCII character per byte. Line numbers are low byte-high byte 16-bit binary and are stored with the text (section 8.1.1 or 8.2.1). The amount of memory space available for the current program may be changed by means of the LOCK command (section 5.7.1). The number of bytes available to (by LOCK or by default), but left unused by, the current program may be obtained with the SIZE function (section 3.3). These unused bytes are used for storage of the two-byte, one-byte, and string arrays (section 3.2).

One may not use the LOCK command to set the boundary at any page that already has current program text in it, nor at any page below the bottom of the text area. If the LOCKed area is exceeded during the entering of a statement of the current program, Control Basic will print "SORRY" and will accept no more statement input until more space is made available, either by deleting some present lines or by unLOCKing more text area.

Control Basic allocates storage in "pages", or 256-byte units. Model CB-308 has roughly the following organization:

page 0 - not used
pages 1,2, and 3 - used for I/O routines, variables, and stack storage
pages 4 through 31 - text area for current program and arrays
pages 32 through 227 and 240 through 255 - used for SAVEd program files
pages 228 through 239 - storage of Control Basic program itself

Model MCB-216 has a slightly different organization:

pages 0 through 15 - storage of Control Basic program itself
pages 16 through 31 - used for previously-stored programs in PROM
pages 32 and 33 - used for variables and stack storage
pages 34 and 35 - text area for current program and arrays
pages 36 through 255 - used for SAVEd program files

Much more detailed information on memory allocation including listings of the data areas in RAM will be found in sections 8.1.1 (model CB-308) and 8.2.1 (model MCB-216).

CHAPTER 5

Control Basic Commands and Statements

5.1 Assignment Commands

5.1.1 LET Command

The LET command can be used to evaluate expressions and store the values in variables, the double byte array @(I), and the single byte array &(J). More than one variable or array element may be set by a single LET command. Some examples of the use of LET are:

```
10 LET A=234-5*6, @(A-3)=4*A, &(25)=18
20 LET U0=(A<B)+AND(ABS(X-1),%FF)+:*
30 LET A0=B0=0
40 LET @(0)=-1, &(0)=0
50 A=3, B=2*A
```

Note that statement 30 will not set both A0 and B0 to zero. The second equal sign (=) in this command is a compare operator (see section 3.4); thus only A0 will be altered according to whether B0 is equal or is not equal to 0.

Note also that after statement 40 is executed, the value of @(0) will not be -1. This is because &(2*I) is the same as the low order byte of @(I), and &(2*I+1) is the same as the high order byte of @(I). Thus &(1) will be 255 and @(0) will be -256.

The word "LET" is optional. Thus statement 50 sets A to 3 and B to 6.

5.1.2 PUT Command

The PUT command can be used to evaluate expressions and store the values (which must be between 0 and 255) into absolute memory locations. The form of PUT is

```
PUT(x) = a,b,c,...
```

where a, b, and c are expressions whose values are stored in consecutive memory locations beginning with the address which is the value of the expression x. Either single or consecutive memory locations may be altered with the PUT command. An example of the use of PUT is

```
     10 PUT(%800)=%11,AND(X+3,%FF),(X+3)/%100,%CD,%A4,%EC,%C9
     20 PUT(X+3) = :H, :O, :W, :D, :Y, %D
```

where part of a machine language subroutine (LD HL,X+3; CALL ECA4H; RET) is PUT into memory locations %800 through %806, and the string of ASCII code for "HOWDY <CR>" is put into locations X+3 through X+8. A better way of unpacking the low and high bytes of X+3 in statement 10 is to initialize the two byte array @(I) to the address, X+3, and then use the one byte array, &(I) or &(I+1) as appropriate, to specify low or high byte:

```
      5 LET @(0)=X+3
     10 PUT(%800)=%11, &(0), &(1), %CD, %A4, %EC, %C9
```

The counterpart of PUT is GET , but GET is a function as opposed to a command. GET(x) can be used in any expression and returns the contents of absolute memory location x.

5.2 Control Commands

5.2.1 IF Command

The IF command consists of the word IF followed by an expression, and then followed by one or more other commands. An example is:

```
10 IF A<B LET X=3; PRINT 'A IS LESS THAN B'
```

The command above tests the value of the expression A<B. If it is true (i.e., if it is not zero), the commands in the rest of the line of this statement will be executed. If the value of the expression is false (i.e., if it is zero), the rest of this statement will be skipped over and execution will continue with the next statement. Note that the word "THEN" is not used.

5.2.2 GOTO Command

The GOTO command consists of the word GOTO followed by an expression. An example is:

```
10 GOTO 120
```

This statement causes the program being executed to jump to statement 120. Note that a GOTO command cannot be followed by a semi-colon and other commands. It must end in a carriage return. The statement

```
10 GOTO A*10+B
```

causes the program to jump to a variable statement number as computed from the value of the expression.

Indiscriminate use of GOTO statements makes a program difficult to follow and should be avoided. On the other hand combining short and logically related commands in a single statement can make a program easier to understand and is highly recommended. Consider the following example of a game; this program uses many GOTOs but does not take advantage of multiple-command statements.

```
 10 PRINT "NUMBERS, NUMBERS, WHO CAN GUESS MY NUMBERS?"
 20 LET X=RND(100)
 30 LET N=0
 40 PRINT "I HAVE A NUMBER BETWEEN 1 AND 100"
 50 INPUT "YOUR GUESS" G
 60 LET N=N+1
 70 IF N>20 GOTO 130
 80 IF G>X GOTO 160
 90 IF G<X GOTO 180
100 PRINT "YOU GOT IT IN", N, " GUESSES"
110 PRINT "LET'S PLAY AGAIN"
120 GOTO 20
130 IF G=X GOTO 100
140 PRINT "NO, IT WAS", X
150 GOTO 110
160 PRINT "TRY A SMALLER NUMBER"
170 GOTO 50
180 PRINT "TRY A BIGGER NUMBER"
190 GOTO 50
```

This is a working program, but it could have been much cleaner and easier to understand with half of the GOTO statements deleted and with a few of the other statements combined:

```
 10 PRINT "NUMBERS, NUMBERS, WHO CAN GUESS MY NUMBERS?"
 20 LET X=RND(100), N=0
 40 PRINT "I HAVE A NUMBER BETWEEN 1 AND 100"
 50 INPUT "YOUR GUESS" G; LET N=N+1
 70 IF (N>20)*(G#X) PRINT "NO, IT WAS", X; GOTO 110
 80 IF G>X PRINT "TRY A SMALLER NUMBER"; GOTO 50
 90 IF G<X PRINT "TRY A BIGGER NUMBER"; GOTO 50
100 PRINT "YOU GOT IT IN", N, " GUESSES"
110 PRINT "LET'S PLAY AGAIN"; GOTO 20
```

5.2.3 FOR Command

The FOR and NEXT commands are used to set up loops in Control Basic. The FOR statement consists of the word FOR followed by a variable or a double byte array element, an equal sign, an expression, the word TO, another expression, and optionally the word STEP and a third expression. The variable or the two byte array element is called the control variable of the loop. It is set to the value of the first expression before entering the loop. The second expression is the limit of the control variable. It is evaluated and stored internally at entry. The third expression is the step size, and it is also evaluated and stored at entry. It can be positive, negative, or zero. If the step size is not specified, it is assumed to be +1. After the step size is stored, execution of Control Basic continues with the next statement or the next command of a multiple-command statement.

5.2.4 NEXT Command

The NEXT command consists of the word NEXT followed by the control variable of the loop. The control variable is updated by the amount of the step size and is then compared with the limit of the control variable. If it is within the limit (inclusive), Control Basic will loop back to the command that follows the FOR command and repeat the loop. If the updated control variable is outside the limit, the internal storage for this loop is cleared and Control Basic proceeds to execute the statement which follows the NEXT statement. The following example will illustrate the use of FOR-NEXT loops with varying STEP sizes:

```
 10 PRINT "TEST",
 20 FOR T=1 TO 3
 30 PRINT T,
 40 NEXT T
 50 PRINT; PRINT 'COUNT DOWN',
 60 FOR C=100 TO 98 STEP -1
 70 PRINT C;
 80 NEXT C
 90 PRINT '    ...'; PRINT 'STEP',
100 FOR @(3)=10 TO -15 STEP -5
110 PRINT @(3),
120 NEXT @(3)

OK
>RUN
TEST        1       2       3
COUNT DOWN   100     99      98      ...
STEP        10      5       0       -5      -10     -15
```

5.2.5. Notes on FOR-NEXT Loops

The following are several unusual features of Control Basic FOR-NEXT loops which are worth mentioning. Numbers 4 through 7 below are not recommended if the user is attempting to write clean, easily debugged code.

1. The loop will be executed at least once no matter what the initial value, step size, and the limit of the control variable are.

2. If the step size is 0, the loop will never end unless the control variable is altered inside the loop.

3. After the loop is finished, the control variable will have the final updated value, which is outside the limit.

4. Since Control Basic is interpreted rather than compiled, it is perfectly acceptable to put the NEXT statement physically before the FOR statement as long as there are GOTOs to make Control Basic "see" the FOR before the NEXT.

5. It is also acceptable to have unequal numbers of FORs and NEXTs, as long as there are IFs and GOTOs so that Control Basic will not "see" a NEXT without first encountering a FOR with the same control variable.

6. It is acceptable to have a GOTO out of the loop and later GOTO back into the loop. It is also acceptable to have a GOTO out of the loop and never come back into the loop. In the latter case the loop remains "active"; it takes some stack space but is harmless otherwise.

7. When a new FOR command having a control variable that is the same as an old but still active FOR loop is encountered, the old loop is purged.

8. FOR-NEXT loops can be nested as long as each level uses a different control variable. The depth is limited only by stack space.

9. If a NEXT commmand for the outer loop is encountered inside the inner loop when using nested FOR-NEXT loops, the inner loop is purged automatically.

Note that the above apply to Cromemco Control Basic but do not necessarily apply to other Basic-like languages. For example the following program does not work on many of them:-

```
 10 REM  TRY THIS WITH BOTH A>0 AND A<0
 15 INPUT A
 20 FOR J=1 TO 3
 30 IF A<0 GOTO 50
 40 NEXT J
 50 PRINT J
 60 FOR I=1 TO 3
 70 FOR J=1 TO 3
 80 NEXT J
 90 NEXT I
100 GOTO 15
```

Although this program works with Control Basic, it's a bit obscure.

5.3 Subroutines

5.3.1 GOSUB Command

The GOSUB statement is similar to the GOTO statement except that (a) the current statement number and position within the statement are stored internally (this allows control to be transferred back by RETURN); and (b) other commands separated by semi-colons can follow it in the statement. The following are legal GOSUB statements:

```
     1Ø GOSUB 12Ø
     2Ø GOSUB A*1Ø+B
     3Ø GOSUB 12Ø; IF C<=Ø GOTO 1Ø
     4Ø REM  IF C>Ø THE GOTO ABOVE WILL NOT BE EXECUTED
```

Statement 1Ø will cause the execution to jump to statement 12Ø; when a RETURN command (see following section) is encountered, control will transfer back to the next following statement or command of a multiple-command statement. Statement 2Ø will cause the execution to jump to a variable statement number as computed from the value of the expression A*1Ø+B.

5.3.2 RETURN Command

A RETURN command must be the last command in a statement and followed by a carriage return (i.e., it cannot be followed by a semi-colon and other commands). When a RETURN command is encountered, it will cause the execution to jump back to the command following the most recent GOSUB command.

An active FOR-NEXT loop in the calling program is no longer active in the subroutine but will be restored to be active after the subroutine RETURNs to the calling program. Any active FOR-NEXT loop in the subroutine itself will be purged automatically when RETURN is encountered. Variables are always global; therefore subroutines can have no local variables.

GOSUBs can be nested. The depth of nesting is limited only by the stack space and can be no more than 24 levels deep. GOSUBs are also recursive with the limitation that all variables and arrays are global. For example, consider the function F(N) of positive integers N such that:

```
F(1)=1
F(N)=N*F(N-1) for N>1
```

One can write the following test program to illustrate the function F(N) using recursive GOSUBs (underlined words are user-typed):

```
 10 INPUT 'GIVE ME A SMALL POSITIVE INTEGER' N
 20 GOSUB 100
 30 PRINT 'F(', #1,N, ')=', F
 40 STOP
100 IF N=1 LET F=1; RETURN
110 N=N-1; GOSUB 100; N=N+1
120 F=N*F; RETURN

OK
>RUN
GIVE ME A SMALL POSITIVE INTEGER:6
F(6)=720
```

Note that F(N) as defined above is simply the factorial function.

5.3.3 RUN Command

The RUN command consists of the word RUN followed optionally by a page number. The RUN command is a subroutine call on a grand scale. If a page number is given, it calls the Control Basic program SAVEd (or EPROMed) on that page.[9] If a page number is not given, it calls the Control Basic program in the active text area. The RUN and GOSUB commands differ in that a GOSUB command calls a subroutine that is within the same program as the calling routine, whereas a RUN command calls a subprogram which is in a separate file. A subprogram in a separate file has the advantage that it has its own set of line numbers. Since RUN may be used in a statement, the following is a legal program:

```
10 RUN 40
20 PRINT "END OF FACTORIAL"
```

This would run the program given above in section 5.3.2, assuming it had been SAVEd on page 40. Thus, it is easy to see how different programs could share the same set of debugged "library" subprograms.

The RUN command can be nested and is recursive. As it is for GOSUB, the depth of nesting is limited by stack space, and one should bear in mind that all variables and arrays are global.

5.3.4 AUTORUN Command

The AUTORUN command is a feature of model MCB-216 Control Basic only. If the first line in a Basic program stored on page 10H begins with the command AUTORUN, then that program (on page 10H) will run automatically each time the computer is reset.

The console serial port of the computer will automatically be set to 9600 baud. However, it is easy for the program to change the baud rate to another value.

For example, if the following program is stored on page 10H,

```
     10 AUTORUN
     20 OUT(0) = %84
     30 RUN %11
```

the baud rate will be changed to 300 with one stop bit (see the SCC Input/Output Register Description in the SCC Instruction Manual), and then the program beginning on page 11H will be RUN. This process occurs every time the computer is hardware reset.

It should be noted that during normal execution (i.e., in all cases except after a hardware reset or power-on) the line beginning with the word "AUTORUN" is treated as a REMark.

5.3.5 STOP Command

The STOP command consists of the word STOP followed by a carriage return. It cannot be followed by a semi-colon and other commands on the same line. The STOP command goes with the RUN command in the same way that RETURN goes with GOSUB. The principle difference between the two sets of commands is that RUN-STOP may be used to call a subprogram in a separate file with its own set of line numbers.

STOP returns control from a subprogram to the calling program (which will continue RUNning); however, if the program which is RUNning when STOP is encountered is the current program in the text area, execution is terminated and the prompt is given. If the end of file is reached in a subprogram before a STOP statement is encountered, Control Basic will also return to direct mode instead of to the calling program. An example will illustrate the way STOP passes control from subprogram to calling program. The following is a program which has been previously stored (SAVEd or EPROMed) at page 32:

```
10 PRINT "THIS IS A PROGRAM ON PAGE 32"
20 RUN 33
30 PRINT "END OF PROGRAM ON PAGE 32"
40 STOP
```

and the next listing is a program which has been previously stored at page 33:

```
10 PRINT 'BUT THIS PROGRAM IS ON PAGE 33'
20 STOP
```

RUNning the program on page 32 will produce the following result:

```
OK
>RUN 32
THIS IS A PROGRAM ON PAGE 32
BUT THIS PROGRAM IS ON PAGE 33
END OF PROGRAM ON PAGE 32
```

Note that this will have no effect on the current program in the text area if there is one. This means that one does not need to LOAD a program to RUN it, a most useful feature (see section 5.3.3 for more information).

5.3.6 CALL Command

The CALL command is used to call a machine language subroutine. The word CALL is followed by an expression and then a list of arguments enclosed in parentheses. The value of the expression is the absolute address of the entry of the machine language subroutine. The argument list consists of none, or one or more arguments. Each argument is a variable, a one byte array element, or a two byte array element. Arguments are separated from one another by commas, and the entire list is enclosed in a pair of parentheses.

The machine language subroutine can be stored in memory by a loader, an assembler, a monitor program, or by Control Basic itself. To store a machine language subroutine with Control Basic, one may use the LET or PUT commands. The example of section 5.1.2 illustrates this. We can now CALL this machine language routine to print the word "HOWDY" on the console with the statement:

```
    30 CALL %800
```

[This example will work only if there is a console printing routine whose starting address is at ECA4H, as there is for model CB-308 Control Basic.] Note that no arguments were passed in this CALL statement.

Arguments are passed by addresses and these addresses are stored in the stack before Control Basic passes control to the machine language subroutine. The arguments are stored in a "Last In-First Out" or LIFO format; thus, the last argument on the statement line will be the first one POPed off the stack. The number of arguments is passed in the C register. The subroutine must POP the stack (C) times even if it does not need the arguments. Other than this arrangement of the stack, the subroutine can change all registers without any limitation. When the subroutine is finished, it uses a RET (Z80 instruction) to POP the stack once more, which also returns control to Control Basic. Another example illustrating argument passing is:

```
    10 CALL %2000 (A, &(3*B+1), @(3))
```

In this example the subroutine at memory location 2000H is called and passed the three arguments, A, &(3*B+1), and @(3). Upon entrance to the machine routine, the C register contains the number 3. The stack pointer points to the address of @(3), which is followed by the address of &(3*B+1), the address of A, and finally the return address.

5.4 Input and Output Commands

5.4.1 PRINT Command

The PRINT command is used to print all types of output on the console device, including strings, constants in both decimal and hexadecimal, and the values of variables, arrays, and expressions. Its format is the word "PRINT" followed by a list of items separated by commas. The items in the list may include any of the following:

1. Constants (decimal, hexadecimal, ASCII).

2. Variables (A through Z and A0 through Z0).

3. One-byte, two-byte, and string array elements.

4. Expressions formed from any of the above, arithmetic and compare operators, and the legal functions.

5. Strings surrounded by matched pairs of single or double quotes.

6. Format control characters described later in this section.

A sample PRINT command follows which shows several of these items. Note the use of both quoted and unquoted strings, a variable, and a constant expression:

```
          10 PRINT 'THIS ', "OR THAT ", $(A+3), X, 2*3-5/4

     OK
     >RUN
     THIS OR THAT PLUS    -9     5
```

The string array $(A+3) has been set elsewhere to the string "PLUS", and the variable X has been previously set equal to -9.

The items in the print list can also be format controls. The allowable characters to use as format controls are summarized briefly below, followed by a more detailed description and some illustrations.

#	specify leading spaces or hexadecimal
%	specify number of hexadigits.
,	(preceding) specify additional leading spaces
,	(succeeding) specify no CR-LF at end of item
_	tab to a specific column

The character "#" followed by a number n means print leading spaces (if necessary) to make numbers at least n spaces wide. Spaces can also be added by means of extra commas between items. The format control "#%%" means print numbers in 4 hexadigits, whereas "#%" means print the low order byte of a number only, in 2 hexadigits. A format control stays effective until changed by another format control or until the PRINT command ends. The default format if no control is specified is #6. The following example will illustrate these points:

```
      10 PRINT 1, -2, 345, -6789
      20 PRINT #3, 1, -2, 345, -6789
      30 PRINT #%%, 1, -2, 345, -6789
      40 PRINT #%%, 1,, -2,, 345,, -6789
      50 PRINT #%, 1,, -2,, 345,, -6789

  OK
  >RUN
       1    -2   345 -6789
    1 -2345-6789
  0001FFFE0159E57B
  0001 FFFE 0159 E578
  01 FE 59 7B
```

The underline or back-arrow character (ASCII 5FH) is used as a format control to indicate tabbing to a particular column on the console device. It should be followed by a number n which is the column number; this will cause the cursor or print head to move to that position. The print positions are numbered from left to right beginning with 1. If n is 1, the cursor (print head) will move to the left margin without a line feed. For model MCB-216 of Control Basic, the cursor will move to position n independent of its present position, i.e, it may move either left or right. Model CB-308, however, allows movement only to the right of the present position.

The PRINT command generates a carriage return and line feed at the end of the last item in the command. However, if there is a comma at the very end of the line, the CR-LF is not generated. The following example will illustrate both the tab feature just discussed and the suppressed CR-LF:

```
        1Ø PRINT 'THIS',
        2Ø PRINT ' IS ',
        3Ø PRINT 'PRINTED ON ONE LINE'.
        4Ø PRINT 'WE HAVE', _1Ø, 'O = O', _1Ø, '- / /'

    OK
    >RUN
    THIS IS PRINTED ON ONE LINE
    WE HAVE  Θ ≠ Ø
```

Note that statement 4Ø of this example is somewhat special purpose in that it will work correctly only with model MCB-216 and with a console device which will print non-destructive spaces such as a teletype.

5.4.2 <u>INPUT Command</u>

The INPUT command is used to prompt the operator to type in an expression or a string to be saved by the program in a variable, an array element, or a string array. The word INPUT is followed by a list of items (any of the above) separated by commas. The items may also be strings (quoted in single or double quotes) or tab controls for the terminal. These items look and behave exactly like those used in a PRINT statement (see section 5.4.1). For example, the underline or back-arrow character (ASCII 5FH) may be used to specify INPUT at a particular column on the console terminal. Following are two examples which will illustrate some of the features of INPUT statements. The first example shows tab control and the second shows how to input character strings (underlined sections are to be user-typed):

```
    10 INPUT "INPUT 'A' HERE" A, #30, "AND 'B' THERE" B
    20 PRINT #0, A, #30, B

OK
>RUN
INPUT 'A' HERE:123
                              AND 'B' THERE:456
123                           456
```

Note that the quoted letters 'A' and 'B' are considered part of the strings shown because they are in single quotes and the strings are in double quotes. This could be reversed (letters in doubles; strings in singles); however, if both the strings and the letters are quoted with the same symbol, Control Basic will not understand and will print "WHAT?" (see Chapter 6).

```
    10 INPUT "GIVE ME A STRING: ", $(23)
    20 PRINT "YOU TYPED '", $(23), "'"
    30 INPUT X, @(11)
    40 PRINT X, @(11)

OK
>RUN
GIVE ME A STRING: THIS IS A STRING.
YOU TYPED 'THIS IS A STRING.'
X:3*5+5*6
@(11):X+1
    45     46
```

In this example the operator answered the first input inquiry by typing "THIS IS A STRING.", answered the second one by "3*5+5*6", and answered the third one by "X+1". Note that in statement 30 where a variable and an array element are the input items, the names of the variables, "X" and "@(11)", are also printed automatically by Control Basic as prompts. If the operator makes an error in response to the prompt, Control Basic will also retry that part of the INPUT and reprint the prompt. When the input item is a string array as it is in statement 10, there is no automatic prompt and whatever the operator types is always accepted. We now rerun the same program giving several unusual inputs to see their effect:

```
OK
>RUN
GIVE ME A STRING: %?#!*
YOU TYPED '%?#!*'
X:%?#!*
WHAT?
X:2/Ø
HOW?
X:@(8ØØØ)
SORRY.
X:2+1
@(11):(3+4)*5)
WHAT?
@(11):(3+4)*5
      3    35
```

Programmers not wishing to take advantage of the automatic variable name prompt can supply their own in a pair of single or double quotes preceding the variable or array name without a comma between them. The next example will illustrate:

```
    1Ø INPUT "PLEASE GIVE ME YOUR ","WEIGHT (IN LBS)" X
    2Ø LET Y=(1Ø*X+11)/22
    3Ø PRINT 'THAT IS ABOUT ', #Ø, Y,\" KILOGRAMS"

OK
>RUN
PLEASE GIVE ME YOUR WEIGHT (IN LBS):HUNDRED AND FORTY
WHAT?
WEIGHT (IN LBS):14Ø
THAT IS ABOUT 64 KILOGRAMS
```

Note that only the prompt "WEIGHT (IN LBS)" is repeated; the string "PLEASE GIVE ME YOUR " is separated from the input item by a comma and is thus not repeated following an incorrect response. The expression in statement 2Ø handles the unit conversion and rounding in integer arithmetic.

5.4.3 OUT Command

The OUT command is used to output a byte of data to any I/O port. The format of the command is

```
OUT(x) = y
```

where x is the port number and is an expression with a value between 0 and 255 (0-FFH), and y is an expression also with a value between 0 and 255, which is the actual byte of data to be output.

The counterpart of OUT is IN, but IN is a function rather than a command. The input function IN(x) is used to read data or status information in from an I/O port. The port number x is again an expression with a value between 0 and 255 (0-FFH). This function, like any other function, cannot stand for a command by itself. Thus in the following example illustrating the use of the OUT command and the IN function, statement 10 is invalid while the other statements are legal.

```
10 IN(3)
20 PRINT #0, IN(3), SIZE, LOC
30 LET A=AND(IN(B+1),%40)
40 OUT(B+1)=A
```

5.5 Non-Executable Commands

5.5.1 REM Command

The word "REM" may be used in a Control Basic statement to denote a REMark line. This "command" is non-executable and everything following the REM on that statement line (preceding the CR) is ignored by the Interpreter. REM may also appear on the same line with other commands in a multiple-command statement; everything preceding the REM command will be executed but the rest of the line is again ignored. There are several commands which must be the last command on a line; these cannot be followed by even a REM command. They are: GOTO, RETURN, and STOP. [Also note that the word "REMARK" may be used instead of "REM" if you wish since the letters "ARK" are ignored anyway.] The following example illustrates the use of REM:

```
1Ø REM
2Ø REM   *** Title of Program ***
3Ø REM
4Ø PRINT "RECIPE"; REM PRINT "COOKBOOK"
```

In this example statements 1Ø, 2Ø, and 3Ø will be ignored during execution of the program; statement 4Ø will cause Control Basic to print the word "RECIPE" but not the word "COOKBOOK".

5.5.2 AUTORUN Command

Note that AUTORUN is treated exactly like a REMark in every case except when the computer hardware reset is pressed, in which case AUTORUN causes the Control Basic program at page 16($1Ø) to begin executing. AUTORUN is a feature of the MCB-216 model of Control Basic only. For more information see section 5.3.4.

5.6 Editing Commands (direct only)

5.6.1 NEW Command

The NEW command is used to delete the current program stored in the text area. It is a direct command which is executed by typing "NEW <CR>" from the console device. NEW is executed automatically upon entering Control Basic initially and upon issuing a LOAD command (section 5.7.4).

5.6.2 LIST Command

The LIST command is used to print some or all of the statements of the current program in the text area in numerical order on the console device. It is used frequently in entering, editing, and verifying Control Basic programs. There are several legal forms of LIST; the simplest is to type "LIST <CR>", which will then list the entire current program from beginning to end. A more complete format is

```
LIST x,n
```

where x is the beginning line number (a constant) and n is the number of lines to be LISTed. Both these values are optional, and the comma (,) should be omitted if n is. The following examples will illustrate these points further:

LIST 120	prints all statements beginning with line number 120.
LIST 120,3	prints at most (i.e., fewer will be printed if fewer are present) 3 statements beginning with line 120.
LIST ,20	prints at most 20 statements starting with the beginning line of the current program.
L.90,1	prints only statement number 90, if there is one. Note the abbreviated form of LIST (Chapter 4).

5.7 Storage and File Commands (direct only)

5.7.1 LOCK Command

The LOCK command is used to change the boundary between the current program text area and the area reserved for SAVEing program files. A sample LOCK command is

```
LOCK 40
```

which will set the boundary between the text area and the SAVE area at 2800H (page 40D=page 28H). Thus pages 4 through 39 (CB-308) or 34 through 39 (MCB-216) become allocated to the current program text and pages 40 through 255 can be used to SAVE files. The LOCKed value (see "TXTLMT" in Memory Allocation, section 8.1.1 or 8.2.1) is set when Control Basic is cold started to 32 (20H) for model CB-308 and 36 (24H) for model MCB-216. If these values are lowered by means of the LOCK command after a program has been entered into the text area, Control Basic will print "SORRY" if the page number specified would include part of that current program. For example, suppose the current program resides in pages 4 through AH. The command "LOCK %A" will print "SORRY" but the command "LOCK %B" will be accepted and executed.

If the LOCK command is used to increase the size of the current program area, it is up to the user to make sure that this does not endanger any previously SAVEd files. The allowable range of the LOCK command is 5-255 (5H-FFH) for CB-308 and 35-255 (23H-FFH) for MCB-216.

5.7.2 SAVE Command

The SAVE command is used to save the current program in a particular page of memory. The page number given should be above the LOCKed text area and generally coincident only with read/write memory space. A sample SAVE command is

```
SAVE %43
```

where the current program will be saved in memory beginning at 4300H. When the SAVE command is finished, Control Basic will print a message indicating how many pages of memory were used. For example the message

```
SAVED ON PAGE %43 TO %45
```

means that the current program now occupies all of pages 43H and 44H, and, all or part of page 45H. If the copy does not compare with the original (for example, if there is no RAM at the specified page), Control Basic will print "SORRY". "SORRY" will also be printed if one attempts to SAVE a program in pages 0 through one less than the LOCKed number (i.e., 0 through 31 for CB-308 and 0 through 35 for MCB-216, as they are initially configured). This region is reserved by Control Basic for variables, stack, and the current text (see section 8.1.1 or 8.2.1).

The SAVE command may also be used in conjunction with a 32K Bytesaver to program 2716 (2516) PROMs. Remember to turn on the program power of the Bytesaver board before issuing the SAVE command. Used in this way, SAVE will take several minutes to execute due to the wait states required by the 2716's while programming.

5.7.3 EPROM Command

The EPROM command may be used to save the current program in blank or erased 2708 PROMs. The PROMs must be loaded onto a Cromemco Bytesaver board residing in currently addressed memory. A convenient place to have the PROMs reside when using model CB-308 of Control Basic is from addresses F000H to FFFFH, or simply the top half of the Bytesaver which contains CB-308. A sample EPROM command is

```
EPROM 64
```

where the current program will be saved in PROMs addressed beginning at 4000H (page 64D=page 40H). The page number must be divisible by 4 (since 2708's are 1K bytes long, or 4 pages). This command will take several minutes to execute, where the amount of elapsed time will depend on the length of the program being saved.

When the EPROM command is finished, Control Basic will print a message indicating how many pages of memory were used (always a multiple of four pages).

If the copy does not compare with the original, Control Basic will print "SORRY". See the SAVE command, section 5.7.2, for information on programming 2716 (2516) PROMs.

5.7.4 LOAD Command

Programs which have been saved with the SAVE or EPROM commands may be copied back to the text area for editing by means of the LOAD command. This process performs an automatic "NEW"; in other words the current program in the text area is deleted. An example use of LOAD is

```
LOAD %43
```

where the file stored on page 43H (or at 4300H in memory) is copied into the text area. If no program is stored at page 43H, Control Basic prints "SORRY". Also, since the length of any given stored program is contained in the first two bytes of the first page (see section 8.1.1 or 8.2.1), Control Basic knows immediately if it will fit into the text area. If it will not fit, "SORRY" is printed and no LOADing takes place. The size of the text area may be increased (to allow for LOADing very long programs) by means of the LOCK command, section 5.7.1.

5.8 Console Terminal Commands (direct only)

5.8.1 WIDTH Command

The WIDTH command may be used to compensate for the non-standardization of console terminals by allowing the user to set both a soft and hard screen width or margin. A sample WIDTH command is

```
WIDTH 72,80
```

which sets the soft margin at column 72 and the hard margin at column 80. When the soft margin has been exceeded during output, Control Basic will generate a carriage return-line feed sequence after any space character (and thus, at the ends of words). When the hard margin has been exceeded, it generates an immediate CR-LF. The values above are good choices for an 80-column CRT. The default WIDTH for Control Basic model CB-308 is 60,60 and for MCB-216 it is 72,72. The allowable usable range for the first number is 0-129 and for the second is 0-255.

Since WIDTH is a direct command, it cannot be set by a statement in a program. However, it is sometimes useful to be able to change the screen width in a particular program. One method of doing this is to use the PUT command to alter the RAM locations where the soft and hard margins are stored (see "MARGN" in Memory Allocation, section 8.1.1 or 8.2.1). An example of this (for CB-308) is

```
10 PUT(%3E2) = 72,80
```

which has the same effect as the WIDTH example above. A similar command may be used to set the margins for MCB-216:

```
10 PUT(%21E3) = 72,80
```

5.8.2 NULL Command

The NULL command is used to set the number of nulls transmitted by Control Basic after any carriage return-line feed sequence. This allows the user to set carriage return time for hardcopy devices such as teletypes. A sample NULL command is

```
NULL 3
```

which causes Control Basic to transmit 3 null characters after every CR-LF. The default value for NULL upon entering Control Basic (either model) is Ø. The allowable, usable range of the value is Ø-128.

Introducción:

El término microcomputador ha sido usado para describir virtualmente cada tipo de pequeño dispositivo de cómputo diseñado en - los últimos 5 años.

El mayor impacto de la tecnología de gran integración se ha encontrado en la tecnología MOS (Metal Oxide Semiconductor), con ella es posible fabricar sistemas de cómputo utilizando un pequeño número de componentes.

Para tener una mejor comprensión del funcionamiento de un sistema de cómputo basado en un microprocesador a continuación daremos una descripción más o menos detallada de la arquitectura del microprocesador Z-80, su funcionamiento interno y su relación - con el mundo exterior.

Cualquier sistema de cómputo consiste de 3 partes:

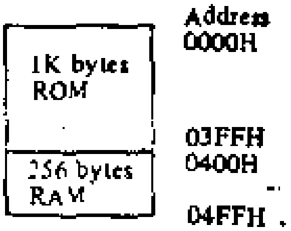
1.- CPU (Unidad Central de Proceso).
2.- Memoria
3.- Interfases con dispositivos periféricos o (I/O) entrada/salida.

El CPU es el corazón del sistema, en particular para un sistema de cómputo basado en un microprocesador, este último será la parte más importante.

[illegible] the input is an 8 bit status word. The [illegible] standard tri-state driver while the output data could [illegible] standard TTL latch. [illegible] this example we have used a Z80-PIO for the I/O circuit. This single circuit attaches to the data bus as shown and provides the required 16 bits of TTL compatible I/O. (Refer to the Z80 PIO manual for details on the operation of this circuit.) Notice in this example that with only three LSI circuits, a simple oscillator and a single 5 volt power supply, a powerful computer has been implemented.

ADDING RAM

Most computer systems require some amount of external Read/Write memory for data storage and to implement a "stack." Figure 9.0-2 illustrates how 256 bytes of static memory can be added to the previous example. In this example the memory space is assumed to be organized as follows:



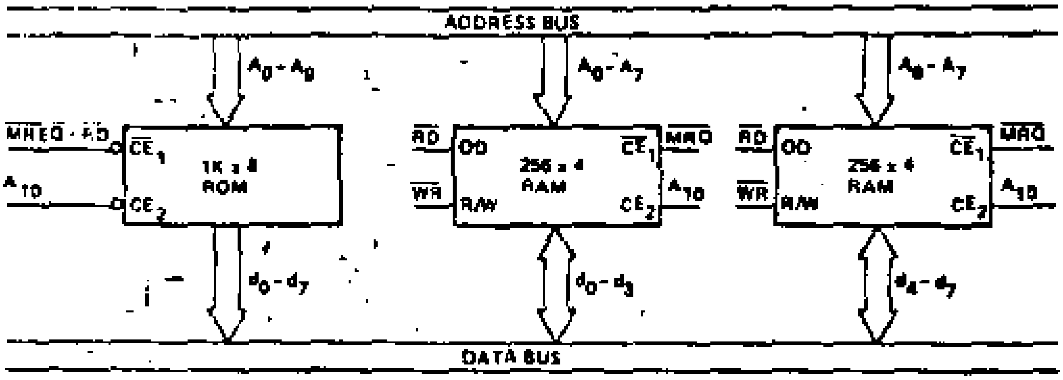


FIGURE 9.0-2
ROM & RAM IMPLEMENTATION EXAMPLE

In this diagram the address space is described in hexidecimal notation. For this example, address bit A_{10} separates the ROM space from the RAM space so that it can be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder will be required to form the chip selects.

MEMORY SPEED CONTROL

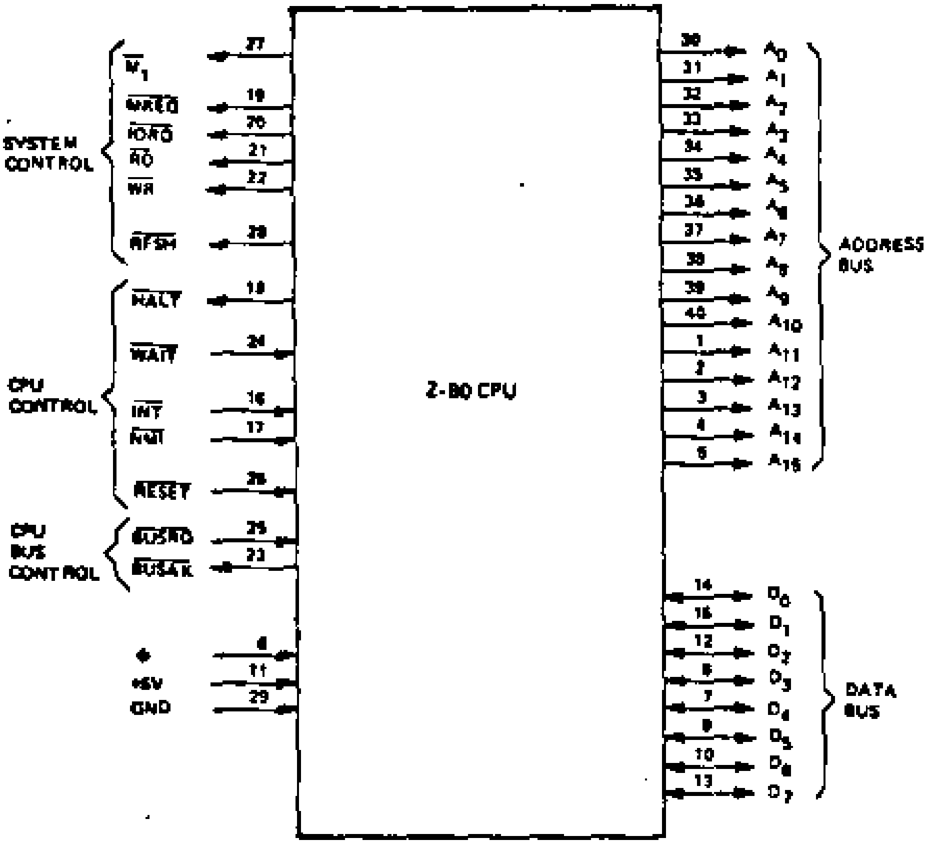
For many applications, it may be desirable to use slow memories to reduce costs. The $\overline{\text{WAIT}}$ line on the CPU allows the Z-80 to operate with any speed memory. By referring back to section 4 you will notice that the memory access time requirements are most severe during the M1 cycle instruction fetch. All other memory accesses have an additional one half of a clock cycle to be completed. For this reason it may be desirable in some applications to add one wait state to the M1 cycle so that slower memories can be used. Figure 9.0-3 is an example of a simple circuit that will accomplish this task. This circuit can be changed to add a single wait state to any memory access as shown in Figure 9.0-4.

Z-80 CPU.- Descripción del integrado.

El Z-80 se encuentra empaquetado en un circuito integrado de 40 pins (patas), en la siguiente figura se describe la función de cada una de ellas y en el texto su actividad y significado electrónico.

3.0 Z-80 CPU PIN DESCRIPTION

The Z-80 CPU is packaged in an industry standard 40 pin Dual In-Line Package. The I/O pins are shown in figure 3.0-1 and the function of each is described below.



Z-80 PIN CONFIGURATION
FIGURE 3.0-1

A_0-A_{15}
(Address Bus)

Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.

D_0-D_7
(Data Bus)

Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$
(Machine Cycle one)

Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M1}$ is generated as each op code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH or FDH. $\overline{M1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

$\overline{MREQ}$
(Memory Request)

Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{\text{IORQ}}$
(Input/Output Request)

Tri state output, active low. The $\overline{\text{IORQ}}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{\text{IORQ}}$ signal is also generated with an $\overline{\text{M1}}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.

$\overline{\text{RD}}$
(Memory Read)

Tri-state output, active low. $\overline{\text{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{\text{WR}}$
(Memory Write)

Tri-state output, active low. $\overline{\text{WR}}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{\text{RFSH}}$
(Refresh)

Output, active low. $\overline{\text{RFSH}}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{\text{MREQ}}$ signal should be used to do a refresh read to all dynamic memories.

$\overline{\text{HALT}}$
(Halt state)

Output, active low. $\overline{\text{HALT}}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{\text{WAIT}}$
(Wait)

Input, active low. $\overline{\text{WAIT}}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.

$\overline{\text{INT}}$
(Interrupt Request)

Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{\text{BUSRQ}}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{\text{IORQ}}$ during M_1 time) is sent out at the beginning of the next instruction cycle. The CPU can respond to an interrupt in three different modes that are described in detail in section 5.4 (CPU Control Instructions).

$\overline{\text{NMI}}$
(Non Maskable Interrupt)

Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{\text{INT}}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{\text{NMI}}$ automatically forces the Z-80 CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a $\overline{\text{BUSRQ}}$ will override a $\overline{\text{NMI}}$.

$\overline{\text{RESET}}$

Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes:

1) Disable the interrupt enable flip-flop
2) Set Register I = 00_H
3) Set Register R = 00_H
4) Set Interrupt Mode 0

During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{\text{BUSRQ}}$
(Bus Request)

Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.

$\overline{\text{BUSAK}}$
(Bus Acknowledge)

Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

Φ

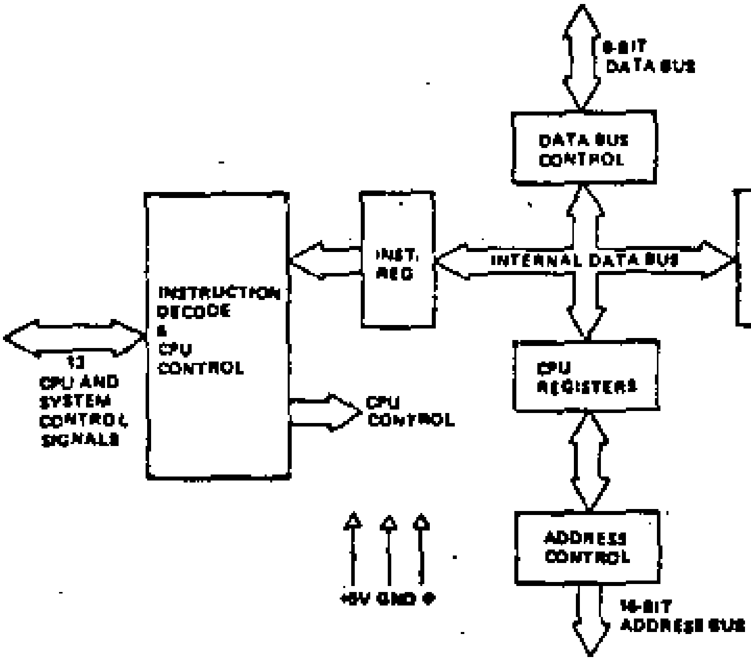
Single phase TTL level clock which requires only a 330 ohm pull-up resistor to +5 volts to meet all clock requirements.

Z-80 Arquitectura Interna.

En la siguiente figura se incluye la arquitectura del CPU.

2.0 Z-80 CPU ARCHITECTURE

A block diagram of the internal architecture of the Z-80 CPU is shown in figure 2.0-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.



Z-80 CPU BLOCK DIAGRAM
FIGURE 2.0-1

2.1 CPU REGISTERS

The Z-80 CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2.0-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z-80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.
2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

Registros:

El CPU Z80 contiene 208 (bits) celdas de memoria estática de acceso aleatorio de lectura/escritura accesibles al programador, ordenadas formando registros según se muestra en la siguiente figura en donde:

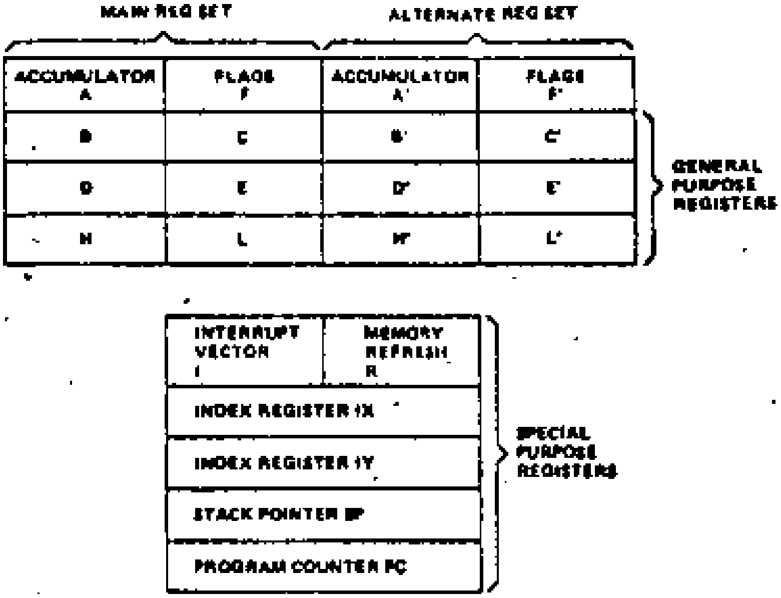
PC.- Contador del programa.

SP.- Apuntador a pila o stack.

IX.-
} Registros de índice.
IY.-

I .- Registro de dirección de página en interrupciones.

R .- Registro para refrescamiento de memoria.



Z-80 CPU REGISTER CONFIGURATION
FIGURE 2.0-2

3. **Two Index Registers (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z-80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

5. **Memory Refresh Register (R).** The Z-80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with with a single exchange instruction so that he may easily work with either pair.

Dos conjuntos de registros de trabajo accesibles al programador forman la unidad de almacenamiento interno del CPU, pudiéndose intercambiar a gusto del programador a través de una sola instrucción lo cual simplificará el manejo de interrupciones reduciendo el tiempo de atención a la rutina de interrupción.

Unidad Aritmética/Lógica (ULA).

La ULA del microprocesador Z-80 permite la ejecución de las siguientes operaciones:

Operaciones de Acumulador:

Incrementa, decrementa.
Corrimientos y rotaciones.
Prende, apaga o prueba un bit.

Operaciones Aritméticas:

Suma, resta.

Operaciones Lógicas:

AND, OR, OR Exclusiva, Comparación.

Banderas:

S	Z	Ø	H	Ø	P/V	N	C

(Carry): Bandera de acarreo: se genera como un noveno bit del acumulador en operaciones de suma y como (Borrow) bit de deuda en la resta.

(Zero): Bandera que se enciende cuando el resultado de una opeción es zero.

(Negative Sign): Bandera de signo. Se enciende si el resultado de una operación es negativo, esta es una copia del 7 bit o más significativo del acumulador.

(Parity/Overflow): Para operaciones lógicas esta bandera indica la paridad del resultado. Prendiéndose si la paridad es par. Para operaciones aritméticas en 2 complementa con cantidades seguidas.

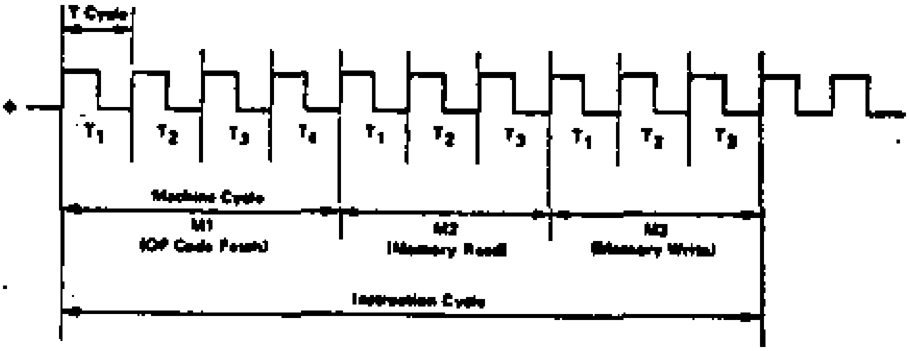
(H,S): Banderas para manejo de cantidades en BCD.

4.0 CPU TIMING

The Z-80 CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write
I/O device read or write
Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T cycles and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T cycles long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 10, the exact timing for each instruction is specified.



BASIC CPU TIMING EXAMPLE
FIGURE 4.0-0

All CPU timing can be broken down into a few very simple timing diagrams as shown in figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)
4.0-2. Memory data read or write cycles
4.0-3. I/O read or write cycles
4.0-4. Bus Request/Acknowledge Cycle
4.0-5. Interrupt Request/Acknowledge Cycle
4.0-6. Non maskable Interrupt Request/Acknowledge Cycle
4.0-7. Exit from a HALT instruction

Registro de Instrucciones y Control.

En cada operación de fetch la instrucción obtenida es colocada en el registro de instrucciones, el cual es entrada de la unidad de control en donde se lleva a cabo la decodificación y secuenciación de señales de control necesarias para la ejecución de la instrucción, tanto internas como externas al CPU.

Ciclos del CPU.

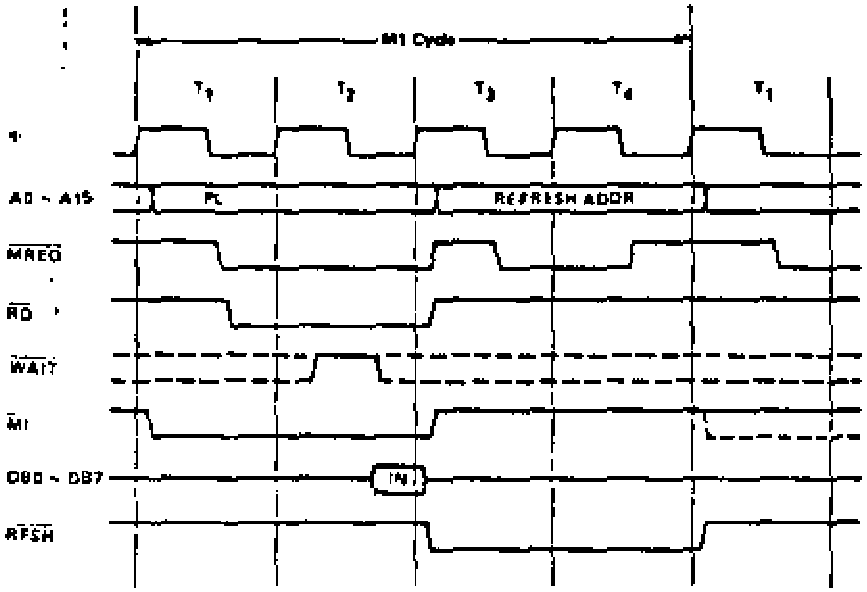
T Período básico de reloj.

M Ciclos de máquina.

$M_1 \ldots M_3$ Ciclos de instrucción.

INSTRUCTION FETCH

Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of $\overline{MREQ}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{RD}$ line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the $\overline{RD}$ and $\overline{MRQ}$ signals. Thus the data has already been sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4 the lower 7 bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that a $\overline{RD}$ signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The $\overline{MREQ}$ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during $\overline{MREQ}$ time.



INSTRUCTION OP CODE FETCH
FIGURE 4.0-1

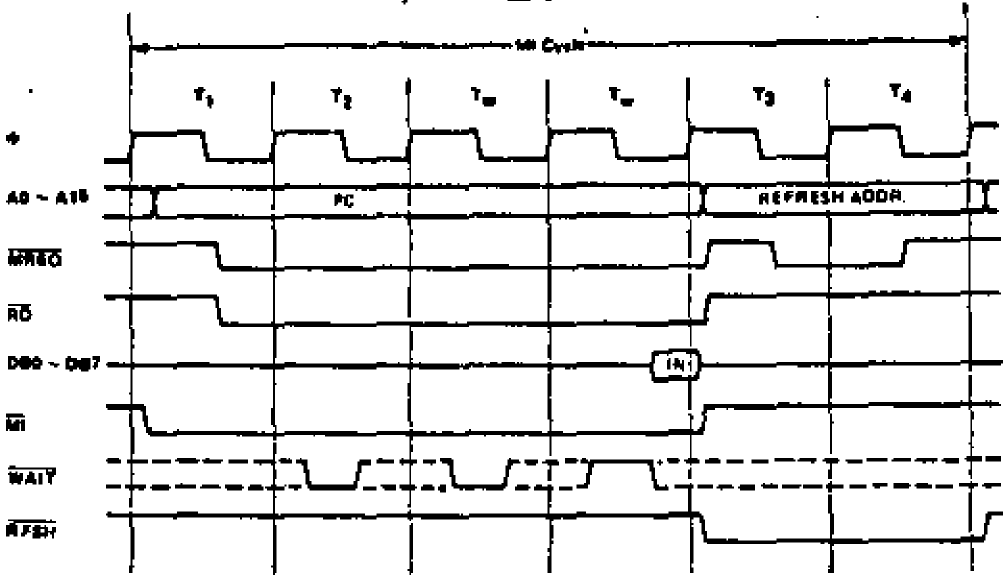
Figure 4.0-1A illustrates how the fetch cycle is delayed if the memory activates the $\overline{WAIT}$ line. During T2 and every subsequent Tw, the CPU samples the $\overline{WAIT}$ line with the falling edge of Φ. If the $\overline{WAIT}$ line is active at this time, another wait state will be entered during the following cycle. Using this technique the read cycle can be lengthened to match the access time of any type of memory device.

Las siguientes páginas resumen la actividad del CPU en función del tiempo para cada operación básica o ciclo básico del Z80.

La explicación será complementada durante la sesión correspondiente del curso y no ahondaremos más sobre este tema en estos apuntes.

Ciclo M_1 Fetch del código de operación.
Ciclos de lectura/escritura a memoria.
Ciclos de entrada/salida.
Ciclo de requerimiento y agotación del bus.
Ciclo de interrupción.
Ciclo de interrupción no enmascarable.
Ciclos durante la instrucción de alto.

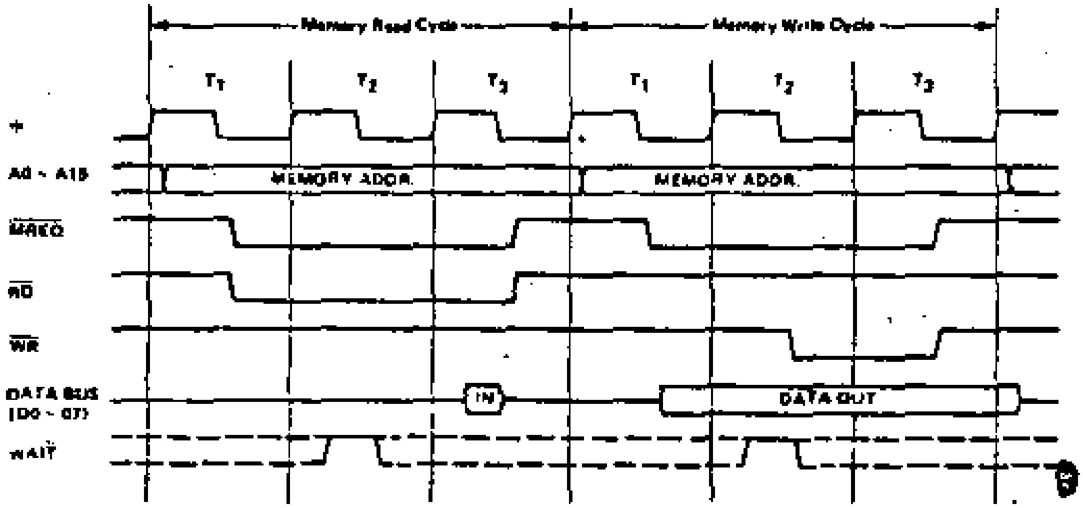
17



INSTRUCTION OP CODE FETCH WITH WAIT STATES
FIGURE 4.0-1A

MEMORY READ OR WRITE

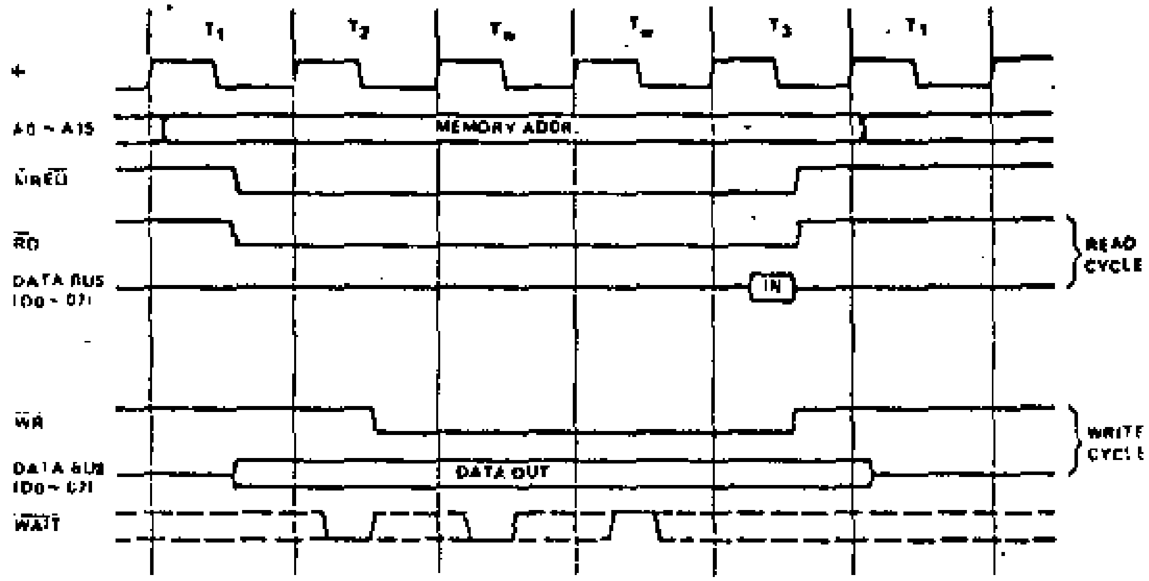
Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the $\overline{WAIT}$ signal. The $\overline{MREQ}$ signal and the $\overline{RD}$ signal are used the same as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the $\overline{WR}$ signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.



MEMORY READ OR WRITE CYCLES
FIGURE 4.0-2

20

Figure 4.0-2A illustrates how a $\overline{\text{WAIT}}$ request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.



MEMORY READ OR WRITE CYCLES WITH WAIT STATES
FIGURE 4.0-2A

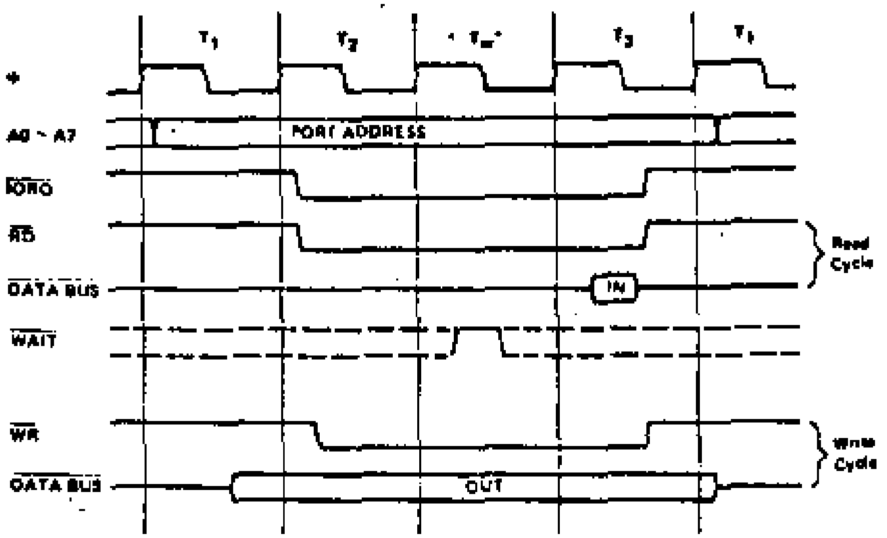
INPUT OR OUTPUT CYCLES

Figure 4.0-3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the IORQ signal goes active until the CPU must sample the $\overline{\text{WAIT}}$ line is very short and without this extra state sufficient time does not exist for an I/O port to decode its address and activate the $\overline{\text{WAIT}}$ line if a wait is required. Also, without this wait state it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time the $\overline{\text{WAIT}}$ request signal is sampled. During a read I/O operation, the $\overline{\text{RD}}$ line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the $\overline{\text{WR}}$ line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

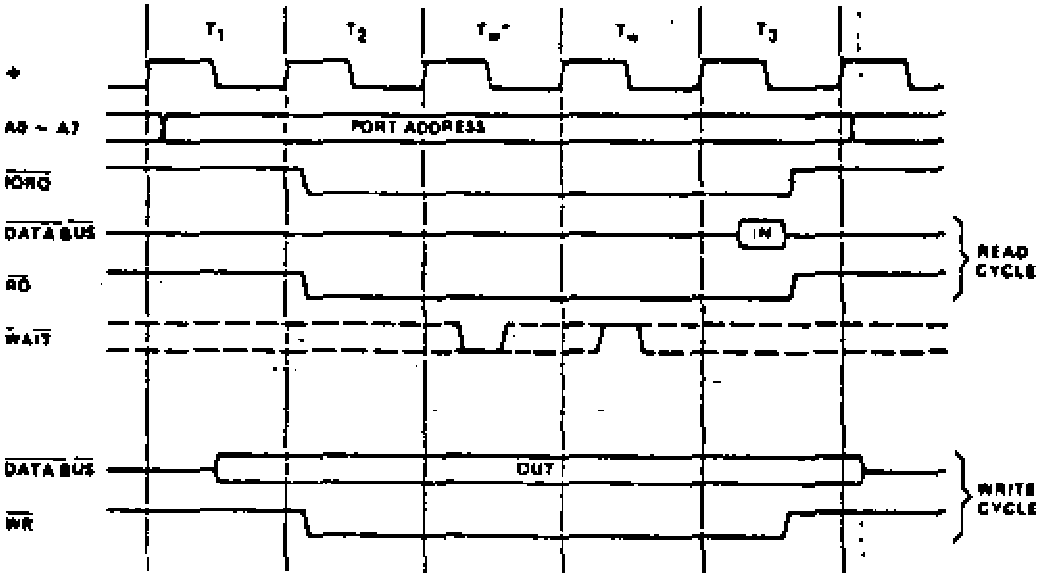
Figure 4.0-3A illustrates how additional wait states may be added with the $\overline{\text{WAIT}}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

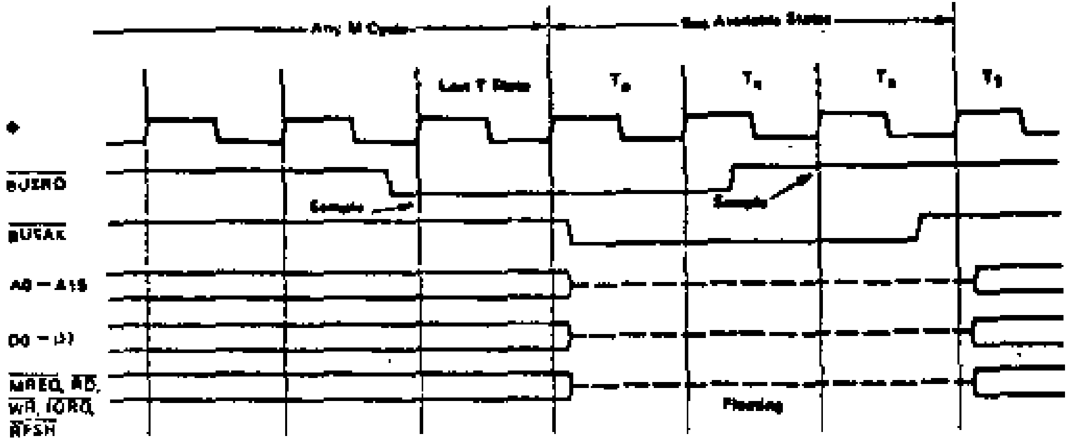
Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{\text{BUSRQ}}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{\text{BUSRQ}}$ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either a $\overline{\text{NMI}}$ or an $\overline{\text{INT}}$ signal.



INPUT OR OUTPUT CYCLES
FIGURE 4.0-3



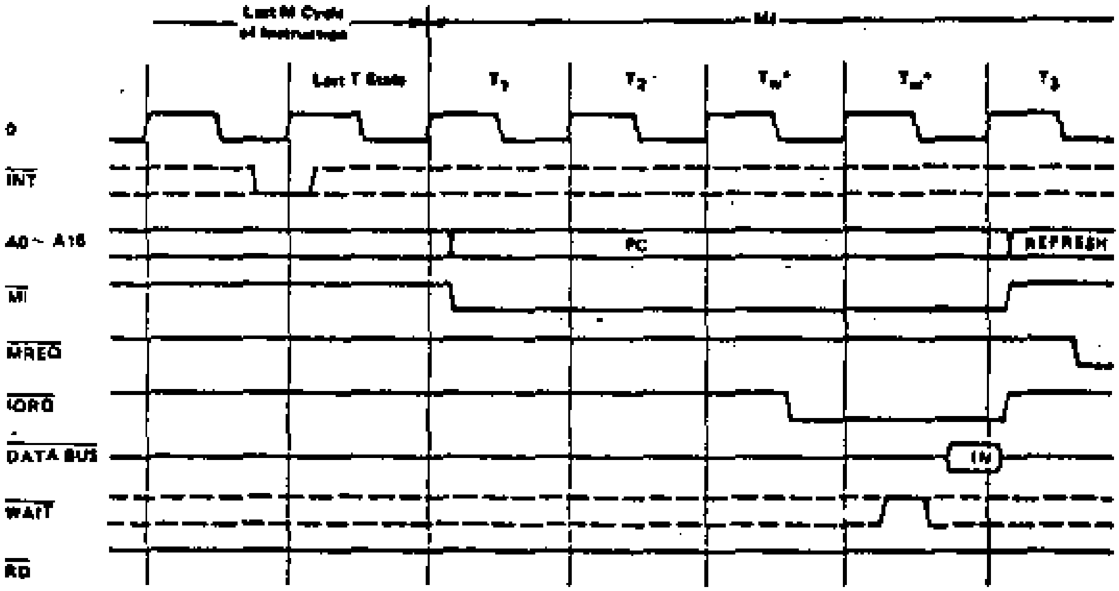
INPUT OR OUTPUT CYCLES WITH WAIT STATES
FIGURE 4.0-3A



BUS REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4.0-4

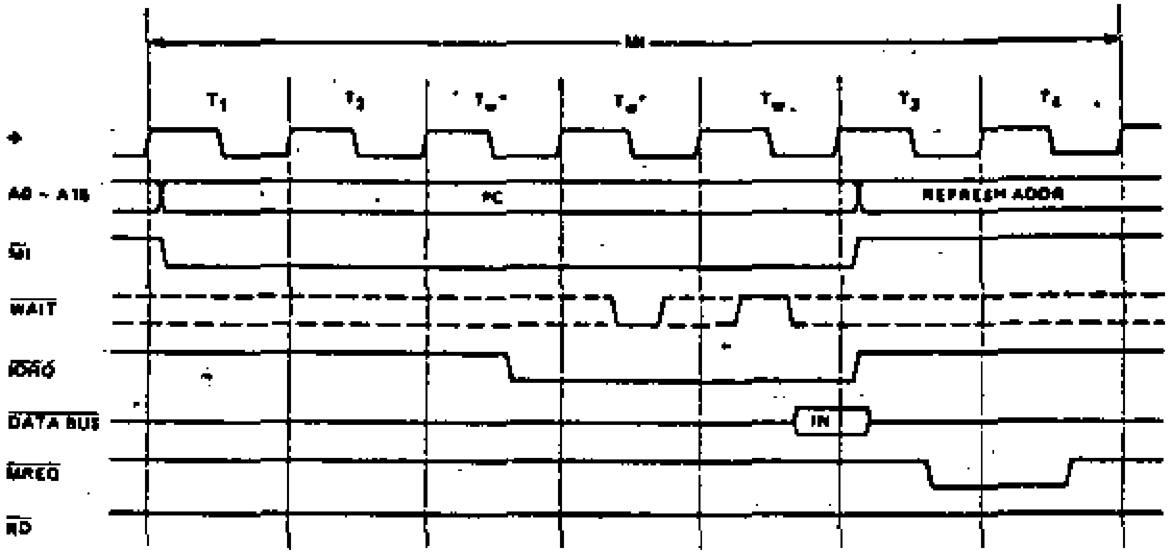
INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSRQ}$ signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the $\overline{IORQ}$ signal becomes active (instead of the normal $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector. Refer to section 8.0 for details on how the interrupt response vector is utilized by the CPU.



INTERRUPT REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4.0-5

Figure 4.0-5A illustrates how additional wait states can be added to the interrupt response cycle. Again the operation is identical to that previously described.



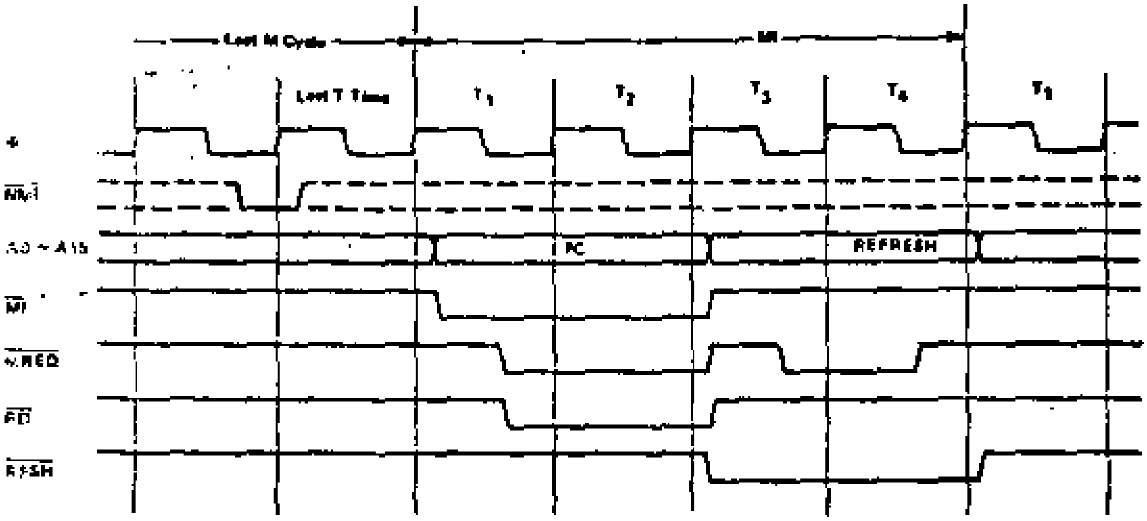
INTERRUPT REQUEST/ACKNOWLEDGE WITH WAIT STATES
FIGURE 4.0-5A

NON MASKABLE INTERRUPT RESPONSE

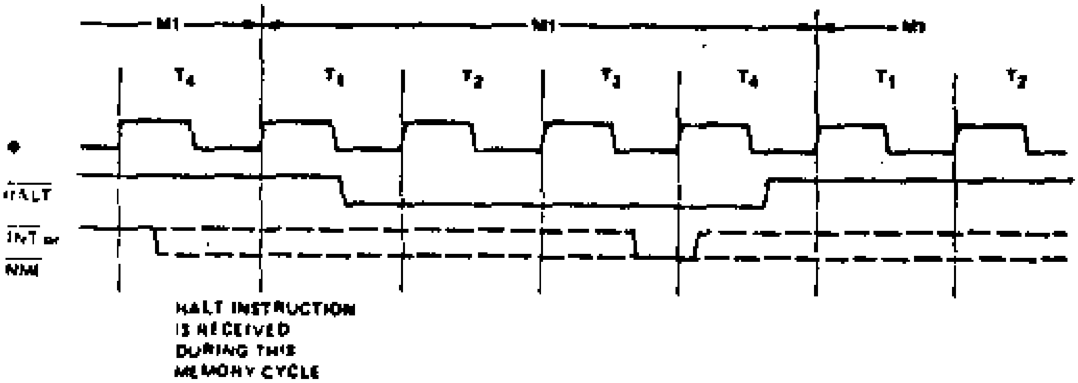
Figure 4.0-6 illustrates the request/acknowledge cycle for the non maskable interrupt. This signal is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

Whenever a software halt instruction is executed the CPU begins executing NOP's until an interrupt is received (either a non maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in figure 4.0-7. If a non maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it has highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.



NON MASKABLE INTERRUPT REQUEST OPERATION
FIGURE 4.0-6



HALT EXIT
FIGURE 4.0-7

Características Eléctricas.

11.0 ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.1W

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

- **D.C. CHARACTERISTICS**

T_A = 0°C to 70°C, V_{cc} = 5V ± 5% unless otherwise specified

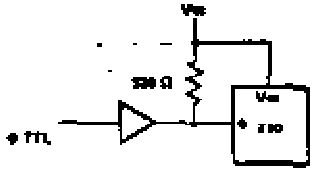
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{cc}[1]		V_{cc}	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{cc}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH}= -100µA
I_{CC}	Power Supply Current			200	mA	t_c= 400nsec
I_{LI}	Input Leakage Current			10	µA	V_{IN}=0 to V_{cc}
I_{LOH}	Tri-State Output Leakage Current in Float			10	µA	V_{OUT}=2.4 to V_{cc}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	µA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	µA	$0 \le V_{IN} \le V_{cc}$

- **CAPACITANCE**

T_A = 25°C, f = 1 MHz

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
C_{ϕ}	Clock Capacitance		35	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance		5	pF	
C_{OUT}	Output Capacitance		10	pF	

[1] Clock Driver



An external clock pull-up resistor of (330Ω) will meet both the A.C. and D.C. clock requirements.

24

A.C. CHARACTERISTICS

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min.	Max.	Unit	Test Condition
Φ	t_c $t_w(\Phi H)$ $t_w(\Phi L)$ $t_{r,f}$	Clock Period Clock Pulse Width, Clock High Clock Pulse Width, Clock Low Clock Rise and Fall Time	.4 180 180 	[12] 30	μsec nsec nsec nsec	
A_{0-15}	$t_{D(AD)}$ $t_{F(AD)}$ t_{acm} t_{aci} t_{ca} t_{caf}	Address Output Delay Delay to Float Address Stable Prior to $\overline{MREQ}$ (Memory Cycle) Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle) Address Stable From $\overline{RD}$ or $\overline{WR}$ Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	 [1] [2] [3] [4]	200 100 	nsec nsec nsec nsec nsec nsec	C_L=100pF
D_{0-7}	$t_{D(D)}$ $t_{F(D)}$ $t_{S\Phi(D)}$ $t_{S\overline{\Phi}(D)}$ t_{dcm} t_{dci} t_{cdf}	Data Output Delay Delay to Float During Write Cycle Data Setup Time to Rising Edge of Clock During M1 Cycle Data Setup Time to Falling Edge of Clock During M2 to M5 Data Stable Prior to $\overline{WR}$ (Memory Cycle) Data Stable Prior to $\overline{WR}$ (I/O Cycle) Data Stable From $\overline{WR}$	 100 100 [5] [6] [7]	350 100 	nsec nsec nsec nsec nsec nsec 	C_L=100pF
	t_H	Any Hold Time for Setup Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$ $t_{DH\Phi(MR)}$ $t_{DH\overline{\Phi}(MR)}$ $t_{w(\overline{MRL})}$ $t_{w(\overline{MRH})}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low $\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High $\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High Pulse Width, $\overline{MREQ}$ Low Pulse Width, $\overline{MREQ}$ High	 [8] [9]	130 130 130 	nsec nsec nsec nsec nsec	C_L=50pF
$\overline{IORQ}$	$t_{DL\Phi(IR)}$ $t_{DL\overline{\Phi}(IR)}$ $t_{DH\Phi(IR)}$ $t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low $\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low $\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High $\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		110 130 130 150	nsec nsec nsec nsec	C_L=50pF
$\overline{RD}$	$t_{DL\Phi(RD)}$ $t_{DL\overline{\Phi}(RD)}$ $t_{DH\Phi(RD)}$ $t_{DH\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low $\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low $\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High $\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		130 150 130 150	nsec nsec nsec nsec	C_L=50pF
$\overline{WR}$	$t_{DL\Phi(WR)}$ $t_{DL\overline{\Phi}(WR)}$ $t_{DH\Phi(WR)}$ $t_{w(\overline{WRL})}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low $\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low $\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High Pulse Width, $\overline{WR}$ Low	 [10]	110 130 130 	nsec nsec nsec nsec	C_L=50pF
$\overline{M1}$	$t_{DL(M1)}$ $t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ Low $\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		180 250	nsec nsec	C_L=30pF
$\overline{RFSH}$	$t_{DL(RF)}$ $t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low $\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		300 300	nsec nsec	C_L=30pF
$\overline{WAIT}$	$t_{s(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L=50pF
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	$t_{w(\overline{NML})}$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_{s(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{BUSAK}$	$t_{DL(BA)}$ $t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low $\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		150 150	nsec nsec	C_L=50pF
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	70		nsec	
	$t_{F(C)}$	Delay to Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		100	nsec	

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 130$

[2] $t_{aci} = t_c - 140$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 80$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 100$

[5] $t_{dcm} = t_c - 200$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 330$

[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 80$

[8] $t_{w(\overline{MRL})} = t_c - 80$

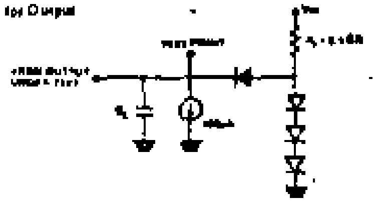
[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_f - 70$

[10] $t_{w(\overline{WR})} = t_c - 80$

NOTES

1. Data should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and $\overline{IORQ}$ are both active.

2. Load circuit for Output

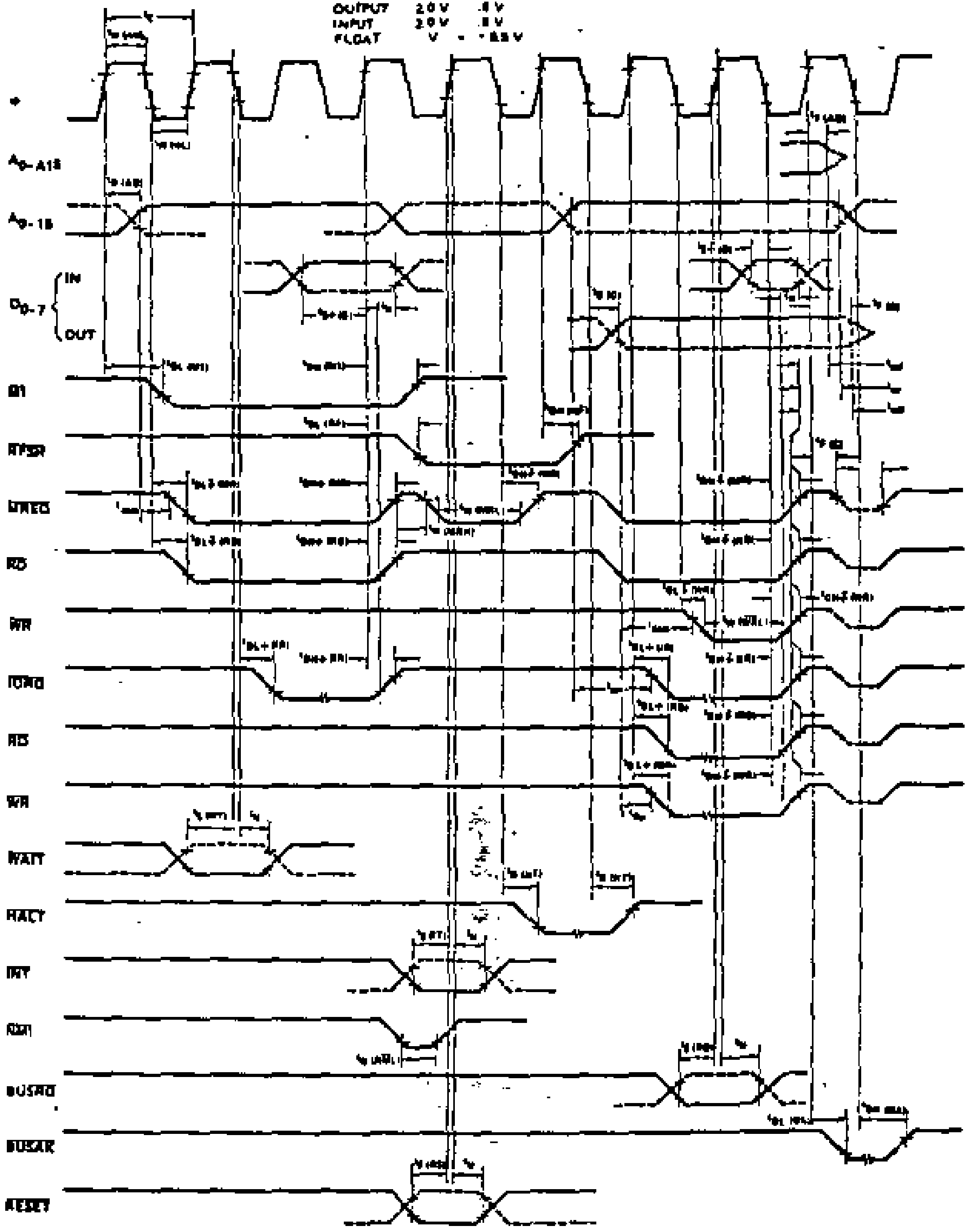


3. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

4. The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	4.2 V	.8 V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	ΔV	±0.5 V



DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

UNIDAD DE COMUNICACION PARALELA

Julio, 1981

01

MOSTEK

Z80 MICROCOMPUTER DEVICES

Technical Manual

MK 3881 PARALLEL I/O CONTROLLER

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE NO. TITLE PAGE NO.

LIST OF TABLES

TABLE NO. TITLE PAGE NO.

1.0 INTRODUCTION

The Z80 Parallel I/O Circuit is a programmable, two port device which provides a TTL compatible interface between peripheral devices and the Z80-CPU. The CPU can configure the Z80-PIO to interface with a wide range of peripheral devices with no other external logic required. Typical peripheral devices that are fully compatible with the Z80-PIO include most keyboards, paper tape readers and punches, printers, PROM programmers, etc. The Z80-PIO utilizes N channel silicon gate depletion load technology and is packaged in a 40 pin DIP. Major features of the Z80-PIO include:

- Two independent 8 bit bidirectional peripheral interface ports with 'handshake' data transfer control
- Interrupt driven 'handshake' for fast response
- Any one of four distinct modes of operation may be selected for a port including:

  Byte output
  Byte input
  Byte bidirectional bus (Available on Port A only)
  Bit control mode
  All with interrupt controlled handshake
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic
- Eight outputs are capable of driving Darlington transistors
- All inputs and outputs fully TTL compatible
- Single 5 volt supply and single phase clock required.

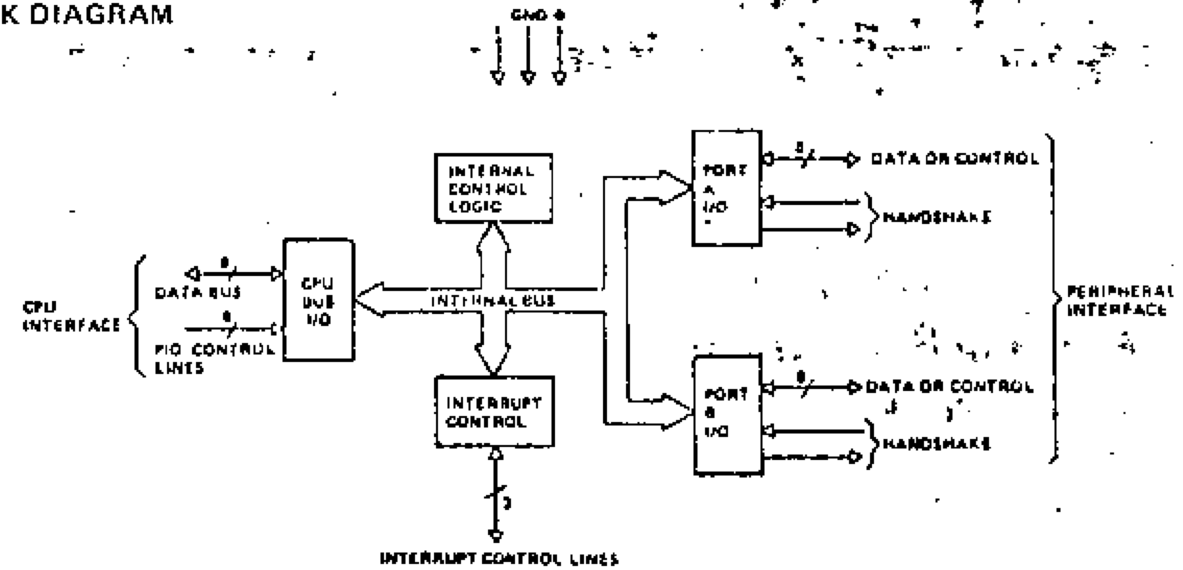
One of the unique features of the Z80-PIO that separates it from other interface controllers is that all data transfer between the peripheral device and the CPU is accomplished under total interrupt control. The interrupt logic of the PIO permits full usage of the efficient interrupt capabilities of the Z80 CPU during I/O transfers. All logic necessary to implement a fully nested interrupt structure is included in the PIO so that additional circuits are not required. Another unique feature of the PIO is that it can be programmed to interrupt the CPU on the occurrence of specified status conditions in the peripheral device. For example, the PIO can be programmed to interrupt if any specified peripheral alarm conditions should occur. This interrupt capability reduces the amount of time that the processor must spend in polling peripheral status.

05

2.0 PIO ARCHITECTURE

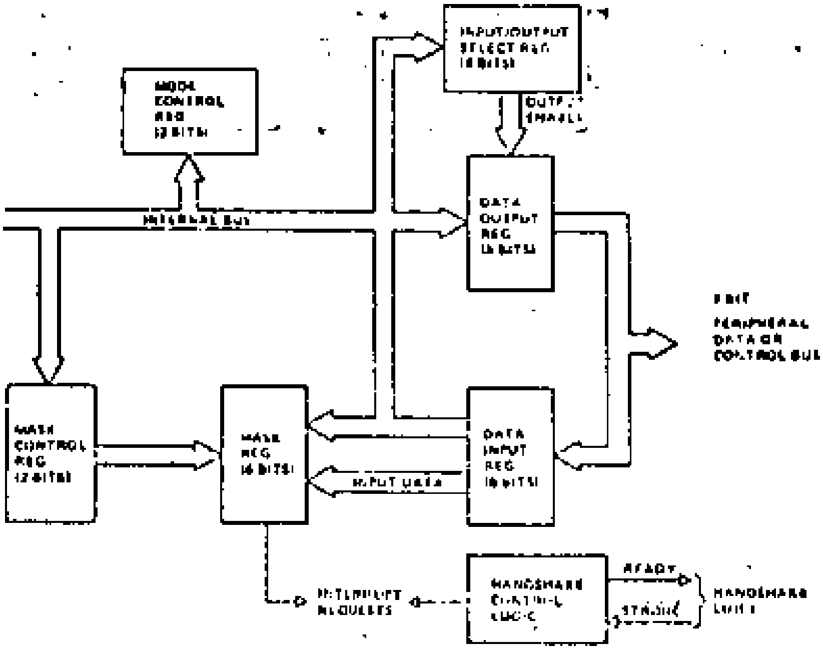
A block diagram of the Z80 PIO is shown in figure 2.0-1. The internal structure of the Z80-PIO consists of a Z80 CPU bus interface, internal control logic, Port A I/O logic, Port B I/O logic, and interrupt control logic. The CPU bus interface logic allows the PIO to interface directly to the Z80-CPU with no other external logic. However, address decoders and/or line buffers may be required for large systems. The internal control logic synchronizes the CPU data bus to the peripheral device interfaces (Port A and Port B). The two I/O ports (A and B are virtually identical and are used to interface directly to peripheral devices.

PIO BLOCK DIAGRAM
Figure 2.0-1



The Port I/O logic is composed of 6 registers with "handshake" control logic as shown in figure 2.0-2. The registers include: an 8 bit data input register, an 8 bit data output register, a 2 bit mode control register, an 8 bit mask register, an 8 bit input/output select register, and a 2 bit mask control register.

PORT I/O BLOCK DIAGRAM
Figure 2.0-2



The 2-bit mode control register is loaded by the CPU to select the desired operating mode (byte output, byte input, byte bidirectional bus, or bit control mode). All data transfer between the peripheral device and the CPU is achieved through the data input and data output registers. Data may be written into the output register by the CPU or read back to the CPU from the input register at any time. The handshake lines associated with each port are used to control the data transfer between the PIO and the peripheral device.

The 8-bit mask register and the 8-bit input/output select register are used only in the bit control mode. In this mode any of the 8 peripheral data or control bus pins can be programmed to be an input or an output as specified by the select register. The mask register is used in this mode in conjunction with a special interrupt feature. This feature allows an interrupt to be generated when any or all of the unmasked pins reach a specified state (either high or low). The 2-bit mask control register specifies the active state desired (high or low) and if the interrupt should be generated when all unmasked pins are active (AND condition) or when any unmasked pin is active (OR condition). This feature reduces the requirement for CPU status checking of the peripheral by allowing an interrupt to be automatically generated on specific peripheral status conditions. For example, in a system with 3 alarm conditions, an interrupt may be generated if any one occurs or if all three occur.

The interrupt control logic section handles all CPU interrupt protocol for nested priority interrupt structures. The priority of any device is determined by its physical location in a daisy chain configuration. Two lines are provided in each PIO to form this daisy chain. The device closest to the CPU has the highest priority. Within a PIO, Port A interrupts have higher priority than those of Port B. In the byte input, byte output or bidirectional modes, an interrupt can be generated whenever a new byte transfer is requested by the peripheral. In the bit control mode an interrupt can be generated when the peripheral status matches a programmed value. The PIO provides for complete control of nested interrupts. That is, lower priority devices may not interrupt higher priority devices that have not had their interrupt service routine completed by the CPU. Higher priority devices may interrupt the servicing of lower priority devices.

When an interrupt is accepted by the CPU in mode 2, the interrupting device must provide an 8-bit interrupt vector for the CPU. This vector is used to form a pointer to a location in the computer memory where the address of the interrupt service routine is located. The 8-bit vector from the interrupting device forms the least significant 8 bits of the indirect pointer while the I Register in the CPU provides the most significant 8 bits of the pointer. Each port (A and B) has an independent interrupt vector. The least significant bit of the vector is automatically set to a 0 within the PIO since the pointer must point to two adjacent memory locations for a complete 16-bit address.

The PIO decodes the RETI (Return from interrupt) instruction directly from the CPU data bus so that each PIO in the system knows at all times whether it is being serviced by the CPU interrupt service routine without any other communication with the CPU.

3.0 PIN DESCRIPTION

A diagram of the Z80-PIO pin configuration is shown in figure 3.0-1. This section describes the function of each pin.

D_7-D_0

Z80 CPU Data Bus (bidirectional, tristate)
This bus is used to transfer all data and commands between the Z80-CPU and the Z80-PIO. D_0 is the least significant bit of the bus.

B/A Sel

Port B or A Select (input, active high)
This pin defines which port will be accessed during a data transfer between the Z80-CPU and the Z80-PIO. A low level on this pin selects Port A while a high level selects Port B. Often Address bit A_0 from the CPU will be used for this selection function.

C/D Sel

Control or Data Select (input, active high)
This pin defines the type of data transfer to be performed bwtween the CPU and the PIO. A high level on this pin during a CPU write to the PIO causes the Z80 data bus to be interpreted as a command for the port selected by the B/A Select line. A low level on this pin means that the Z80 data bus is being used to transfer data between the CPU and the PIO. Often Address bit A_1 from the CPU will be used for this function.

$\overline{CE}$

Chip Enable (input, active low)
A low level on this pin enables the PIO to accept command or data inputs from the CPU during a write cycle or to transmit data to the CPU during a read cycle. This signal is generally a decode of four I/O port numbers that encompass port A and B, data and control.

Φ

System Clock(input)
The Z80-PIO uses the standard Z80 system clock to synchronize certain signals internally. This is a single phase clock.

$\overline{M1}$

Machine Cycle One Signal from CPU (input, active low)
This signal from the CPU is used as a sync pulse to control several internal PIO operations. When M1 is active and the RD signal is active, the Z80-CPU is fetching an instruction from memory. Conversely, when M1 is active and IORQ is active, the CPU is acknowledging an interrupt. In addition, the M1 signal has two other functions within the Z80-PIO.

1. M1 synchronizes the PIO interrupt logic.

2. When M1 occurs without an active RD or IORQ signal the PIO logic enters a reset state.

$\overline{IORQ}$

Input/Output Request from Z80-CPU (input, active low)
The $\overline{IORQ}$ signal is used in conjunction with the B/A Select, C/D Select, $\overline{CE}$, and $\overline{RD}$ signals to transfer commands and data between the Z80 CPU and the Z80-PIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are active, the port addressed by B/A will transfer data to the CPU (a read operation). Conversely, when $\overline{CE}$ and $\overline{IORQ}$ are active but $\overline{RD}$ is not active, then the port addressed by B/A will be written into from the CPU with either data or control information as specified by the C/D Select signal. Also, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the interrupting port will automatically place its interrupt vector on the CPU data bus if it is the highest device requesting an interrupt.

$\overline{RD}$

Read Cycle Status from the Z80-CPU (input, active low)
If $\overline{RD}$ is active a MEMORY READ or I/O READ operation is in progress. The $\overline{RD}$ signal is used with B/A Select, C/D Select, $\overline{CE}$ and $\overline{IORQ}$ signals to transfer data from the Z80-PIO to the Z80-CPU.

IEI

Interrupt Enable In (input, active high)
This signal is used to form a priority interrupt daisy chain when more than one interrupt driven device is being used. A high level on this pin indicates that no other devices of higher priority are being serviced by a CPU interrupt service routine.

IEO

Interrupt Enable Out (output, active high)
The IEO signal is the other signal required to form a daisy chain priority scheme. It is high only if IEI is high and the CPU is not servicing an interrupt from this PIO. Thus this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$

Interrupt Request (output, open drain, active low)
When INT is active the Z80-PIO is requesting an interrupt from the Z80-CPU.

A_0-A_7

Port A Bus (bidirectional, tri-state)
This 8 bit bus is used to transfer data and/or status or control information between Port A of the Z80-PIO and a peripheral device. A_0 is the least significant bit of the Port A data bus.

$\overline{A\ STB}$

Port A Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: The positive edge of this strobe is issued by the peripheral to acknowledge the receipt of data made available by the PIO.

2) Input mode: The strobe is issued by the peripheral to load data from the peripheral into the Port A input register. Data is loaded into the PIO when this signal is active.

3) Bidirectional mode: When this signal is active, data from the Port A output register is gated onto Port A bidirectional data bus. The positive edge of the strobe acknowledges the receipt of the data.

4) Control mode: The strobe is inhibited internally.

A RDY

Register A Ready (output, active high)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: This signal goes active to indicate that the Port A output register has been loaded and the peripheral data bus is stable and ready for transfer to the peripheral device.

2) Input mode: This signal is active when the Port A input register is empty and is ready to accept data from the peripheral device.

3) Bidirectional mode: This signal is active when data is available in Port A output register for transfer to the peripheral device. In this mode data is not placed on the Port A data bus unless $\overline{A\ STB}$ is active.

In the control mode this signal is disabled and forced to a low state.

B_0-B_7 Port B Bus (Bidirectional, tristate)
This 8 bit bus is used to transfer data and/or status or control information between Port B of the PIO and a peripheral device. The Port B data bus is capable of supplying 1.5ma @ 1.5V to drive Darlington transistors. B_0 is the least significant bit of the bus.

$\overline{\text{B STB}}$ Port B Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal is similar to that of $\overline{\text{A STB}}$ with the following exception:

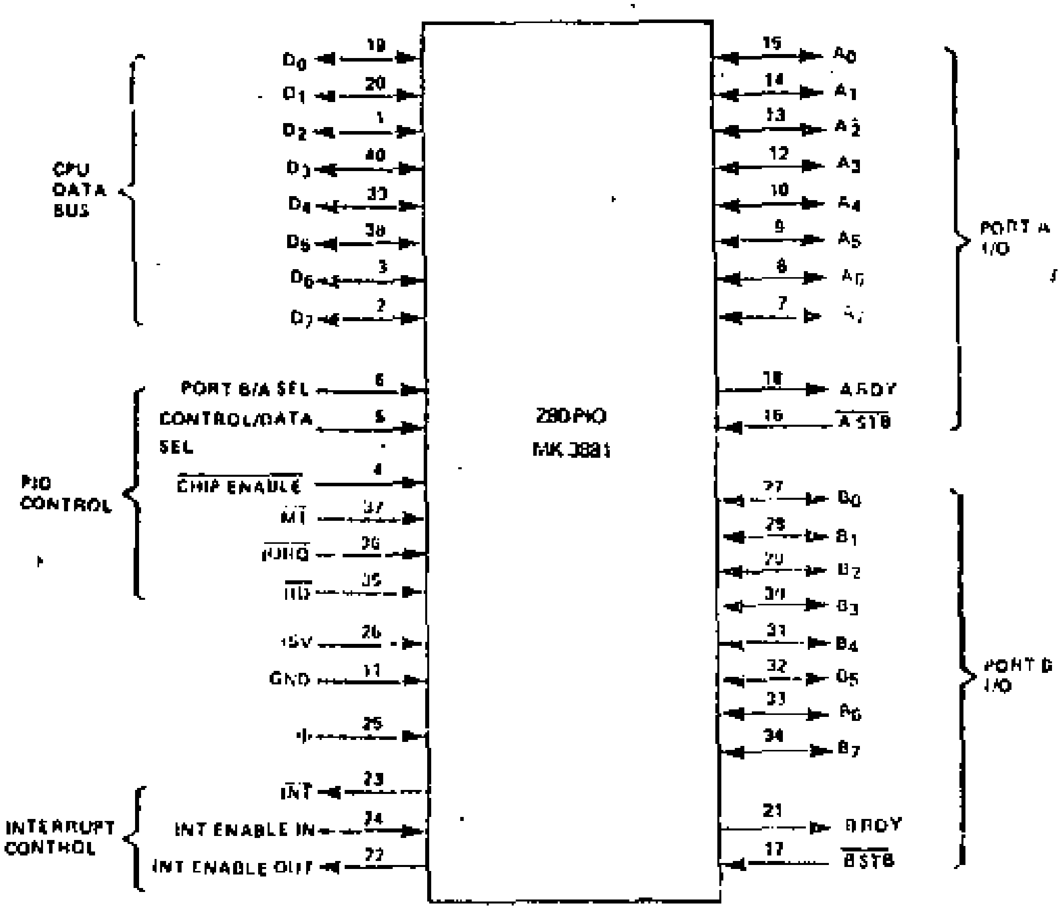
In the Port A bidirectional mode this signal strobes data from the peripheral device into the Port A input register.

B RDY Register B Ready (output, active high)
The meaning of this signal is similar to that of A Ready with the following exception:

In the Port A bidirectional mode this signal is high when the Port A input register is empty and ready to accept data from the peripheral device.

PIO PIN CONFIGURATION

Figure 3.0-1



4.0 PROGRAMMING THE PIO

4.1 RESET

The Z80-PIO automatically enters a reset state when power is applied. The reset state performs the following functions:

1) Both port mask registers are reset to inhibit all port data bits.

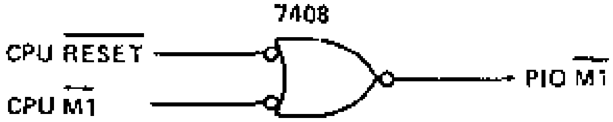
2) Port data bus lines are set to a high impedance state and the Ready "handshake" signals are inactive (low). Mode 1 is automatically selected.

3) The vector address registers are not reset.

4) Both port interrupt enable flip flops are reset.

5) Both port output registers are reset.

In addition to the automatic power on reset, the PIO can be reset by applying an $\overline{M1}$ signal without the presence of a $\overline{RD}$ or $\overline{IORQ}$ signal. If no $\overline{RD}$ or $\overline{IORQ}$ is detected during $\overline{M1}$ the PIO will enter the reset state immediately after the $\overline{M1}$ signal goes inactive. The purpose of this reset is to allow a single external gate to generate a reset without a power down sequence. This approach was required due to the 40 pin packaging limitation. It is recommended that in breadboard systems and final systems with a "Reset" push button that a $\overline{M1}$ reset be implemented for the PIO.



A software RESET is possible as described in Section 4.4, however, use of this method during early system debug may not be desirable because of non-functional system hardware (bus buffers or memory for example).

Once the PIO has entered the internal reset state it is held there until the PIO receives a control word from the CPU.

4.2 LOADING THE INTERRUPT VECTOR

The PIO has been designed to operate with the Z80 CPU using the mode 2 interrupt response. This mode requires that an interrupt vector be supplied by the interrupting device. This vector is used by the CPU to form the address for the interrupt service routine of that port. This vector is placed on the Z80 data bus during an interrupt acknowledge cycle by the highest priority device requesting service at that time. (Refer to the Z80-CPU Technical Manual for details on how an interrupt is serviced by the CPU). The desired interrupt vector is loaded into the PIO by writing a control word to the desired port of the PIO with the following format:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

signifies this control word is an interrupt vector

D0 is used in this case as a flag bit which when low causes V7 thru V1 to be loaded into the vector register. At interrupt acknowledge time, the vector of the interrupting port will appear on the Z80 data bus exactly as shown in the format above.

4.3 SELECTING AN OPERATING MODE

Port A of the PIO may be operated in any of four distinct modes: Mode 0 (output mode), Mode 1 (input mode), Mode 2 (bidirectional mode), and Mode 3 (control mode). Note that the mode numbers have been selected for mnemonic significance; i.e. 0=Out, 1=In, 2=Bidirectional. Port B can operate in any of these modes except Mode 2.

The mode of operation must be established by writing a control word to the PIO in the following format:

D7	D6	D5	D4	D3	D2	D1	D0
M1	M0	X	X	1	1	1	1

X=unused bit

mode word

signifies mode word to be set

Bits D7 and D6 from the binary code for the desired mode according to the following table:

D7	D6	MODE
0	0	0 (output)
0	1	1 (input)
1	0	2 (bidirectional)
1	1	3 (control)

Bits D5 and D4 are ignored. Bits D3-D0 must be set to 1111 to indicate "Set Mode".

Selecting Mode 0 enables any data written to the port output register by the CPU to be enabled onto the port data bus. The contents of the output register may be changed at any time by the CPU simply by writing a new data word to the port. Also the current contents of the output register may be read back to the Z80 CPU at any time through the execution of an input instruction.

With Mode 0 active, a data write from the CPU causes the Ready handshake line of that port to go high to notify the peripheral that data is available. This signal remains high until a strobe is received from the peripheral. The rising edge of the strobe generates an interrupt (if it has been enabled) and causes the Ready line to go inactive. This very simple handshake is similar to that used in many peripheral devices.

Selecting Mode 1 puts the port into the input mode. To start handshake operation, the CPU merely performs an input read operation from the port. This activates the Ready line to the peripheral to signify that data should be loaded into the empty input register. The peripheral device then strobes data into the port input register using the strobe line. Again, the rising edge of the strobe causes an interrupt request (if it has been enabled) and deactivates the Ready signal. Data may be strobed into the input register regardless of the state of the Ready signal if care is taken to prevent a data overrun condition.

Mode 2 is a bidirectional data transfer mode which uses all four handshake lines. Therefore only Port A may be used for Mode 2 operation. Mode 2 operation uses the Port A handshake signals for output control and the Port B handshake signals for input control. Thus, both A RDY and B RDY may be active simultaneously. The only operational difference between Mode 0 and the output portion of Mode 2 is that data from the Port A output register is allowed on to the port data bus only when $\overline{\text{A STB}}$ is active in order to achieve a bidirectional capability.

Mode 3 operation is intended for status and control applications and does not utilize the handshake signals. When Mode 3 is selected, the next control word sent to the PIO must define which of the port data bus lines are to be inputs and which are outputs. The format of the control word is shown below:

D7	D6	D5	D4	D3	D2	D1	D0
I/O$_7$	I/O$_6$	IO/$_5$	I/O$_4$	I/O$_3$	I/O$_2$	I/O$_1$	I/O$_0$

If any bit is set to a one, then the corresponding data bus line will be used as an input. Conversely, if the bit is reset, the line will be used as an output.

During Mode 3 operation the strobe signal is ignored and the Ready line is held low. Data may be written to a port or read from a port by the Z80-CPU at any time during Mode 3 operation. (An exception to this is when Port A is in Mode 2 and Port B is in Mode 3). When reading a port, the data returned to the CPU will be composed of input data from port data bus lines assigned as inputs plus port output register data from those lines assigned as outputs.

4.4 SETTING THE INTERRUPT CONTROL WORD

The interrupt control word for each port has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Enable Interrupt	AND/OR	High/Low	Masks follows	0	1	1	1

D6–D4: used in Mode 3 only

D3–D0: signifies interrupt control word

If bit D7=1 the interrupt enable flip flop of the port is set and the port may generate an interrupt. If bit D7=0 the enable flag is reset and interrupts may not be generated. If an interrupt occurs while D7=0, it will be latched internally by the PIO and passed onto the CPU when PIO Interrupts are Re-Enabled (D7=1). Bits D6, D5 and D4 are used mainly with Mode 3 operation, however, setting bit D4 of the interrupt control word during any mode of operation will cause a pending interrupt to be reset. These three bits are used to allow for interrupt operation in Mode 3 when any group of the I/O lines go to certain defined states. Bit D6 (AND/OR) defines the logical operation to be performed in port monitoring. If bit D6=1, and AND function is specified and if D6=0, an OR function is specified. For example, if the AND function is specified, all bits must go to a specified state before an interrupt will be generated while the OR function will generate an interrupt if any specified bit goes to the active state.

Bit D5 defines the active polarity of the port data bus line to be monitored. If bit D5=1 the port data lines are monitored for a high state while if D5=0 they will be monitored for a low state.

If bit D4=1 the next control word sent to the PIO must define a mask as follows:

D7	D6	D5	D4	D3	D2	D1	D0
MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

Only those port lines whose mask bit is zero will be monitored for generating an interrupt.

The interrupt enable flip flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Int Enable	X	X	X	0	0	1	1

If an external Asynchronous interrupt could occur while the processor is writing the disable word to the PIO (03H) then a system problem may occur. If interrupts are enabled in the processor it is possible that the Asynchronous interrupt will occur while the processor is writing the disable word to the PIO. The PIO will generate an INT and the CPU will acknowledge it, however, by this time, the PIO will have received the disable word and de-activated its interrupt structure. The result is that the PIO will not send in its interrupt vector during the interrupt acknowledge cycle because it is disabled and the CPU will fetch an erroneous vector resulting in a program fault. The cure for this problem is to disable interrupts within the CPU with the DI instruction just before the PIO is disabled and then re-enable interrupts with the EI instruction. This action causes the CPU to ignore any faulty interrupts produced by the PIO while it is being disabled. The code sequence would be:

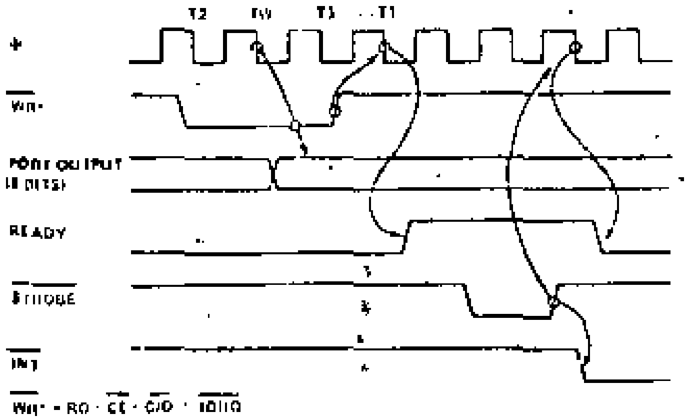
```
LD  A,03H
DI               ; DISABLE CPU
OUT (PIO),A      ; DISABLE PIO
EI               ; ENABLE CPU
```

5.0 TIMING

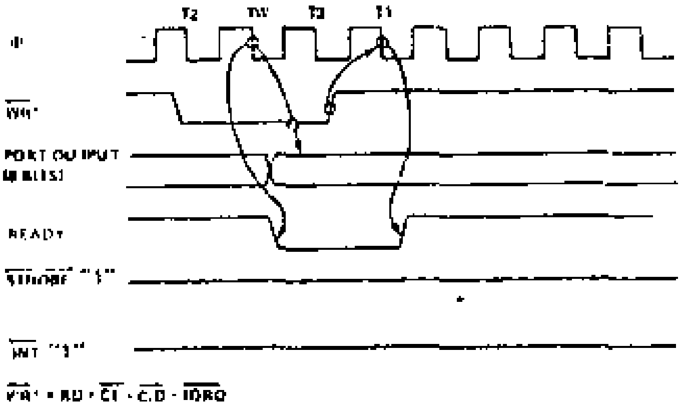
5.1 OUTPUT MODE (MODE 0)

Figure 5.0-1a illustrates the timing associated with Mode 0 operation. An output cycle is always started by the execution of an output instruction by the CPU. A $\overline{\text{WR}}$* pulse is generated by the PIO during a CPU I/O write operation and is used to latch the data from the CPU data bus into addressed port's (A or B) output register. The rising edge of the $\overline{\text{WR}}$* pulse then raises the READY line after the next falling edge of Φ to indicate that data is available for the peripheral device. In most systems, the rising edge of the READY signal can be used as a latching signal in the peripheral device. The READY signal will remain active until a positive edge is received from the $\overline{\text{STROBE}}$ line indicating that the peripheral has taken the data shown in Figure 5.0-1a. If already active, READY will be forced low 1½ Φ cycles after the falling edge of $\overline{\text{IORQ}}$ if the port's output register is written into. READY will return high on the first falling edge of Φ after the rising edge of $\overline{\text{IORQ}}$ as shown in figure 5.0-1b. This action guarantees that READY is low while port data is changing and that a positive edge is generated on READY whenever an Output instruction is executed.

MODE 0 (OUTPUT) TIMING
Figure 5.0-1a


T2
TW
T3
T1
Φ
$\overline{\text{WR}}$*
PORT OUTPUT (8 BITS)
READY
$\overline{\text{STROBE}}$
$\overline{\text{INT}}$
$\overline{\text{WR}}$* = RD · $\overline{\text{CE}}$ · C/D · $\overline{\text{IORQ}}$

MODE 0 (OUTPUT) TIMING
Figure 5.0-1b

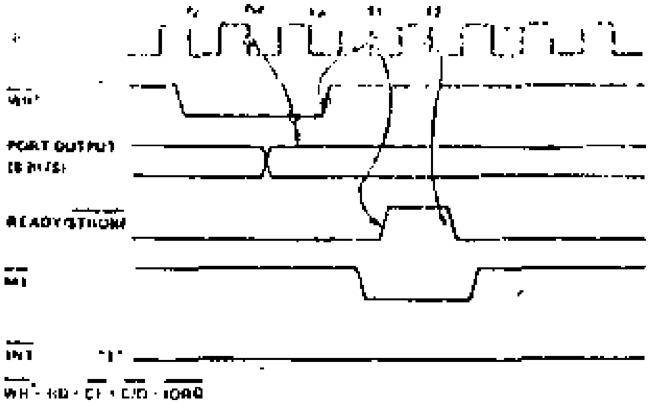

T2
TW
T3
T1
Φ
$\overline{\text{WR}}$*
PORT OUTPUT (8 BITS)
READY
$\overline{\text{STROBE}}$ "1"
$\overline{\text{INT}}$ "1"
$\overline{\text{WR}}$* = RD · $\overline{\text{CE}}$ · C.D · $\overline{\text{IORQ}}$

By connecting READY to $\overline{\text{STROBE}}$ a positive pulse with a duration of one clock period can be created as shown in Figure 5.0-1c. The positive edge of READY/$\overline{\text{STROBE}}$ will not generate an interrupt because the positive portion of $\overline{\text{STROBE}}$ is less than the width of $\overline{\text{M1}}$ and as such will not generate an interrupt due to the internal logic configuration of the PIO.

If the PIO is not in a reset status (i.e. a control mode has been selected), the output register may be loaded before Mode 0 is selected. This allows port output lines to become active in a user defined state. For example, assume the outputs are desired to become active in a logic one state, the following would be the initialization sequence:

a) PIO RESET
b) Load interrupt Vector
c) Select Mode 1 (input) (automatic due ro RESET)
d) Write FF to Data Port
e) Select Mode 0 (Outputs go to "1's")
f) Enable Interrupt if desired

MODE 0 (OUTPUT) TIMING READY TIED TO STROBE
Figure 5.0-1c

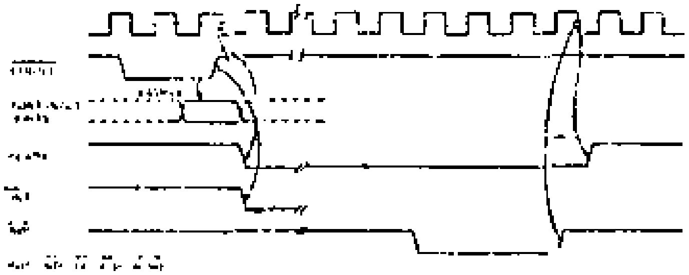


5.2 INPUT MODE (MODE 1)

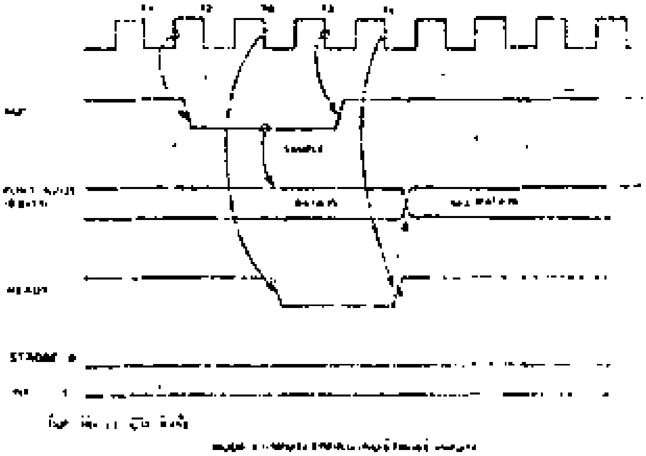
Figure 5.0-2 illustrates the timing of an input cycle. The peripheral initiates this cycle using The $\overline{\text{STROBE}}$ line after the CPU has performed a data read. A low level on this line loads data into the port input register and the rising edge of the $\overline{\text{STROBE}}$ line activates the interrupt request line ($\overline{\text{INT}}$) if the interrupt enable is set and this is the highest priority requesting device. The next falling edge of the clock line (Φ) will then reset the READY line to an inactive state signifying that the input register is full and further loading must be inhibited until the CPU reads the data. The CPU will in the course of its interrupt service routine, read the data from the interrupting port. When this occurs, the positive edge from the CPU $\overline{\text{RD}}$ signal will raise the READY line with the next low going transition of Φ, indicating that new data can be loaded into the PIO.

Since RESET causes READY to go low a dummy Input instruction may be needed in some systems to cause READY to go high the first time in order to start "handshaking".

MODE 1 (INPUT) TIMING
Figure 5.0-2a



MODE 1 (INPUT) TIMING (NO STROBE INPU
Figure 5.0-2b



If already active, READY will be forced low one and one-half Φ periods following the falling edge of $\overline{\text{IORQ}}$ during a read of a PIO port as shown in Figure 5.0-2b. If the user strobes data into the PIO only when READY is high, the forced state of READY will prevent input register data from changing while the CPU is reading the PIO. Ready will go high again after the rising edge of the $\overline{\text{IORQ}}$ as previously described.

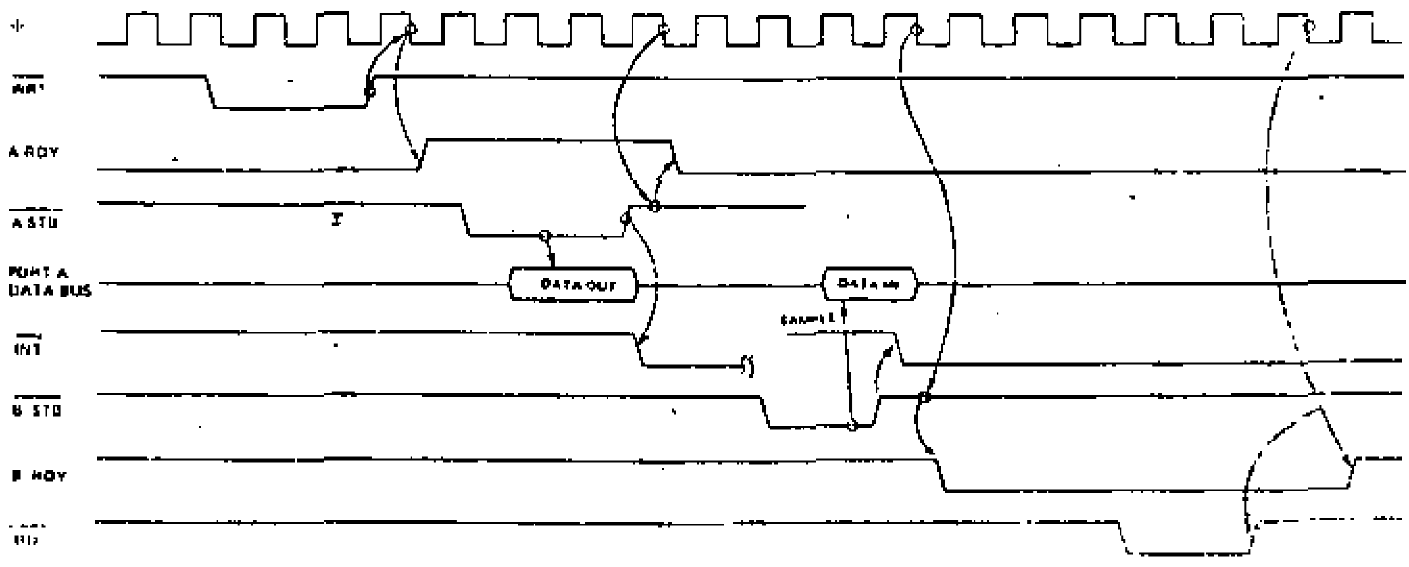
5.3 BIDIRECTIONAL MODE (MODE 2)

This mode is merely a combination of Mode 0 and Mode 1 using all four handshake lines. Since it requires all four lines, it is available only on Port A. When this mode is used on Port A, Port B must be set to the Bit Control Mode. The same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input transfer interrupt during Mode 2 operation of Port A. Ambiguity is avoided if Port B is operated in a polled mode and the Port B mask register is set to inhibit all bits.

Figure 5.0-3 illustrates the timing for this mode. It is almost identical to that previously described for Mode 0 and Mode 1 with the Port A handshake lines used for output control and the Port B lines used for input control. The difference between the two modes is that, in Mode 2, data is allowed out onto the bus only when the A $\overline{\text{STROBE}}$ is low. The rising edge of this strobe can be used to latch the data into the peripheral since the data will remain stable until after this edge. The input portion of Mode 2 operates identically to Mode 1. Note that both Port A and Port B must have their interrupts enabled to achieve an interrupt driven bidirectional transfer.

PORT A, MODE 2 (BIDIRECTIONAL) TIMING

Figure 5.0-3



WR* = RD · CE · C/D · IORQ
RD* = RD · CE · C/D · IORQ

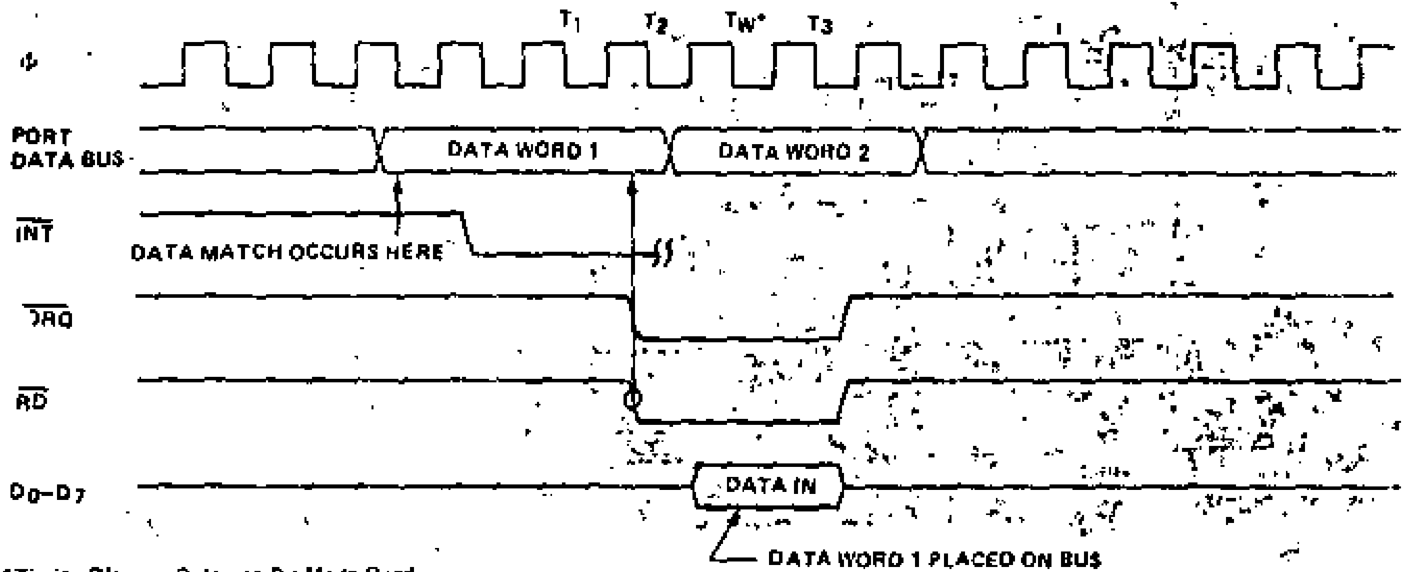
The peripheral must not gate data onto a port data bus while $\overline{\text{A STB}}$ is active. Bus contention is avoided if the peripheral uses $\overline{\text{B STB}}$ to gate input data onto the bus. The PIO uses the $\overline{\text{B STB}}$ low level to sample this data. The PIO has been designed with a zero hold time requirement for the data when latching in this mode so that this simple gating structure can be used by the peripheral. That is, the data can be disabled from the bus immediately after the strobe rising edge. Note that if $\overline{\text{A STB}}$ is low during a read operation of Port A (in response to a $\overline{\text{B STB}}$ interrupt) the data in the output register will be read by the CPU instead of the correct data in the data input register. The correct data is latched in the input register it just cannot be read by the CPU while $\overline{\text{A STB}}$ is low. If the $\overline{\text{A STB}}$ signal could go low during a CPU Read, it should be blocked from reaching the $\overline{\text{A STB}}$ input of the PIO while BRDY is low (the CPU read will occur while BRDY is low as the $\overline{\text{RD}}$ signal returns BRDY high).

5.4 CONTROL MODE (MODE 3)

17 The control mode does not utilize the handshake signals and a normal port write or port read can be executed at any time. When writing, the data will be latched into output registers with the same timing as Mode 0. A RDY will be forced low whenever Port A is operated in Mode 3. B RDY will be held low whenever Port B is operated in Mode 3 unless Port A is in Mode 2. In the latter case, the state of B RDY will not be affected.

When reading the PIO, the data returned to the CPU will be composed of output register data from those port data lines assigned as outputs and input register data from those port data lines assigned as inputs. The input register will contain data which was present immediately prior to the falling edge of $\overline{RD}$. See Figure 5.0-4.

MODE 3 TIMING
Figure 5.0-4a



*Timing Diagram Refers to Bit Mode Read

An interrupt will be generated if interrupts from the port are enabled and the data on the port data lines satisfies the logical equation defined by the 8-bit mask control registers. Another interrupt will not be generated until a change occurs in the status of the logical equation. A Mode 3 interrupt will be generated only if the result of a Mode 3 logical operation changes from false to true. For example, assume that the Mode 3 logical equation is an "OR" function. An unmasked port data line becomes active and an interrupt is requested. If a second unmasked port data line becomes active concurrently with the first, a new interrupt will not be requested since a change in the result of the Mode 3 logical operation has not occurred. Note that port pins defined as outputs can contribute to the logical equation if their bit positions are unmasked.

If the result of a logical operation becomes true immediately prior to or during $\overline{M1}$, an interrupt will be requested after the trailing edge of $\overline{M1}$, provided the logical equation remains true after $\overline{M1}$ returns high.

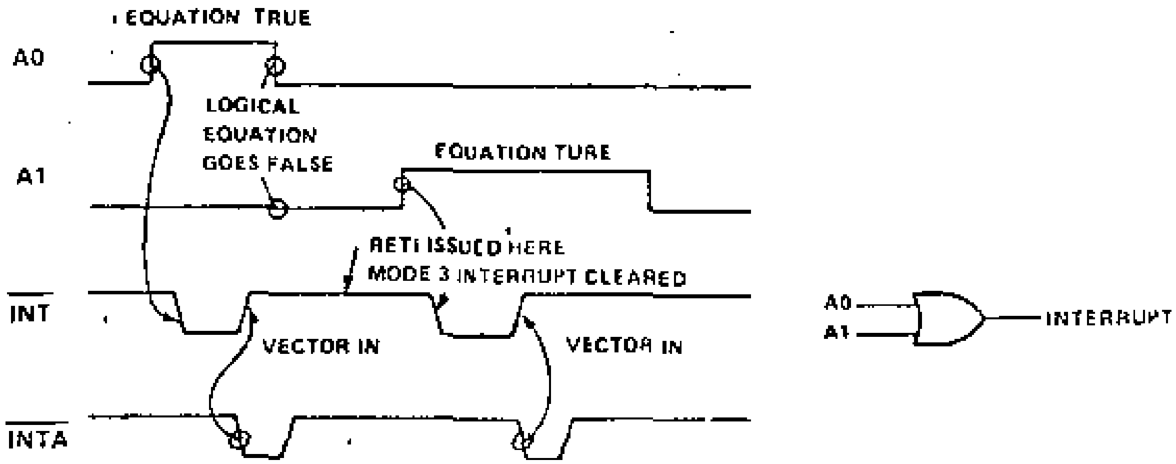
Figure 5.0-4b is an example of Mode 3 interrupts. The port has been placed in Mode 3 and OR logic selected and signals are defined to be high. All but bits A0 and A1 are masked out and are not monitored thereby creating a two input positive logic OR gate. In the timing diagram A0 is shown going high and creating an interrupt ($\overline{INT}$ goes low) and the CPU responds with an Interrupt Acknowledge cycle ($\overline{INTA}$). The PIO port with its interrupt pending sends in its Vector and the CPU goes off into the Interrupt Service Routine. A0 is shown going inactive either by itself or perhaps as a result of action taken in the Interrupt Service Routine (making the logical equation false). An arrow is shown at the point in time where the Service Routine issues the RETI instruction which clears the PIO interrupt structure. A1 is next shown going high making the logical equation-true and generating another interrupt. Two important points need to be made from this example:

1) A1 must not go high before A0 goes low or else the logical equation will not go false — a requirement for A1 to be able to generate an interrupt.

2) In order for A1 to generate an interrupt it must be high after the RETI issued by A0's Service Routine clears the PIO's Interrupt structure. In other words, if A1 were a positive pulse that occurred after A0 went low (to make the equation false) and went low before the RETI had cleared the Interrupt Structure it would have been missed. The logic equation must become false after the INTA for A0's service and then must be true or go true after RETI clears the previous interrupt for another interrupt to occur.

MODE 3 EXAMPLE

Figure 5.0-4b



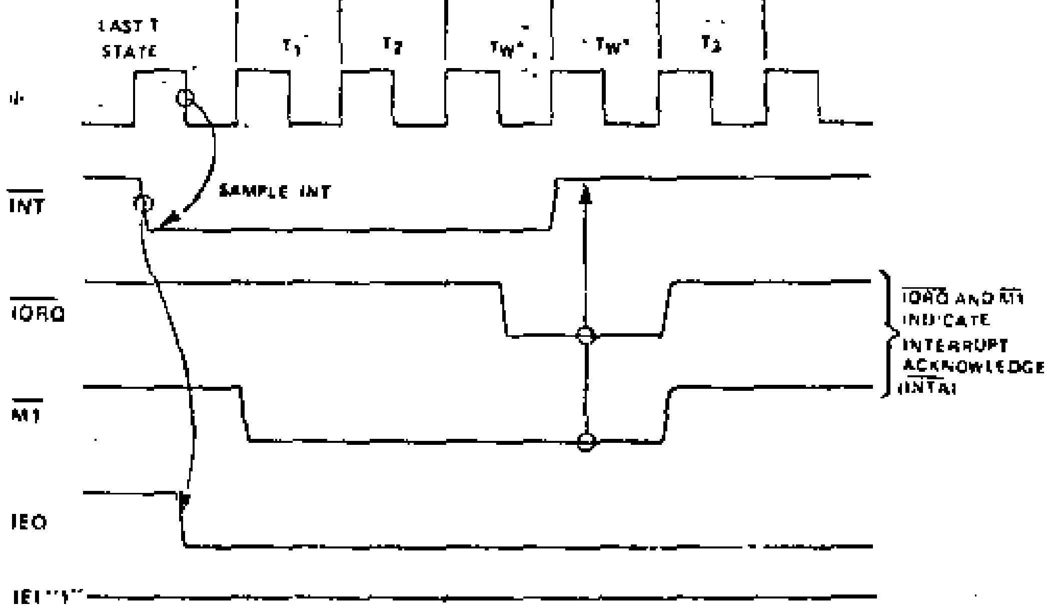
6.0 INTERRUPT SERVICING

Some time after an interrupt is requested by the PIO, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). During this time the interrupt logic of the PIO will determine the highest priority port which is requesting an interrupt. (This is simply the device with its Interrupt Enable Input high and its Interrupt Enable Output low). To insure that the daisy chain enable lines stabilize, devices are inhibited from changing their interrupt request status when $\overline{M1}$ is active. The highest priority device places the contents of its Interrupt vector register onto the Z80 data bus during interrupt acknowledge.

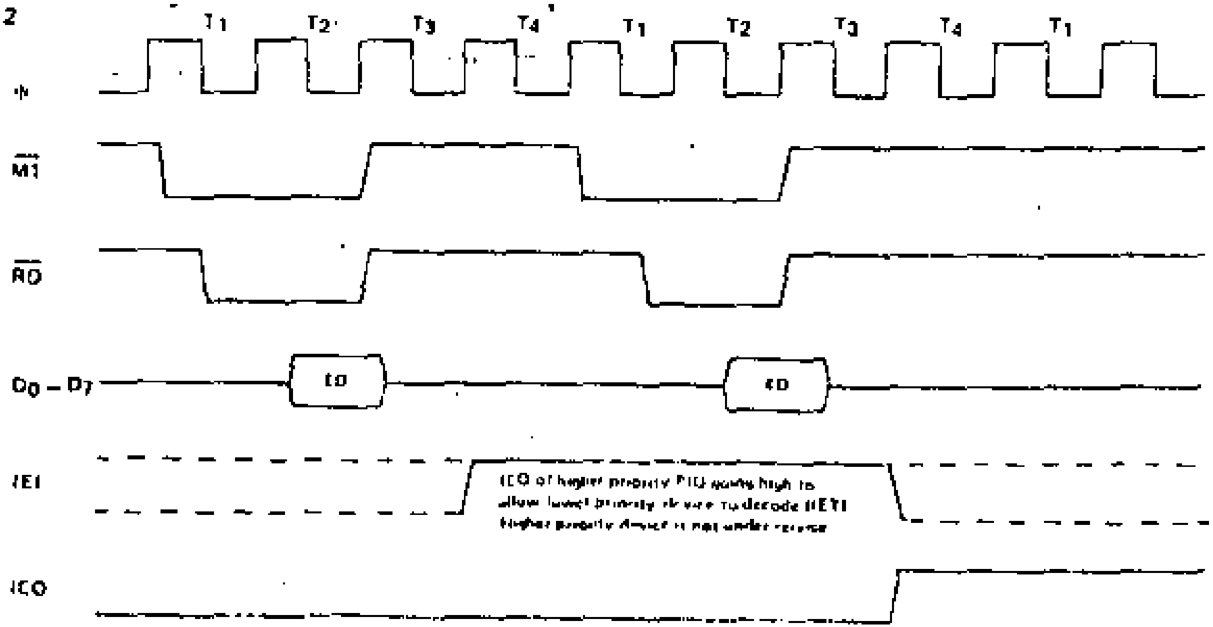
Figure 6.0-1 illustrates the timing associated with interrupt requests. During $\overline{M1}$ time, no new interrupt requests can be generated. This gives time for the Int Enable signals to ripple through up to four PIO circuits. The PIO with IEI high and IEO low during INTA will place the 8-bit interrupt vector of the appropriate port on the data bus at this time.

If an interrupt requested by the PIO is acknowledged, the requesting port is 'under service'. IEO of this port will remain low until a return from interrupt instruction (RETI) is executed while IEI of the port is high. If an interrupt request is not acknowledged, IEO will be forced high for one $\overline{M1}$ cycle after the PIO decodes the opcode 'ED'. This action guarantees that the two byte RETI instruction is decoded by the proper PIO port. See Figure 6.0-2.

INTERRUPT ACKNOWLEDGE TIMING
Figure 6.0-1

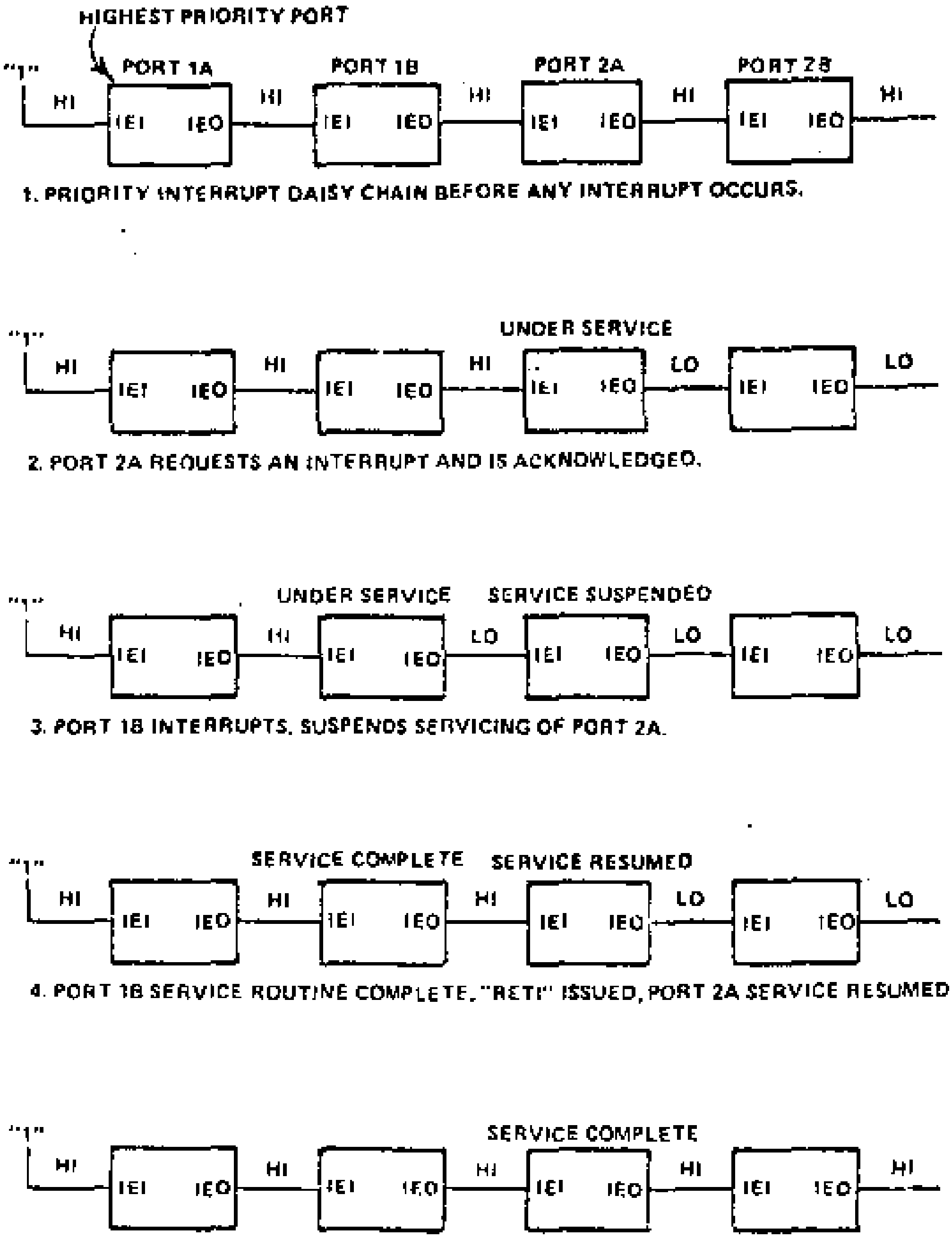


RETURN FROM INTERRUPT CYCLE
Figure 6.0-2


T1
T2
T3
T4
T1
T2
T3
T4
T1
ϕ
M1
RD
D0 – D7
ED
4D
IEI
IEO of higher priority PIO goes high to allow lower priority device to decode RETI
Higher priority device is not under service
IEO

DAISY CHAIN INTERRUPT SERVICING

Figure 6.0-3



1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.

2. PORT 2A REQUESTS AN INTERRUPT AND IS ACKNOWLEDGED.

3. PORT 1B INTERRUPTS, SUSPENDS SERVICING OF PORT 2A.

4. PORT 1B SERVICE ROUTINE COMPLETE, "RETI" ISSUED, PORT 2A SERVICE RESUMED.

5. SECOND "RETI" INSTRUCTION ISSUED ON COMPLETION OF PORT 2A SERVICE ROUTINE.

Figure 6.0.3 illustrates a typical nested interrupt sequence that could occur with four ports connected in the daisy chain. In this sequence Port 2A requests and is granted an interrupt. While this port is being serviced, a higher priority port (1B)-requests and is granted an interrupt. The service routine for the higher priority port is completed and a RETI instruction is executed to indicate to the port that its routine is complete. At this time the service routine of the lower priority port is completed.

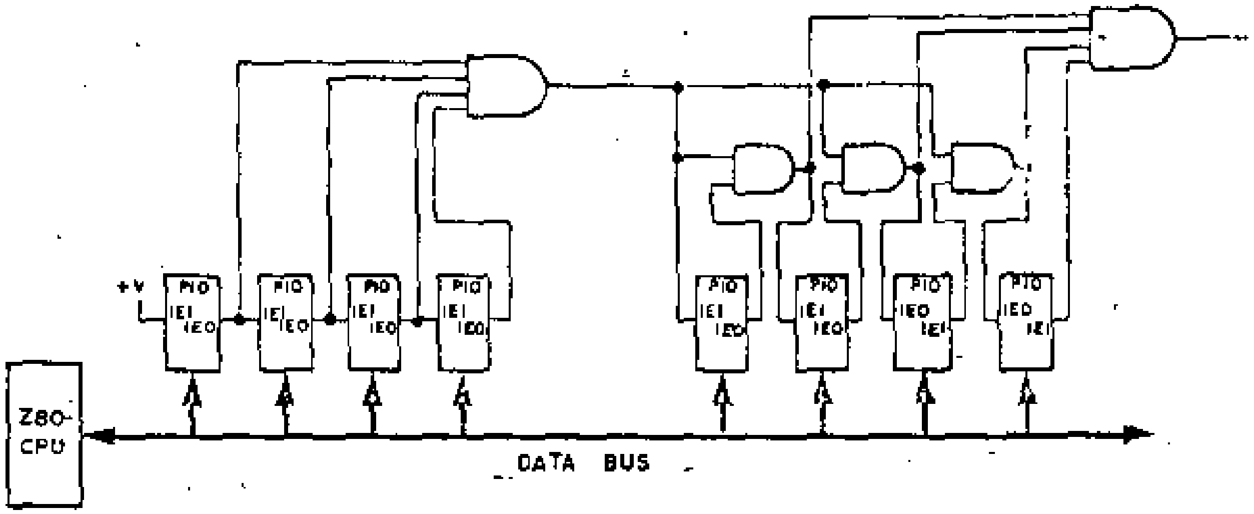
7.0 APPLICATIONS

7.1 EXTENDING THE INTERRUPT DAISY CHAIN

Without any external logic, a maximum of four Z80-PIO devices may be daisy chained into a priority interrupt structure. This limitation is required so that the interrupt enable status (IEO) ripples through the entire chain between the beginning of $\overline{M1}$, and the beginning of $\overline{IORQ}$ during an interrupt acknowledge cycle. Since the interrupt enable status cannot change during $\overline{M1}$, the vector address returned to the CPU is assured to be from the highest priority device which requested an interrupt.

If more than four PIO devices must be accommodated, a "look-ahead" structure may be used as shown in figure 7.0 1. With this technique more than thirty PIO's may be chained together using standard TTL logic.

A METHOD OF EXTENDING THE INTERRUPT PRIORITY DAISY CHAIN
Figure 7.0 1



7.2 I/O DEVICE INTERFACE

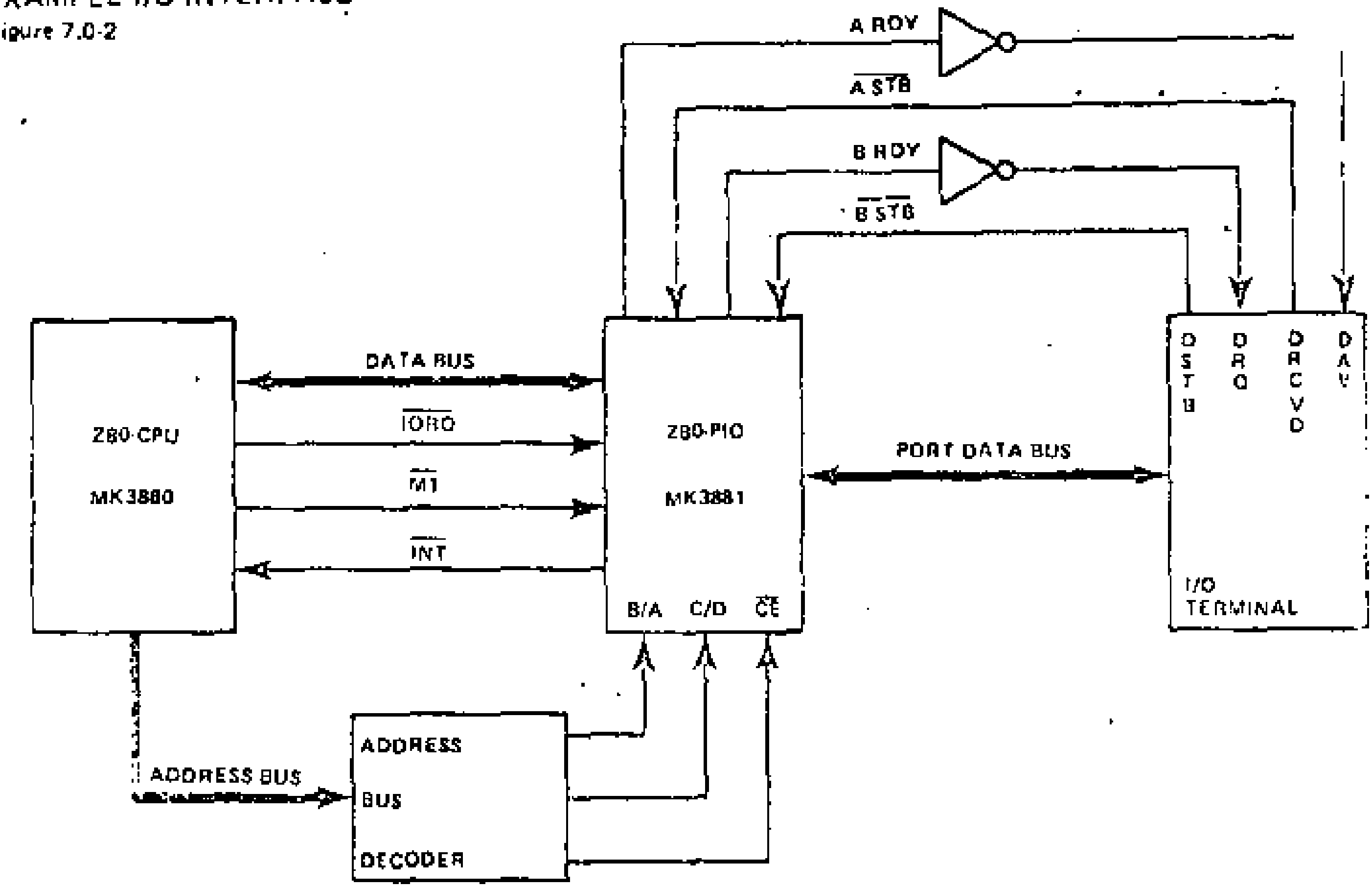
In this example, the Z80-PIO is connected to an I/O terminal device which communicates over an 8 bit parallel bidirectional data bus as illustrated in figure 7.0-2. Mode 2 operation (bidirectional) is selected by sending the following control word to Port A:

EXAMPLE I/O INTERFACE
Figure 7.0-2

D7	D6	D5	D4	D3	D2	D1	D0
1	0	X	X	1	1	1	1

MODE CONTROL

EXAMPLE I/O INTERFACE
Figure 7.0-2



Next, the proper interrupt vector is loaded (refer to CPU Manual for details on the operation of the interrupt).

V7	V6	V5	V4	V3	V2	V1	0

Interrupts are then enabled by the rising edge of the first $\overline{M1}$ after the interrupt mode word is set unless that $\overline{M1}$ defines an interrupt acknowledge cycle. If a mask follows the interrupt mode word, interrupts are enabled by the rising edge of the first $\overline{M1}$ following the setting of the mask.

Data can now be transferred between the peripheral and the CPU. The timing for this transfer is as described in Section 5.0.

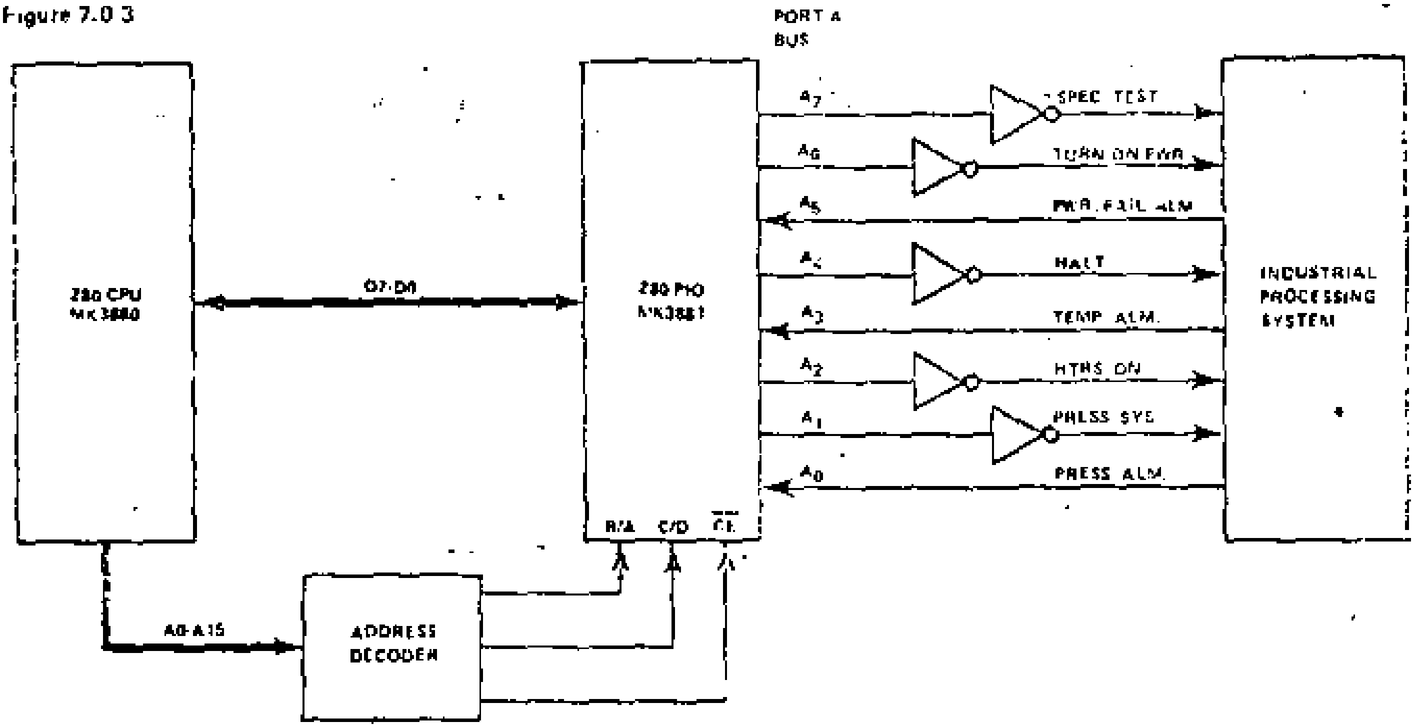
7.3 CONTROL INTERFACE

A typical control mode application is illustrated in figure 7.0-3. Suppose an industrial process is to be monitored. The occurrence of any abnormal operating condition is to be reported to a Z80 CPU based control system. The process control and status word has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Special Test	Turn On Power	Power Failure Alarm	Halt Process [illegible]	Temp Alarm	Temp Heaters On	Pressure On System	Pressure Alarm

CONTROL MODE APPLICATION

Figure 7.0 3



The PIO may be used as follows. First Port A is set for Mode 3 operation by writing the following control word to Port A.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	X	X	1	1	1	1

Whenever Mode 3 is selected, the next control word sent to the port must be an I/O select word. In this example we wish to select port data lines A5, A3, and A0 as inputs and so the following control word is written:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	0	0	1

Next the desired interrupt vector must be loaded (refer to the CPU manual for details

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	V0

An interrupt control word is next sent to the port:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	1	1
Enable Interrupts	OR Logic	Active High	Mask Follows	Interrupt Control			

The mask word following the interrupt mode word is:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	1	0	1	1	0

Selects A5, A3 and A0 to be monitored

Now, if a sensor puts a high level on line A5, A3, or A0, an interrupt request will be generated. The mask word may select any combination of inputs or outputs to cause an interrupt. For example, if the mask word above had been:

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	1	1	0

then an interrupt request would also occur if bit A7 (special Test) of the output register was set.

Assume that the following port assignments are to be used:

$E0_H$= Port A Data
$E1_H$= Port B Data
$E2_H$= Port A Control
$E3_H$= Port B Control

All port numbers are in hexadecimal notation. This particular assignment of port numbers is convenient since A_0 of the address bus can be used as the Port B/A Select and A_1 of the address bus can be used as the Control/Data Select. The Chip Enable would be the decode of CPU address bits A_7 thru A_2 (111000). Note that if only a few peripheral devices are being used, a Chip Enable decode may not be required since a higher order address bit could be used directly.

25

8.0 PROGRAMMING SUMMARY

8.1 LOAD INTERRUPT VECTOR

V7	V6	V5	V4	V3	V2	V1	0

8.2 SET MODE

M1	M0	X	X	1	1	1	1

MODE NUMBER	M_1	M_0	MODE
0	0	0	Output
1	0	1	Input
2	1	0	Bidirectional
3	1	1	Bit Control

When selecting Mode 3, the next word to the PIO must set the I/O Register:

I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

I/O = 1 Sets bit to Input
I/O = 0 Sets bit to Output

8.3 SET INTERRUPT CONTROL

Int Enable	AND/OR	High/Low	Mask Follows	0	1	1	1

USED IN MODE 3 ONLY

If the "mask follows" bit is high, the next control word written to the PIO must be the mask:

MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

MB = 0, Monitor bit
MB = 1, Mask bit from being monitored

Also, the interrupt enable flip flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Int Enable	X	X	X	0	0	1	1

9.0 ELECTRICAL SPECIFICATIONS

9.1 ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	Specified operating range.
Storage Temperature	–65°C to +150°C
Voltage On Any Pin With Respect To Ground	–0.3V to +7V
Power Dissipation	.6W

9.2 D. C. CHARACTERISTICS

Table 9.2-1

T_A = 0°C to 70°C, V_{CC} = 5 V ± 5% unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.6	V_{CC}+.3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.0mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250µA
I_{CC}	Power Supply Current		70*	mA	
I_{LI}	Input Leakage Current		10	µA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	µA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	µA	V_{OUT} = 0.4 V
I_{LD}	Data Bus Leakage Current in Input Mode		±10	µA	$0 \le V_{IN} \le V_{CC}$
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH}=1.5V Port B Only

* 150mA for -4, -10, and -20 devices.

9.3 CAPACITANCE

Table 9.3-1

T_A = 25°C, f = 1 MHz

Symbol	Parameter	Max	Unit	Test Condition
C_ϕ	Clock Capacitance	10	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

9.4A A.C. CHARACTERISTICS MK3880, MK3880-10, MK3880-20, Z80-PIO

Table 9.4-1A $T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 5\%$, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMEN
Φ	t_c	Clock Period	400	[1]	nsec	
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	170	2000	nsec	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	170	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times		30	nsec	
	t_h	Any Hold Time for Specified Set-Up Time	0		nsec	
C/D SEL $\overline{CE}$ ETC.	$t_{S\Phi(CS)}$	Control Signal Set-up Time to Rising Edge of Φ During Read or Write Cycle	280		nsec	
D_0-D_7	$t_{DR(D)}$	Data Output Delay from Falling Edge of $\overline{RD}$		430	nsec	[2]
	$t_{S\Phi(D)}$	Data Set-Up Time to Rising Edge of Φ During Write or $\overline{M1}$ Cycle	50		nsec	$C_L = 50pF$
	$t_{DI(D)}$	Data Output Delay from Falling Edge of IORQ During $\overline{INTA}$ Cycle		340	nsec	[3]
	$t_{F(D)}$	Delay to Floating Bus (Output Buffer Disable Time)		160	nsec	
IEI	$t_{S(IEI)}$	IEI Set-Up Time to Falling Edge of $\overline{IORQ}$ During $\overline{INTA}$ Cycle	140		nsec	
IEO	$t_{DH(IO)}$	IEO Delay Time from Rising Edge of IEI		210	nsec	[5]
	$t_{DL(IO)}$	IEO Delay Time from Falling Edge of IEI		190	nsec	[5] $C_L = 50pF$
	$t_{DM(IO)}$	IEO Delay from Falling Edge of M1 (Interrupt Occurring Just Prior to M1) See Note A.		300	nsec	[5]
$\overline{IORQ}$	$t_{S\Phi(IR)}$	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle	250		nsec	
$\overline{M1}$	$t_{S\Phi(M1)}$	$\overline{M1}$ Set-Up Time to Rising Edge of Φ During $\overline{INTA}$ or $\overline{M1}$ Cycle. See Note B.	210		nsec	
$\overline{RD}$	$t_{S\Phi(RD)}$	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	240		nsec	
A_0-A_7 B_0-B_7	$t_{S(PD)}$	Port Data Set-Up Time to Rising Edge of $\overline{STROBE}$ (Mode 1)	260		nsec	
	$t_{DS(PD)}$	Port Data Output Delay from Falling Edge of $\overline{STROBE}$ (Mode 2)		230	nsec	[5]
	$t_{F(PD)}$	Delay to Floating Port Data Bus from Rising Edge of $\overline{STROBE}$ (Mode 2)		200	nsec	$C_L = 50pF$
	$t_{DI(PD)}$	Port Data Stable from Rising Edge of $\overline{IORQ}$ During $\overline{WR}$ Cycle (Mode 0)		200	nsec	[5]
$\overline{ASTB}$ $\overline{BSTB}$	$t_{W(ST)}$	Pulse Width, $\overline{STROBE}$	150 [4]		nsec nsec	
$\overline{INT}$	$t_{D(IT)}$	$\overline{INT}$ Delay Time from Rising Edge of $\overline{STROBE}$		400	nsec	
	$t_{D(IT3)}$	$\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		650	nsec	
ARDY BRDY	$t_{DH(RY)}$	Ready Response Time from Rising Edge of $\overline{IORQ}$		t_c + 460	nsec	[5] $C_L = 50pF$
	$t_{DL(RY)}$	Ready Response Time from Rising Edge of $\overline{STROBE}$		t_c + 400	nsec	[5]

A. $2.5 t_c > (N-2) t_{DL(IO)} + t_{DM(IO)} + t_{S(IEI)}$ + TTL Buffer Delay, if any

B. $\overline{M1}$ must be active for a minimum of 2 clock periods to reset the PIO.

[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

[2] Increase $t_{DR(D)}$ by 10 nsec for each 50pF increase in loading up to 200pF max.

[3] Increase $t_{DI(D)}$ by 10 nsec for each 50pF increase in loading up to 200pF max.

[4] For Mode 2: $t_{W(ST)} > t_{S(PD)}$

[5] Increase these values by 2 nsec for each 10pF increase in loading up to 100pF max.

9.4B A.C. CHARACTERISTICS [illegible]

Table 9.4-1B $T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 5\%$, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_c $t_{W(\Phi H)}$ $t_{W(\Phi L)}$ $t_{r,f}$	Clock Period Clock Pulse Width, Clock High Clock Pulse Width, Clock Low Clock Rise and Fall Times	250 105 105	[1] 2000 2000 30	nsec nsec nsec nsec	
	t_h	Any Hold Time for Specified Set-Up Time	0		nsec	
C/D SEL CE ETC.	$t_{S\Phi(CS)}$	Control Signal Set-Up Time to Rising Edge of Φ During Read or Write Cycle	145		nsec	
D_0-D_7	$t_{DR(D)}$ $t_{S\Phi(D)}$ $t_{DI(D)}$ $t_{F(D)}$	Data Output Delay From Falling Edge of $\overline{RD}$ Data Set-Up Time to Rising Edge of Φ During Write or $\overline{M1}$ Cycle Data Output Delay from Falling edge of $\overline{IORQ}$ During $\overline{INTA}$ Cycle Delay to Floating Bus (Output Buffer Disable Time)	 50	380 250 110	nsec nsec nsec nsec	[2] $C_L = 50pF$ [3]
IEI	$t_{S(IEI)}$	IEI Set-Up Time to Falling edge of $\overline{IORQ}$ during $\overline{INTA}$ Cycle	140		nsec	
IEO	$t_{DH(IO)}$ $t_{DL(IO)}$ $t_{DM(IO)}$	IEO Delay Time from Rising Edge of IEI IEO Delay Time from Falling Edge of IEI IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring Just Prior to $\overline{M1}$) See Note A.		160 130 190	nsec nsec nsec	[5] [5] $C_L = 50pF$ [5]
$\overline{IORQ}$	$t_{S\Phi(IR)}$	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle	115		nsec	
$\overline{M1}$	$t_{S\Phi(M1)}$	$\overline{M1}$ Set Up Time to Rising Edge of Φ During $\overline{INTA}$ or $\overline{M1}$ Cycle. See Note B.	90		nsec	
$\overline{RD}$	$t_{S\Phi(RD)}$	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	115		nsec	
A_0-A_7, B_0-B_7	$t_{S(PD)}$ $t_{DS(PD)}$ $t_{F(PD)}$ $t_{DI(PD)}$	Port Data Set-Up Time to Rising Edge of $\overline{STROBE}$ (MODE 1) Port Data Output Delay from Falling Edge of $\overline{STROBE}$ (Mode 2) Delay to Floating Port Data Bus from Rising Edge of $\overline{STROBE}$ (Mode 2) Port Data Stable from Rising Edge of $\overline{IORQ}$ During $\overline{WR}$ Cycle (Mode 0)	230	 210 180 180	nsec nsec nsec nsec	 [5] $C_L = 50pF$ [5]
$\overline{ASTB}$ $\overline{BTSB}$	$t_{W(ST)}$	Pulse Width, $\overline{STROBE}$	150 [4]		nsec nsec	
$\overline{INT}$	$t_{D(IT)}$ $t_{D(IT3)}$	$\overline{INT}$ Delay Time from Rising Edge of $\overline{STROBE}$ $\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		440 660	nsec nsec	
ARDY, BRDY	$t_{DH(RY)}$ $t_{DL(RY)}$	Ready Response Time from Rising Edge of $\overline{IORQ}$ Ready Response Time from Rising Edge of $\overline{STROBE}$		t_c + 410 t_c + 360	nsec nsec	[5] $C_L = 50pF$ [5]

* $2.5 t_c > (N-2) t_{DL(IO)} + t_{DM(IO)} + t_{S(IEI)}$ + TTL Buffer Delay, if any

[illegible] must be active for a minimum of 2 clock periods to reset the PIO.

[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

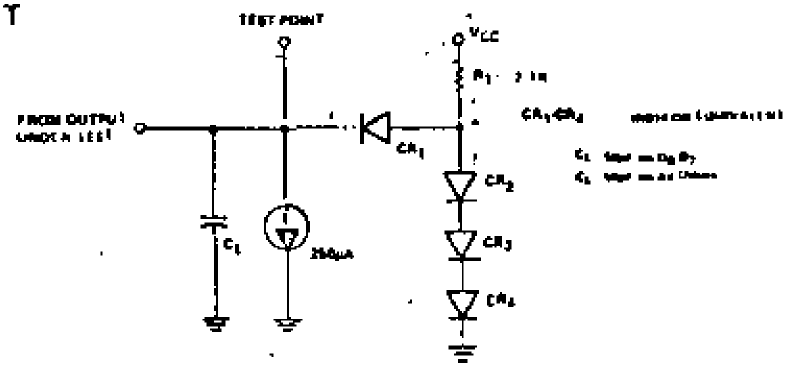
[2] Increase $t_{DR(D)}$ by 10 nsec for each 50pF increase in loading up to 200pF max.

[3] Increase $t_{DI(D)}$ by 10 nsec for each 50pF increase in loading up to 200pF max.

[4] For Mode 2: $t_{W(ST)} > t_{S(PD)}$

[5] Increase these values by 2 nsec for each 10pF increase in loading up to 100pF max.

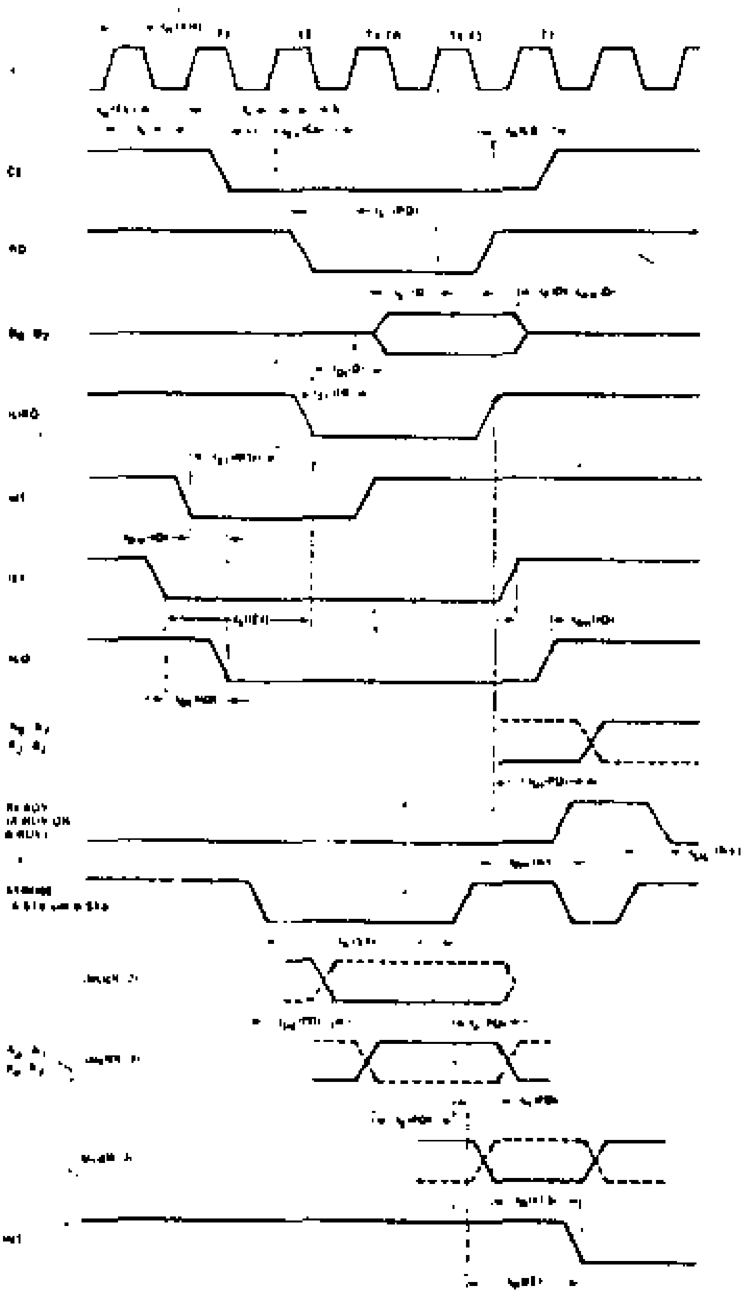
OUTPUT LOAD CIRCUIT
Figure 9.4-1



9.5 TIMING DIAGRAM

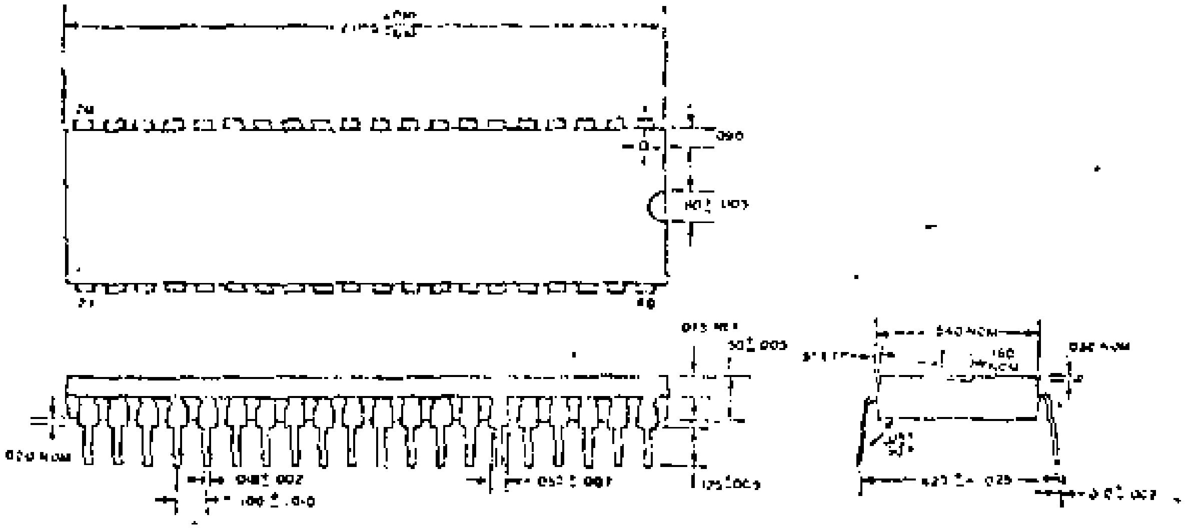
Timing measurements are made at the following voltages, unless otherwise specified.

	"1"	"0"
CLOCK	4.2V	0.8V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	ΔV	+0.5V

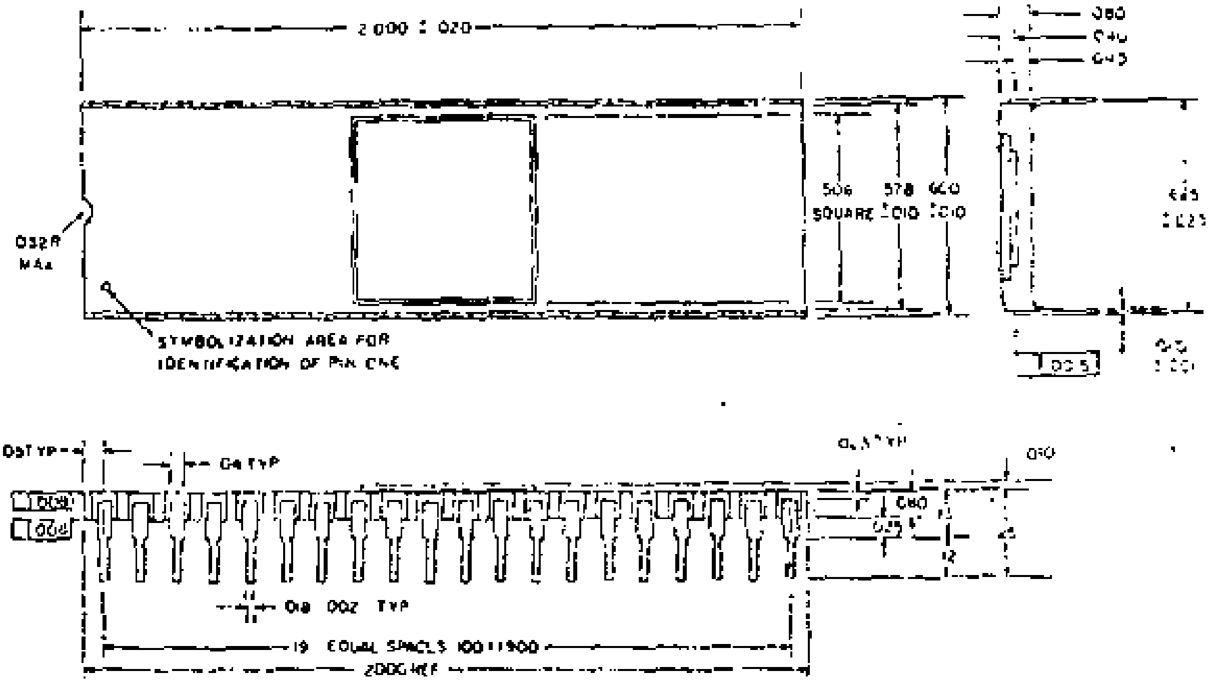


10.0 PACKAGE DESCRIPTION AND ORDERING INFORMATION

PACKAGE DESCRIPTION – 40 PIN DUAL IN-LINE PLASTIC PACKAGE



PACKAGE DESCRIPTION – 40 PIN DUAL IN-LINE CERAMIC PACKAGE



ORDERING INFORMATION

PART NO.	DESIGNATOR	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3881N	Z80 PIO	Plastic	2.5 MHz	0° to 70°C
MK3881P	Z80-PIO	Ceramic	2.5 MHz	
MK3881N-4	Z80A-PIO	Plastic	4.0 MHz	
MK3881P-4	Z80A-PIO	Ceramic	4.0 MHz	
MK3881P-10	Z80-PIO	Ceramic	2.5 MHz	-40° to +85°C
MK3881P-20	Z80 PIO	Ceramic	2.5 MHz	-55° to +125°C

DESCRIPCION DEL MONITOR ZBUG

ZBUG es un programa escrito para el Z80 de 2048 bytes, el cual permite al usuario meter y corregir programas a nivel de lenguaje de máquina. Este programa, está en una memoría del tipo ROM, el monitor ZBUG usa un teclado hexadecimal para meter datos y 6 desplíegues de 7 segmentos para tener lectura de datos en hexadecimal.

El ZBUG incluye también subrutinas para cargar y vaciar programas, a través de una INTERFACE de cassete que permite la utilización de grabadoras de cassete, de muy bajo costo. Un programador de Eproms tipo 2716 de 2 Kbytes se incluye - en este pequeño sistema y las subrutinas para realizar la programación, se encuentran en el monitor ZBUG. Estas memorias una vez programadas se podrán utilizar en los receptáculos que se proporcionan y así tener programas de aplica - ción en forma permanente.

Otra característica del monitor es su capacidad para - diagnósticos avanzados como son las trampas múltiples (BREAK POINT), ejecución de instrucciones paso a paso y la posibilidad de desplegar y modificar los registros del Z80, todos estos diagnósticos generalmente se encuentran en sistemas no simples, pero que son de gran ayuda para el usuario.

En resumen el monitor proporciona al usuario con un control completo sobre la ejecución, detección y corrección - de un programa que se esté desarrollando.

DESCRIPCION DEL ZBUG:

* Botón de Reset

Este switch forza al CPU a inicializarse y empieza la ejecución del programa en la dirección 0000H. El monitor - recide en los primeros 2 kbytes del bloque de memoria.

El monitor inicializa el apuntador de Stack en la 23C0H.

* Tecla monitor.

Esta tecla se usa para suspender la ejecución de programa y regresar el control al ZBUG, (produce una interrupción del tipo $\overline{NMI}$)

* Tecla examinar memoria

Esta tecla se usa para examinar y cambiar localidades de un puerto, para su uso entran 2 dígitos que representan la dirección del puerto, posteriormente, esta tecla se activa y el dato de ese puerto se muestra.

* Tecla examinar registros

Con esta tecla, se pueden examinar y cambiar los siguientes registros: A, B, C, D, E, F, H, L, I, IFF, PC, IX, IY

* Tecla examinar los registros alternos

Como la tecla anterior, se puede examinar los registros alternos:

A', B', C', D', E', F', H', L'

* Breakpoints

El ZBU6 tiene la posibilidad de poner hasta 5 trampas en el programa del usuario, lo que se hace, es cambiar el código operacional del programa y sustituirlo con un RST8(CFH) se guarda el código en una tabla.

Al encontrarse este RST8, los registros del Z80 se guardan en el mapa de registros, para ser analizados. El uso de la tecla paso a paso (single step), cancela todos los Breakpoints.

* Tecla paso a paso (single-step)

Esta tecla, le permite al usuario ejecutar una instrucción a la vez,de programas que están en desarrollo, de esta manera poder interrogar el Status completo del CPU.

* Tecla ejecutar

Esta tecla, ordena al CPU comenzar la ejecución del programa del usuario en la localidad que se especifique o en la -- que esté apuntando el contador de programa.

- MAPA DE MEMORIA -

2800H	NO USADA
27FFH 2400 H	RAM USUARIO 1 K BYTE
23FFH 23CIH	TABLA DE BREAKPOINTS Y MEMORIA DE TRAZADO DEL CPU
23COH 23A9H	MAPA DE REGISTROS
23A8H 2390H	AREA DEL STACK
238F 2000	RAM USUARIO
1FFFH 1800	NO USADA
17FFH 1000 H	PROM 2
OFFFH 0800H	PROM 1
07FF 0000	ZBUG MONITOR

DECODIFICACION DE PUERTOS.

Los puertos de entrada y salida, se encuentran totalmente decodificados.

DIRECCION DEL PUERTO	DISPOSITIVO
80H - 83H	Z80 - PIO
88H - 8BH	SEGMENTOS
8CH - 8FH	DIGITOS

P IO

80H	PA	DATA REGISTER
81	PB	DATA REGISTER
82	PA	CONTROL REGISTER
83	PB	CONTROL REGISTER

LJEMPLO 1:

Escribir al despliegue.

El siguiente programa mueve el carácter 8 de derecha a izquierda, a lo largo del despliegue.

```
            ORG         2000H; Punto de Entrada
D20ms       EQU         064FH;Subrutina para retraso de
                              20milisegundos
SEGMEN      EQU         88H   ;Puerto que selecciona seg-
                              mentos

DIGITO      EQU         8CH   ;Puerto que selecciona el dí
                              gito

MONITOR     EQU         66H   ;Reentrada al monitor
            LD          A,00H
            OUT         (Segmen),A;Activa todos los seg -
                                   mentos
            LD          H,01H
LOOP        OUT         (Dígito),A;Selecciona Primer dígito
            CALL        D20ms
            CALL        D20ms
            CALL        D20ms
            CALL      D20ms

            ~~CALL~~      D20ms       ; Retardo de aprox. 100ms

            RLCA     ; Rotación al siguiente dígito

              JR        LOOP      ; Regresa
```

- INTERRUPCION POR P10 -

```
2000        3E 21      LD      A,21H
2002        ED 47      LD      I,A   inicializa I = 21
2004        3E00       LD      A,00
2006        DE 82      OUT     (82H),A Vector de Interr.
                                      V7 V6V5V4V3V2V10
2008        3E4F       LD      A,4FH;P10 Modo de operación
                                       Entrada   01XXIIII

200A        D382       OUT     (82), A
200C        3E87       LD      A,87H    Interr. control MODE
                                      [1XXX,0111]

200E        D382       OUT     (82),A
2010    .   ED5E       IM 2          Modo 2 de interrupción
```

```
2012        FB                EI   Habilitar interrupción
2013        76                HALT
```

TABLA DE SERVICIO DE INTERRUPCION

```
2100        0022                ; Apunta a la Dirección 2200
```

RUTINA DE SERVICIO DE INTERRUPCION

```
2200       - - - -

2201        ED4D        RETI     ;Regreso de interrupción
```

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

SPECTRUM (The Microcomputer invades the production line)

The MCS-48 Single Component Microcomputer

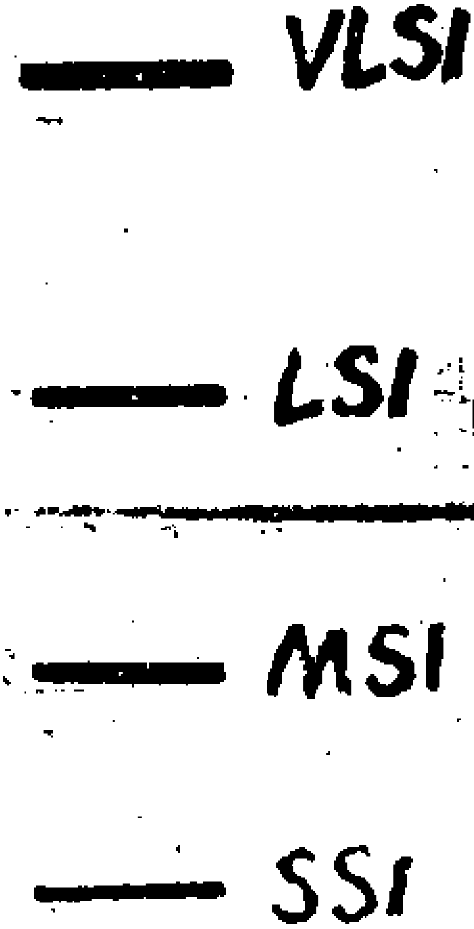
ELECTRONICS (Two versions of 16-bit chip span microprocessor, minicomputer needs)

COMPUTER (Architecture of a new microprocessor)

COMPUTER (The intel MCS-48 microcomputer family; A critique)

ELECTRONICS (VLSI, LSI, MSI, SSI,)

Julio, 1981



by William R. Blood, Jr. Motorola Inc., Integrated Circuits Division, Mesa, Ariz.

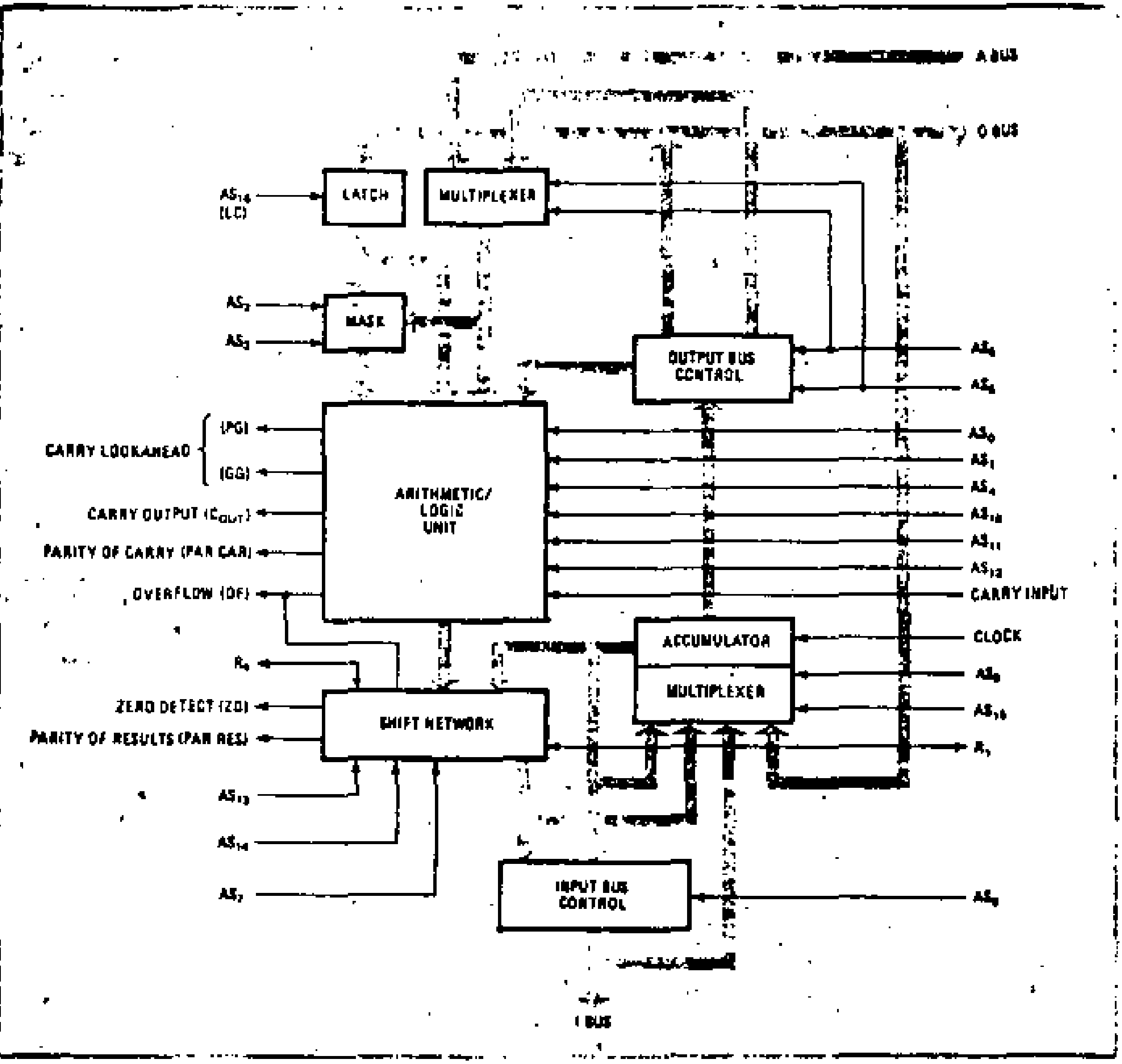
□ In the realm of microprocessors, bipolar large-scale integrated circuitry has evolved very differently from the metal-oxide-semiconductor LSI technologies. Its big selling point is its speed, which can only be optimized for any given microprocessor application if the designer has control of the processor's bus structure, word size, and instruction set. The need for such control has led to the bit-slice approach in bipolar LSI circuits, which is quite unlike the more general-purpose byte orientation of slower MOS microprocessors.

The fastest bipolar technology is emitter-coupled logic, and ECL is the basis for the M10800 family of standard bit-slice parts.

The 10800 family

There are nine members in the 10800 ECL bit-slice family. Each handles 4-bit-wide data paths, but since each is designed around the slice concept, it can parallel itself to build a processor of any given word width. Moreover, each contains data ports for easy interconnection to other LSI circuits. The family includes:

- The MC10800 basic 4-bit arithmetic-and-logic unit.
- The MC10801 microprogram-control circuit.
- The MC10802 timing controller.



1. Processor element. The MC10800 4-bit arithmetic-and-logic unit slice is the heart of Motorola's family of high-speed emitter-coupled-logic circuits. Structured around three buses, two of which are bidirectional, the chip is capable of handling binary and binary-coded-decimal data.

- The MC10803 memory-interface circuit.
- The MC10804 and MC10805, which are 4- and 5-bit level translators for hooking ECL to TTL.
- The MC10806, a dual-access buffer memory.
- The MC10807 5-bit bus transceiver.
- The MC10808 programmable multibit shifter.

The parts also hook onto compatible ECL memories, such as the MCM10146 1,024-bit random-access memory.

The speed of the family is mainly attributable to new circuit design techniques, rather than any breakthrough in integrated-circuit processing. An example is the internal logic that operates off a −2-volt supply, which is better suited than a 5-V supply to the operation of multiplexers, registers, and some other commonly used logic elements. Those elements can thus be easily integrated with those circuit elements like adders that are better built with the series-gated ECL structures powered by the conventional −5.2-V supply. Further, since the 10800 family of parts employs the same fabrication process as the MCM10146 high-speed 1,024-by-1-bit ECL RAM, they enjoy all the benefits of long-established, high-volume production.

The ALU chip

At the heart of the family is the MC10800 4-bit arithmetic-and-logic-unit (ALU) slice, which was the first in the family of standard ECL products to be developed. The chip performs the logic, arithmetic, and shift functions required to execute various machine instructions. Because the part was the first built with the new −2-V logic design, circuit complexity was held to the equivalent of a conservative 350 gates. The area of the chip, which employs standard design rules and double-layer metalization, is less than 15,000 square mils

The microcomputer invades the production line

'Mechanotechniques' yield to the 'silent revolution' of electronics

There's a quiet revolution going on in the manufacturing world that is slowly changing the look of the factory floor. Spearheaded by mini- and microcomputer technologies, electronics is controlling and monitoring processes far more efficiently and quietly than clangy electromechanical relays—and with minimum human supervision.

Although the application of electronics to manufacturing has been somewhat slow, despite the availability of the technology, distinct trends can be observed. These include:

- A proliferation of microcomputers in process-control applications, mainly brought on by their low prices, flexibility of reconfiguration, and suitability for dedicated functions.
- A new generation of robots for small-parts assembly, and the appearance on the production line of smarter robots. Some can make machining and processing decisions.
- An increase in the use of electronic tools, such as lasers and electron-beam guns, to treat materials.
- New electronic data-logging instruments that now monitor more manufacturing processes than scores of human operators formerly did.
- Greater use of computer software through computer-aided design, computer-aided manufacturing, and design automation to cut lengthy production times and increase throughput rates.

A niche for microcomputers

The declining cost of microcomputer products has increased their use as industrial controllers. An 8-bit microcomputer system on a single printed-circuit (PC) board can now be purchased for $300 to $400—far lower than the $1500 to $2000 for a minicomputer to do the same task. And the microcomputer's use as an industrial controller is more efficient; a minicomputer has far more computing power than its application needs. In addition the microcomputer, with its smaller size—and hence less costly software program—is easier to reconfigure for changing applications.

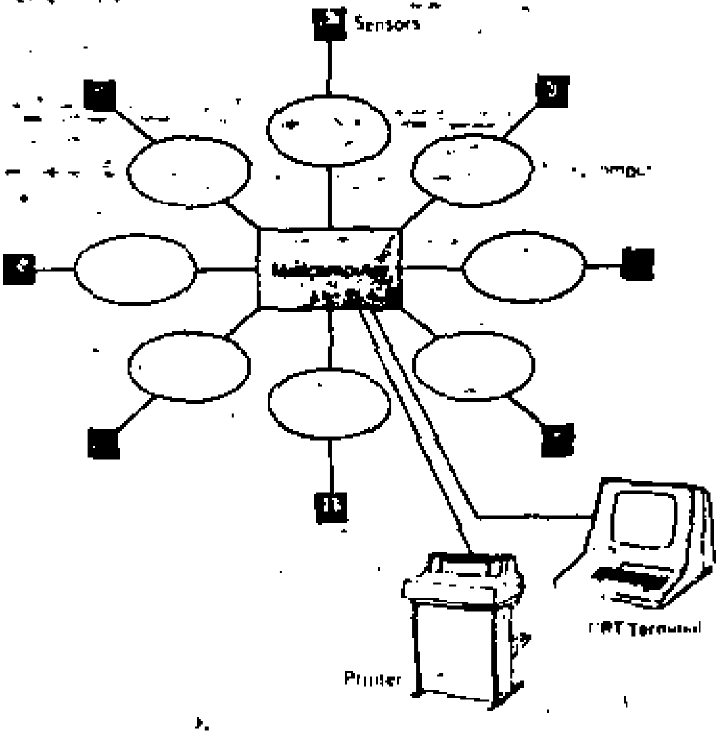
Many microcomputers in industry are being used as dedicated controllers. Typically several microcomputers, each handling a specific function and all tied together by a minicomputer, can be found in a plant. The minicomputer performs the larger data-manipulation task and acts as an interface between the microcomputers and a monitoring station, which may be a CRT terminal (Fig. 1). A printer may also be included.

A high-volume application like automotive sheet-metal gauging illustrates the microcomputer's use. The conventional method is to use visual and manual inspection when sampling and checking sheet-metal parts for correct dimensions. This is done offline and only at infrequent intervals, because it takes a relatively long time. With a microcomputer, several gauging sensors can be set up to scan samples online. The job can be done faster and more frequently, and this ensures that fewer out-of-specification sheet-metal parts will get through. Inspections are also more accurate.

The low cost of microcomputer PC boards has, in many cases, simplified the servicing of electronic control systems. A defective microcomputer board in a dedicated process-control system can simply be isolated and discarded and a replacement quickly put in. Larger and more expensive microcomputers, however, require troubleshooting and repair by service personnel in the plant; the PC boards are simply too costly to discard.

This poses a problem: plant service personnel are mostly house electricians and ill-equipped to repair complex electronic equipment. The shortage of technically proficient maintenance workers is part of a larger problem that the manufacturing industries face as the computer is applied in its many forms: production supervisors and

[1] Because of its low cost and reprogramming flexibility, the microcomputer is being used in many dedicated industrial applications. Typically several microcomputers are tied into a minicomputer, which performs larger data-manipulation and analysis tasks.



Roger Allan, Associate Editor

0018-9235/79/0100-0053$00.75 © 1979 IEEE

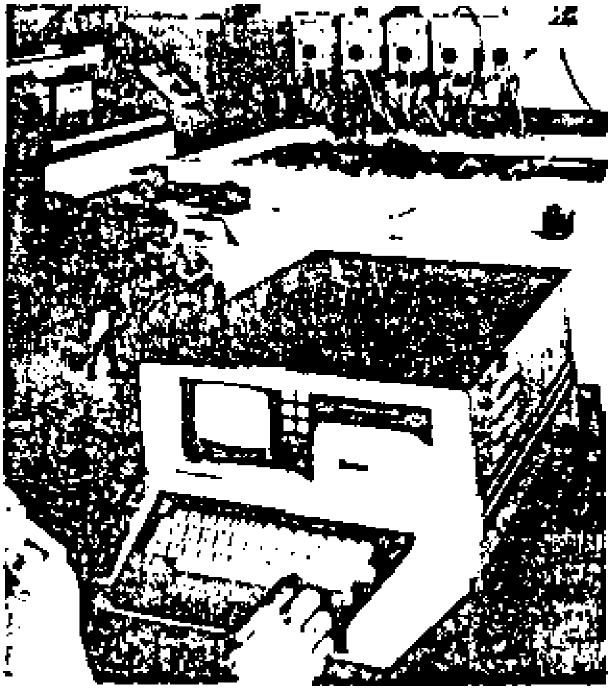
managers [illegible] and computer technology (both hardware and software aspects) as they are with the process they manage.

An indication of the growing use of mini- and microcomputers in manufacturing is the rise in sales of single-card I/O PC boards for interfacing with process variables. Typically such boards contain A/D converters, D/A converters, signal-conditioning components, and multiplexing circuitry. Some require little or no additional circuits and can be interfaced to the sensor directly. Nearly all such PC boards are designed to plug into and interface directly with the mini- or microcomputer they're designed for; the link usually is made in the computer's card-cage housing.

Recently smart digital programming and controller instruments for process control have begun to appear. Typical of such equipment is the microcomputer-based DCP7700 from Honeywell, Fort Washington, Pa. It combines in one box a variable-setpoint vs time programmer with a three-mode controller. The programmer/controller has a keyboard through which an operator can enter and control process inputs. The unit can store up to nine separate programs consisting of up to 200 ramp or soak segments plus event switches.

Analog Devices in Norwood, Mass., has gone several steps further with a total microcomputer-based, closed-loop, real-time measurement and control system (Fig. 2). Known as MACSYM II, the system has a CRT and keyboard and is designed for scientific and industrial applications requiring the acquisition, storage, computation, reduction, presentation, and outputting of high- and low-level signals from universally used sensors (thermocouples, strain gauges, resistive temperature devices, etc.). The system makes use of a high-level language called MACBasic, an extension of Basic, to allow nontechnical operators to use it. As many as 256 channels of analog input data can be accommodated with plug-in PC cards that interface directly to nearly any known process variable.

[2] This closed-loop system from Analog Devices is designed for total process measurement and control by an unskilled operator. The system, MACSYM II, interfaces with all popular industrial sensors.



More robots are appearing

New robots for parts assembly are proving more attractive for industrial batch assembly (nearly three-fourths of all assembly operations in the U.S. are of the batch-assembly variety). Until recently most robots were used for such operations as welding, parts transfer, die casting, forging, and the operation of punch presses. Recent robots have included versions with high intelligence. They can recognize poor metal welds and improperly positioned machine tools as well as do the work.

At the recent Third Industrial Robot Conference and Exhibition in Chicago, Unimation Inc. of Danbury, Conn., introduced the first commercially available microprocessor-controlled robot specifically designed for the assembly of small items, such as electronic components and hardware. Known as Puma (programmable universal manipulator for assembly), it was developed jointly by Unimation and General Motors in Detroit.

GM is using several Pumas for small-parts assembly. The small robot can repeatedly position an object, staying within a tolerance of 0.004 in (0.100 mm), which is only slightly thicker than a human hair. The heart of the 175-lb (79-kg) robot is a Digital Equipment Corp. LSI-11 microprocessor, which controls five other microprocessors, each dedicated to one of five robot arm axes. The motions correspond to waist rotation, shoulder rotation, elbow rotation, wrist bend, and hand rotation. The robot can lift up to 7.7 lb (3.5 kg), including the weight of its end manipulator. Under maximum load, its arm-tip velocity is 3.3 ft/s (101 cm/s). Arm and controller may be separated from each other by as much as 10 ft (3.3 meters) by means of a cable assembly.

Early last year the Westinghouse Electric Corp. Research and Development Center in Pittsburgh submitted to the National Science Foundation a second-phase proposal on programmable automation of batch-assembly operations. The first phase of this study, "Programmable assembly research technology transfer to industry," was completed in October 1977, and it led to the preliminary design of an automated system for small motors (Fig.3). Help for the study was supplied by SRI International of Menlo Park, Calif.; the Charles Stark Draper Laboratory at the Massachusetts Institute of Technology in Cambridge, and the University of Massachusetts in Amherst.

As part of the first-phase study for the National Science Foundation, Westinghouse conducted a worldwide review of programmable-assembly technology. It analyzed about 60 different Westinghouse product lines before deciding on small motors as the most adaptable to programmable automatic assembly. About 450 different motor styles were assembled experimentally. The average batch size was 600, and there were an average of 13 changeovers per shift. The pilot system, known as APAS (adaptable programmable assembly system), was tested over a three-month production schedule.

Richard G. Abraham, Westinghouse's manager for programmable automation, explains: "Although robots are important in the automation of batch-assembly operations, it is equally important to have low-cost, microprocessor-based visual-inspection systems for checking incoming parts and for ascertaining proper assembly steps and style changes. It is also important to stress the need for programmable-parts presentation equipment and software to manage the changeover required for different product styles."

Advances are being made in group control of robots. At the Leningrad Polytechnical Institute in the Soviet Union, researchers have devised a computer system that controls up to 20 sensing robots in real time. The researchers expect to increase this handling capacity to 40 robots as more sophisticated machines are developed with built-in controllers. In the present project, an operator at the system's control panel directs specific robots to carry out dedicated tasks and to take corrective actions, if necessary.

The cell concept

Last year the Foundation of Scientific and Industrial Research at the University of Trondheim in Norway set up what it calls the first full-scale laboratory production line to use a cellular concept. Under this, manufacturing operations are broken down into "cells," each at a different plant. Each cell is responsible for the manufacture of specific subassemblies for a particular product. The cells are interconnected by a network of material and subassembly supply lines. The Norwegian project manufactured complex diesel-engine parts for Wickman Manufacturing of Norway.

In the cellular concept (Fig. 4) the output per manhour of labor is reported to be considerably higher than that of conventional manufacturing methods, even when the additional costs of the interconnecting material and subassembly supply lines are taken into account. Each cell has at its core one robot.

The cell system can be operated day and night, requiring worker participation only during the day. All that is needed for unattended night operation is proper planning, so the robot has an ample supply of materials and enough storage area for completed subassemblies. The robot is a six-arm model manufactured by Cincinnati Milacron, Cincinnati, Ohio.

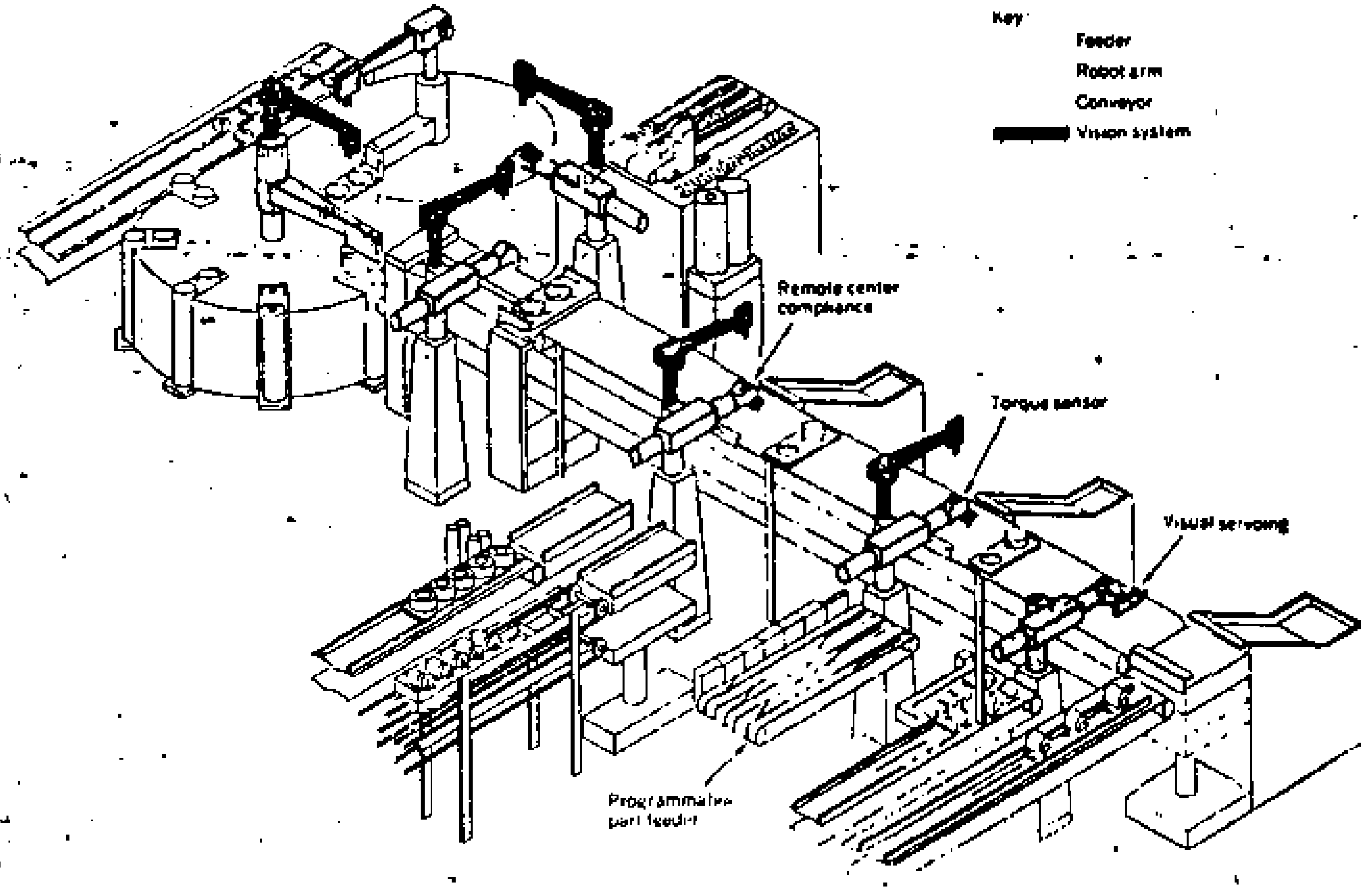
According to Prof. Oyvind Bjorke, head of the Norwegian foundation's production engineering laboratory, the cell concept, with its flexibility for manufacutring different parts, has proved particularly useful in Norway.

"This cell concept," he says, "has been quite beneficial to us in terms of increasing productivity, mainly due to our social and geographical conditions. Norway is a country that has a small and scattered population, which is desirable for national security reasons but is undesirable for manufacturing. We cannot get a large concentration of skilled workers in any one industrial center. This is like bringing the mountain to Mohammed if you can't bring Mohammed to the mountain."

Lasers for materials treatment

Be it drilling, cutting, welding, treating, or the removal of materials, the laser can be found taking on more of these tasks in the plant as its precision and power are increased and its price drops further. Manufacturers are finding the new lasers more reliable, better designed for

[3] A pilot automated assembly line for manufacturing small motors has been designed by Westinghouse Electric's Research and Development Center as part of a National Science Foundation study on programmable automation. Assisting have been SRI International, the Charles Stark Draper Laboratory of the Massachusetts Institute of Technology, and the University of Massachusetts.



industry use, and simpler to operate than those previously available.

With recent improvements in output power, CO$_2$ lasers in the 5–15-kW range are becoming more popular for metal cutting and treating. In fact, laser treatment of materials is the fastest-growing segment of all laser sales, next to their applications in research.

One of the most dynamic and promising areas of industrial laser applications is in platemaking for printing, where faster printing turnarounds and lower costs are major advantages. Laser scanners are used to expose printing plates as large as 48 by 60 inches (122 by 152 cm) directly from the pasted-up page, thus eliminating several in-between steps required with conventional methods. Many medium- and small-circulation newspapers are using laser platemakers.

There also have been advances in computer control of lasers. At Western Electric's Engineering Research Center near Princeton, N.J., a high-speed, computer-controlled laser has been developed to spot-weld miniature-relay terminals. The laser's spot-welding speed of 20 times per second is four times faster than the resistance welding system it replaces. The system consists of a 200-watt (average power) pulsed Nd:YAG (neodymium:yttrium-aluminum-garnet) laser and an X–Y positioning table, whose position accuracy is within 0.0001 in (0.00025 cm). The laser and positioning table are under the control of a microprocessor, which monitors position coordinates by use of linear encoders.

Another favorite and recent technological tool for welding and hardening metals is the electron-beam gun. Its use for selective heat treating of metals offers advantages over other techniques: it is faster to use than the laser, more accurate than induction-heating systems, and more selective in resolution than standard flame-hardening techniques. And it is more energy efficient than all three. These advantages, however, aren't without drawbacks. Electron-beam systems are still very expensive.

An electron-beam welding process was recently developed by Technical Materials Inc. of Lincoln, R.I., for welding formerly incompatible metals in strips of unlimited lengths. The process makes possible the welding of gold with copper, silver with copper, invar with stainless steel, and steel with copper. The key to the welding process is a triode electron gun that generates the electron beam. A high-stability alumina insulator is shielded from problem-causing vapors and is precisely positioned for minimal thermal expansion. Gun operation is accomplished with a 60-kV potential, which compares favorably with a potential of about 150 kV needed by most electron-beam welders. Advanced electronic controls for the gun's electro-optical focusing system are responsible for the lower potential.

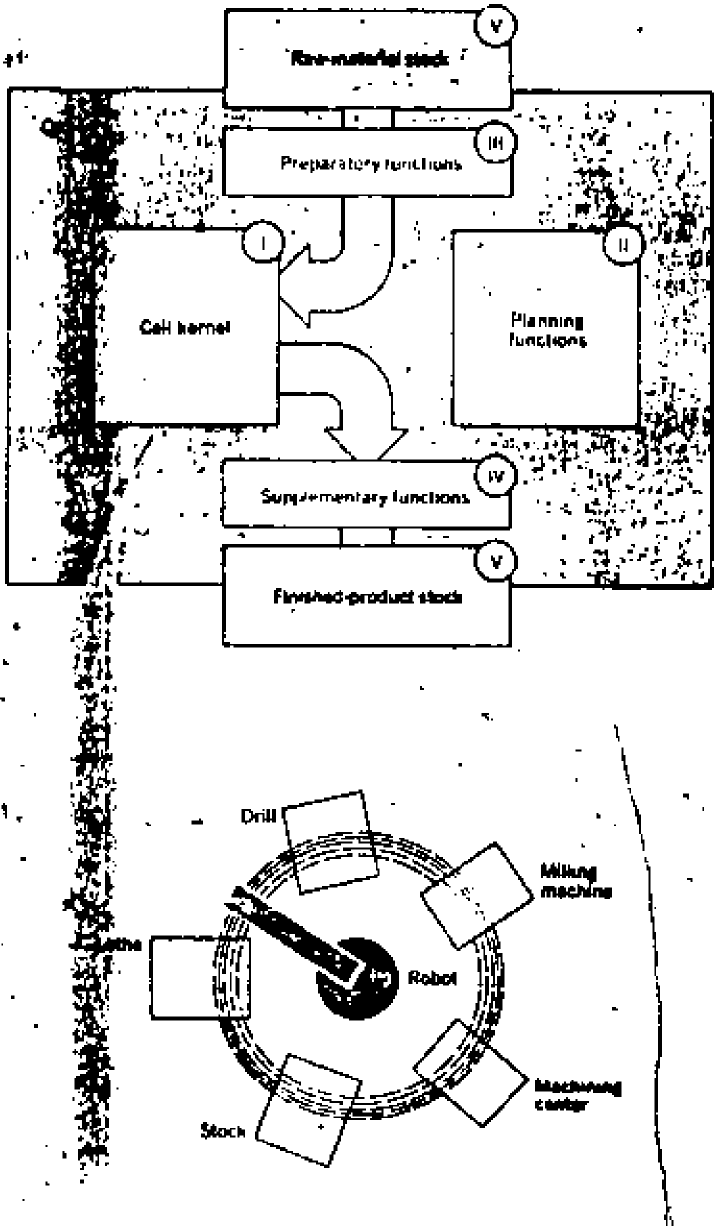
Keeping tabs on the process

Thanks to microprocessor technology, inexpensive data-logging instruments are making process control more scientific. This, in turn, has made possible better control over processes, because of the availability of more accurate and systematic data-collection and analysis techniques.

Data loggers can be found in nearly every industrial application, from food processing and materials treatment to papermaking and the generation of steam. The high speed and accurate data-collection advantages now possible have made obsolete more expensive and slower equipment, like pen-chart recorders.

A typical data-logging application is recorded in the files of the John Fluke Manufacturing Co. in Mountlake Terrace, Wash. To obtain more control over the final product, a producer of electrically conducting aluminum bus bars used a Fluke Model 2240A data logger to keep track of temperature and voltage data. The data were collected in point-for-point pairs along each bar to determine relative conductivity as a function of the bar's shape, alloy, and type. The reason for measuring temperature in addition to voltage was to provide better data correlation. The instrument also monitored the bus bars for dangerous overheating at five separate points every second. And it was equipped to notify operators of overheating and to turn the process off if necessary. All of the collected data were fed to a minicomputer for analysis.

[4] The cell concept of decentralized manufacturing has at its core a robot programmed to work day and night, attended by humans only during the day. The system increases throughputs in job-lot manufacturing. Proposed by the Foundation of Scientific and Industrial Research at the University of Trondheim in Norway, it is being used with a Cincinnati Milacron six-arm robot to manufacture complex diesel-engine parts.



The challenge of technology transfer

Many U.S. manufacturing experts say the technology to increase productivity has been here for some time. Knowing how to put it to practical use rapidly is really the problem, they add. The laser, for example, has been a research tool in the laboratory for well over a decade, but only in the last few years has it been applied successfully in manufacturing plants. Similarly the minicomputer was available long before it was applied in industry on a wide scale.

Perhaps the problem runs deeper than what is perceived. Prof. Gustav Olling, chairman of Bradley University's Department of Manufacturing Technology in Peoria, Ill., and a former practicing engineer with extensive industrial experience, puts it this way:

"U.S. academic researchers and more practical production/manufacturing individuals are not in tune with each other. Each group needs to get more involved with each other's experiences—the academician with the production manufacturing person's real-life experiences in the plant and the production person with the academician's theoretical contributions. This will also ensure an easier transition for engineering graduates into industry."

Prof. Olling cites foreign countries with more advanced industrial applications of electronics than the U.S.—Japan, Germany, and Norway. In these countries, he says, academic and industrial people work hand in hand to solve industry's problems. In many cases, promotions to professorships are based on the individual's industrial experience as well as scientific contributions. In some countries a professor who lectures in a university may also work in industry to solve automation problems. Prof. Oyvind Bjorke's work on cellular manufacturing, described in this report, is an example.

As part of its long-range plans, Computer Aided Manufacturing International, a Texas-based research organization, has an education/industry committee of which Prof. Olling is chairman. Its objective is to stimulate cooperation between industrial, educational, and professional groups through information exchanges, common research areas, faculty and student exchanges, and the development of educational programs in computer-aided design and computer-aided manufacturing.

The trend is to incorporate more intelligence in data-logging instruments and to provide them with wider ranges of options, since some of their users are not technically proficient and industrial applications differ widely.

Greater use of CAD, CAM, and DA

Industry has slowly begun to make greater use of computer-aided design (CAD), computer-aided manufacturing (CAM), and design automation (DA) as manufacturing has become more complex and productivity has not kept pace with rising costs for labor and materials. In countries like Japan, where high industrial productivity is a national goal, CAD is used in nearly every industry. For example, the government-owned Nippon Telegraph and Telephone Corp. recently opened a service to the public that provides via telephone the supporting software for developing LSI microprocessor programs. The information comes from the data banks of a national computer.

Although total use of CAD and CAM—the completely automated factory—is still far off, manufacturing experts are emphasizing the potential of each technique. Computer Aided Manufacturing International of Arlington, Tex., a research organization composed of leading worldwide industrial companies, has been developing computer programs and manufacturing strategies to help automate production systems. These are intended to lead to higher manufacturing productivity.

One such program is CAPP (computer-aided process planning), a program that links CAD and CAM. CAPP features a parts-coding scheme that allows almost any part in a factory to be described from a few fundamental geometric shapes. With a CRT terminal and a keyboard, a designer in a plant using CAPP can call up at once, or even create instantly, a part needed for manufacturing. There is no need for time-consuming engineering drawings. All of the part's manufacturing characteristics are described in detail on the CRT screen.

One reason why more industrial manufacturers don't use more programs like CAPP is the high initial cost of software preparation and generation. Although CAPP has proved cost-effective for industrial applications, it is being used in few job-lot companies. They are skeptical of investing the $100 000 needed typically to build just a classifying data base with CAPP. For a 20 000-item job lot, 30 to 40 weeks are needed to build the classifying data base, and about one-third of this time is spent planning how to put such a data base together.

Automating the design of PC boards

An area in manufacturing where application of the computer's power has made notable progress is in the design of PC boards. Here DA, design automation, is using the computer not only to generate precision design data (such as PC-board artwork) but also to lay out optimally the circuit components and interconnections on the board. With CAD, computer-aided design, component placement and interconnection are performed manually.

DA has been applied successfully in the design of dense PC boards with high-speed ICs like ECL (emitter-coupled logic), a job virtually impossible to achieve without the computer's aid. Given a logic diagram of such basic data as the components to be used, the size of the PC board, and the required wiring, a DA program can produce interconnection patterns for the most complex multilayered PC boards.

A leader in applying DA to PC-board design, Automated Systems Inc. in El Segundo, Calif., recently introduced a software module (Place II) that makes it possible to route the interconnections for large PC boards of high density, a job that previously could not be done automatically. For example, using the Place II module, a designer can lay out PC cards as large as 15 by 18 inches (38 by 46 cm) with 300 different devices. This is done by sophisticated techniques that allow the computer to group circuit elements according to similar functions. ◆

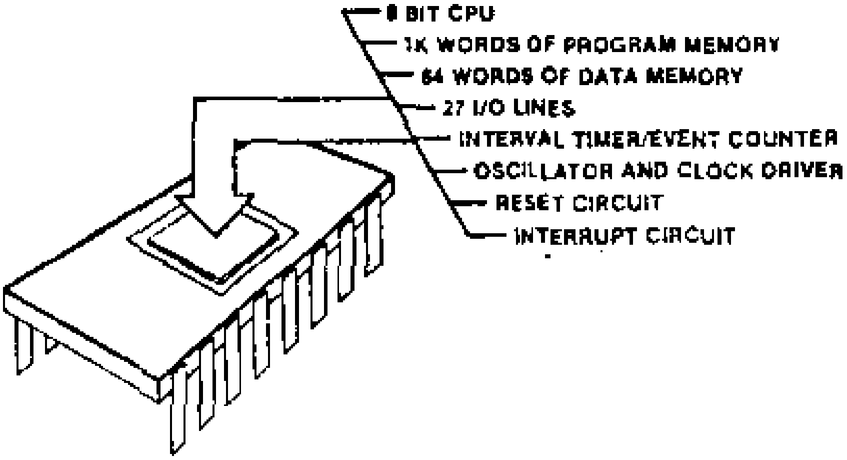
THE MCS-48™ FAMILY

Notes: /

8048 — MICROCOMPUTER WITH ROM
8748 — MICROCOMPUTER WITH EPROM
8035 — MICROCOMPUTER WITHOUT ROM
8243 — I/O EXPANDER
8355 — ROM PROGRAM MEMORY AND I/O EXPANDER
8755 — EPROM PROGRAM MEMORY AND I/O EXPANDER
8155 — DATA MEMORY AND I/O EXPANDER

- *The Basic Family will be expanded with additional I/O and processor elements.*
- *Most 8080 peripherals and standard memory products are directly compatible.*

ON CHIP FEATURES



8 BIT CPU
1K WORDS OF PROGRAM MEMORY
64 WORDS OF DATA MEMORY
27 I/O LINES
INTERVAL TIMER/EVENT COUNTER
OSCILLATOR AND CLOCK DRIVER
RESET CIRCUIT
INTERRUPT CIRCUIT

- *Totally self contained. All that is required is 5 Volts.*

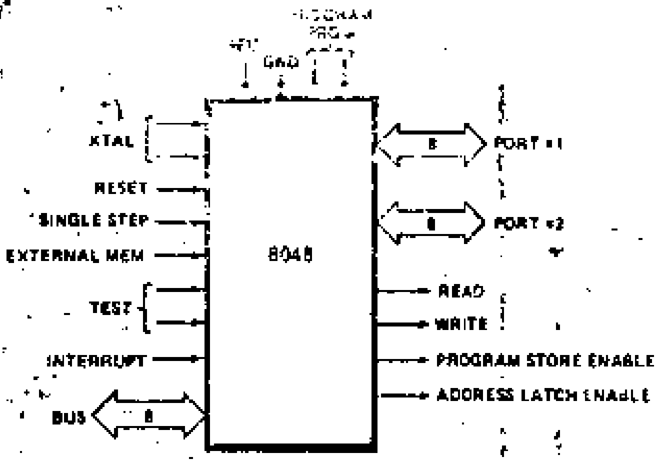
SPECIAL FEATURES

- SINGLE 5V SUPPLY
- 40 PIN DIP
- PIN COMPATIBLE ROM AND EPROM
- 2.5 μsec CYCLE
- ALL INSTRUCTIONS 1 OR 2 CYCLES
- SINGLE STEP
- 8 LEVEL STACK
- 2 WORKING REGISTER BANKS
- RC, XTAL, OR EXTERNAL FREQUENCY SOURCE
- ÷ 3 OR ÷ 15 CLOCK OUTPUT

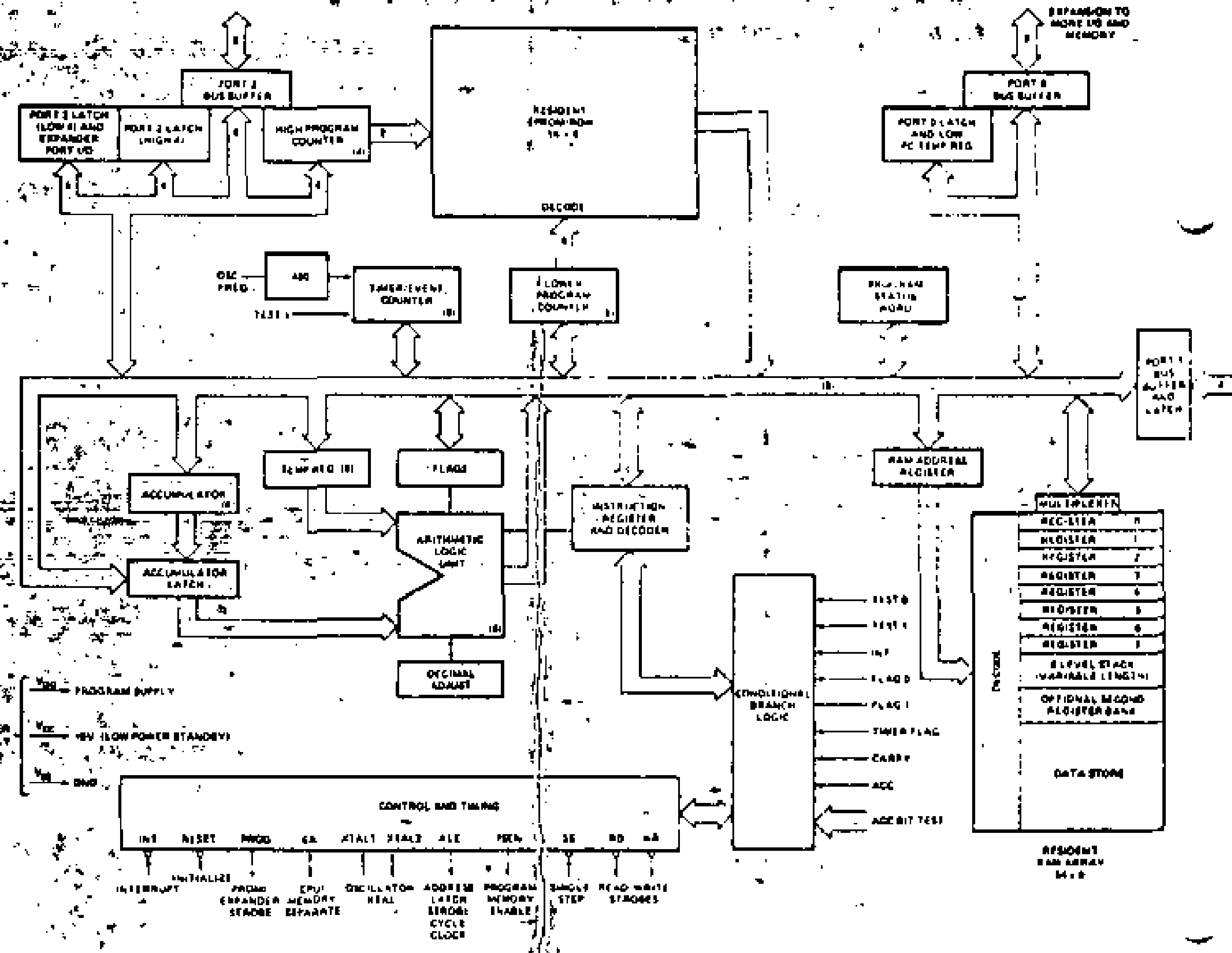
- *All contained on the single chip.*
- *Low power standby on ROM version.*

409

Notes: 7



- Inputs and outputs can be mixed on a single port.
- Bus can be used to provide a standard interface for expansion or latched in one chip applications.
- All I/O lines are TTL compatible.

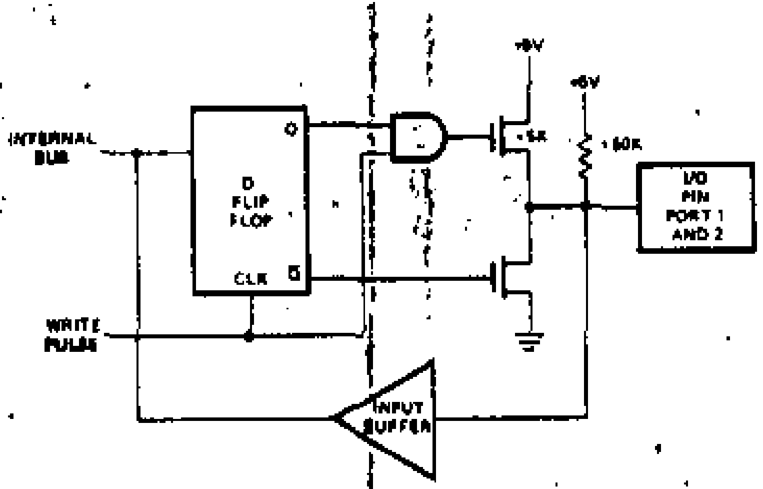


- All major functional computer elements are integrated on the single chip.

.2

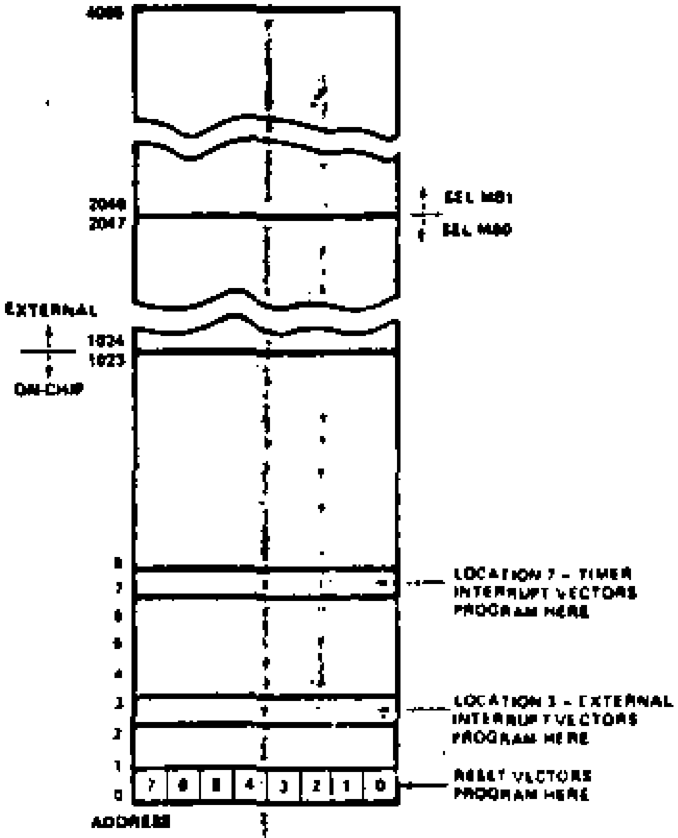
Notes: 3

"QUASI BI-DIRECTIONAL" PORT STRUCTURE



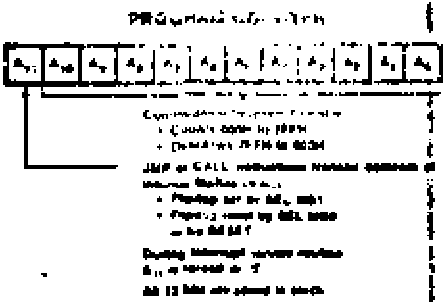
- *Allows inputs and output to be mixed on a single port.*
- *All output ports can also be read for diagnostics and temporary storage purposes.*

MCS-48 PROGRAM MEMORY MAP

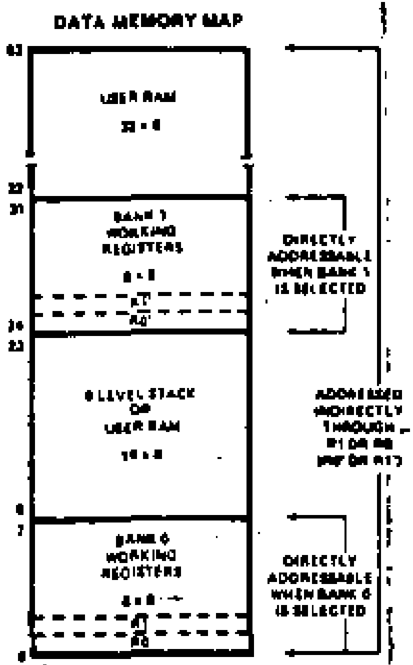

4095
2048
2047
SEL MB1
SEL MB0
EXTERNAL
1024
1023
ON-CHIP
8
7
6
5
4
3
2
1
0
7
6
5
4
3
2
1
0
ADDRESS
LOCATION 7 – TIMER INTERRUPT VECTORS PROGRAM HERE
LOCATION 3 – EXTERNAL INTERRUPT VECTORS PROGRAM HERE
RESET VECTORS PROGRAM HERE

- *8035 all address references are external.*
- *8048/8748 address references below 1K are internal and all references above 1K are automatically routed to the external bus.*
- *70% of all applications fall below 1K.*

Notes:



- *The 12 bit program counter can address 4K of program memory.*
- *Last bit of program counter can be directly manipulated by the instruction software.*

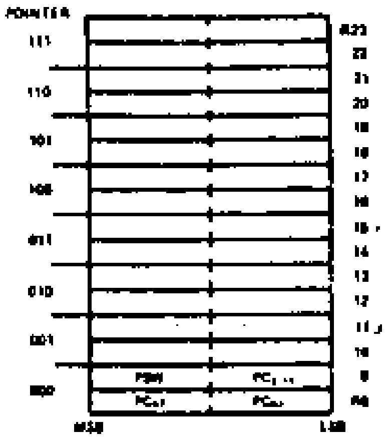


IN ADDITION R0 OR R1 (R0' OR R1') MAY BE USED TO ADDRESS 256 WORDS OF EXTERNAL RAM

- *Additional pages of 256 x 8 can be externally addressed.*
- *Second register bank and stack depth are optional, allowing resident memory to be from 32 x 8 to 56 x 8.*
- *Bank switch is done with one command.*
- *Stack stores the program counter and status word.*

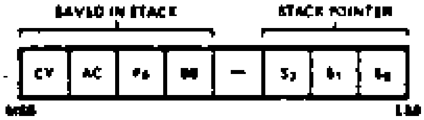
Notes:

PROGRAM COUNTER STACK



- 8 level stack provides for generous subroutine nesting.
- Program status word is stored next to program.counter in the stack.
- The stack depth is optional. Unused stack locations may be used for data memory.

PROGRAM STATUS WORD (PSW)

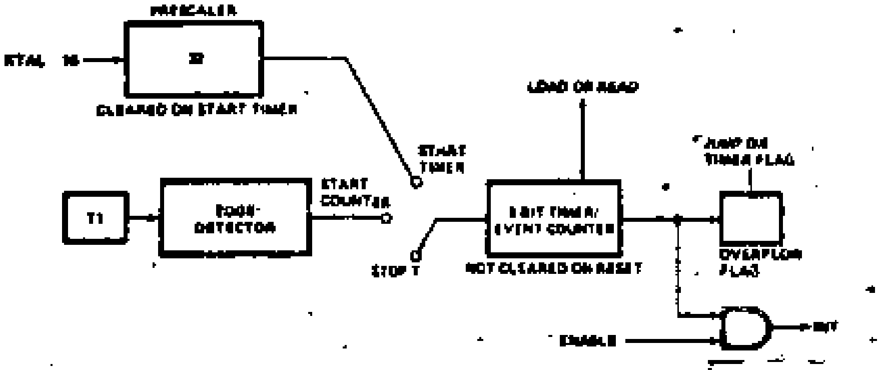


CY	CARRY
AC	AUXILLARY CARRY
F0	FLAG 0
BS	REGISTER BANK SELECT

- Status word can be directly manipulated by the software.
- Machine state is automatically restored after a return from interrupt.

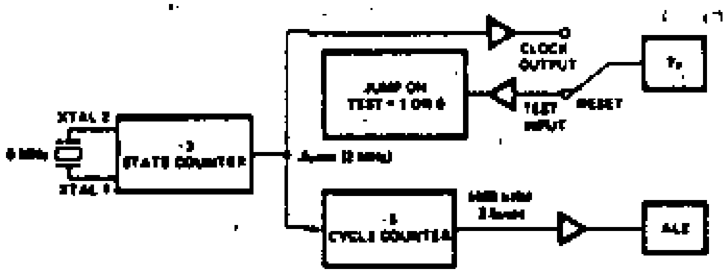
TIMER/EVENT COUNTER

Notes:



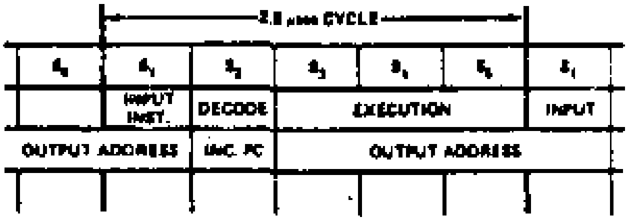
- *Test 1 (T1) is specified to be counter input if that option is selected.*
- *All options can be selected under software control.*
- *Timer or counter can be preset, read, stopped, and started.*
- *Timer/counter overflows at 255, causing set flag or optional interrupt.*

DIAGRAM OF 8048 CLOCK UTILITIES



- *ALE signal is always available as a clock output.*
- *Test 0 (T0) may be specified to be clock out under software control.*

INSTRUCTION CYCLE

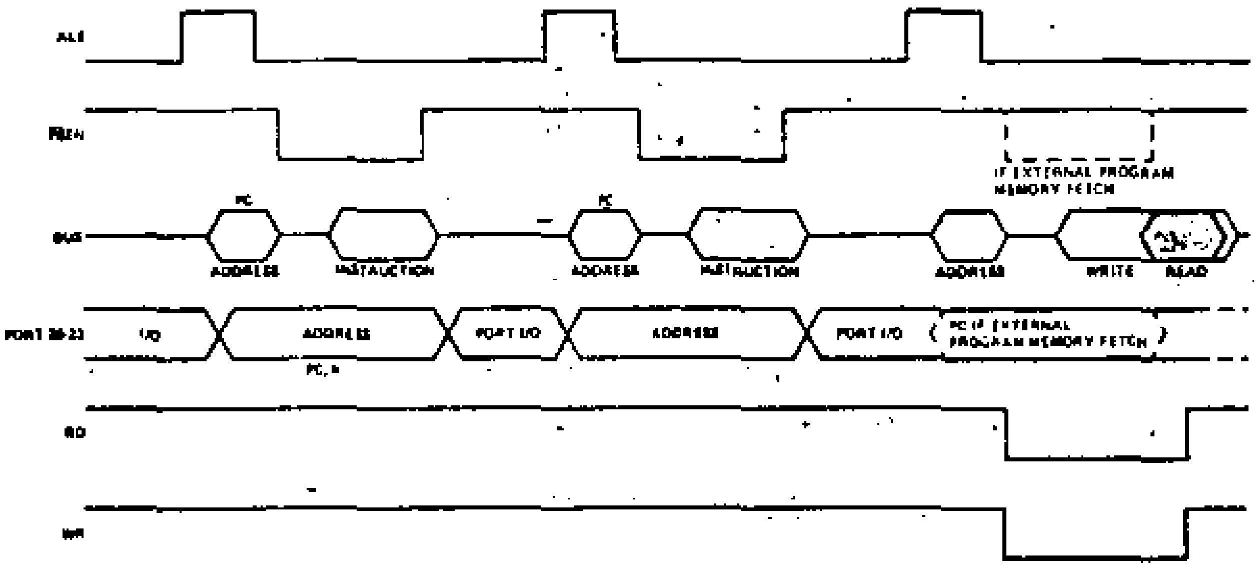


2.5 µsec CYCLE

S_5	S_1	S_2	S_3	S_4	S_5	S_1
	INPUT INST.	DECODE	EXECUTION			INPUT
OUTPUT ADDRESS		INC. PC	OUTPUT ADDRESS			

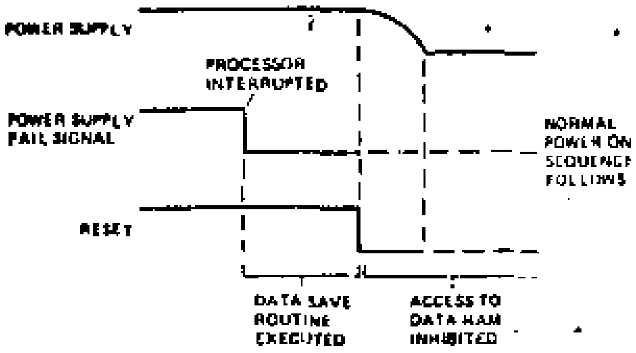
- *All instruction cycles consist of 5 internal states.*
- *Overlapped operation allows for fast instruction execution.*

MCS-48™ CYCLE TIMING



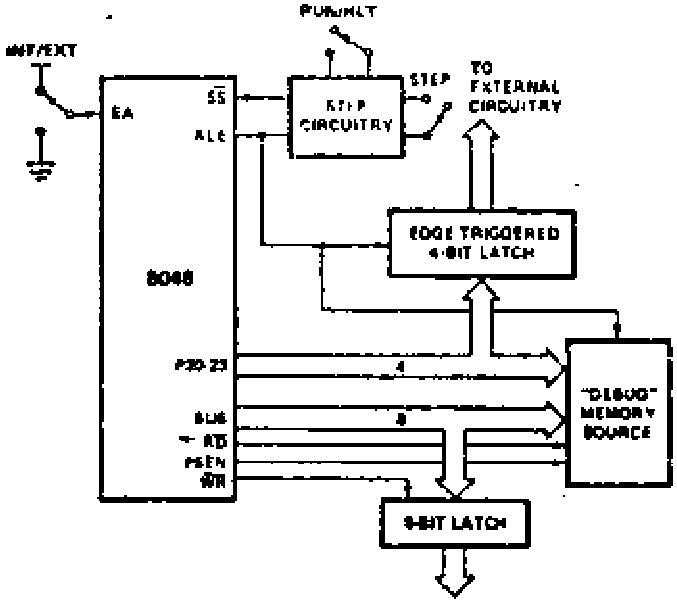
- *One period of ALE designates a cycle.*
- *Port 2 I/O data is restored after external memory fetch.*
- *Cycle timing is compatible to standard and custom memory and I/O devices.*

POWER DOWN SEQUENCE


POWER SUPPLY
PROCESSOR INTERRUPTED
POWER SUPPLY FAIL SIGNAL
NORMAL POWER ON SEQUENCE FOLLOWS
RESET
DATA SAVE ROUTINE EXECUTED
ACCESS TO DATA RAM INHIBITED

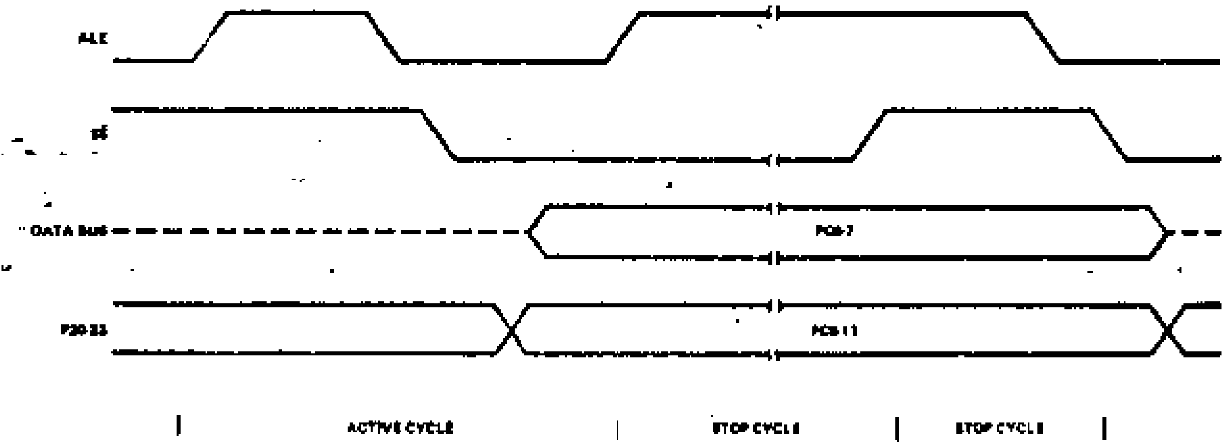
Notes:

- *Power down mode allows data to be maintained in 64 RAM registers while removing power from remainder of circuitry.*



- *EA pin allows for a separation of internal memory and CPU.*
- *All program execution can be routed outside to a debug or test memory for quick modification.*

SINGLE STEP TIMING



- *Single step allows processor to execute one instruction at a time. While stopped, address of next instruction is available externally.*

Notes:

SPECIAL INSTRUCTION SET FEATURES

FOR BCD ARITHMETIC:

- Decimal Adjust A
- Swap 4-bit Nibbles of A
- Exchange lower nibbles of A and Register
- Rotate A left or right with or without Carry

FOR LOOKUP TABLES:

- Load A from Page 3 of ROM (Address in A)
- Load A from Current Page of ROM (Address in A)

- *The 4 bit operations allow MCS-48 to perform with many of the advantages of the 4 bit microcomputers.*
- *Look-up tables facilitate the implementation of complex algorithms with a minimum of execution time.*

SPECIAL INSTRUCTION SET FEATURES

FOR LOOP COUNTERS:

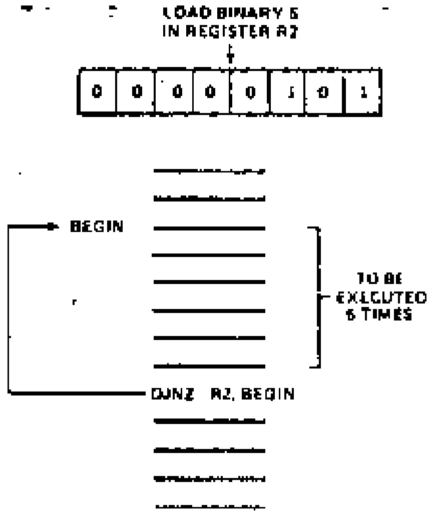
- Decrement Register and Jump if not zero.

FOR BIT MANIPULATION:

- AND to A (immediate data or Register)
- OR to A (immediate data or Register)
- XOR to A (immediate data or Register)
- AND to Output Ports
- OR to Output Ports
- Jump Conditionally on any bit in A

- *Bit manipulation essential for I/O operations and logical design with microcomputers.*
- *Loop counter allows program loops to be implemented.*

USING A LOOP COUNTER



EXPANSION CAPABILITIES

Notes:

Program Memory (to 4K words)
- Standard ROM
- Standard EPROM
- ROM and I/O
- EPROM and I/O

Data Memory (to 320 words)
- Standard RAM
- Low Power RAM
- RAM and I/O

Input/Output (unlimited)
- MCS-48 I/O Expander
- MCS-80 I/O Devices
- Peripheral Microprocessors

Special Interfaces
- MCS-80 Peripherals

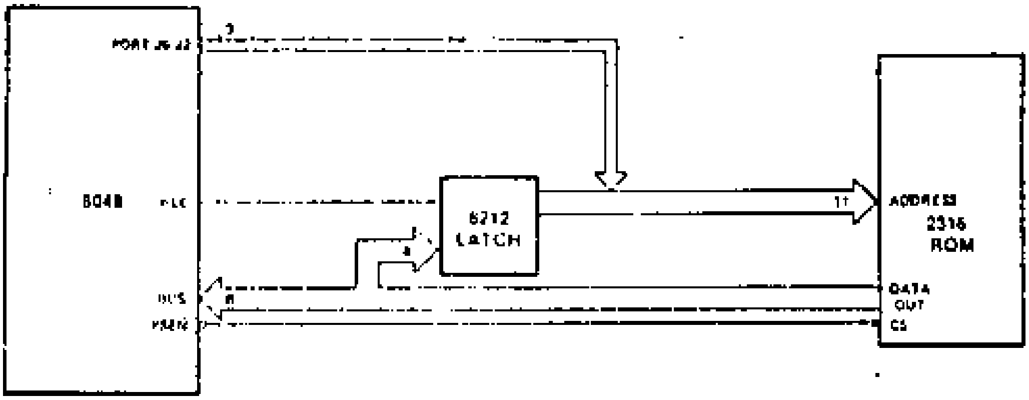
- *Full memory and I/O expansion provided without extra logic.*
- *Both standard and custom products available.*

EXPANDING PROGRAM MEMORY

8708	1K x 8 EPROM
8308	1K x 8 ROM
8755	2K x 8 EPROM with I/O
8355	2K x 8 ROM with I/O
8316	2K x 8 ROM

- *Both standard ROM/PROMs and custom ROM/PROMs are available for program memory expansion.*

EXPANDING MCS-48™ PROGRAM MEMORY USING STANDARD MEMORY PRODUCTS



USING 2K x 8 ROM

- *Basic 3K program memory system as shown.*
- *8212 demultiplexes address from bus.*
- *Port 2 contains high order address lines that are static for duration of the transfer, after which port data is restored.*

Notes:

17
/P

EXPANSION CAPABILITIES

- Program Memory (to 4K words)
  - Standard ROM
  - Standard EPROM
  - ROM and I/O
  - EPROM and I/O
- Data Memory (to 320 words)
  - Standard RAM
  - Low Power RAM
  - RAM and I/O
- Input/Output (unlimited)
  - MCS-48 I/O Expander
  - MCS-80 I/O Devices
  - Peripheral Microprocessors
- Special Interfaces
  - MCS-80 Peripherals

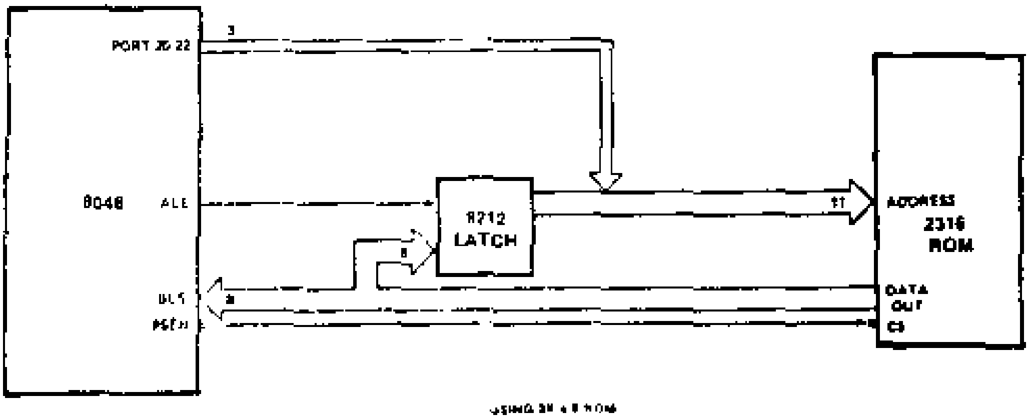
- *Full memory and I/O expansion provided without extra logic.*
- *Both standard and custom products available.*

EXPANDING PROGRAM MEMORY

8708	1K x 8 EPROM
8308	1K x 8 ROM
8755	2K x 8 EPROM with I/O
8355	2K x 8 ROM with I/O
8316	2K x 8 ROM

- *Both standard ROM/PROMs and custom ROM/PROMs are available for program memory expansion.*

EXPANDING MCS-48™ PROGRAM MEMORY USING STANDARD MEMORY PRODUCTS



- *Basic 3K program memory system as shown.*
- *8212 demultiplexes address from bus.*
- *Port 2 contains high order address lines that are static for duration of the transfer after which port data is restored.*

Notes: 19

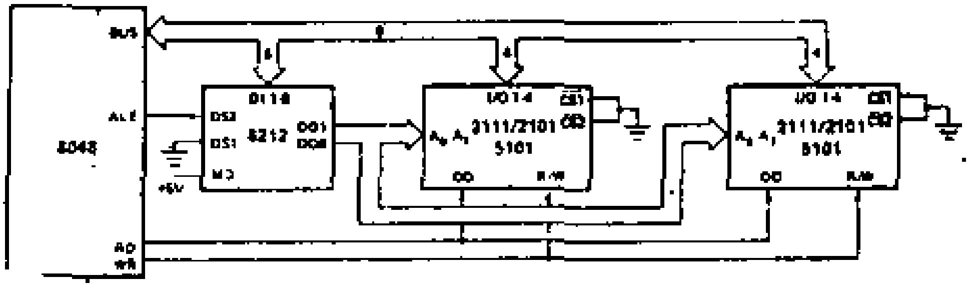
2C

EXPANDING DATA MEMORY

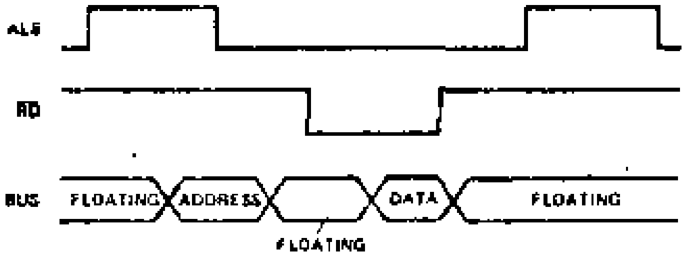
8101	256 x 4 Static RAM
8111	256 x 4 Static RAM
5101	256 x 4 CMOS Static RAM
8155	256 x 8 RAM/Timer/I/O

- *Compatible with most static RAMs, including low power 5101.*
- *8048 also contains a low power standby pin for the resident 64 x 8 data memory.*

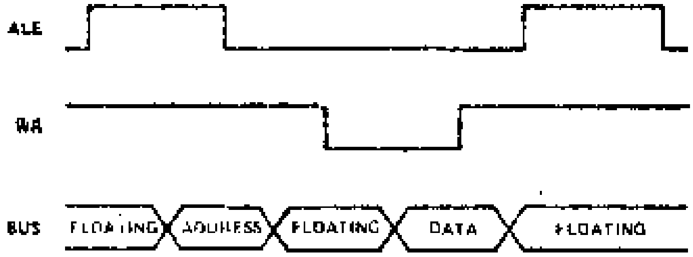
8048 INTERFACE TO 256 X 8 STANDARD MEMORIES



READ FROM EXTERNAL DATA MEMORY

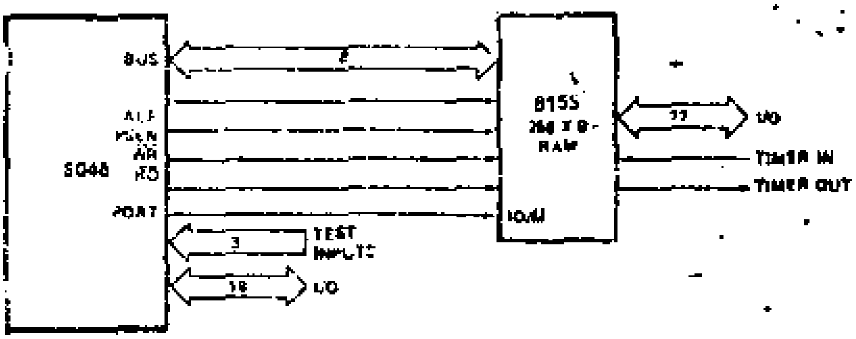


WRITE TO EXTERNAL DATA MEMORY



20 8048 INTERFACE TO 256 X 8 STANDARD MEMORIES

Notes:



- *Equivalent to five standard components.*
- *Several 8155s can be used to expand data memory.*
- *The timer feature is programmable, similar to 8253.*
- *The I/O is also programmable, similar to 8255.*

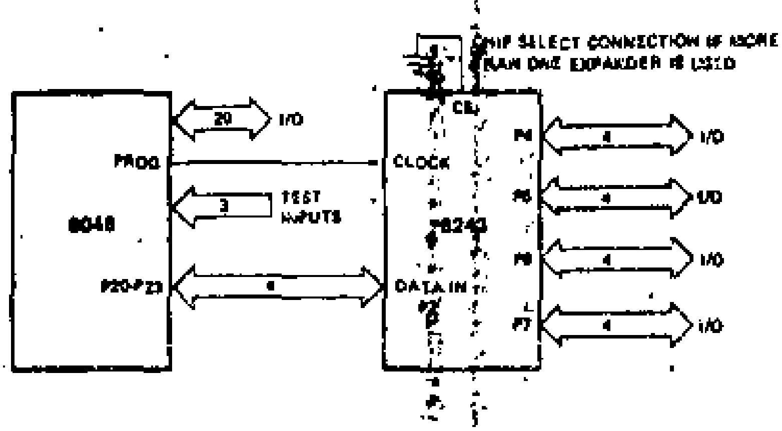
EXPANDING I/O

8243	I/O Expander (16 lines)
8212	I/O Latch (8 lines)
8255	General Purpose I/O (24 lines)
8155	RAM with I/O (20 lines)
8355	ROM with I/O (16 lines)
8755	EPROM with I/O (16 lines)

- *A large selection of devices are available to provide simple input and output facilities.*
- *Integrating I/O with memory provides for minimum package count.*
- *8255 provides optional I/O strobes.*

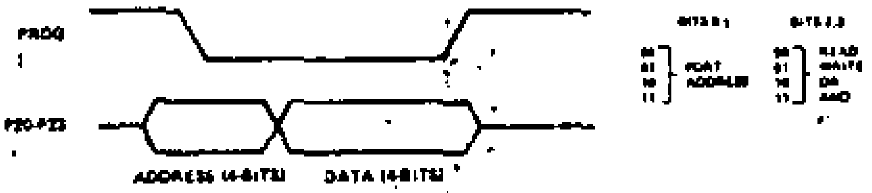
Notes:

EXPANDER INTERFACE



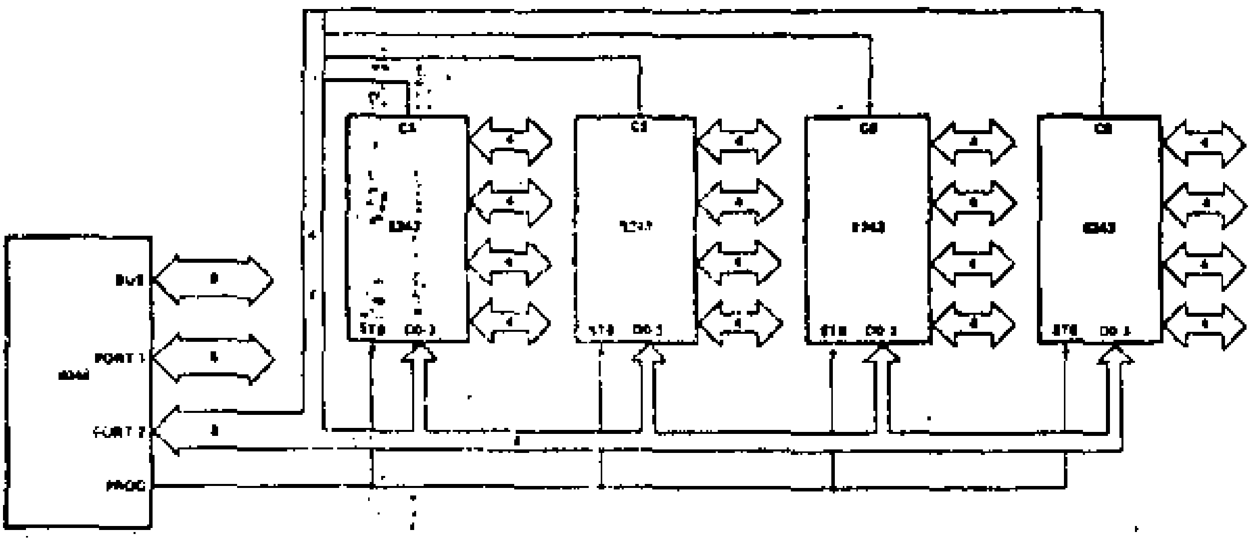
- Expansion can also take place on bus port.

OUTPUT EXPANDER TIMING



- Several 8243s may be appended to the port 2 4 bit bus. Addressing is accomplished via chip select.
- Program line strobes data.

LOW COST I/O EXPANSION

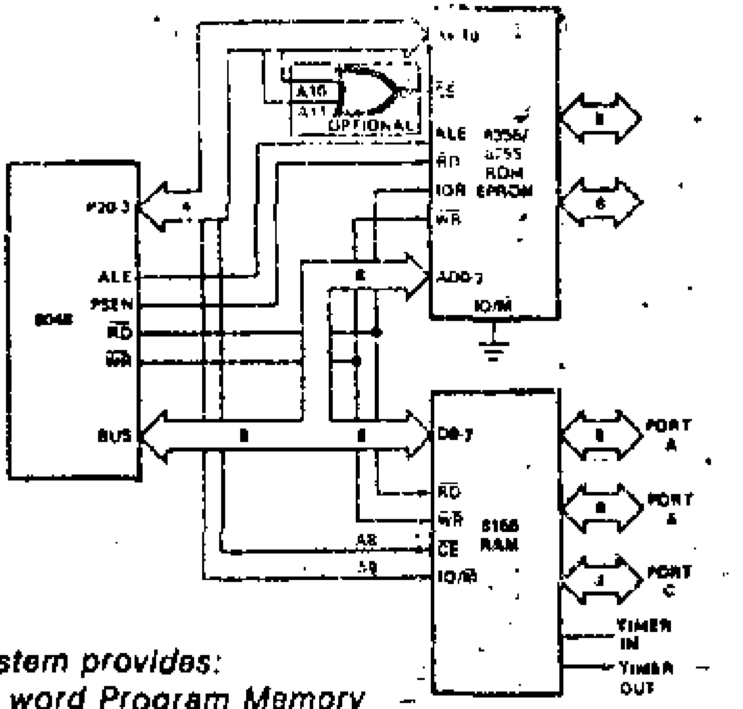


23

THE THREE COMPONENT MCS-48 SYSTEM

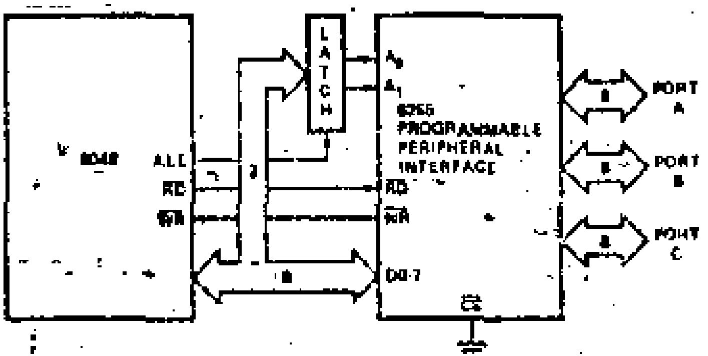
8155/8355

Notes:

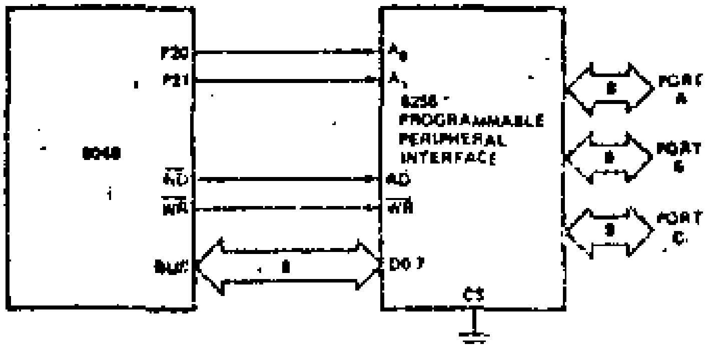


- System provides:
  3K word Program Memory
  320 word Data Memory
  53 I/O Lines
  2 Timer/Counters

INTERFACE TO MCS-80 PERIPHERALS



OPTION #1



OPTION #2

- OPTION #1 - Ports are addressed as data RAM
- OPTION #2 - Ports are addressed via output port

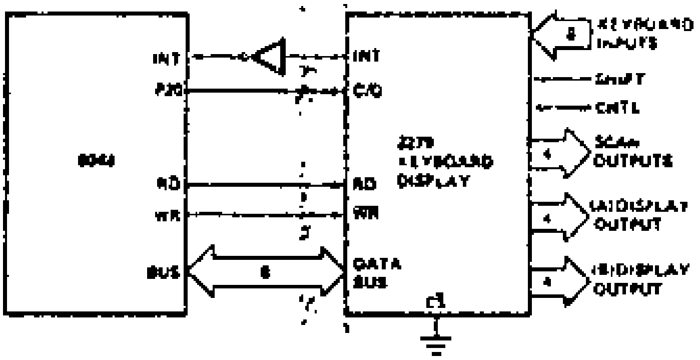
SPECIAL PERIPHERALS

Notes:

8214	Priority Interrupt
8251	USART
8279	Keyboard/Display
8253	Interval Timer

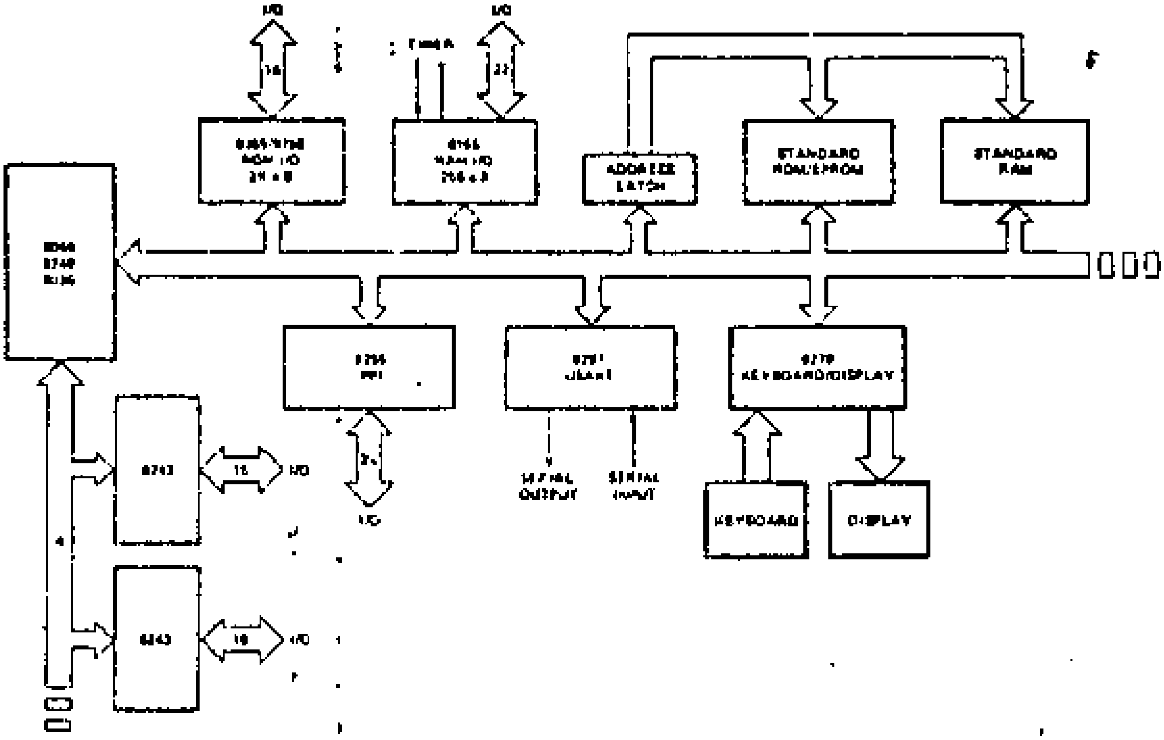
- *While most MCS-80 peripherals are MCS-48 compatible, the ones shown are the most popular.*
- *Peripherals reduce system cost by providing low component count and specialized interfaces.*

KEYBOARD/DISPLAY INTERFACE



- *Provides easy interface to a 64 key matrix and two 16 digit displays.*

MCS-48 EXPANSION CAPABILITY

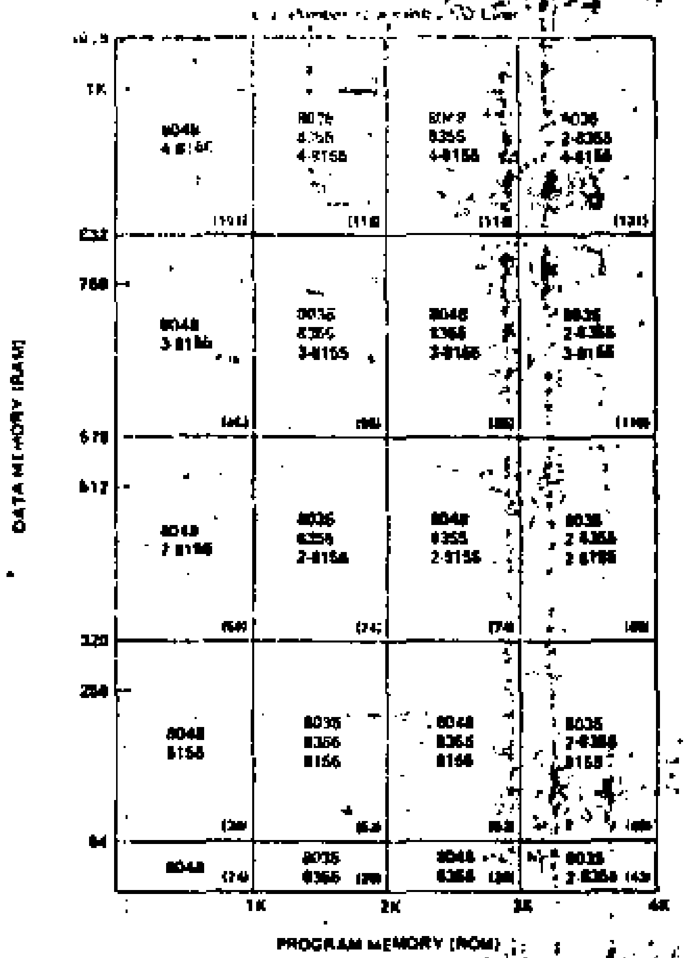


- *All modes of expansion may be used simultaneously.*

THE EXPANDED MCS-48 SYSTEMS

Notes:

25



- *The 8035 allows the user to match his program memory requirements exactly.*

DEVELOPMENT SUPPORT

- **INTELLEC® Assembler**
- **UPP PROM Programmer**
- **PROMPT 48**
- **ICE-48™**
- **User's Library**
- **Application Engineers**
- **Training Courses**

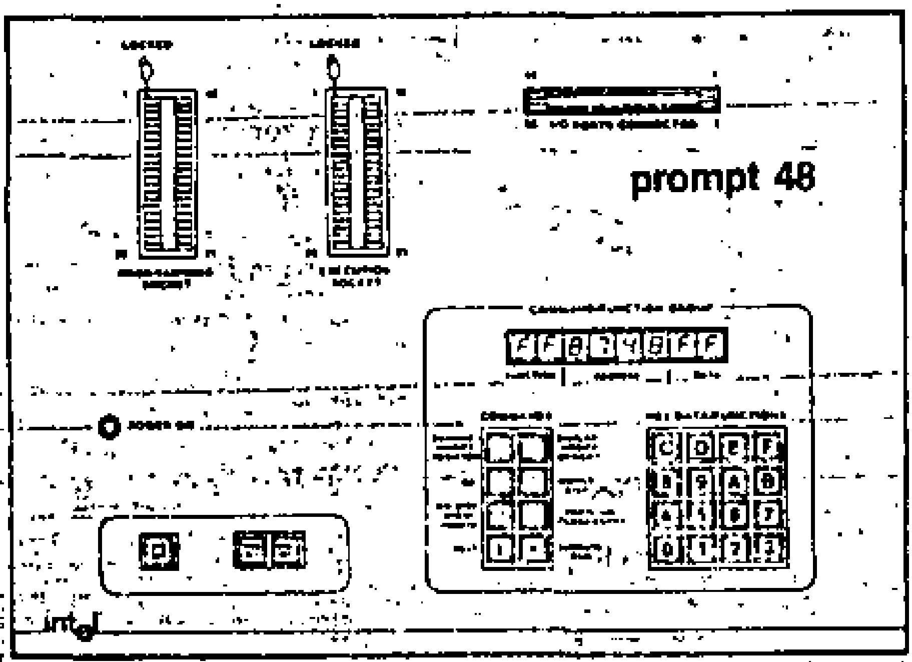
- *Development support is as important as components.*
- *MCS-48™ training is available and affords the best opportunity to learn the family.*

Notes:

PROMPT 48 FEATURES

- 8748 EPROM Programmer
- Hex Keyboard/Display
- Real Time Execution
- Single Step
- 8 Break Points
- Examine/Modify all Internal Registers
- TTY Interface (or RS232)
- Hex Tape Load/Dump (TTY)
- All I/O Ports available to User.

- *Low cost.*
- *Self contained bench top enclosure.*



INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM FEATURES

- Resident MCS-48 Macro Assembler
- Universal PROM Programmer Module
- ICE-48™
- Disk Operating System
- High Speed Peripherals

- *A self contained microcomputer development laboratory.*

Two versions of 16-bit chip span microprocessor, minicomputer needs

Larger, 48-pin package addresses 8 megabytes of memory; regularity of instruction set makes programming easy

by Masatoshi Shima, Zilog Inc., Cupertino, Calif.

□ A microprocessor would gain ready acceptance if it could fit immediately into applications of current 8- and 16-bit microprocessors and at the same time have an advanced architecture that was expandable to ensure long product lifetime. The Z8000 from Zilog Inc. meets the first goal easily—and does so with 10 times the throughput of existing microprocessors. To meet the second goal, the Z8000 has departed from the traditional byte-oriented microprocessor design and moved toward the more regular architecture of minicomputers.

With many of the architectural features of minis and some pluses as well, the Z8000 is designed for minicomputer as well as microcomputer applications. To begin with, it handles seven data types, from bits to word strings, and offers eight selectable addressing modes. Its 81 distinct operation codes combine with the various data types and addressing modes to form a rich 414-instruction set more powerful than that of most minicomputers. Moreover, the set exhibits a high degree of regularity: more than 90% of the instructions can use any of five main addressing modes with 8-bit byte, 16-bit word, and 32-bit long-word data types.

Among its architectural resources are a large number of on-chip registers—24 16-bit registers in all—that dramatically reduce the number of memory references needed in programming. Sixteen of those registers are general-purpose, and all except one can be used as index registers without restrictions.

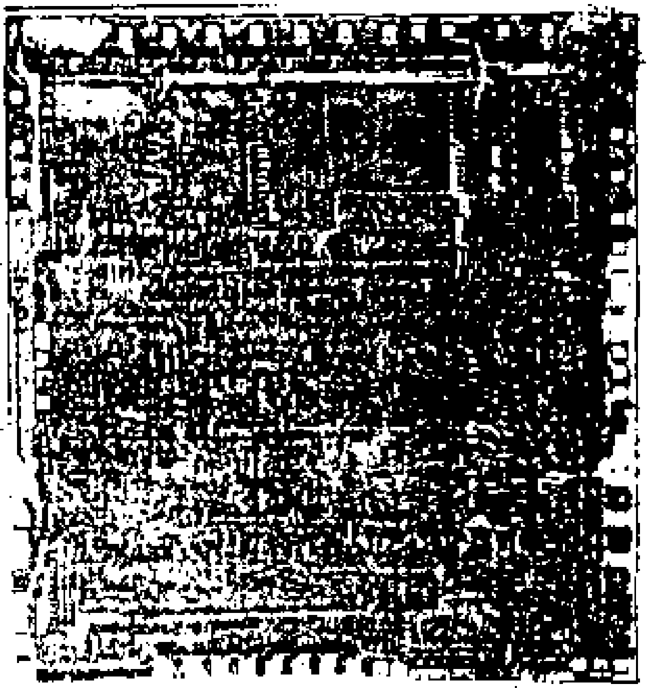
Also aiming at minicomputer applications is the Z8000's large direct-memory-addressing capability of 8 megabytes. Instead of treating it as linear space, however, the Z8000 organizes memory into a set of 128 segments of up to 65,536 bytes each. A segmented space is closer to the way the programmer uses memory—each procedure and data space, either local or global, resides in its own segment. To further facilitate use of all that space, a memory-management chip will work with the Z8000 in performing the dynamic relocation and memory protection needed in a large system.

Two versions

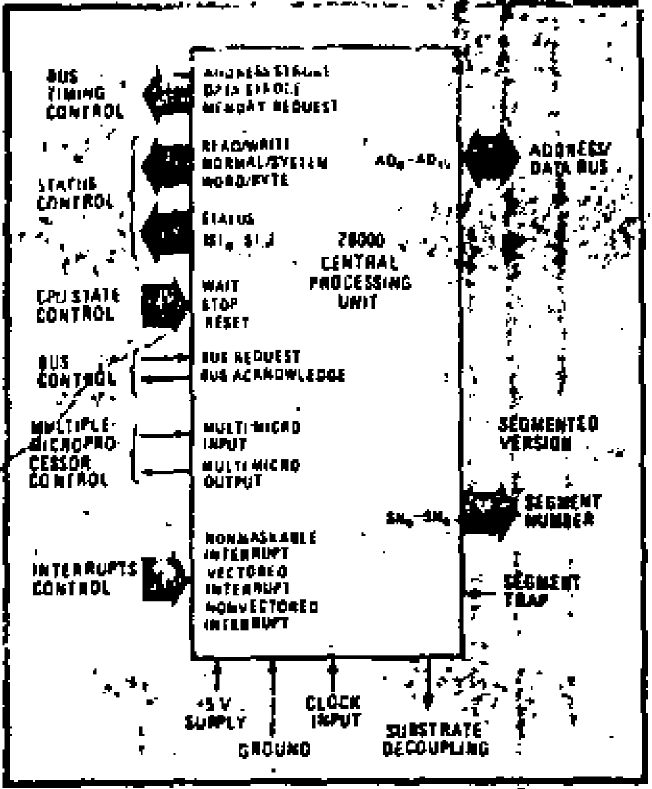
But because the Z8000 must satisfy existing microprocessor needs, two versions are offered. Besides the 48-pin memory-segmented version with 23 lines that addresses 8 megabytes, a 40-pin chip is offered with 16 lines to address 64 kilobytes—the equivalent of one segment. Expansion is guaranteed from the 40-pin to the 48-pin version: the segmented Z8000 can run any nonsegmented code in any one of its 128 segments using a load-program-status instruction.

Finally, the Z8000 boasts two operating modes, system and normal, that keep operating-system and applications programming separate, as in computer systems. Each mode has a separate stack, and the arrangement isolates global features like privileged instructions from normal programming.

An n-channel metal-oxide-semiconductor chip built with scaled-down depletion-load silicon-gate technology, the Z8000 squeezes about 17,500 transistors into an area of 238 by 256 mils. Its density—148 gates per square millimeter—surpasses that of previous microprocessors (see also "Genealogy of the Z8000," p. 83). The chip uses a 5-volt supply and requires a single-phase 4-megahertz (250-nanosecond) clock for timing. As at



Dense. The 16-bit Z8000 central processing unit is built with scaled n-channel depletion-load silicon-gate technology. The chip, which crams about 17,500 transistors into a 238-by-256-mil or 39.3-millimeter-square area, has a density of about 148 gates/mm².



1. Two versions. The Z8000 16-bit microprocessor comes with its 40-pin nonsegmented version, which has 16 lines for directly addressing 64 kilobytes of memory. The minicomputer-like 48-pin version adds 7 segment-address lines; it can address 8 megabytes.

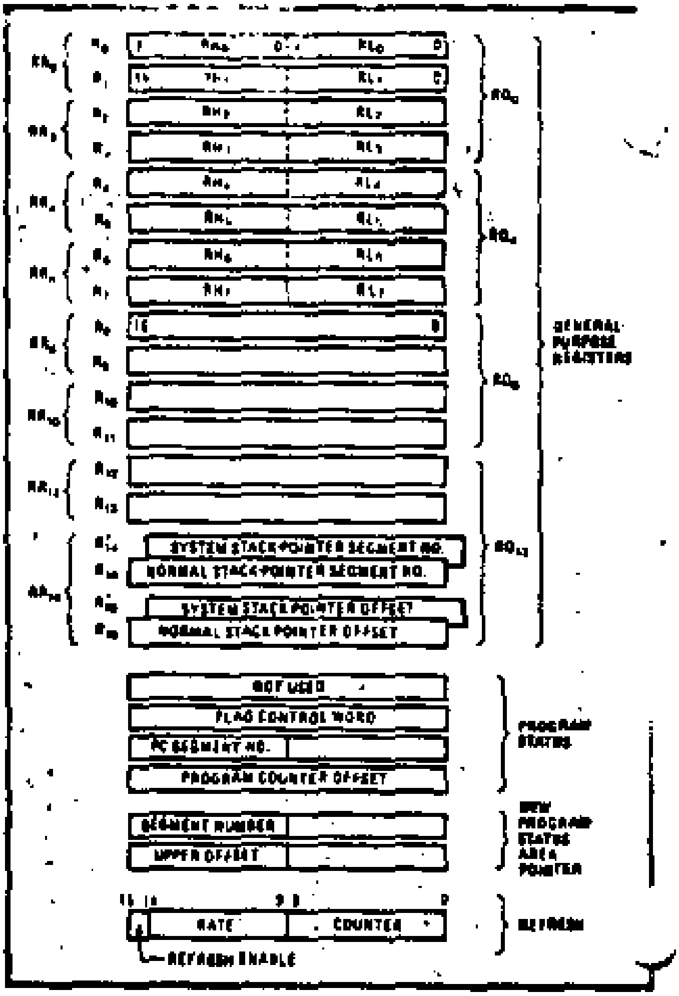
least three clock cycles in the central processing unit are required for one memory cycle, the Z8000 needs memory devices with a cycle time of 750 ns and an access time of 430 ns.

As shown in Fig. 1, the Z8000 has in addition to its address and data buses and clock and power-supply inputs six types of control buses: bus-timing, status, CPU-state, interrupt, bus, and multiple-microprocessor control. The three bus-timing control outputs coordinate the data flow over the chip's address/data lines. An address strobe signals that addresses are valid, and a data strobe times the window for valid data in and out of the CPU. The memory-request line is a timing signal that eases interfacing to dynamic memory.

CPU status

The next bus provides information on the CPU's status. A read/write line gives early status of the forthcoming cycle, while a normal/system line indicates which of the two modes the CPU is in for the current cycle. A word/byte line indicates whether the CPU is accessing 16 bits of data or 8 bits. The four status-control lines form a 4-bit word that indicates several bus statuses, including memory-request, stack, first- and subsequent-word instruction fetch, interrupt acknowledgments, internal operation, and others.

The next of the control buses are three CPU-state inputs. The reset line initializes the CPU. A wait line signals the CPU that data transfer is not ready. The stop line halts internal CPU operation (although dynamic memory is still refreshed). The CPU can be stopped each time the first word of an instruction is fetched.



2. Registers. Sixteen 16-bit registers are organized into high and low bytes (RH and RL), 32-bit long-words (RR), and 64-bit quad-words (RQ). Four words, including the program counter, contain the program status and two more point to the new-program-status area.

A pair of lines governs the control of all the Z8000's buses. Driving the bus-request input low instructs the CPU to put all its address/data, bus-timing, and status-control lines into a high-impedance state so that other devices can use them. The CPU signals it has relinquished control with its bus-acknowledge output.

Another pair of lines is used with certain instructions to coordinate multiple-microprocessor systems. The multi-micro output line issues a request, while the input line recognizes outside requests. Thus any CPU in a multiple-microprocessor system can, for example, exclude all other asynchronous CPUs from being able to access a critical resource.

Finally, there are three interrupt inputs and, in the segmented version of the Z8000, a trap input. Interrupts are asynchronous events triggered typically by peripherals needing the CPU's attention, and traps are synchronous events resulting from the execution of specific instructions that occur each time the instruction is executed with the same set of data. The two are handled in a similar fashion by the Z8000.

The Z8000 is a register-oriented machine, placing little constraint on the use of its 16 general-purpose

Genealogy of the Z8000

The changes in microprocessor architecture from the first-generation 8-bit devices to the fourth-generation Z8000 have been both swift and dramatic. Developments have been guided alternately by technology limitations and by the hardware and software demands of users.

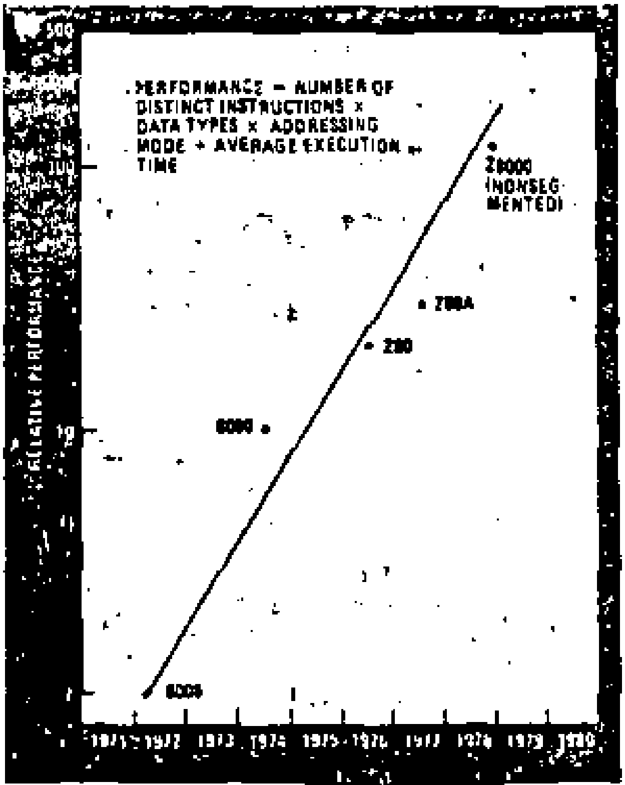
Thus, the shortcomings of the first 8-bit microprocessor, the 8008 developed in 1971, were technological. Metal-oxide-semiconductor processing was relatively new and set limits to circuit complexity. Also, microprocessors were actually offshoots of calculator designs, being developed by semiconductor and not computer houses. So performance and features left much to be desired.

Advances in processing technology gave microprocessor designers a powerful tool with which to build the next generation of microprocessors—n-channel silicon-gate MOS that boosted circuit speeds by a factor of four over previous p-channel technology. Thus, the 8080, born in 1974, began the second generation of microprocessors.

By the time the third generation of microprocessors entered the design stage, users had become more sophisticated and were involved in high-level languages. Data-processing applications grew in popularity, and the disk-operating system was introduced. It was those software requirements that indicated the areas needing improvement, and the Z80 addressed the problem with software-oriented features. It added a large number of new instructions and a second register set, two index registers, and better interrupt handling. Still, because the Z80 maintains source-code compatibility with the 8080, many critical bottlenecks were inherited.

The Z80 marked the final exploitation of the original microprocessor structure and instruction format. Attempts to add capabilities would require two or three 8-bit instruction fetches—and exceedingly poor use of memory bandwidth and space. Moreover, the increasing popularity of high-level languages, plus a demand for much larger addressing space fueled by the plummeting costs of memory, outstripped the capabilities of an 8-bit microprocessor. The various trends toward large programs, complex distributed intelligent systems, and advanced memory management all pointed to a 16-bit architecture. But it was Zilog Inc.'s conclusion that a chip with minicomputer performance could not last a decade without 32-bit operations and memory segmentation. Thus it chose the more advanced approach of the Z8000.

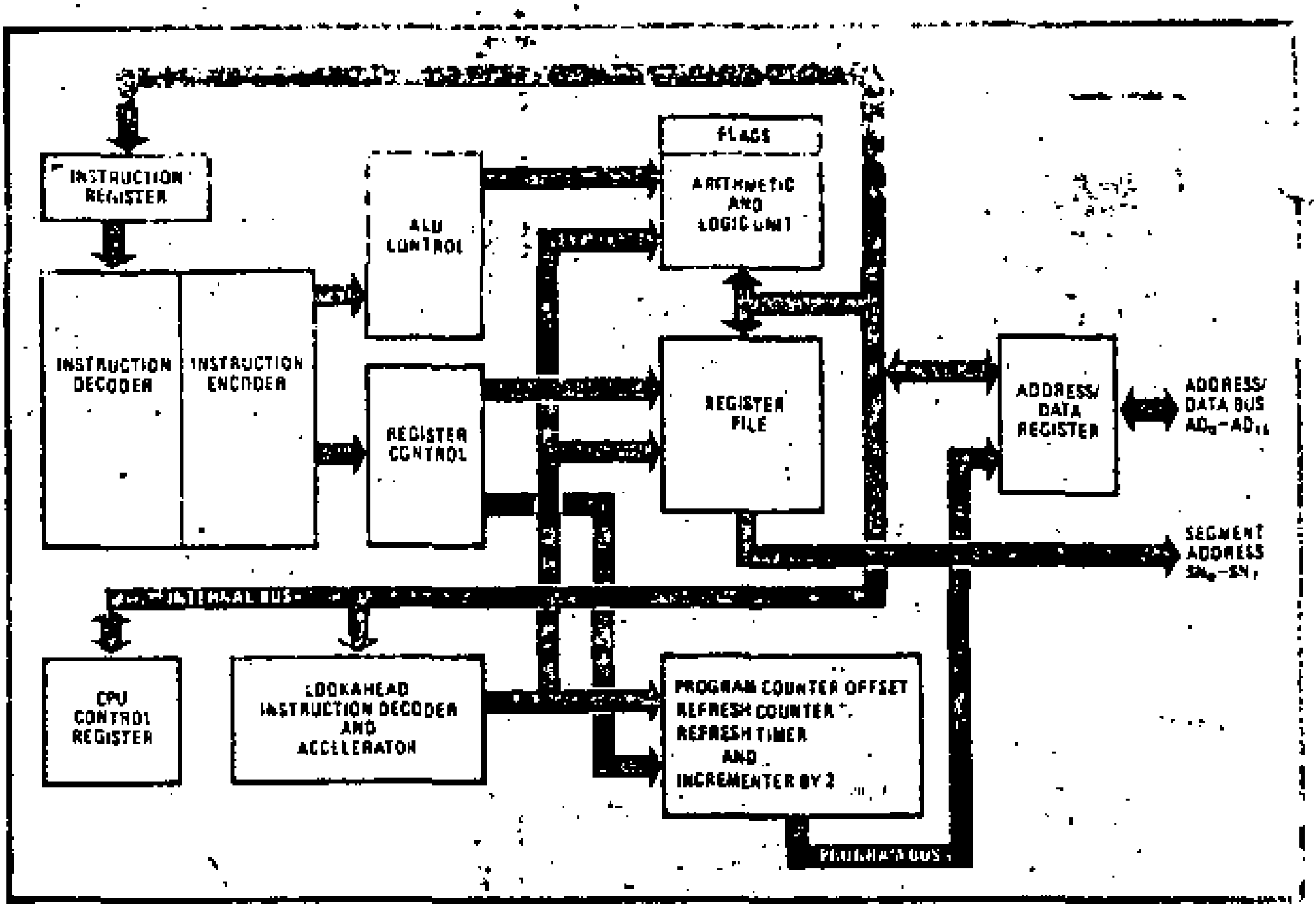
The table compares microprocessors. The companion graph indicates relative performance; though the equation provides no absolutes, it serves as an indicator for both hardware and software since it takes instructions, addressing, data types and speed into account.



COMPARISON OF MICROPROCESSOR CHARACTERISTICS

	8080	Z80	Z80A	Z8000
Date of initial production	1974	1976	1977	1978
Power consumption (W)	1.2	1.0		1.6
Number of transistors	4,800	8,200		17,500
Number of gates	1,600	2,733		5,833
Chip size (mm^2)	22.3	27.1	22.4	39.2
Density (gates/mm^2)	72	101	122	148
Number of distinct instructions*	34	52		81
Combination of number of distinct instructions and data types*	39	60		149
Combination of number of distinct instructions, data types and addressing modes*	65	178		414

*The numbers represent a conservative counting method. The user sees much larger number of instructions in assembly language notation.



3. Architecture. The Z8000 boosts throughput with a look-ahead instruction decoder and accelerator on its internal bus. Thanks in part to the regularity of the instruction set, an instruction actually begins execution while it is entering the instruction register.

registers. Indeed, with but one exception (the stack pointer), no registers are ever implied in an instruction and none whatever have special restrictions. Bottlenecks found in early microprocessor designs, like dedicated accumulators, are thus avoided, so that programming is efficient and straightforward. All 16 of the 16-bit registers (R_0–R_{15}) can be used as accumulators. All except R_0 can be used as index registers, base registers, and as memory pointers for indirect addressing.

A flexible register architecture

As shown in Fig. 2, the flexibility of the registers is afforded by a unique arrangement of overlaps and pairs. The 16 8-bit registers (RH_0–RH_7 and RL_0–RL_7), all of which may be used as accumulators, are overlapped with the first eight 16-bit registers (R_0–R_7). The eight 32-bit long-word registers (RR_0–RR_{14}) are register pairs, and the four 64-bit quad-word registers (RQ_0–RQ_{12}), which are used by a few instructions such as multiply, divide, and extend sign, are register quadruples.

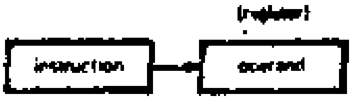
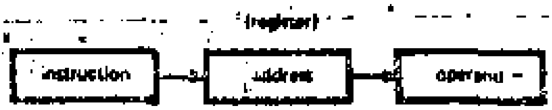
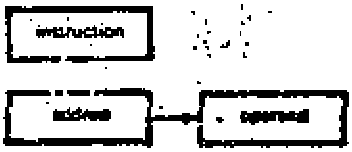
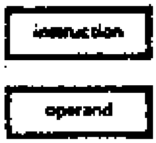
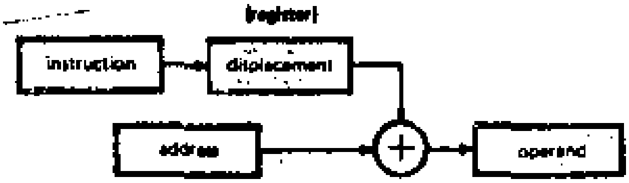
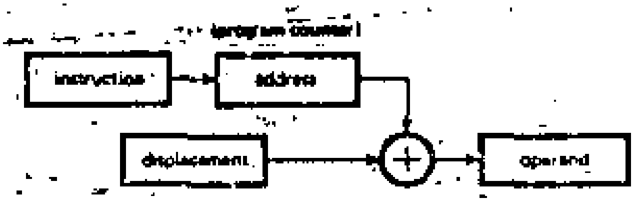
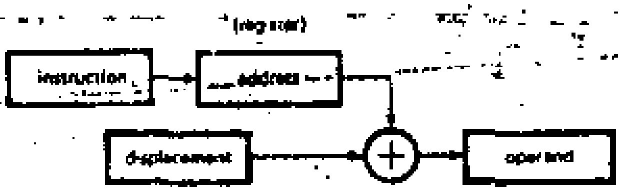
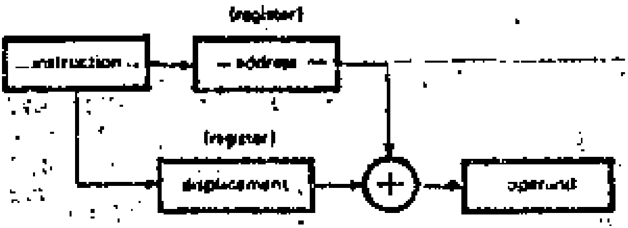
In the nonsegmented version of the chip, the last 16-bit general-purpose register, R_{15}, is the stack pointer. In the segmented version, the last two registers, R_{14} and R_{15} (or long-word register RR_{14}), are needed to hold the stack pointer, with R_{14} storing the segment number while R_{15} contains the offset. The only instructions that use the stack pointer exclusively are call, call relative, return, and return from interrupt; the push and pop instructions can use any register as a stack pointer. However, all instructions can manipulate the stack pointer, since it is in the general-purpose register group.

The two running modes of the Z8000 each have a copy of the stack pointer—one for the system mode and another for the normal mode—as implied by the primed registers R_{14}' and R_{15}' in Fig. 2. Although the stacks are separated, the normal stack registers can be accessed in the system mode by using the load-control-word instruction. Having two sets of stack pointers facilitates task-switching when interrupts or traps occur. The normal stack is always kept clear of system information, since the information saved on the occurrence of interrupts or traps is always pushed on the system stack before the new program status is loaded.

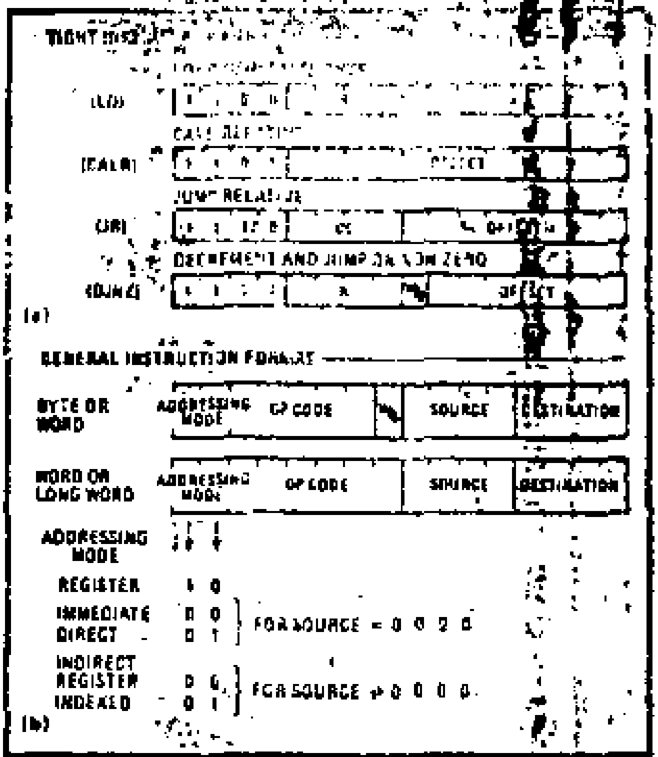
In addition to the general-purpose registers, there are the program-status registers, which contain the flags, control bits, and program counter. In the 40-pin nonsegmented version of the Z8000, the program status is held in two 16-bit registers: the first is the flag and control word, the second is the program counter. In the segmented version, program status is a full four words: the flag and control word, a two-word program counter, and a word reserved for future use.

Another register holds the pointer for the new-program-status area. It comprises two words in the segmented version and one word in the nonsegmented version. Lastly, a refresh register contains a 9-bit counter for automatic refresh of dynamic memories.

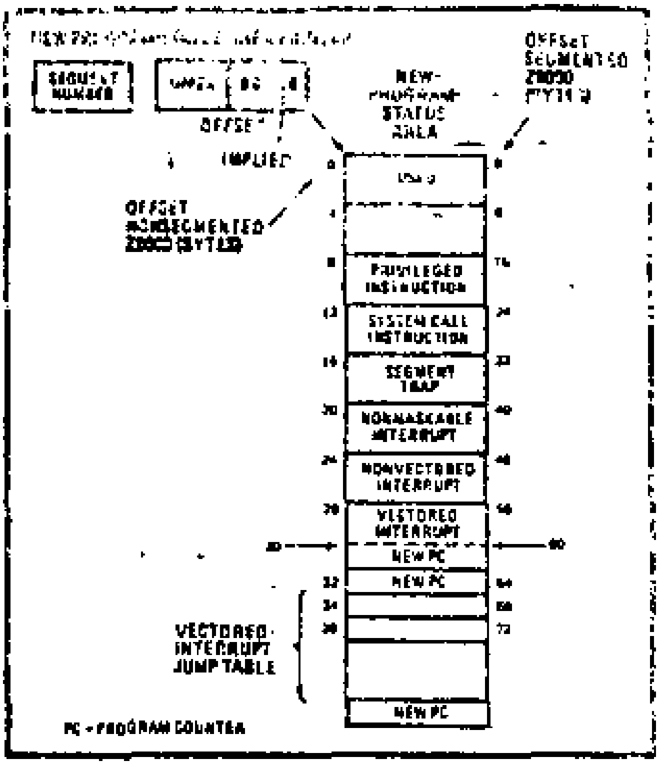
The Z8000 executes instructions by stepping through

Mode	Diagram	Operand value
Register		the content of the register
Indirect register		the content of the location whose address is in the register
Direct address		the content of the location whose address is in the instruction
Immediate		in the instruction
Index	 (register) instruction displacement address operand	the content of the location whose address is the address in the instruction, offset by the content of the working register
Relative address	 (program counter) instruction address displacement operand	the content of the location whose address is the content of the program counter, offset by the displacement in the instruction
Base address	 (register) instruction address displacement operand	the content of the location whose address is the address in the register, offset by the displacement in the instruction
Base index	 (register) instruction address (register) displacement operand	the content of the location whose address is the address in the register, offset by the displacement in the register

Many modes. Over 90% of the Z8000's instructions work with any of five main addressing modes, proof of the chip's software regularity. A load-addressing instruction that accepts all eight modes accommodates any other operand addressing scheme desired



4. Instruction formats. Tight format (a) uses only one 16-bit word, is executed quickly, and saves memory. General format (b), used for bytes, words, or long words, specifies the addressing mode using the 2 address-mode bits and the source register number.



6. Program status. The new-program-status area pointer is two words long; the 7 most significant bits in the second word specify the beginning of an area in memory from which the new program status is fetched in response to interrupts and traps.

a set of the following basic machine cycles: memory read or write, input/output-device read or write, and internal data execution. Since the memory cycle uses three clock cycles to fetch the instruction or data from memory or to write data into memory, each machine cycle also uses a minimum of three clock cycles, though for complex operations it can extend to as many as eight.

The matter of timing

Ideally, for optimum throughput, all instruction time should be memory-cycling time; no clock cycles should be wasted on other phases of the instruction cycle. Simulations of a wide variety of benchmark programs have shown that, on the average, the effective memory cycle time (also called the bus-utilization time or bus efficiency) of the Z8000 is 80% to 85% of the instruction time and up to 90% if jump instructions are excluded. This efficiency is a significant improvement over the 65% to 70% of the 8-bit Z80 microprocessor.

One reason for the high efficiency of the Z8000 is its look-ahead instruction decoder and accelerator, shown in the architectural block diagram of Fig. 3. Since the look-ahead is tied to the internal bus, and since the instruction set is very regular, an instruction can actually begin execution while it is still being stored in the instruction register. The look-ahead makes for a significant improvement in throughput, for example, in the case of direct and indexed memory addressing (the most frequently used addressing modes after register address), in which the Z8000 does not require any additional clock cycles to decode the instruction in deciding whether it is short or long offset. The load-register-to-register instruction has been optimized to require only the three clock cycles of its memory access. In most instructions, in fact, the data-manipulation time is fully overlapped with the fetching of the first word of the next instruction.

Throughout the design of the Z8000, meticulous attention was paid to accelerating and optimizing each instruction in proportion to its statistical importance. Some instructions and data references are aligned in a single word to speed execution, simplify logic, and get a larger range when the relative addressing mode is used.

To further increase execution speed, as well as to reduce memory usage, the most frequently used instructions in the Z8000 have beed coded as one word. Among these are jump relative, decrement and jump on nonzero, load immediate byte, load immediate word, and call relative. Moreover, the sophisticated, interruptible, preprogrammed block and string instructions can execute memory-to-memory data manipulations as fast as 888,000 bytes per second.

Extra instructions

A number of powerful instructions not found on previous microprocessors were added to the Z8000 repertoire. There are those that handle the new data types—instructions like multiply and divide that manipulate 32-bit long-words—and other instructions that load and store multiple words. And there are instructions that increment and decrement the contents of any register or memory location by any number from 1 to 16. Finally, multiple addressing modes for the push, pop, load, and store instructions enhance performance.

An important part of microprocessor design is the

The Z8000 family

Even a minimum system based on a Z8000 microprocessor executes instructions fast and is easy to program. Soon to be available is a family of associated chips that will extend these advantages to complex Z8000-based computer systems and networks.

The members of the family include:

- The Z-MMU memory-management unit that takes care of memory segmentation and protection and address translation.
- The Z-UPC universal peripheral controller, a Z8 single-chip microcomputer used as a general-purpose programmable peripheral device with the Z8000.
- The Z-CIO counter and parallel-I/O chip, with three programmable 16-bit counters, two 8-bit bidirectional I/O ports, and a 4-bit I/O port.
- The Z-SIO serial-I/O circuit, which has two full-duplex channels and is capable of handling asynchronous and bisynchronous protocols at data rates of up to 880 kilobytes per second.
- The Z-MBU microprocessor buffer unit, a 128-by-8-bit first-in, first-out buffer that can be cascaded to any depth to connect asynchronous parallel processors to the Z8000 when multiprocessing.
- The Z-FIFO first-in first-out buffer memory, also 128 by 8 bits, for expanding the Z-MBU depth or interfacing I/O ports to user equipment.
- The Z-bus RAMs, a pair of random-access memory chips—a 2,048-by-8-bit fully static RAM and a self-refreshing 4,096-by-8-bit pseudo-static RAM—for small local storage.

instruction format, since logic complexity (and hence chip size) depends heavily upon its complexity. Designing into the instruction set total software regularity (where all instructions can use all data types and addressing modes) is ideal, and that goal is one towards which the Z8000 has striven.

Of the eight selectable addressing modes (see table), the five main modes—register, indirect register, immediate, direct address, and indexed address—can be used with nearly all instructions, excepting a few such as rotate and shift instructions. The three other addressing modes—relative address, base address, and base indexed address—have been added to all load and store instructions. To save memory space, the relative addressing mode applies additionally to jump, call, and decrement and jump on non-zero instructions. Some instructions have built-in autoincrementing and auto-decrementing addressing modes. Finally, a load-address instruction, which can use all of the eight addressing modes, supports even the most sophisticated operand-addressing schemes.

Instruction formats

The formats for Z8000 instructions are shown in Fig. 4. The 2 most significant bits in the instruction word determine whether the tight instruction format (a) or the general instruction format (b) is used. Use of the tightly coded instruction—a single word—reduces instruction-memory usage and speeds execution.

As long as the 2 most significant bits are not both 1s, the general instruction format applies. Those 2 bits in conjunction with the source-register field in the instruction are sufficient for specifying any of the five main addressing modes. As shown in Fig. 4b, an all-zero source specification distinguishes immediate or direct addressing from indirect and indexed addressing, both of which require a source register. Source and destination-register fields in the instruction format are 4 bits wide for addressing the 16 general-purpose registers.

The Z8000 does not have memory-to-memory arithmetic instructions. However, it performs memory-to-memory transfers on a sophisticated set of preprogrammed block-transfer and string-manipulation instructions and offers store immediate, push immediate, and compare immediate instructions. That arrangement provides a more compact instruction format with more op codes available for additional instructions than would be possible with the general memory-to-memory addressing mode used in the Digital Equipment Corp. PDP-11 minicomputer, which has two sets of addressing modes and register fields.

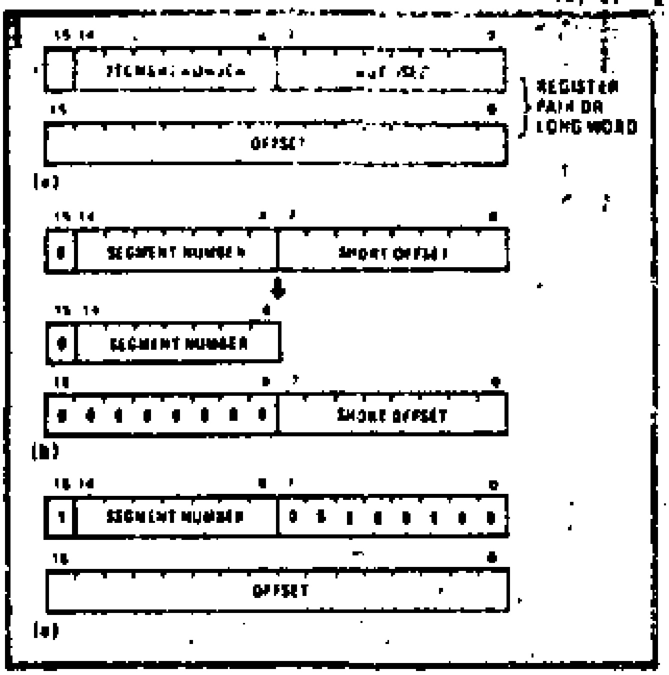
Interrupts and traps

The Z8000's seven interrupts and traps, both internal and external, are arranged in priority. The three interrupts are all external inputs: nonmaskable interrupt, vectored interrupt, and nonvectored interrupt. The vectored and nonvectored interrupts are maskable. Of the four traps, the only external one is the segment input, which is found in only the 48-pin segmented version of the chip. The remaining three traps occur when certain instructions limited to the system mode are called in the normal mode, or for the system-call instruction, or for an illegal instruction. The descending priority order of the traps and interrupts is: internal traps, nonmaskable interrupts, segment trap, and vectored and nonvectored interrupt.

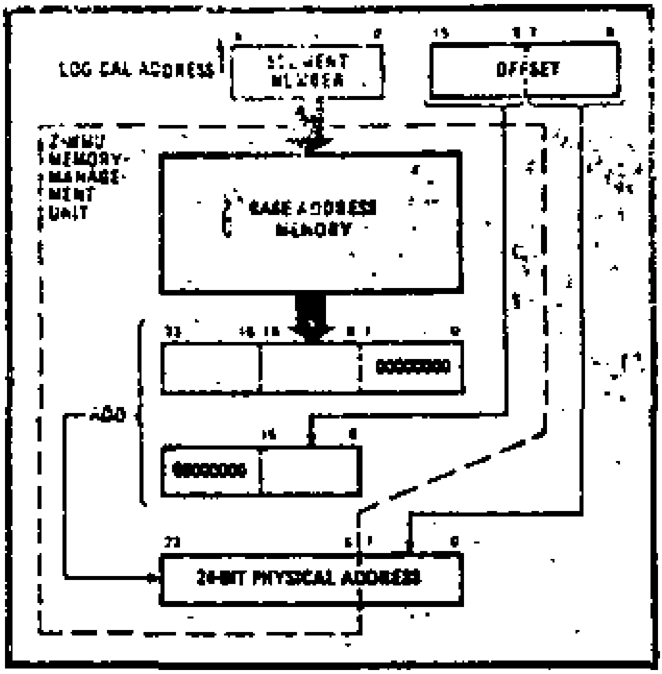
When an interrupt or trap occurs, the program status, which is contained in two 16-bit words in the nonsegmented version and three words in the segmented version, is pushed onto the system stack followed by an additional word. This extra word typically indicates the reason for the occurrence.

In the case of an internal trap, the reason word is the first word of the trapped instruction. In the case of the segment trap and for all interrupts, the reason is the vector on the data bus that is read by the CPU during the interrupt- or trap-acknowledge machine cycle.

The previous program status thus having been pushed on the system stack, a new program status is fetched from the new-program-status table (Fig. 5) that is specified by the new-program-status area pointer. As in Fig. 2, that pointer is the most significant byte in the new-program-status area pointer register. In the case of the segmented version of the Z8000, the pointer is two words in all—the segment number is specified by the 7 most significant bits of its second word. After the interrupt or the trap has terminated, a reset sequence is entered. A new program status is then fetched from a fixed location



6. **Address representation.** Segmented addresses appear as a long word (32 bits) when represented in a register or in memory (a). In an instruction, however, an address can be a single word (b) or a long word (c) if it is within the first 256 locations of a segment.



7. **Memory management.** The Z-MMU memory-management chip carries out the memory relocation from logical to physical address by adding the 16-bit offset value to a 24-bit base address associated with each segment. Two Z-MMUs handle all 128 segments.

in memory at the beginning of segment 0.

Facilitating the separation of operating-system programming from applications programming are the system and normal operating modes of the Z8000. The distinction is made by privileged instructions, which can only be executed in the system mode and are trapped when encountered in the instruction flow of normal-mode operation. Those instructions include all input/output instructions, halt, enable/disable interrupt, load control word, store control word, load new program status, return from interrupt, and all multiple-microprocessor instructions.

High-level languages, sophisticated operating systems, large programs and data bases, and decreasing memory prices are all accelerating the trend towards larger memory space in microcomputer systems. But even when it is available, questions are raised: how is it best accessed by a programmer? and what memory-management mechanism best allows the system to manage its memory on the user's behalf? In answer, the Z8000 proposes a segmented addressing scheme.

The segment number is an unsigned 7-bit integer ranging from 0 to 127; the offset is an unsigned 16-bit integer ranging from 0 to 65,535.

When represented in a register, a segmented address is always a register pair or long word (Fig. 6a). The two words may be manipulated separately or together by any of the word and long-word register operations. All segmented addresses exist in memory as a long word.

A segmented address in an instruction, however, has two different forms: either with a long offset (Fig. 6c), in which the address occupies two words, or with a short offset that is one word. The short offset, which, as shown in Fig. 6b, implies that the most significant 8 bits of the offset are all zero, can be used whenever the address is within the first 256 locations of a segment. That representation permits very dense encoding of addresses and is convenient not only for indexed addressing, but for direct addressing when short data segments are used or when subroutines start at the beginning of a segment.

Memory-management chip

Those addresses manipulated by the programmer, used by the instructions, and appearing at the output of the Z8000 are called logical addresses. Transforming the logical addresses, which comprise the segment and offset concatenation, into a 24-bit physical address is the job of the Z-MMU memory-management unit (see "The Z8000 family," p. 87).

That transformation of logical address into a physical address, called relocation, is performed by this chip as shown in Fig. 7. A 24-bit origin or base is logically associated with each segment. To form the 24-bit physical address, the Z-MMU adds the 16-bit offset to the base for the given segment. (In operation, the Z8000 sends out the segment number half a clock period ahead of the 16-bit offset address to compensate for the time the unit needs to do this.) Thus the Z8000 can directly address half of a 16-megabyte physical memory space.

In addition to relocation, the Z-MMU provides segment management and protection from undesired writes. Each such unit stores 64 segment entries that consist of the segment base address and its attributes, size, and status. Segments can vary in size from 256 bytes to 64 kilobytes in increments of 256 bytes.

Using a pair of these units with the Z8000 accommodates all of the 128 segment numbers. Moreover, several Z-MMUs can be used together to accommodate several translation tables, although only a single pair may be enabled at any one time. □

Architecture of a New Microprocessor

Bernard L. Peuto
Zilog Inc.

Increased capabilities, architectural compatibility, and clearly defined interfaces were the chief architectural goals of Zilog's new Z8000 microprocessor family. Here is an account of how those goals were met for two members of that family—the Z8000 CPU and the MMU.

The Z8000 family is a new set of microprocessor components (CPU, CPU support chips, peripherals, and memories) which supports the Z8000 architecture. The account of how architectural goals were selected and achieved for two key members of this family—the Z8000 CPU and the memory management unit—illustrates how much of a challenge microprocessor architecture represents to the semiconductor industry. MOS technology shows enormous potential; but it is still difficult to use because of limitations on pin count, power dissipation, speed, and complexity.[1]

Since this discussion is restricted to technical issues, we will not allude to the many additional factors (marketing considerations, human considerations, self-imposed restrictions, etc.) which make architecture such a fascinating and difficult discipline. Furthermore, no attempt has been made to exhaustively describe the Z8000 architecture and components. Interested readers should consult the specific manuals for a more complete description.[2,3]

The goals of the Z8000 architecture: increased capabilities, architectural compatibility, increased clarity

The primary reason for introducing a new system architecture is to significantly improve the control and processing capabilities of microprocessors while maintaining their price/performance advantages. Technical advances have permitted the implementation of substantially increased processor power, but the most significant motivation for a new component family is generality. Only through such a family could we provide for architecturally compatible growth over a wide range of processing power requirements.

Our approach was a staged system architecture which attempts to provide new components, enhanced features, and new functions, while protecting the user's investment in hardware and software. The Z8000 family supports a single unified architecture for all small, medium, and high-end user applications which are implemented using a mix of components within the same family.

The goals of the Z8000 architecture can be grouped into three categories: increased capabilities, architectural compatibility over a wide range of processing powers, and increased clarity. In all these cases the resulting architectural features apply either to the basic architecture (that seen by an applications programmer) or to system architecture (that seen by a system designer or an operating system programmer).

Increased capabilities. All existing 8-bit microprocessors and many 16-bit minicomputers suffer from having a small address space. So, one of our goals was to provide access to a large address space (8M bytes). A second goal was to provide more resources in terms of registers (16 general-purpose 16-bit registers), in terms of data types (from bits to 32 bits), and in terms of additional instructions compared to existing microprocessors (multiply and divide, multiple register saving instructions, specialized instructions for compiler support etc.).

To facilitate complex applications it was important to support multiprogramming with good hardware support of task switching, interrupts, traps, and two execution modes. Operating systems also required a good hardware protection system.

Finally, we wanted to increase overall system performance. This resulted in the choice of an implementation using a 16-bit-wide data path to memory.

0018-9162/79/0200-0010$00.75 © 1979 IEEE

00(36

Architectural compatibility. One of the important lessons learned from previous computer system designs is that the design of a new family architecture is a rare occurrence. One way to apply this lesson is to design a unified architecture compatible over a wide range of processing powers. If we anticipate user growth from small to large systems within a family architecture, then such an approach can significantly increase its life.

The two versions of the Z8000™ (a 40-pin unsegmented and a 48-pin segmented version) are designed to achieve this goal, but many other features contribute indirectly to the family compatibility. For small aplications an unsegmented Z8000 with one or more 64K-byte address spaces can be used. For medium applications, a segmented Z8000 and one memory management unit allows direct access to 4M bytes of address space. For large applications a segmented Z8000 and multiple pairs of MMUs allow the use of several 8M-byte address spaces.

Since the segmented Z8000 can run in an unsegmented mode, both systems are compatible. Finally, to achieve even larger processing power through hardware replication, the architecture provides basic mechanisms for both multiprocessing and distributed processing.

Clarity. Clarity in an architecture is a measure of how well key interfaces are defined and specified. This is an elusive but important goal in a family where new and unforeseen components will be added during the life of its architecture.

We felt bus protocols were so important that we developed an independent specification for the Z-bus along with the individual device manuals.

Clarity in terms of the basic architecture means regularity and extendability of the instruction set, as well as the general and simple handling of the operating system interfaces. Clarity in terms of the system architecture means a well-defined method of communication between the various components. The key link between these components is the Z-bus, which is a shared system bus. In the section on communication with other devices, we describe some of the various types of bus protocols. At Zilog we felt this was so important that we developed an independent specification for the Z-bus along with the individual device manuals.[4]

Comparison with other system architectures

We are convinced that the differences between microprocessor system architecture and large computer system architecture are not sufficient to require a different design approach, although they certainly influence the details of design compromises. The last section of this paper deals with implementation tradeoffs and illustrates some particular compromises. (In a few places we mix implementation considerations with descriptions of architectural tradeoffs. Despite the importance of separating an architecture from its implementation, we found that this separation is often absent during the actual creation of a new architecture.)

Two differences between conventional computer systems and microprocessor systems have the greatest impact: price structure and component boundary differences. For high-end LSI systems, it makes sense to have one unified architecture, but unlike their computer family counterparts (IBM 360/370, PDP-11) different implementations cannot be justified on a price/performance basis. Speed and performance are mainly dependent on the state of technology, and therefore, for a given application, a user will waste the speed willingly since another slower implementation would cost the same. This does not exclude different versions of one implementation, which reflect only different test and production criteria such as package type, functional temperature range, and even speed range.

Most computer systems have both external and internal interfaces. External interfaces which define system boundaries are often standardized (e.g., the IBM channel interface or the DEC unibus). The internal interfaces of most mini or large computer systems are essentially hidden. In contrast, the component boundaries of a microprocessor-based system represent actual interfaces, and most users must be familiar with them as well as with external interfaces. Because the component interfaces are more visible and often must be more general, the microprocessor-oriented system bus emerges as a key standardization link to allow a wider mix of components and designs.

The basic architecture

Address space considerations. It is advantageous to have more than one address space, with each address space as large as possible. In the Z8000, memory references and I/O references are viewed as references to different address spaces. The I/O space is discussed in the section below on communication with other devices. Memory references may be instructions or data and stack accesses, with each type of access possible in either system or normal modes. The Z8000 distinguishes between each of these reference possibilities by using different combinations of its status lines. Separating the various address spaces can be used to increase the total number of addressable bytes and to achieve protection. The size of each address space depends on the versions of the Z8000 used. The 40-pin package version allows each address space to be at most 64K bytes, the 48-pin package version allows each address space to be at most 8000K bytes.

The 40-pin version is intended for systems, often used as dedicated systems, where the program and data spaces are small. In this case, relocation is not usually important. Using the different address spaces, one has a simple way to address in practice up to 4 x 64K bytes (with a maximum of 6 x 64K bytes). Some simple protection is achieved by separating these spaces in hardware.

The 48-pin version with one or more MMUs is intended for the medium to large applications where relocation and better memory protection are important.[3] In these cases, status information can also be used to separate between address spaces by using multiple MMUs. But it is also essential to achieve the detailed memory protection required. (It is possible to use the 48-pin version without an MMU.) In these high-end applications, the address spaces are so large that one is unlikely to exhaust them. Experience with large computers shows that 8M bytes is probably adequate. The current implementation of the Z8000 uses 8M-byte address spaces, but the architecture provides for 31-bit address (2147M bytes).

In both versions, the Z8000 allows direct access to each address space. Direct access means that the addresses used in instructions or registers have as many bits as the address space size requires. In other schemes the effective address is a combination of a shorter field in the instruction and other extension bits often found in an implied register. Despite the shorter address fields, we believe this "indirect access" does not save bytes, because extra instructions must be used to load and save the implied registers, which are typically in short supply.

Registers. The Z8000 is primarily a memory-to-register architecture. This characteristic does not entirely exclude other organizations, and mechanisms exist in the Z8000 to support them. For example, memory-to-memory operations are supported for strings, whereas stack operations are supported for procedure and process changes. This choice provides upward compatibility with the Z80. A register architecture also results in good performance, since register accesses are made at a greater speed than memory accesses in the current implementation.

Experience with register-oriented machines seems to confirm that four general-purpose registers are not enough and that a "proper" number is between eight and 32.[5] The Z8000 supports bytes, words (16-bit), and long words (32-bit), and a few instructions even use quadruple-word (64-bit) data elements. If we choose 16, 16-bit registers allow eight 32-bit registers as well as four 64-bit registers (Figure 1). Since addresses are 32 bits, the necessity of at least eight 32-bit registers was obvious. The impact of the 4-bit register field on the instruction format depends also on the number of address modes and operands. Sixteen registers allowed a reasonable tradeoff, whereas 32 registers would have resulted in too few one-word instructions.

With one minor restriction any register can be used by any instruction as an accumulator, source operand, index, or memory pointer. This regularity of the structure is so important that it is worthwhile to sacrifice any possible encoding improvements in instruction formats which could result from dedicating registers to special functions. Encoding improvements based on instruction frequency, so that frequent instructions use one word, are more effective in saving space without having a negative effect on the architecture.

Why not have specialized registers? The difficulty lies in the fact that the restrictions caused by dedication are inconsistent with one another.

Most applications dedicate the available registers to specific functions. For example, most high-level languages require a stack pointer and a stack frame pointer. Then why not, one might argue, have specialized registers? The difficulty lies in the fact that the restrictions caused by dedication are inconsistent with one another. If the architecture supplies only general-purpose registers, the user is free to dedicate them to specific usages for his application without restrictions. This is important in the context of microprocessors where user applications are not well known and where high-level languages are still used infrequently.

For example, the Z8000 allows software stacks to be implemented with any register. There are also two hardware supported stacks, but the registers used are still general-purpose and can participate in any operation. There is no allocated stack frame pointer, since any register can be used by means of the proper combination of addressing modes. The savings realized by register specialization are unattractive when the given function can still be performed simply. The loss that would result from restricting the applications would be too great. In contrast, significant savings result from excluding R0 from use as an index or memory pointer. This exclusion allows one to distinguish between the indexed and direct addressing modes which use the same combination of the instruction address mode field. The price is small, since R0 still can be an acumulator or source register and 15 others accumulator, index, and/or memory pointers are available. In this case the restriction made sense.

Another decision to be made about registers is their size. Since the architecture handles multiple data types we must have multiple data register sizes, which can hold each data type. The solution of the problem is implemented in the architecture by pairing registers, two 1-byte registers make a word register, two word registers make a long word register, etc.

Data types. Users would like to have as many directly implemented data types as possible. A data type is supported when it has a hardware representa-

006 38

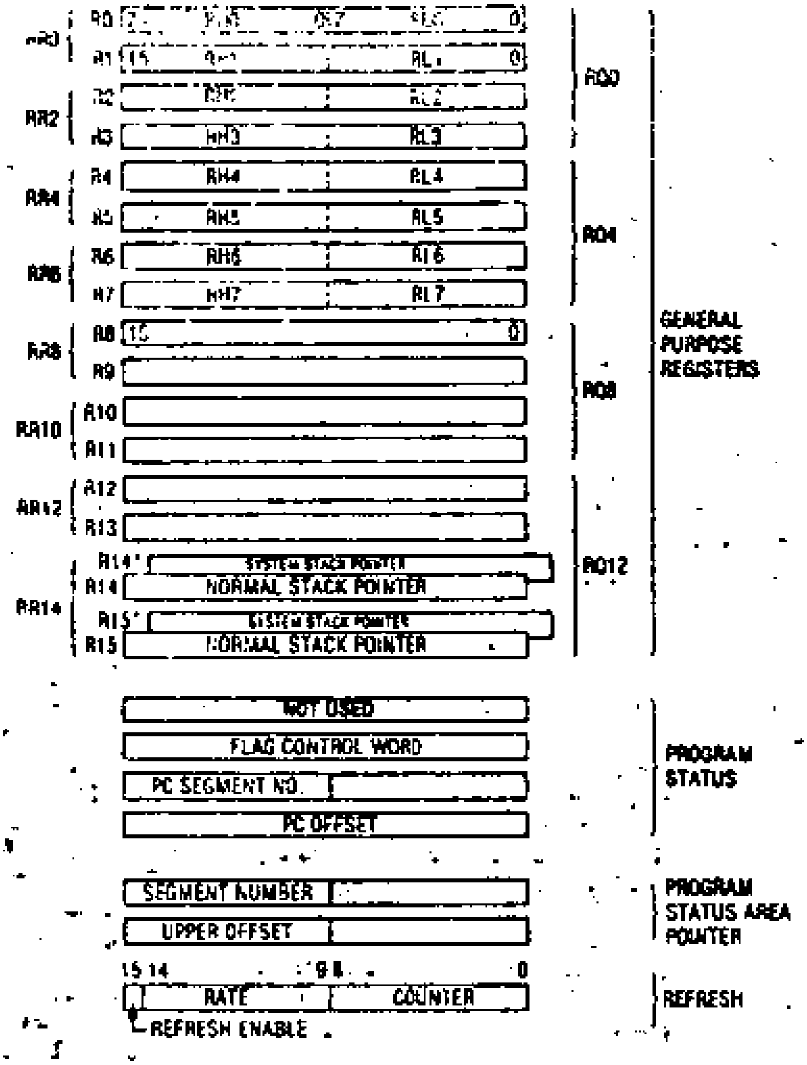


Figure 1. CPU registers (segmented version).

tion and instructions which directly apply to it. New data types can always be simulated in terms of basic data types, but hardware support provides faster and more convenient operations. At the same time, a proliferation of fully supported data types complicates the architecture and the implementations.

The Z8000 supports several primitive types in the architecture and provides expansion mechanisms. The basic data types are obviously the ones expected to be used most frequently. The extended data types are built using existing data types and manipulated using existing instructions.

The basic data type is the byte, which is also the basic addressable element. All other data types are referenced using their first byte address and their length in bytes. The architecture also supports the following data types: bytes (8 bits), words (16 bits), long words (32 bits), bytes, and word strings. In addition, bits are fully supported and addressed by number within a byte or word. BCD digits are supported and represented as two 4-bit digits in 1 byte. One consequence of this data type organization is that byte, word, and long-word registers are needed to support them. The Z8000 even provides quadruple register—another extension—used in long-word manipulation.

Other data types are supported by using one of the preceding data types; for example, addresses are manipulated as long words, and each element (segment number or offset) can be manipulated as a byte or a word. Instructions are one to five-word strings, the program status is four words, etc.

As the family grows, support for new data types will be added. The architecture will need to support them in its registers or in memory if they do not fit in registers (as strings are implemented today). But most important, the architecture will have to support the addition of new instructions to its repertoire.

Instructions. In designing an instruction format the architect must decide how to allocate a limited number of bits to the opcode field, address mode field, and other operand subfields. Instruction usage statistics are the best source of data to influence decisions about instruction set format.[1, 5, 7] Behind their usage lies a strong technical position: we do not

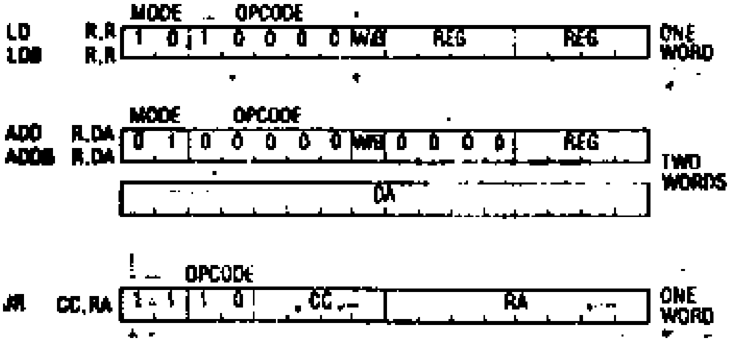


Figure 2. Examples of instruction formats (nonsegmented version).

believe that any one of the various instruction set structures—register oriented, memory oriented, stack oriented, symmetrical, or asymmetrical, etc.—are always better when used exclusively. Thus the task of the architect is to decide what his most important goals are, and for each of them adapt the best features of the various structures so that on the average, and for his set of goals, an optimum solution can be found. We do not believe that the optimum will be very sharp; it will be more like a range of applications for which the resulting composite structure works well. We decided to use a register structure for compatibility, multiple word instructions for speed, memory-to-memory instructions for strings, stack structure for process control and procedure support, "short" instruction for byte density improvement, etc.

Instruction format consideration. The Z8000 has over 110 distinct instruction types; several instruction formats are illustrated in Figure 2. The opcode field specifies the type of instruction (for example, ADD and LD). The mode field indicates the addressing modes (for example, Register (R), Direct Address (DA). The data element type (W/B) and register designator fields complete the basic instruction fields. Long word instructions use a different opcode value from their byte or word counterpart. Frequent instructions are encoded in a single word, and less frequent instructions which use more than two operands use two words. There are often additional fields for special elements such as immediate values or condition code descriptors (CC). Instructions can designate one, two, or three operands explicitly. The instruction TRANSLATE AND TEST is the only one with four operands and is also the only one with an implied register operand.

Several restraints can guide the proper choice of an instruction format. A large number of opcodes (used or reserved) is very important: having a given instruction implemented in hardware saves bytes and improves speed. But one usually needs to concentrate more on the completeness of the operations available on a particular data type rather than on adding more and more esoteric instructions which, if used frequently, will not significantly affect performance. Great care must be given to the problem of expanding the instruction set so, for example, new data types can be added.

Addressing modes. The Z8000 has eight addressing modes: *register* (R), *indirect register* (IR), *direct address* (DA), *indexed* (X), *immediate* (IM), *base address* (BA), *base indexed* (BX), and *relative address* (RA). Several other addressing modes are implied by specific instructions such as autoincrement or autodecrement.

Although a very large number of addressing modes is beneficial, usage statistics demonstrate that not all combinations of operands, address modes, and operators are meaningful.[8] The five basic addressing modes of R, IR, DA, X, and IM are the most frequently used and apply to most instructions with more than one address mode. For two-operand instructions, statistics show that most of the time the destination is a register. Other cases of addressing mode combinations and less basic addressing modes are associated with special instructions. Thus, the frequent combination of autodecrement for the destination operand with the five basic address modes for the source operand is provided by the PUSH instruction. The combination of autoincrement addressing modes for both source and destination operands is one of the block move instructions. In essence, the address mode field space has been traded for opcode field space. This allows more instructions and combinations while staying within a one-word format.

The price for this tradeoff is the infrequent occurrence of pairs or triples of instructions simulating a missing addressing mode. This situation occurs in most instruction sets in any case.

Code density. Because current microprocessors are restricted to primitive pipeline structures, their speed is largely dependent on the number of executed instruction words. Therefore, code density is not only important because of program size reduction but also because of speed improvement. One would like to encode in the smallest number of bits the most frequent instructions. The basic instruction size increment was chosen to be a word for reasons dealing with alignment, speed penalties, and hardware complexity. Thus the most frequent one and two-operand instructions take one word in their register or register-to-register forms. Less frequent instructions or instructions which use more than two operands use at least two words.

The Z8000 goes even further by selecting several special instructions as "short" instructions which take only one word, when normally they would take two words. These instructions, such as LOAD BYTE REGISTER IMMEDIATE and LOAD WORD REGISTER IMMEDIATE (for small immediate values), CALL RELATIVE, and JUMP RELATIVE, are so frequent statistically that they deserve such special treatment.

A one-word JUMP RELATIVE and DECREMENT AND JUMP ON NON-ZERO also have a very significant impact on speed. The short offset mechanism used by addresses (and described below) is also designed to allow one-word addresses. Compared to previous microprocessors, the largest reduction in size and increase in speed results from the Z8000's consistent

G:4.0

[illegible] machine used to exemplify in Figure 2 [illegible].

High-level language support. For microprocessor users, the transition from assembly language to high-level languages will allow greater freedom from architectural dependency and will improve ease of programming.[8] It is easy and tempting to adapt a computer architecture to execute a particular high-level language efficiently.[9] Most programming languages act as a filter and can be supported by a subset of available hardware with greater efficiency.[10] But efficiency for one particular high-level language is likely to lead to inefficiency for unrelated languages. The Z8000 will be used in a wide variety of applications, and we know that a large number of users will still be using assembly languages. Since the Z8000 is a general-purpose microprocessor, language support has been provided only through the inclusion of features designed to minimize typical compilation and code-generation problems. Among these is the regularity of the Z8000 addressing modes and data types. The addressing structure provided by segmentation should support procedures that result from structured programming. Access to parameters and local variables on the procedure stack is supported by index with short offset address mode as well as base address and base indexed address modes. In addition, address arithmetic is aided by the INCREMENT BY 1 TO 16 and DECREMENT BY 1 TO 16 instructions.

Testing of data, logical evaluation, initialization, and comparison of data are made possible by the instructions TEST, TEST CONDITION CODES, LOAD IMMEDIATE INTO MEMORY, and COMPARE IMMEDIATE WITH MEMORY. Compilers and assemblers manipulate character strings frequently, and the instructions TRANSLATE, TRANSLATE AND TEST, BLOCK COMPARE, and COMPARE STRING all result in dramatic speed improvements over software simulations of these important tasks, especially for certain types of languages. In addition, any register can be used as a stack pointer by the PUSH and POP instructions.

Segmentation. In order to provide for convenient code generation and data access, addresses must also be easy to manipulate. Architectures with direct access to memory typically use a linear address space, so that address arithmetic may be used on the entire address. In this case, addresses are manipulated as one of the data types of the same size. This removes the need to distinguish an address as a new data type. In contrast, the Z8000 has a non-linear address space. Addresses are made of two parts: a 7-bit segment number and a 16-bit offset. Only the offset participates in address arithmetic. The segment number is essentially a pointer to a part of the total address space, which can vary in size from 0 to 64K bytes. The hardware representation of a segmented address is a long word or a register pair (Figure 3), which allows the easy manipulation of each part of the address.

The segmented addresses are one of the key mechanisms used to support both large and small memory systems efficiently. The two versions of the Z8000 implementation, the 40-pin unsegmented and the 48-pin segmented, allow the maintenance of the architectural compatibility and ease the growth between these two application groups. The segmented address space guarantees that each 64K-byte address space of the 40-pin version becomes one of the segments of the 48-pin version. Each 40-pin version's 16-bit address becomes an offset within the segment, and a mode exists in the 48-pin package version in which 40-pin version code can be executed. Furthermore, compatibility with any current 8-bit microprocessor such as the Z80 is easy, and a new microcomputer such as the Z8 can address external data in a shared segment with the Z8000.

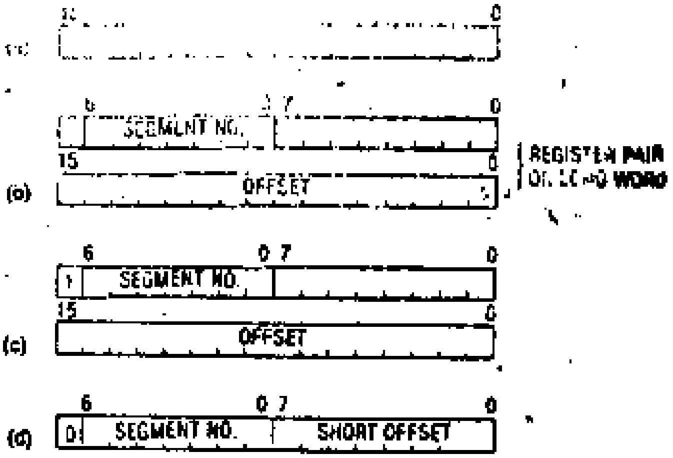


Figure 3. Hardware representation of segmented addresses. Any nonsegmented address is one word, whether it is in a register, memory, or an instruction (Figure 3a). Segmented addresses are always two words in a register or memory (Figure 3b); however, instructions can have one of two forms. The usual case (long offset) requires two words (Figure 3c); however, there is also a short offset form that uses only one word (Figure 3d).

The hardware performance of the Z8000 is also improved by address segmentation. Since a segment number does not participate in arithmetic, it can be put on the bus before the result of an address computation is available. This feature allows the use of MMUs with essentially no impact on memory access time by allowing it to function in parallel with the CPU. Indexing operations are also faster because only a 16-bit addition must be performed. Because of the distinction between the segment number and its offset, one can use shorter addresses without software constraints. Short addresses can use a short offset (fewer than 256 bytes) and thereby reduce program size (Figure 3).

Finally, it is very easy to associate with each of the 128 segments of the address space the protection and dynamic relocation features desirable for larger systems. Relocation allows a user to write his application using logical addresses independent of any physical addresses. Relocation is essential, for example, in a disk-based general data processing system with several users. Relocation is not essential for dedicated applications with code typically residing in

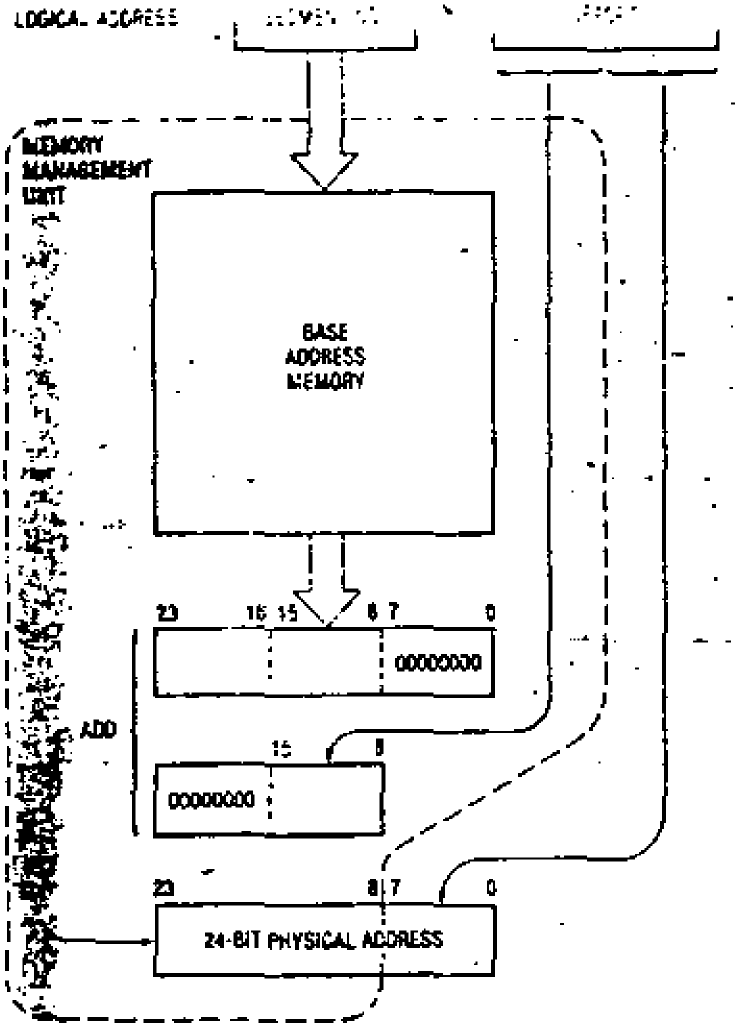


Figure 4. Logical to physical address translation.

ROM. Users whose total memory needs are small are also unlikely to need relocation.

In summary, the choice of a segmented address space has provided—at low cost and with few practical limitations—a powerful solution to the problem of user growth, relocation, and protection as well as virtual memory implementation. We believe that a linear address space could have achieved these results but at a considerably higher price.

The system architecture

Protection facilities. The Z8000 protection facilities can be divided into system protection features and memory protection features. Experience with large computers has demonstrated the advantages of having at least two execution modes with different access rights to hardware facilities. The Z8000 provides the system and normal modes for this purpose. A simple protection system results from the presence of these two modes and their [illegible] defined [illegible] which [illegible] interrupts, traps, and mode changes. Programs in normal mode which attempt to execute a privileged instruction will cause a trap and a change to system mode. The switch from user to system mode can also be caused by the system call instruction. These mechanisms enforce protection and help in designing reliable and efficient operating systems with clean user interfaces. Several other traps are required to achieve a consistent system: segmentation trap, privileged instruction trap, and undefined instruction trap.

A desirable memory protection scheme is one for which protection information (read only, read write, execute only, system only, size of data or code, etc.) is easily associated with the data and code structures of a given application. It is also one for which a large number of different types of protection information can be verified.

The relocation and memory protection mechanisms described above are provided by an external device: the memory management unit.[3] To provide relocation and protection features directly on the Z8000 would have demanded too much simplification. The external MMU has the further advantage of providing for easier growth by the addition of components. The Z8000 40-pin package does not have to carry the burden of the unused advanced relocation and protection features, although some form of protection can be achieved by hardware separation of the different address spaces. With multiple MMUs, the 48-pin package user can control the relocation and protection complexity desired in his application.

The memory management unit. The MMU performs three functions: (1) address translation of logical address to physical address using dynamic relocation, (2) memory protection, and (3) segment management. The addresses manipulated by the programmer, used by the instructions, and output by the Z8000 are called logical addresses. The MMU uses these logical addresses, composed of a 7-bit segment number and 16-bit offset, and transforms them into a 24-bit physical address (Figure 4). A 24-bit origin or base is logically associated with each segment. To form a 24-bit physical address, the 16-bit offset is added to the base for the given segment. In effect, with the help of one memory management device, the Z8000 can address 8M bytes directly within a 16M-byte physical memory space. The reasons for the choice of a large physical address space include an expectation that large systems will want to use extra bits for complex resource management purposes.

Each segment is given a number of attributes when it is initially entered into the MMU. When a memory reference is made, the protection mechanism checks these attributes against the status information from the CPU. If a mismatch occurs, a trap is generated which interrupts the CPU. The CPU can then check the MMU status registers to determine the cause of the trap. Segment attributes include segment size and type (read only, system only, execute only, in-

4-2.

valid DMA, invalid CPU, etc.) Other segment protection features include a write warning zone useful for stack operations.

When a memory protection violation is detected, a write inhibit line guarantees that memory will not be incorrectly changed. The invalid DMA and CPU bits indicate that the entry cannot be used by the DMA or CPU respectively, because either the segment number is illegal or the segment entry is not loaded. This fast feature, in conjunction with the segment history information (segment "changed" and segment "referenced" bits) and the segmentation trap mechanism, allows the implementation of a virtual segmented memory system.

The MMU comes in a 48-pin package (Figure 5). The chip inputs are the segment number, the upper 8 bits of the offset, and status information from the CPU. The outputs from the segment chip are the upper 16 bits of the 24-bit physical address and the segmentation trap line. Since the memory management device processes only the upper 8 bits of the offset, the lower 8 bits go directly to memory. This is equivalent to having zeros in the 8 lower bits of the 24-bit origin. Thus, the memory management device only needs to store the upper 16 bits of each base address. Segment limit protection is done in the memory management device, and thus segments can be protected in increments of 256 bytes.

Each MMU stores 64 segment entries that consist of the segment base address, its attributes, size, and status. A pair of MMUs support the 128 segments available in an address space. Additional MMUs can be used to accommodate multiple translation tables. Using the status information provided with each reference, pairs of MMUs can be enabled dynamically.

The memory management device functions constantly while memory references are made, but its translation and protection tables are loaded and unloaded as an I/O peripheral. To achieve this, the memory management device has chip select, address strobe, data strobe, and read/write lines. The Z8000 special byte I/O instructions that use the upper byte of the data bus can load or unload the memory management device.

Mode switching: interrupt and trap handling. From small users in dedicated process control applications to large users in general-purpose data processing applications, asynchronous events such as interrupts and synchronous events like traps must be handled. When these events occur, the state of any currently executing program must be saved during what is generally called a task switch or process switch. The users benefit from the availability of many interrupts and traps. They also benefit from a fast, easy, and uniform handling of process switching.

Peripherals using interrupts have widely varying constraints on interrupt processing time. To solve this problem, peripherals with the same characteristics are often associated with one of several interrupts. A priority enforced among the several interrupts allows the required processing time to be guaranteed. Enabling or disabling the various interrupts is the mechanism used to enforce this processing priority.

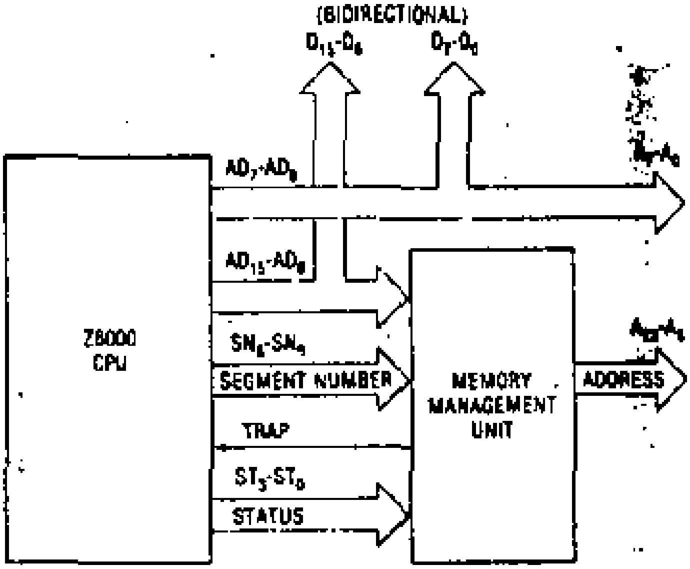


Figure 5. Memory management device with Z8000 CPU.

In the Z8000, we felt that three levels of interupts were sufficient. A *non-maskable interrupt* represents a catastrophic event which requires special handling to preserve system integrity. In addition there are two maskable interrupts: *non-vectored interrupts* and *vectored interrupts*, which correspond to a fixed mapping of interrupt processing routines and to a variable mapping of interrupt processing routines depending on the vector presented by the peripheral to the Z8000.

Both interrupts and traps result in similar process switches. Information related to the old process (its program status) is saved on a special system stack with a code describing the reason for the switch.. This allows recursive task switches to occur while leaving the normal stack undisturbed by system information. The state of the new process (its new program status) is loaded from a special area in memory—the program status area—designated by a pointer resident in the CPU (see Figure 6).

The use of the stack and of a pointer to the program status area are specific choices made to allow architectural compatibility if new interrupts or traps are added to the architecture. The choice of the two modes of execution has a strong impact on the design of clean user interfaces. Experience has shown that in large systems the normal mode instruction set and the user interfaces together constitute the most important element in achieving architectural compatibility.

Communication with other devices: the Z-bus. The Z-bus is the shared bus which links all the components of the Z8000 family.[4] The variety and performance requirements of the components are so different that in fact the Z-bus is composed of five buses:

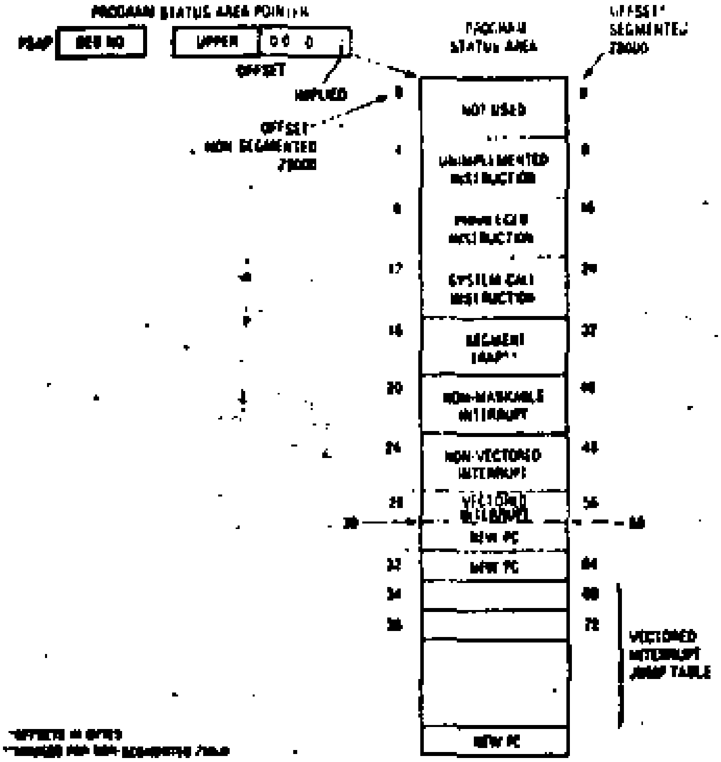


Figure 6. Program status area.

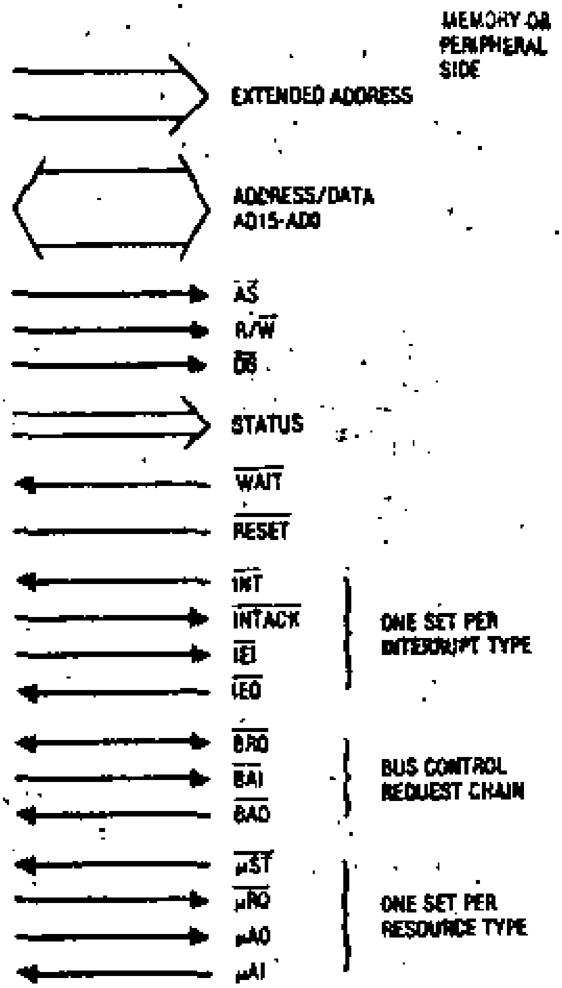

MEMORY OR PERIPHERAL SIDE
EXTENDED ADDRESS
ADDRESS/DATA AD15-AD0
AS
R/W
DS
STATUS
WAIT
RESET
INT
INTACK
IEI
IEO
ONE SET PER INTERRUPT TYPE
BRQ
BAI
BAO
BUS CONTROL REQUEST CHAIN
μST
μRQ
μAO
μAI
ONE SET PER RESOURCE TYPE

Figure 7. Z-bus signals.

a memory bus, an I/O bus, an interrupt bus, and two resource request buses (Figure 7).

The Z-bus is called a "shared" bus because several components can use it. A bus user is a CPU or a peripheral which can usually generate one or more bus transactions such as memory data request or an I/O request. Identical bus transactions cannot take place at the same time, but serialization mechanisms allow sequential use of the Z-bus. Architecturally, the buses can be grouped into two structures. The I/O structure uses the I/O bus and the interrupt bus. The memory structure uses the memory bus with or without address extensions. Both structures can use the resource request bus and the mastership request bus.

Each bus consists of a set of signals and the protocols which preside over the various types of transactions. Part of each protocol is the timing relationship between relevant signals. The Z8000 CPU provides most of these timing relations. The advantage of such a choice is the significant reduction in the number of components required to build such a system. One consequence is that bus transactions cannot be aborted or delayed freely since some devices, especially memory, have specific timing constraints. The most important consideration for the Z-bus is the need to interface to multiplexed address and data lines of the Z8000 CPU which must fit in 40- and 48-pin packages. The Z-bus maintains these multiplexed address and data lines. Very little speed could be gained by demultiplexing these lines for memory references since memories are themselves multiplexed. The most important advantage of a multiplexed Z-bus is the direct addressability of peripheral internal registers. This feature allows the construction of complex peripherals which maintain a simple program interface.

The Z-bus is known as a transparent or asynchronous bus. Z8000 components do not require that their clocks be synchronized with the CPU clock. The signals used by each transaction provide all the necessary timing. This concept is important: it allows, for example, I/O references to be independent of the speed and clock frequencies required by other Z-bus transactions.

I/O bus versus memory bus. The I/O and memory buses are the most important. The Z8000 family architecture distinguishes between memory and I/O spaces and thus requires specific I/O instructions. This architectural separation allows better protection and has a nicer potential for extension. The I/O and memory buses use a 16-bit address/data bus, which allows 16-bit I/O addresses and 8- or 16-bit data elements. Memory addresses are 16 bits for the 40-pin package or extended to 23 bits using the segmented version. Thus, the memory bus is in fact a logical address bus. The increased speed requirements of future microprocessors is likely to be achieved by tailoring memory and I/O references to their

G 44

respective characteristic reference patterns and by using simultaneous I/O and memory referencing. These future possibilities require an architectural separation today. Memory-mapped I/O is still possible, but we feel the loss of protection and potential expandability are too severe to justify memory-mapped I/O by itself.

Both the I/O and memory buses need address, data, and control signals. One important implementation decision was to overlap the signals used by the memory and I/O buses on the same Z8000 CPU pins, with the obvious exception of the status signals used to distinguish between the two types of bus requests. For the current Z8000 implementation the resulting reduction in number of pins is significant. In contrast the impossibility of doing concurrent memory and I/O referencing is not very significant since their speeds are essentially the same.

In addition, memories and peripherals both benefit from the availability of early status information defining the bus transaction type (I/O versus memory, read versus write) ahead of the actual transaction so that bidirectional drivers and other hardware elements can be enabled before the reference. The status lines of the Z8000 CPU provide this type of early status.

The I/O structure. Since many peripherals are connected with one CPU, the I/O bus is shared and serialization must be provided. One solution involves using a master/slave protocol. The CPU is a master which can initiate an I/O transation at any time. The peripherals are slaves which *participate* in a transaction only when requested by the master. In order to find out if a peripheral needs to be serviced the master can poll each in turn. The Z-bus also provides a faster way of getting the attention of a master: an interrupt bus. In contrast, with the I/O transaction data bus, each peripheral sharing the interrupt bus may "try" to use it simultaneously. The interrupt bus uses an interrupt line, interrupt acknowledge line, and two more lines used to form a daisy chain. The daisy chain is an implementation of a distributed arbitration policy between the requests. Priority of processing is determined by the position in the daisy chain, and peripherals can be preempted. Interrupt vectors are used to determine the identity of the peripherals requesting service via an interrupt.

Other buses. The two resource request buses are used to request the control of the Z-bus from the CPU and to request control of any generalized resource.

The Z8000 CPU or any Z-bus compatible CPU does not need to request the bus to access it as a master, and is, therefore, the default master. Other devices can request bus mastership, but they must go through a non-preemptive distributed arbitration using another daisy chain. The CPU always relinquishes the bus at the end of its current bus transaction.

The resource request chain is a generalization of that concept in which each resource requestor has equal importance and can use the resource in a non-preemptive manner. This mechanism in the Z8000 CPU permits one to implement in software the kind of exclusion and serialization mechanisms needed for multiple distributed systems with critical resource sharing.

Multiprocessing. In the context of today's large mainframe systems characterized by multiple processes sharing one processor, one is tempted to design distributed processing systems with many low-cost microprocessors running dedicated processes. Such an approach distributes intelligence towards the peripherals, results in modularization, and permits easier development and growth. Unfortunately, in the past, the problem with such an approach has been software and not hardware. Thus one cannot be expected to provide detailed solutions in hardware to a software problem that has not been solved yet. However, some basic mechanisms have been provided to allow the sharing of address spaces: large segmented address spaces and the external MMU make this possible, and a resource request bus is provided which in conjunction with software provides the exclusion and serialization control of shared critical resources. These mechanisms and new peripherals like the Z-FIO have been designed to allow easy asynchronous communication between different CPUs.

Implementation tradeoffs

The key family decision: producibility. Confronted with the problem of designing a new LSI-based system architecture, we could have ignored package size considerations by accepting packages with 64 or more pins, or we could have ignored mass production technology constraints by using die sizes larger than 280 mils square. Such solutions are often justified in the implementation of an existing computer system. The component boundaries, package limitations, and technological limitations are secondary to achieving the goal of exact membership in the computer family. But if one were to design a new system architecture with the same lack of constraints, the individual component would not be price-competitive—only the total system would be. A new system architecture based on this approach could only be used to design yet another traditional computer.

The Z8000 family provides basic, general-purpose blocks out of which a system solution to most problems can be implemented.

The Z8000 family market is intended to be much broader, and each component of the family must be economically viable. The staged introduction of components which are economically viable by themselves allows us to serve the market from very small configurations to very large configurations by using more components, in any combination. Not only do we believe that this approach does not restrict

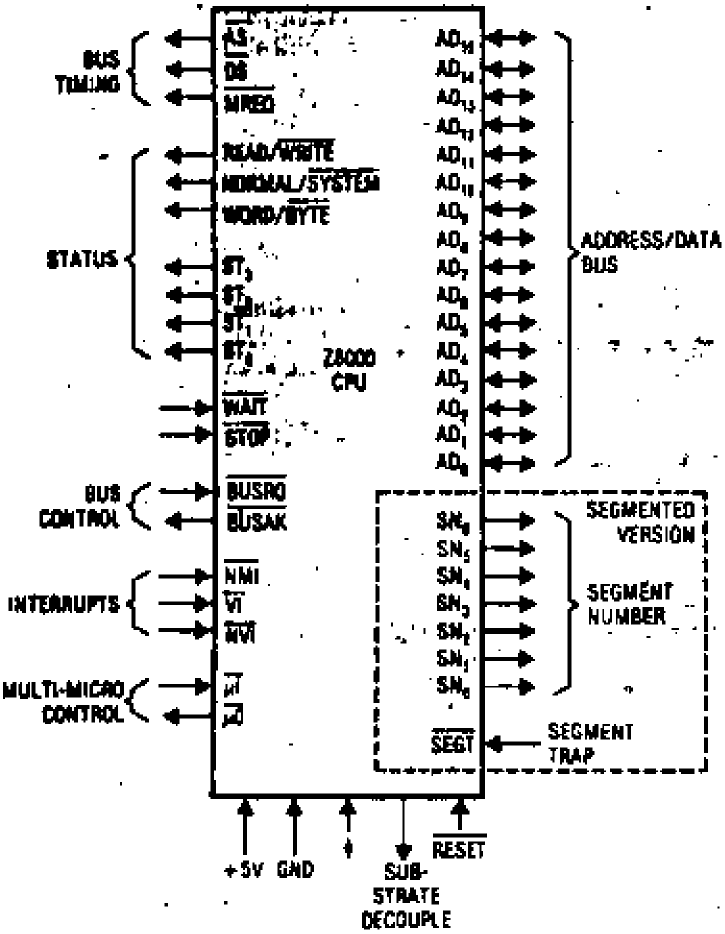


Figure 8. Z8000 pin functions.

system architectural possibilities, but we also believe that the family will be more effective because it will grow with its customer.

The Z8000 family does not always attempt to provide specific architectural solutions, often implemented in hardware, to all system architecture problems. Instead, it provides basic, general-purpose blocks out of which a system solution to most problems can be implemented. The multi-microprocessor and distributed system capabilities of the Z8000 family illustrate the use of open-ended mechanisms to solve a variety of architectural problems, while the memory management of address space illustrates a specific problem supported by a specific solution—the MMU. However, other solutions more appropriate to a particular problem can be used and an advance in the state of the art might be mapped into a new device for the family.

This vision of the family often results in components more powerful and complex than an application may require. The user should not take this as a cause for alarm, but rather as the reason his applications growth will be easier.

Basic CPU implementation decisions. The Z8000 currently uses a 16-bit data bus (Figure 8), an internal register array of 16-bit registers, and a 16-bit parallel ALU. These implementation decisions, which were guided by the technological and practical considerations, have a strong impact on performance.

To achieve good performance with the instruction format and data type envisioned for the Z8000, only a 16-bit bus seens adequate; a 32-bit bus would have necessitated using an unacceptable 56-pin or larger package. Optimal performace is obtained with this chosen bus width if the size of the frequently used register-to-register operations becomes one word. The choice of ALU and internal register widths is a tradeoff between speed of the most frequent operations and the chip area needed to implement a wider ALU or data path inside the CPU.

None of these implementation decisions should limit the architecture. Instructions are from one to five words long, and data types and addresses are not limited to 16 bits. For example, 32-bit words are one of the main data types of the machines, and addresses occupy two words. The *address* mechanism illustrates the strong distinction between an architecture and its implementation. The architectural address representation uses a 32-bit word of which 8 bits are reserved and 1 is a short format/long format descriptor. Thus, the Z8000 architecture provides up to 31-bit addresses, but only 23 are currently implemented and 23 pins of the current package are allocated to addresses.

MMU tradeoffs. The MMU and its relation to the Z8000 CPU illustrate tradeoffs that a microprocessor architect and designer team must make to ensure component manufacturability.

To achieve the goals of good architectural compatibility for high-end systems, it was necessry to include the protection and relocation mechanicms described above. But if all desired features were implemented as a one-chip CPU/MMU combination, it would have been too large and, therefore, uneconomical. And if a reduced set of features were implemented, it would have been architecturally too primitive. Thus, the choice was made to maintain all features and use two chips. This new organization has several significant advantages, such as a capability for multiple MMUs, and allows the access of a DMA device to the MMU.

Given the choice of an external MMU, the next set of decisions concerns package size and circuit speed. Having each relocated segment start on a word boundary would have required a 64-pin package and a very fast 24-bit adder (in fact, a 16-bit adder and 8 bits of carry propagation). In contrast, the decision to start segments on 256-byte boundaries allows the use of a 48-pin package, a fast 8-bit adder, and 8 bits of carry propagation. The latter solution is technically superior and places practically no restriction on the architecture. Segment granularity can be viewed as an implementation restriction and not as an architectural restriction.

Making the 8 low-order bits of the offset go directly to memory also significantly reduces memory access time. Since dynamic memories use these bits first, most of the MMU relocation time is hidden during a normal memory access. The availability of segment numbers earlier than the associated offset bits reinforces this advantage and allows the MMU to result in essentially no memory access speed reduction. Each MMU entry also requires 8 bits less for base and segment size value. This is important: it is desirable to pack as many entries as possible per MMU. With 64 entries a 2K-bit memory is needed, which is technologically difficult in view of the amount of logic surrounding this memory and the complexity of its organization.

The fact than an MMU is only connected to the upper byte of the data bus requires the use of special I/O instructions for its loading and obliges us to replace the possible use of an automatic demand loading of entries by explicit instruction loading. To compensate for the time penalty associated with the loading of potentially unused entries, multiple MMUs are used. They not only allow the implementation of 128 entries, but pairs of MMUs can be automatically enabled by the system and normal mode pins affecting a full environment switch at electronic speed.

We feel this example illustrates one important design approach: to compromise as little as possible on advanced architectural features but to accept compromises which result in implementation ease in order to achieve economical components.

Conclusion

The architectural sophistication of the new 16-bit microprocessors is rapidly approaching the level of the minicomputer and large computer. Problems such as component families, large address spaces, bus standards, I/O structures, software investments, and architectural compatibility are being directly addressed. Some of the solutions to these problems are known, and therefore the transition from 8-bit microprocessors was relatively easy. But the challenges ahead—networks, distributed processing, new applications—are much harder. The impact of microprocessors is already enormous, but we feel they will achieve the often-predicted computer revolution only after these new problems are solved. ■

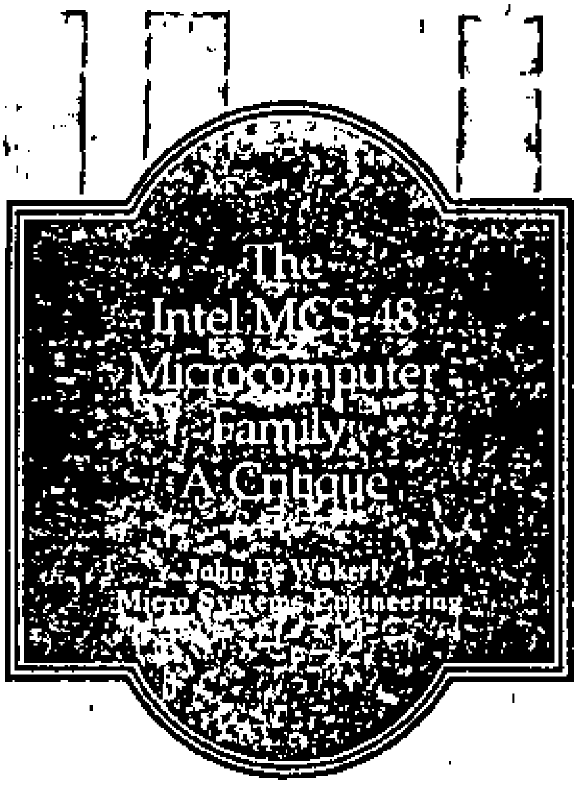
Acknowledgements

The Z8000 family would not exist without the very talented and dedicated designers who contributed to and implemented the ideas described in this paper: Masatoshi Shima for the Z8000, Hiroshi Yonezawa for the MMU, and Ross Freeman for the peripheral devices. Judy Estrin made invaluable contributions to the architecture of the Z8000 and Z8. Many discussions with Charlie Bass, Leonard Shustek, and Forest Baskett have greatly influenced the Z8000. Leonard's instruction set measurements were especially valuable. Dennis Allison, Steve Meyer, Bruce Hunt, and many others must be thanked for their comments on early drafts of this paper.

References

1. B. L. Peuto and L. J. Shustek, "Current Issues in the Architecture of Microprocessors," *Computer*, Vol. 10, No. 2, Feb. 1977, pp. 20-25.
2. Zilog, *Z8000 Technical Manual*, Zilog, Inc., 1979.
3. Zilog, *MMU Technical Manual*, Zilog, Inc., 1979.
4. Zilog, *Z-Bus Specification*, Zilog, Inc., 1979.
5. A. Lunde, "Empirical Evaluation of Some Features of Instruction Set Processor Architectures," *CACM*, Vol. 20, No. 3, Mar. 1977, pp. 143-152.
6. L. J. Shustek, *Analysis and Performance of Computer Instruction Sets*, PhD Dissertation, Dept. of Computer Science, Stanford University, Stanford, Calif., Jan. 1978.
7. B. L. Peuto and L. J. Shustek, "An Instruction Set Timing Model of CPU Performance," *Proc. Fourth Annual Symposium on Computer Architecture*, Mar. 23-25, 1977, pp. 165-178.
8. C. Bass, "PLZ: A Family of System Programming Languages for Microprocessors," *Computer*, Vol. 11, No. 3, Mar. 1978, pp. 34-39.
9. A.' S. Tannenbaum, "Implications of Structured Programming for Machine Architecture," *CACM*, Vol. 21, No. 3, Mar. 1978, pp. 237-246.
10. N. G. Alexander and D. B. Wortman, "Static and Dynamic Characteristics of XPL Programs," *Computer*, Vol. 8, No. 11, Nov. 1975, pp. 41-46.

Bernard L. Peuto is one of the guest editors for this special section; his biography appears with the introduction on p. 9.



The Intel MCS-48 Microcomputer Family: A Critique

by John F. Wakerly
Micro Systems Engineering

A system designer and teacher, who has made liberal use of microcomputers in his own work and whose students have designed 8048 processors, reviews the capabilities and limitations of the MCS-48 family of microcomputers.

The Intel MCS-48 family of single-chip microcomputers contains at least nine different microcomputer chips having a common instruction set but different amounts of on-chip read-only memory, read/write memory, and input/output (see Table 1). According to Intel, the MCS-48 family was originally aimed primarily at the "4-bit market"—users of Intel's 4040 and other low-cost microcontrollers. Recent entries into the family (the 8021, 8022, 8041, and 8741) are increasingly specialized for low-end microcontroller applications. The MCS-48 family has met this market very well.

Table 1.
MCS-48 microcomputers.

PART #	PACKAGE SIZE (pins)	ON-CHIP PROGRAM MEMORY (bytes)	ON-CHIP DATA MEMORY (bytes)	I/O (lines)
8048	40	1K ROM*	64**	27
8748	40	1K EPROM*	64**	27
8035	40	none*	64**	27
8049	40	2K ROM*	128**	27
8039	40	none*	128**	27
8021	28	1K ROM	64	21
8022	40	2K ROM	64	23 plus 2 8-bit A/D conv.
8041	40	1K ROM	64	18 plus master sys. intf.
8741	40	1K EPROM	64	18 plus master sys. intf.

* Expandable to 4K with external chips

** Plus 256 bytes or more of external data memory with external chips

The MCS-48 family was also aimed at a second market—applications that require an expandable, single-chip, general-purpose microcomputer. As shown in Table 2, several expansion chips are available to provide an MCS-48 computer with up to 4K bytes of program ROM, 256 or more bytes of external RWM, and as many I/O bits as a designer would ever need. In addition, the external I/O bus of the MCS-48 family allows easy interfacing of standard 8080/8085-compatible peripheral chips. Nevertheless, the architecture of the MCS-48 family makes it difficult to use in many general-purpose applications, where a more capable 8-bit architecture is required.

Table 2.
MCS-48 expander chips.

PART #	PACKAGE SIZE (pins)	ON-CHIP PROGRAM MEMORY (bytes)	ON-CHIP DATA MEMORY (bytes)	I/O (lines)
8355	40	2K ROM	none	16
8755	40	2K EPROM	none	16
8155/56	40	none	256	27 plus timer/counter
8243	24	none	none	16

Basic architecture

Figure 1 shows the basic structure of an MCS-48 microcomputer chip. (Table 1 gives the facilities available for each of the microcomputers in the MCS-48 family that had been announced by late 1978.) The first member of the family was introduced in late 1976—the 8048 with 1K bytes of on-chip ROM, 64 bytes of RWM, timer/counter, and 27 I/O bits. A detailed description of the entire family can be found in the user's manual published by Intel.[1]

The MCS-48 is a single-accumulator architecture. Program memory and data memory are logically and physically separated (thus, the MCS-48 is not a von

0018-9162/78/0020-0022$00.75 1978 IEEE

4.8

Neumann machine). The maximum program address space (including external ROM) supported by the architecture is 4K bytes. There are a maximum of 256 bytes of on-chip (internal) data memory, of which 128 bytes are implemented in the current family leader, the 8049. In addition to internal data memory, the MCS-48 directly supports 256 bytes of external data memory.

Most MCS-48 family members have 27 I/O pins, arranged as three 8-bit ports, two test inputs, and an interrupt input. Additional pins are provided for such functions as power-on reset, single-stepping, and memory and I/O expansion strobes. One 8-bit port and part of a second are used to form a multiplexed address and data bus for I/O and memory expansion.

The MCS-48 has a single-level interrupt system (only one interrupt in service at a time) and accepts interrupts from two sources—its internal timer/counter and an external interrupt input pin. Interrupt calls and returns automatically push and pop the program counter and certain internal status flags using a stack in the internal data memory.

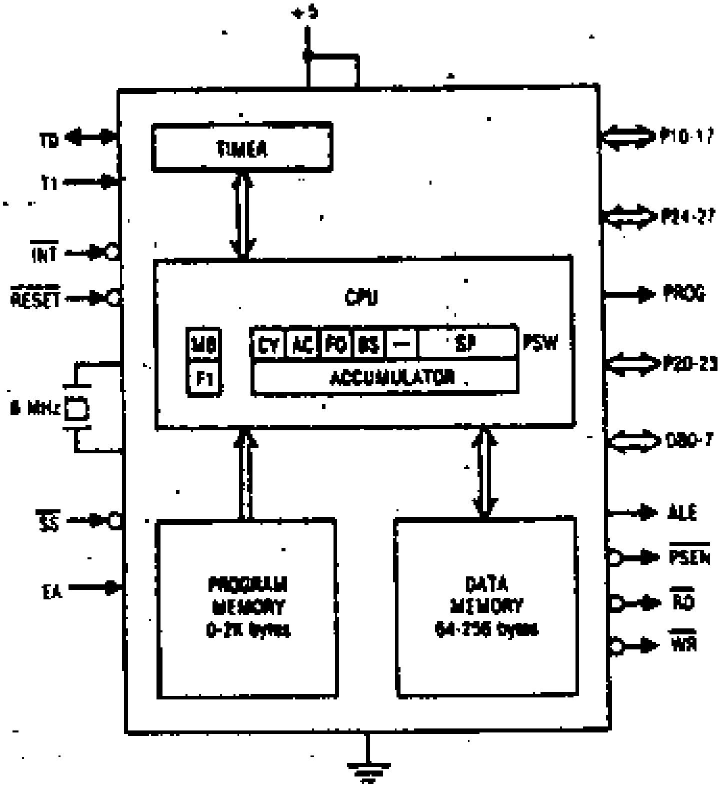


Figure 1. Basic structure of a typical member of Intel's MCS-48 family of microcomputers.

Program store and program control

The MCS-48 architecture supports a maximum of 4K bytes of program store, configured as shown in Figure 2. However, a close look at program-store organization shows that the MCS-48 was originally designed as a 2K-byte machine, with the second 2K-byte capability added as a clumsy afterthought. This creates two problems with the addressing mechanism.

First, the program counter is really only 11 bits and thus addresses instructions only within a 2K-byte bank of program store. Jump and subroutine call instructions likewise specify an 11-bit address. The problem, then, is how to provide a 12th address bit.

Intel's solution is as follows. Provide an internal flag, MB, that can be set and cleared by two instructions (SEL MB1 and SEL MB0, respectively). Whenever a jump or subroutine call is executed, take the 11 low-order PC bits from the instruction, and load the high-order bit from MB. On subroutine calls and returns, push and pop the entire 12-bit address.

There are some problems with this solution. First, in a general sequence of jumps and calls in a 4K system, we don't always know where we came from, and therefore we don't know the current value of MB. So in general, a SEL MBi instruction must precede every jump or call. Naturally the programmer can sometimes avoid this instruction on a case-by-case basis, but this is another thing to worry about.

Having solved the first problem, we think we understand the addressing mechanism until we write our first interrupt routine. Then we wake up in the middle of the night thinking, "Whoops! MB can't be read as part of the processor state PSW. But MB must be set to a new value in order to do jumps within the interrupt routine. How can the old value be restored on return?" We lie awake a few hours dreaming up possible solutions—don't use calls or jumps in in-

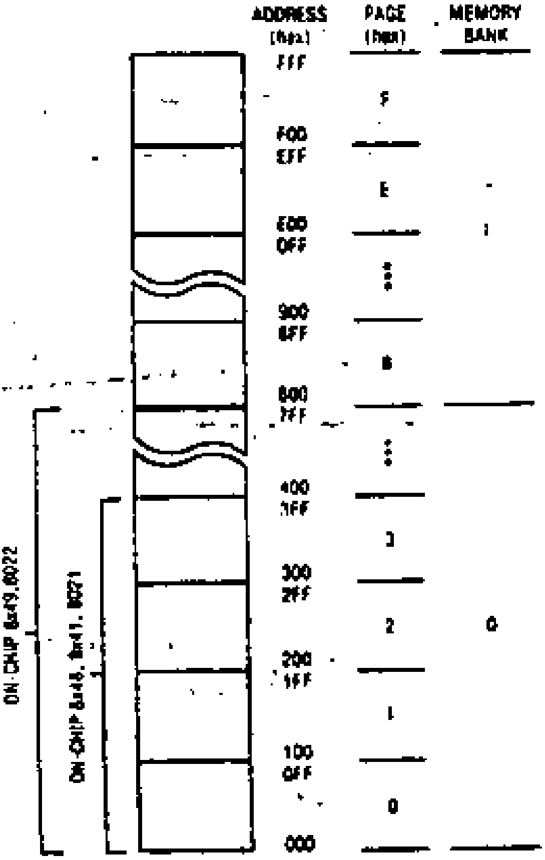


Figure 2. MCS-48 program memory.

How to choose a microcomputer

There are at least six factors to consider in choosing a microcomputer (or microcomputer family) for a product application.

Capability. The μC must have enough ROM, RWM, I/O capability, and speed to satisfy the requirements of the application, plus a design margin. ROM, RWM, and I/O capability can be determined from the manufacturer's literature, while speed is best determined from benchmark programs tailored for the given application.

With some μCs, the amount of ROM, RWM, and I/O can be increased by using extra chips. This expandability helps if the job is initially underestimated or if the marketing department changes the requirements. If the application pushes the absolute memory limits of the μC, it becomes more difficult (and expensive) to develop the programs.

Extensibility. The designer must consider whether improved versions of the μC will be offered. A μC-based product designed in 1979, for example, might be redesigned in 1981 to reduce cost or to add features. It would then be desirable to eliminate extra ROM, RWM, and I/O chips (or avoid having to add them or pick a different μC) by using a new version of the original μC with the extra capability built in. Of course, many products do not undergo this evolution; but if product evolution is expected, the architectural limits of the selected μC should be examined in light of potential application requirements. One can expect lower hardware and software development costs and, probably, lower manufacturing costs if the enhanced product uses an upgraded version of the original μC rather than a completely new one.

Cost. In most areas of the private sector, minimizing cost is a goal, and minimizing μC cost usually means minimizing the number of IC packages. Cost is what prevents the designer from picking a Cray-1 in response to the first factor above, or an IBM 370 in response to the second factor.

If the job is well defined and no product enhancement is anticipated, it is relatively easy to find a minimum-cost μC that will do the job. Otherwise, there are many more tradeoffs to be considered. A simpler μC architecture usually implies a smaller IC die and lower chip cost, but it may also require more chips to support it later. (For example, a μC without a WAIT/READY line may be more difficult to interface to some types of peripherals or memory.) An expandable μC will facilitate later product evolution (if the product is successful), but may increase initial product cost because of instruction efficiency, memory size, I/O pins, or speed sacrificed by the chip designers to make expansion or enhancement possible.

Availability. Many manufacturing organizations require a second source for all components, both to ensure that parts will be available, even if some disaster befalls one source, and to enjoy the normal benefits of competition in a free market.

The designer of a new product is often tempted to select between a μC with one or two sources and one with no sources (yet—"We'll have samples in three months"). It is risky to commit to any part unless your purchasing department can order (and receive) 100 pieces from a distributor's shelf. Manufacturers have been known to slip schedules and even cancel parts.

On the other hand, marketing and cost factors can motivate the selection of a not-yet-available or very new μC. The new μC can give the product a competitive edge in features or performance. Although the new μC may be in short supply and costly initially, it may be cheaper in the long run because it allows a more efficient design with fewer IC packages.

Expected product lifetime should also be compared with the expected lifetime of the μC. Even if it is inexpensive currently, a μC that has been around for a few years may be a bad choice: production quantities may fall and prices rise in a few more years as newer chips are phased into new designs. Of course, this doesn't apply if your company alone is ordering 100,000 pieces per year.

Support tools. Hardware and software support tools are essential for timely development of a μC-based product. The support tools of a newly introduced μC cannot be expected to be as extensive or reliable as those of an established μC family. This encourages the use of an established μC if quick development is needed, or an extensible μC family with reusable tools if product evolution is expected.

Most single-chip μCs are programmed in assembly language, and a good macroassembler is a must. Most manufacturers supply software tools that run on their own development systems. However, if there are more than one or two programmers on the project, the need for good text editors, simulators, and documentation facilities makes it desirable to run all software support tools on a large central computing facility. The appropriate "cross assemblers" and simulators may or may not be offered by the chip manufacturer.

During product development it is obviously necessary to test and change programs running on the product hardware. Since most single-chip μCs ultimately use mask-programmable ROM to store their programs, another means is needed to store and change programs during development without making new masks. Some μCs have pin-compatible versions with on-chip EPROM instead of ROM that allows reuse of the μC chip with different programs. Many have provisions for using external EPROM chips instead of the on-chip ROM. If production quantities are low, or if software changes are expected after product introduction, EPROM versions may be essential.

Besides EPROM facilities, the main support tool provided by the chip manufacturer is the in-circuit emulator, which stores the software program in the RWM of a development system and emulates the μC through a cable and plug inserted in place of the μC in the product. An emulator is a useful tool for debugging both hardware and software. However, with new μCs, it may not be available as soon as the μC chips are, and even if it is, it may still have bugs.

Specific technical factors. Many specific technical factors can be examined in determining whether a μC will do the job at hand—power consumption, speed, TTL compatibility, package size, instruction set. However, once it is determined that the μC can do the job, the other factors above tend to equal or outweigh the technical "niceness" of the μC chip architecture.

50

terrupt routines; determine the value of MB experimentally by doing a jump to a fixed location and seeing whether it winds up in MB0 or MB1; keep a software copy of MB, updating it (with interrupts disabled, of course) every time we do a SEL MB1; make all the code fit in 2K, as we expected to do at the start of the project; and so on. The next morning we read the fine print to discover that the MCS-48 forces the most significant bit of the program counter to 0 during all interrupt routines. We should put all of our interrupt code in the bottom 2K of memory and not touch MB; in fact, we should forget that MB exists!

The requirement to put interrupt code in the bottom 2K makes the MCS-48 very difficult to use as a 4K machine in a real-time application. Not only must the basic interrupt service routine be in the bottom 2K but also any utility routine that might be called by it—that is, any code executed before an interrupt return instruction is executed. This could be well over half the code in an interrupt-driven environment.

But the main problem we find with MCS-48 program store, after writing half of our applications programs, is that the address space is just too small. With only two chips (and soon with just one, I'm sure) we can fill the entire 4K-byte address space of the MCS-48 with code for our original application, new features, diagnostics, and—of course—patches.

Conditional jumps specify an 8-bit target address in the current page; it would be far more useful to have a signed offset from the current address.

The annual halving of the cost of IC memory implies that every year we will need another address bit for the maximum-size application program (since most evolving products tend to use the decreased memory cost to increase features, not to reduce product cost). Clearly, then, a 4K limit is too low for any new architecture, even a single-chip microcomputer.

Besides the 2K memory banks, program store is also divided into 256-byte pages. Conditional jumps specify an 8-bit target address in the current page. It would be far more useful to have a signed offset from the current address; this would increase the likelihood of being able to use the short jump address, since most branch targets are within 128 bytes of the branch instruction.[1] More importantly, it would eliminate the partitioning problem created when many procedures must be packed into the memory space and split across page boundaries.

The only indirect jump instruction also uses an 8-bit target address in the current page. Very strangely, this instruction uses an 8-bit value in the accumulator not as the target address, but as a pointer to a program-store byte in the current page that contains the target address. So the page containing the indirect jump instruction must also contain all of the routines to be jumped to, as well as a silly little table that contains their starting addresses in the page. This not only wastes space and time, but, worse, makes it impossible to dynamically compute a target address after assembly time, since the jump table is in ROM. In my recent experience, three experienced programmers have coded MCS-48 indirect jumps improperly, believing "The manual must have a typo—the contents of the accumulator must be the target address itself." In any case, instructions supporting indirect jumps and calls anywhere in the program store (12-bit address) would be far more useful.

Arithmetic and logical operations

The MCS-48 contains a single accumulator in which arithmetic and logical operations take place. Unary operations on the accumulator are as follows:

increment,
decrement,
clear,
one's complement,
decimal adjust,
swap nibbles,
rotate left,
rotate left with carry,
rotate right, and
rotate right with carry.

Binary operations combine the accumulator and an operand specified by one of the addressing modes described in the next section. The binary operations are:

add,
add with carry,
AND,
OR, and
exclusive OR.

There are also "data-move" operations that load or store the accumulator.

The main difficulty with MCS-48 operations is not the operations themselves but the lack of condition codes for testing their results. Only the accumulator can be tested for zero or negative, and an overflow bit is not provided, making comparisons of signed two's-complement numbers very frustrating.

Operands

Most data moves and binary operations use an on-chip read/write internal data memory (see Figure 3) accessible by two addressing modes: REGISTER and INTERNAL REGISTER INDIRECT. The three other addressing modes are EXTERNAL REGISTER INDIRECT, IMMEDIATE, and ACCUMULATOR INDIRECT.

In REGISTER mode an operand is contained in a register specified by a 3-bit field in the instruction. A flag bit BS, set by a SEL RB instruction, specifies one of two 8-byte register banks, corresponding to internal data memory locations 0-7 if BS is 0 and 24-31 if BS is 1. The specified register may be loaded with an immediate value, moved to or from the accumulator, combined with the accumulator by arithmetic or logical operations, incremented, decremented, or used as a loop counter.

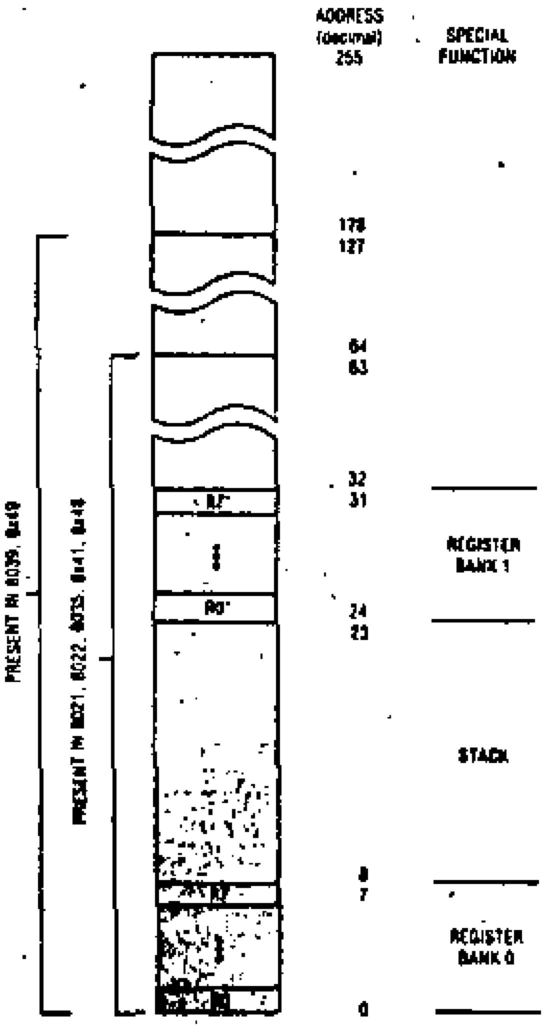


Figure 3. MCS-48 internal data memory.

INTERNAL REGISTER INDIRECT allows either R0 or R1 in the current register bank to be used as an 8-bit pointer to internal data memory. The addressed byte may be loaded with an immediate value, moved to or from the accumulator, combined with the accumulator, or incremented (but for some obscure reason not decremented, even though the necessary "hole" exists in the instruction set).

Locations 8-23 of the internal data memory are reserved for a return address stack (8 entries, 2 bytes per entry). These locations are written by interrupts and subroutine calls and read by interrupt and subroutine return instructions. The stack is too small, making it hard to write procedural code, which is important in larger programs (2K-4K bytes). The programmer must constantly worry about calling sequences and generally enable interrupts only at the top level of the program to avoid overflowing the stack.

There are no instructions to directly push or pop a byte. However, the stack can be rather inconveniently written or read by extracting the stack-pointer field from the PSW, building the appropriate address, and using INTERNAL REGISTER INDIRECT mode.

The architecture also supports up to 256 bytes of external data memory (which resides on a separate chip), accessed by EXTERNAL REGISTER INDIRECT mode. Either R0 or R1 in the current register bank may be used as an 8-bit pointer to external data memory; the addressed byte may be copied into the accumulator or written from the accumulator.

Since pointers are contained in 8-bit registers, the maximum amount of directly accessible data memory supported by the MCS-48 architecture is 256 bytes internal plus 256 bytes external. However, bank switching via I/O bits can be used to address any desired amount of additional external data memory.

The modes for reading operands from program store are rather limited. In IMMEDIATE mode an operand is contained in the byte following the instruction; immediate operands can either be loaded into or combined with the accumulator or be loaded into internal data memory with REGISTER or INTERNAL REGISTER INDIRECT modes.

In ACCUMULATOR INDIRECT mode the accumulator is used as an 8-bit pointer to an operand in either the current page or page 3 of program store; only one type of operation uses this mode, and it loads the accumulator with the specified operand.

A number of instructions specify some "special" operands implicitly, such as the program status word, I/O ports, timer/counter, carry bit, and two 1-bit flags, F0 and F1.

The MCS-48 addressing modes are simple, but they provide most of the facilities a program needs. Still, there are some deficiencies. The most serious problem is the way in which operands in program store are addressed. Since program store only in the current page and in page 3 can be read through a pointer, either lookup tables must all be located in page 3 or the code that reads each table must be in the same page as the table. This is inconvenient if more than one 256-byte translation table is needed. It also makes it difficult to do a ROM checksum self-test routine—a checksum subroutine would have to be placed in every page of program store (and since there is no indirect subroutine call, the main checksum program would have to contain a separate call instruction to each page's checksum routine).

For most programs, the method of indirectly addressing data memory through R0 and R1 is acceptable, but, for some data-structure manipulations, one wishes for one or two more registers that could be used as pointers.

The 256-byte limit on directly addressable internal data memory is too low. The 8049 already contains 128 bytes of RWM, and Intel should soon be able to provide the full 256 bytes of RWM on one chip. The architecture cannot make straightforward use of technology improvements for more RWM once this limit is reached.

Input/output and interrupts

Most MCS-48 microcomputers have three 8-bit I/O ports, as shown in Figure 1. Two of the ports (1 and 2)

52

are "quasi-bidirectional," an interfacing arrangement shown in Figure 4. This type of I/O port was first introduced in the Fairchild F8.[9] In this arrangement, each I/O pin is both an open-drain output and an input pin with a high-impedance pullup to the logic 1 level. When a pin is used for output, the corresponding input buffer is unused except, possibly, for checking the output value. When a pin is used for input, the corresponding output bit must be set to logic 1 so that the I/O device drives only the high-impedance pullup. This can be contrasted with a tristate I/O port, which provides both active pullup and active pulldown in output mode and high impedance in input mode. Electrically, tristate I/O is more desirable, but it requires extra control bits to set the I/O direction for each port or bit. Intel has improved the quasi-bidirectional design by briefly providing active rather than passive pullup whenever a 1 is written to the port, which speeds up 0-to-1 transitions.

What quasi-bidirectional I/O means to the programmer is that input data on the port is logically ANDed with the current output. Ports 1 and 2 are set to all 1's at system reset, and the programmer must leave bits intended for inputs set at output value 1 at all times. The third port (bus) has conventional tristate outputs and can be used for eight strobed inputs, for eight strobed outputs, or for adding external program or data memory.

Four operations on the ports are available:

read input value into accumulator (IN),
load output latch from accumulator (OUTL),
logical AND output latch with immediate mask (ANL), and
logical OR output latch with immediate mask (ORL).

The logical operations allow a program to set or clear any bit or group of bits in one instruction. However, since the mask is an immediate value in program store, the bits to be set or cleared must be known at assembly time. Otherwise, a copy of the output value must be kept in data memory, combined with the mask by logical operations on the accumulator, and loaded into the port. (In general, the quasi-bidirectional interface prevents simply reading the port to get the old value of the output latch.)

A novel "expander-port" arrangement allows four external 4-bit I/O ports to be added to an MCS-48 using a five-wire interface. Again, four operations on the ports are available:

read input value into accumulator,
load output latch from accumulator,
logical AND output latch with accumulator, and
logical OR output latch with accumulator.

Only the low-order four bits of the accumulator are used in these operations. For these ports, dynamic selection of mask bits is possible because the mask is in the accumulator. On the other hand, dynamic selection takes more overhead because the accumulator must be loaded with the mask (and then possibly restored to its old value).

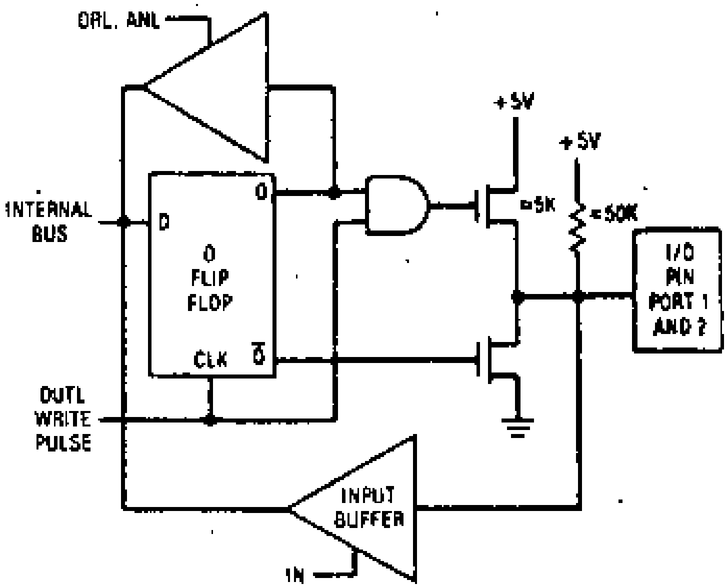


Figure 4. "Quasi-bidirectional" I/O port.

Both the on-chip and expander I/O port instructions contain the port number as an immediate value in the instruction; it is not possible to specify the port number dynamically in a register. This makes it impossible to write reusable I/O handlers for identical devices on different ports of the MCS-48 or of the same expander chip. The same problem exists in the MCS-48's older brother, the 8080. However, it is less serious in the MCS-48 for two reasons. First, the MCS-48 is intended for smaller applications less likely to employ many copies of the same I/O device. Second, the available I/O expansion modes do allow dynamic device selection when each device uses a separate I/O chip.

The processor architecture directly supports only 256 bytes of external data memory and four external 4-bit I/O ports. However, the amount of external data memory and I/O can be increased to any practical amount using on-chip I/O-port bits to implement program-controlled bank switching.

In addition to the I/O ports, an MCS-48 has three additional input pins that can be tested by conditional jump instructions. All are multipurpose pins—T0, which can be set up as a clock output under program control; T1, which can be used as the input to the on-chip timer/counter; and the external interrupt input.

The MCS-48 accepts interrupts from two sources—a level-sensitive input pin and an on-chip timer/counter. When an interrupt is serviced, the 12-bit PC and four status bits (carry, half carry, flag 0, register bank select) are pushed onto the internal stack. Depending on the source, a jump to either location 3 or location 7 is taken. The interrupt system is single-level; interrupt service routines cannot be interrupted. An interrupt return instruction restores the PC and status bits and allows further interrupts to be serviced.

At the time of this writing, the T1 interrupt input is generally useless for counting or timing asynchronous external events, because the current chip

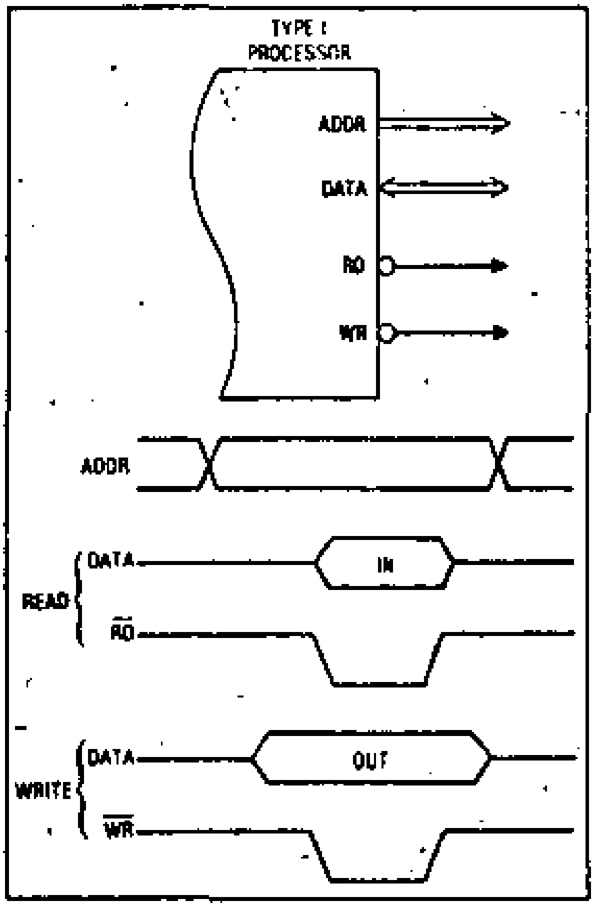


Figure 5. Type 1 bus control signals.

features a poorly designed synchronizer that sometimes misses input edges and hence skips counts. This is the second time in six months that I have seen an LSI chip whose designers were apparently unaware of problems in synchronizer design (the other was the Z80-SIO, which Zilog has since fixed). I would suggest that chip designers read some of the papers on the subject[4] and that academics warn their students of the increasing likelihood of synchronization problems in modern system design.

Ease of programming

Compared to some of the older 4-bit and 8-bit microprocessors, the MCS-48 is a nice machine to program, but it leaves much to be desired compared with an M6801, a Z8, or even an 8085. The single-accumulator architecture, the lack of index registers, and the absence of even a direct data memory addressing mode means that the programmer must constantly be moving things back and forth between the accumulator, the two "pointer" registers R0 and R1, and the rest of the data memory (and keeping track of them!). One may write macros to ease the burden somewhat, at the expense of more inefficient code in the cramped address space. For example, one can write a macro to simulate a direct data-memory-addressing mode:

```
LDA     MACRO MEMADDR
        MOV R0,#MEMADDR
        MOV A,@R0
        ENDM
```

A tale of two buses (or, different strobes for different 'phobes)

A microprocessor memory and I/O bus has many identifying characteristics—data and address word length, multiplexed or nonmultiplexed address and status, separate or memory-mapped I/O, and others.[3] It is interesting to look at two popular read/write clocking arrangements.

Let us call the first technique a Type I (or 1) interface, used by the Intel 8085, 8086, and MCS-48 families. As shown in Figure 5, there are two mutually exclusive control pulses, $\overline{\text{RD}}$ and $\overline{\text{WR}}$, that indicate a read or write operation.

We'll call the second technique Type Z (or 2), used by the Zilog Z8, Z80, Z8000, and also by the Motorola 6800 family. (Perhaps it should be Type M because the M6800 came first, but Z looks more like 2.) It is also used in principle by MCS-48 expander ports. As shown in Figure 6, there is a single control pulse, $\overline{\text{REQ}}$, and a level signal $\text{R}/\overline{\text{W}}$ that indicates which type of operation is to take place. The timing of $\text{R}/\overline{\text{W}}$ is similar to that of an address signal.

Figure 7 shows how to use a Type Z processor with a Type I peripheral chip. The decoding shown in the figure can be easily implemented with one-half of a TTL 74LS139 dual 2-to-4 decoder (this even leaves an extra control input for distinguishing between memory and I/O if desired). Assuming that processor and peripheral speeds are comparable, there should be no problem in satisfying the timing requirement of either the processor or the peripheral chip.

Figure 8 shows an attempt to use a Type I processor with a Type Z peripheral chip. The logical AND of $\overline{\text{RD}}$ and $\overline{\text{WR}}$ from the processor nicely produces $\overline{\text{REQ}}$ for the Type Z peripheral. $\overline{\text{RD}}$ has the correct logic value to serve as $\text{R}/\overline{\text{W}}$, but its timing is a problem. The Type Z peripheral expects $\text{R}/\overline{\text{W}}$ timing to be similar in character to an address signal; that is, it should be valid long before the $\overline{\text{REQ}}$ pulse appears. The only way we could ensure this would be to artificially delay $\overline{\text{REQ}}$ long enough for $\overline{\text{RD}}$ to satisfy the setup time of $\text{R}/\overline{\text{W}}$. Unfortunately, such a delay (unless highly asymmetric) would also delay the trailing edge of $\overline{\text{REQ}}$ until long after $\text{R}/\overline{\text{W}}$ ($\overline{\text{RD}}$) had gone away—again a problem.

The cleanest way to use a Type I processor with Type Z peripheral chips is to use an address line as $\text{R}/\overline{\text{W}}$. For example, the least significant bit of the I/O port address could be reserved as $\text{R}/\overline{\text{W}}$. Hardware decoding of actual port numbers would use the higher-order bits; then software would have to ensure that writes always used odd port addresses, and reads used even.

The conclusion is that it is simple to connect Type I peripherals to Type Z processors, but that the reverse can be difficult. Is this just the way it turned out or was there method to this madness?

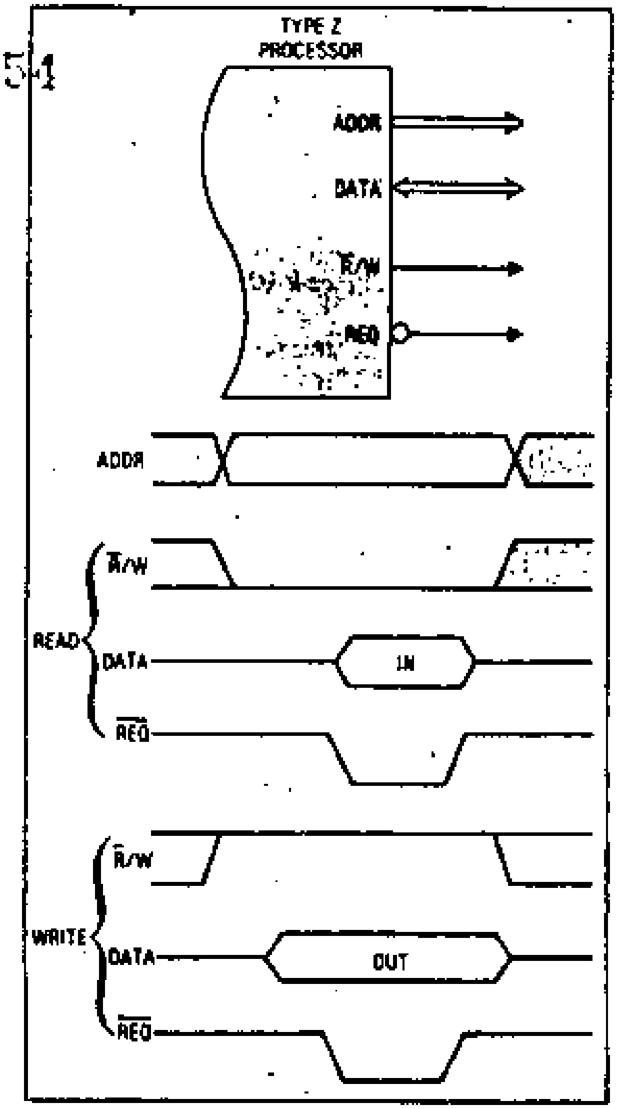


Figure 6. Type Z bus control signals.

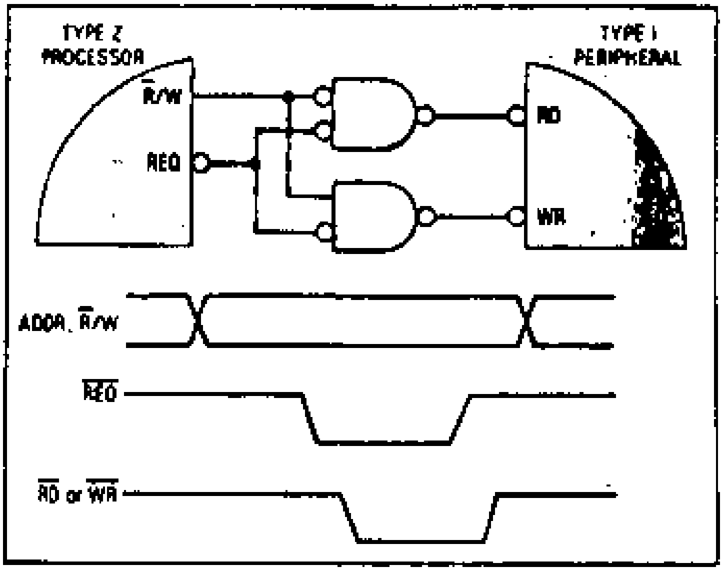


Figure 7. Acceptable Z-to-I interface.

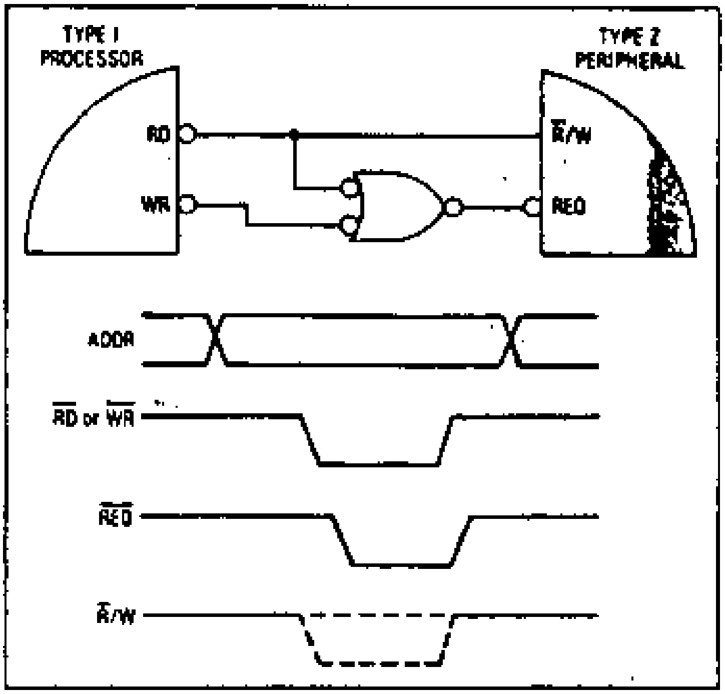


Figure 8. Unacceptable I-to-Z interface.

To use such macros, however, the programmer must give up some registers for use by the macros. In the above example, macros would use R0, leaving only R1 available to the program as a pointer variable. (Buffer copying and other routines requiring two or more pointers get to be a problem.)

The lack of symmetry in the instruction set also creates programming headaches. For example, why are there INC R0, DEC R0, and INC @R0 instructions, but not DEC @R0? Or, why can we conditionally jump on C, Z, T0, and T1 conditions true or false, but on F0, F1, TF, and accumulator bits only true? Except for accumulator bits false, the proper "holes" exist in the instruction set; in fact, it probably took more logic to turn the instructions off than to let them work. I have been told that these "unimportant" instructions leave room for future enhancements, but any worthwhile architectural enhancements would require changes more sweeping than a few special-purpose opcodes.

While nice programs can be written for the MCS-48, they take more effort than those written for a "general-purpose" architecture. Programming difficulty and expense also increase as we bump against the memory limits of the 8048. If we are writing one short (< 1K-byte) program and shipping 50,000 units with it, the programming expense is quite justifiable. If we are writing 10 different 2K- to 4K-byte programs, each of which will be shipped with 5000 units, we might be better off selecting a cleaner (albeit more expensive) machine.

Some electrical characteristics

The MCS-48 uses Intel's reliable *n*-channel silicon-gate MOS process. Several second sources for the family have been announced—AMD, NEC, Signetics, Siemens, Intersil, RCA. Both Intersil and RCA have announced plans for CMOS versions of MCS-48 chips.

The MCS-48 chips use a single +5 volt supply and have logic input and output levels that are fully TTL compatible. As with all MOS microprocessors, the output drive is limited—typically four or five low-power Schottky (LSTTL) unit loads.

MCS-48 chips contain an oscillator to generate the processor clock (nominally 6 MHz) from an external crystal or RC circuit. It is also possible to connect an external clock directly to the oscillator input. The output of the oscillator feeds a divide-by-three counter whose output (nominally 2 MHz) controls the internal states of the processor. Since the divide-by-three counter cannot be synchronized externally, processor I/O cannot be referenced very well to the 6-MHz clock for tricky interfaces, nor can processors be run in lock-step from a common clock in a triplicated microcomputer (author's pet project).

The MCS-48 family satisfied the usual Intel strategy of being the first in the marketplace with an imperfect but useful product.

The MCS-48 chips have an active-low reset input pin connected to an internal high-impedance pullup resistor and Schmitt trigger. Thus, power-on reset can be accomplished by an external 1 mF capacitor. It is a little more difficult to add a logic-controlled reset (for example, by a watchdog timer), since open-collector or discrete transistor drive is required. And unless the driver circuit is sophisticated, a logic-commanded reset will disable the processor for a long time, due to the time constant of the power-on reset circuit—about 200 msec. In any case, reset destroys the state of the processor. It would be nice to have a nonmaskable interrupt (as in the 8085) that could be used for applications such as watchdog timers.

Development tools

Intel supports two major development tools for the MCS-48 family—a cross assembler and an in-circuit emulator, ICE, both of which run on an Intel MDS microcomputer development system. Unfortunately, Intel does not support any MCS-48 assemblers or simulators that run on a large computing system, a necessity for any large development project. However, they can be obtained from independent software houses and consultants such as Microtec.

Intel MDS software for the MCS-48 lacks the consistency one expects from a good set of software tools. For example, there are at least three very different syntaxes that an engineer or programmer might use to change the value of a memory location in the MDS, depending on whether the monitor, ICE, or PROM programmer is being used. The monitor has a nice syntax that allows us to open a location, change it, and continue to the next location with a small number of keystrokes. In ICE, to read and change four locations, we must type (machine type underlined):

```
* CBYTE 144 TO 147
0144H = 01H 3AH BFH 59H
* CBYTE 144=11,27,FF,6A
```

To do the same thing in the PROM programmer, the programmer types:

```
* DISPLAY FROM 144 TO 147
0090 01 3A BF 59
* CHANGE 144=11,27,FF,6A
```

In either syntax, one wastes keystrokes, and it's easy to lose track of the address in a long string. This isn't too terrible until we discover that ICE has interpreted and printed addresses and data in hex, while the PROM programmer has assumed inputs in decimal and printed outputs in hex (except for input FF, which the PROM programmer rejects because it looks like an assembler label!).

Another constant annoyance is that the MDS accepts only the RUB character for deleting characters (echoing the deleted character); backspace is not supported. It would have been easy enough to support both erase characters, making both teleprinter and CRT users happy.

Conclusion

The MCS-48 microcomputer family was a reasonable contribution to the state of the art when it was introduced in 1976, in spite of its flaws. It achieved its design goals and satisfied the usual Intel strategy of being the first in the marketplace with an imperfect but useful product.

The MCS-48 is an acceptable choice for applications with initial estimated requirements of less than 1K bytes of program store, 64 bytes of data memory, and only one or two different programs to be developed. Designers whose applications require more memory or different programs in different chips should seek a more general-purpose architecture, such as the Z8 or 6801.

I have spent much of this article complaining that the MCS-48 is not a general-purpose 8-bit microcomputer. This may seem unfair, since some people at Intel claim the MCS-48 was never intended to go much beyond the old 4-bit market. So why criticize it on that basis? First, to help designers who might otherwise be tempted to use it in a larger application (Intel sales engineers frankly recommend their three-chip 8085 system in such cases). Second, to help designers who have already selected the MCS-48 for a larger application. Third, because discussion of general system requirements should benefit future system designers and chip designers alike. Finally, because Intel does not now offer a clean architecture suitable for the more general-purpose, single-chip, expandable microcomputer market, and I think they should. ■

-000 56

Acknowledgments

I appreciate the comments of my colleagues during the preparation of this article, especially those from Prem Jain, Paul Frantz, Ed Yarwood, and Dave Stearns. The comments of the reviewers were also very helpful. Thanks go to Dennis Allison for suggesting this article over a year ago, and to Bernard Peuto and Len Shustek for making me finish it. Of course, special thanks go to Intel for bringing us the MCS-48.

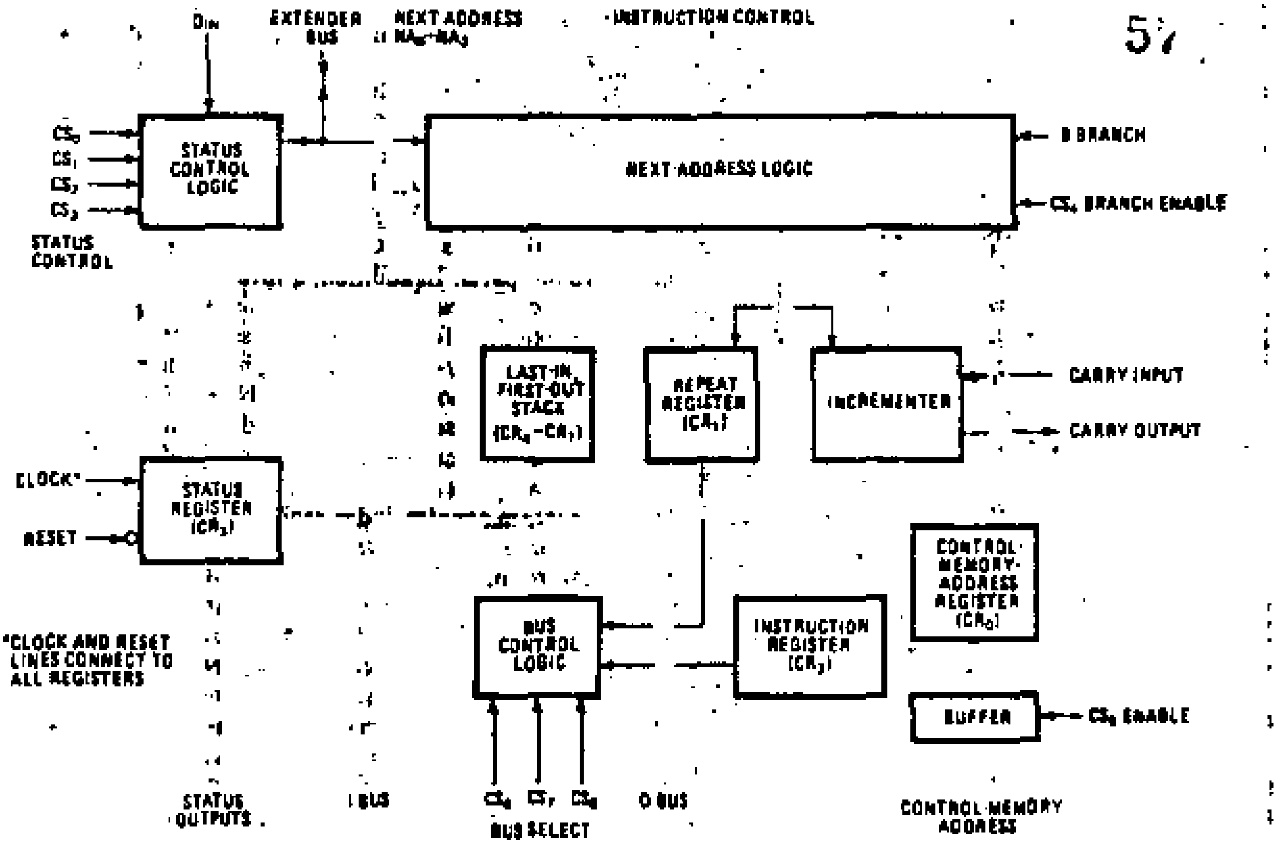
Some of this material was prepared while I was a lecturer at the Digital Systems Laboratory at Stanford University. Other material is adapted from my textbooks, *Microcomputers, Vol. 1: Architecture and Programming* and *Microcomputers, Vol. 2: System Hardware Design*, to be published by John Wiley & Sons in late 1979. All opinions expressed in this article are my own.

References

1. Intel Corporation, *MCS-48 Family of Single Chip Microcomputers User's Manual*, Santa Clara, Calif., July 1978.
2. L. J. Shustek, "Analysis and Performance of Computer Instruction Sets," PhD dissertation, Stanford University, Jan. 1978, available from University Microfilms, Ann Arbor, Mich.
3. J. F. Wakerly, "Microcomputer Input/Output Architecture," *Computer*, Vol. 11, No. 2, Feb. 1977, pp. 26-33.
4. T. J. Chaney, S. M. Ornstein, and W. M. Littlefield, "Beware the Synchronizer," presented at COMPCON-72, IEEE Comput. Soc. Conf., San Francisco, Calif., Sept. 12-14, 1972.
5. T. J. Chaney and C. E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits," *IEEE-TC* (Corresp.), Vol. C-22, No. 4, Apr. 1973, pp. 421-422.
6. M. Pechoucek, "Anomalous Response Times of Input Synchronizers," *IEEE-TC*, Vol. C-25, No. 2, Feb. 1976, pp. 133-139.

John F. Wakerly is an independent consultant and a lecturer at Stanford University, where from 1974 to 1978 he was an assistant professor of electrical engineering. He has published many technical articles in the areas of computer reliability, microprocessors, and digital systems education, and is the author of two textbooks, *Logic Design Projects Using Standard Integrated Circuits* (Wiley, 1976) and *Error-Detecting Codes, Self-Checking Circuits, and Applications* (Elsevier North-Holland, 1978).

Wakerly received the BEE degree from Marquette University in 1970 and the MSEE and PhD from Stanford University in 1971 and 1973, respectively. He was a Hertz Foundation Fellow during his studies at Stanford. He is a member of the IEEE Computer Society, ACM, Sigma Xi, Tau Beta Pi, and Eta Kappa Nu.



2. Controller. The MC10801 microprogram-control chip, which is also a 4-bit-wide slice, generates the microprogram address and provides the logic for complete sequence control. Five address buses interface to microprogram memory, to other parts and to external test points.

The 10800 operates with three data ports, as shown in Fig. 1. The I and the O buses are both 4 bits wide and bidirectional. The third, the A bus, is a 4-bit-wide input-only port. Control of the ALU is by 17 select lines, AS_0 through AS_{16}. The select lines control all circuit functions and determine the source and destination for ALU data. A full set of condition-code outputs, which include parity, carry, overflow, and zero-detect, simplify branch testing. Unique among bit-slice processor elements is the 10800's ability to perform both binary and binary-coded-decimal (BCD) arithmetic with equal ease and speed. Direct BCD arithmetic is gaining in popularity in business computers, process controllers, and test systems where human interface is most often in a BCD format. The chip also features a signal overflow shift network that indicates when an arithmetic left shift has prompted a sign-bit change.

The MC10801 microprogram-control chip is the companion part to the processor element and carries out the sequencing of operations. The development of the 10801, which packs 550 equivalent gates onto a 25,000-square-mil chip, proceeded all the more confidently because of the high yields already obtained with the less complex 10800 part.

The 10801, also a 4-bit-wide slice, is shown in Fig. 2. The control-memory-address register CR_0 holds the microprogram memory address, while the remaining blocks in the figure provide logic for the sequencing operation. Register CR_1, called the repeat register, is a special feature that adds greatly to the 10801's speed and flexibility. Aside from its usefulness as a cycle counter, which allows single instructions or subroutines to be executed a specific number of times by automatically keeping track of loop count, testing for end count, and remaining in or leaving the loop on test result, register CR_1 further provides a return destination for microprogram interrupts.

Register CR_2 is set up to hold a machine instruction starting address or an interrupt vector. Register CR_3, however, is unique to the 10801 in that it interfaces microprogram control to external test points. The register can be loaded with any given bits of status information, whereupon it will test those bits for conditional microprogram jumps. Moreover, its contents can be set or cleared under program control to signal the processor's status. Finally, CR_3 can hold the page address in a word- or page-organized microprogram. In that case, memory address register CR_0 would hold only the microprogram word address.

Registers CR_4-CR_7 form a four-word last-in, first-out (LIFO) stack for nesting subroutines within a program. With the logic built into the 10801, operation of the LIFO is completely automatic. If needed, however, the LIFO can be extended or tested for full stack through the I bus

[illegible] 58

INC	Increment
JMP	Jump to next address inputs
JIB	Jump to I bus
JIN	Jump to I bus and load CR_3
JPI	Jump to primary instruction (CR_3)
JEP	Jump to external port (O bus)
JL2	Jump to next address inputs and load CR_2
JLA	Jump to next address inputs and load address into CR_1
JSR	Jump to subroutine
RTN	Return from subroutine
RSR	Repeat subroutine (load CR_1 from next-address inputs)
RPI	Repeat instruction
BRC	Branch to next address inputs on condition, otherwise increment
BSR	Branch to subroutine on condition; otherwise increment
ROC	Return from subroutine on condition; otherwise jump to next-address inputs
BRM	Branch and modify address with branch inputs (multiway branch)

or the O bus ports. Two branch inputs—the branch (B) and extended branch ($\overline{XB}$)—supply status for conditional microprogram jumps.

The whole part is tied together with 16 instructions that are built into the next-address logic block. These instructions, listed in Table 1, control the source for each new microprogram word address and have been designed to save both microprogram memory size and development time. For example, an 8-bit shift in the ALU can be done with only two microprogram words—a repeat-subroutine instruction (RSR) to load the repeat number (8) into register CR_1, and a repeat instruction (RPI) to perform the eight shifts.

Control of timing

Clock control, often one of the most complex segments of processor design, is implemented with a single chip—the MC10802. As shown in Fig. 3, the chip contains the logic to generate multiple phases, simplify system start and stop, and provide some diagnostic capability.

The 10802 takes a clock input, usually from a crystal oscillator, and splits the signal into separate phases with its four-phase shifter block. The number of different phases can be programmed for two, three, or four phases. The go/halt, run/maintenance, and start inputs control system start and stop operations. The single-cycle/single-phase input helps with diagnostics by advancing the system one clock phase or a complete cycle for each starting signal input. Finally, a synchronizer network built into the part eases interfacing to the start input.

Hooking the bit-slice processor to slower memory and peripherals calls for the MC10803 interface chip. The part is designed for maximum speed: it can simultaneously route data while addressing the memory or peripherals. With 600 equivalent gates packed into a 21,000-square-mil area, the 10803 is actually denser than the 10801 microcontroller. The difference is due to a less complex metalization pattern used in the memory interface chip.

The 10803 has its own ALU. Also organized as 4-bit slices, several 10803s may be connected in parallel to meet any particular system data and address requirements. As shown in Fig. 4, the circuit has data and address ports for interfacing to peripheral equipment, plus the I bus and O bus for connecting directly to other 10800 parts. In addition, a fifth port with pointer inputs to the ALU can be used as a source of address modifiers or constants for memory addressing.

A memory-address register holds the memory address while a memory-data register buffers incoming or outgoing data. A separate four-word register file stores information that is needed in the course of memory addressing, such as the page addresses or the value of the program counter, index register, or stack pointer.

Some select inputs to the data-interface logic have control of a total of 17 data-transfer operations between buses and registers. Other select inputs control the function of the ALU and determine the source and destination of data through microfunctions and destination-decoding logic. Although the ALU is normally used for memory addressing, it can perform seven basic functions—add, subtract, OR, AND, exclusive-OR, shift-left, and shift-right—on a wide variety of data sources.

Certain systems can take advantage of the 10803 to reduce parts count. A peripheral controller, for example, transfers and formats data and usually requires little arithmetic capability. Such a system uses the 10803 both for input/output (I/O) control and as its main ALU, eliminating the need for a 10800.

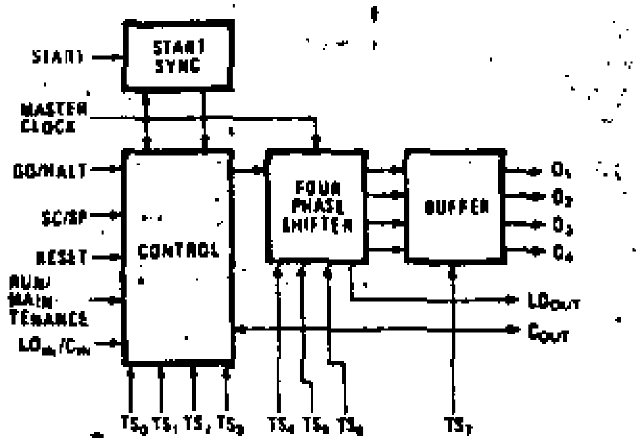
Tying the parts into a system

The simplicity with which the parts of the 10800 bit-slice family can be assembled into a microprogrammable 16-bit minicomputer or signal processor is evident from Fig. 5. Although structures will vary depending on the application, the example illustrates several key features. Eleven LSI chips, together with a few medium-scale integrated parts and supporting high-speed memory devices, are all that is required.

Bus ports on the 10800, 10801, and 10803 directly interconnect. The 10801 microprogram controller supplies an address to microprogram memory, selecting one microprogram word. Each word is divided into groups of bits called fields (represented by the broad arrows at bottom). Each field independently controls a system section. Since all fields are present at the same time in each microprogram word, the various system sections can operate simultaneously for maximum system speed.

A system function performed by all the fields in one microprogram word is called a microinstruction. Several microinstructions may be required for one machine instruction. System performance, therefore, is determined by the number of microinstructions in a system and the speed of each microinstruction. Microinstruction cycle time for a 16-bit 10800-family-based system is about 100 nanoseconds.

The operation of the system in Fig. 5 can be explained



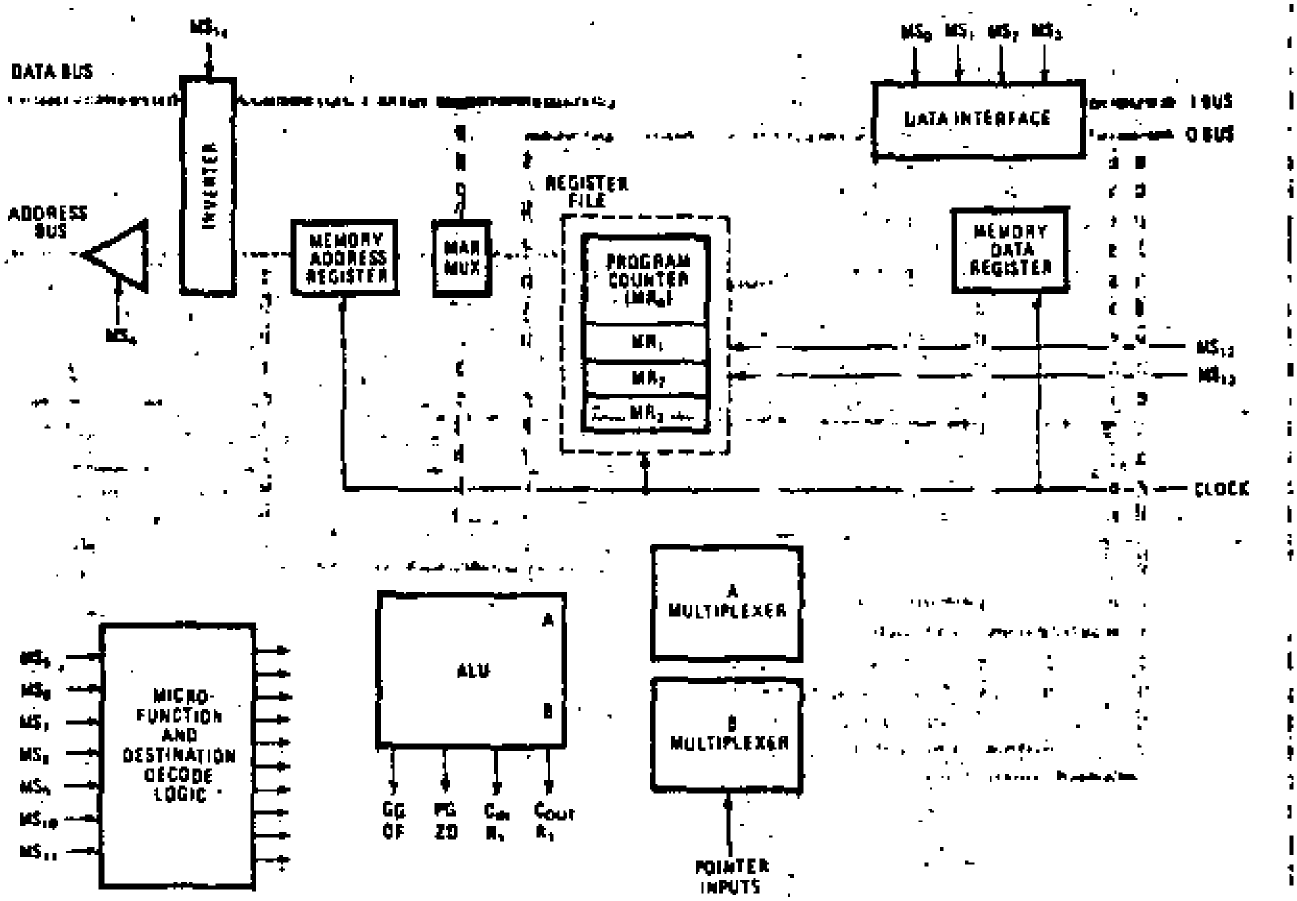
3. Timer. Many of the usual timing problems in bit-slice processor design are handled by the MC10802 timing function chip, including system start, stop, and clock control for diagnostics. A start synchronizer input simplifies interfacing to front-panel switches.

in terms of the relationship of microprogram fields to the LSI blocks. Two fields controlling the 10801 generate each new microprogram address. An instruction field selects one of the 16 program-flow instructions given in Table 1, and once selected, all logic needed to execute the instruction is contained in the 10801. However, some instructions require additional information. For example, a jump-to-next-address or jump-to-subroutine instruction requires a destination, which is supplied by the next-address microprogram field. An important feature of the 10801 is the ability to route next-address data through the O-bus port for ALU or memory interface constants, bit-mask patterns, and offsets when the field is not required for microprogram flow.

Branch control is a third field associated with microprogram addressing. Most programs have to make a large number of flow decisions either from ALU condition codes, such as zero-detection, overflow, and sign bit, or from external test points. Under command of the control field, those status signals are multiplexed into the 10801 branch inputs through control logic. Branch instructions that have been built into the 10801 include branch on condition, branch to subroutine, and branch and modify.

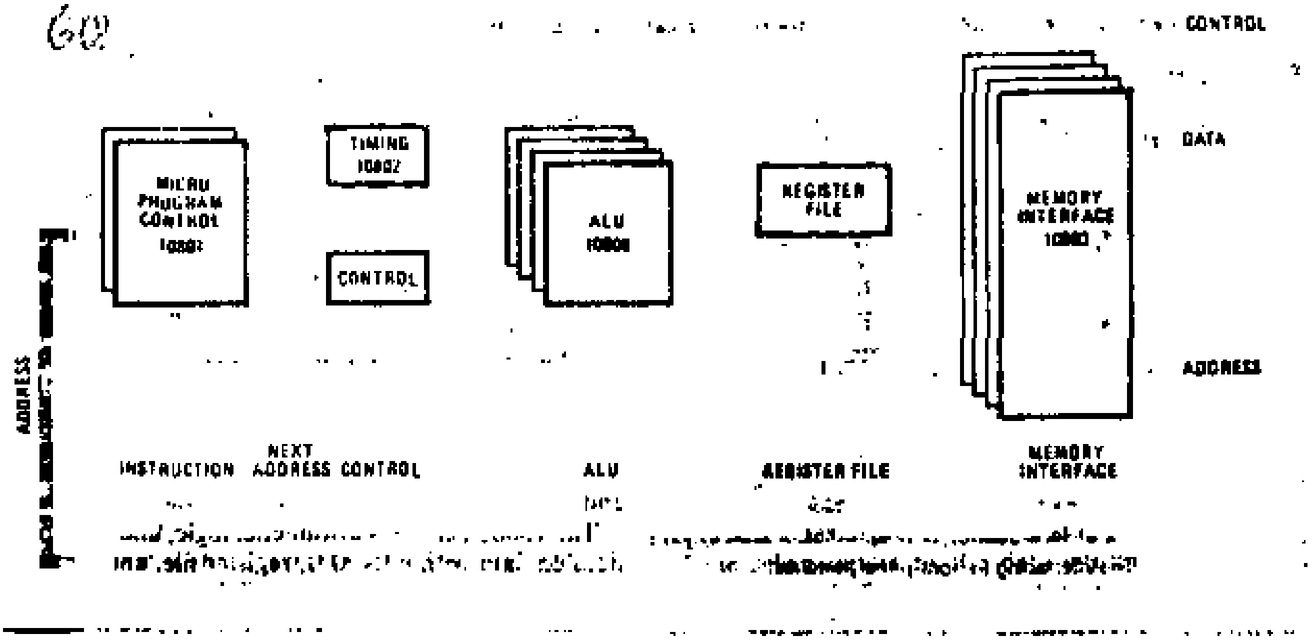
The 10800 performs arithmetic, logic, and shift operations on data within its ALU, register file, and/or memory interface. The bus structure of the processor in Fig. 5 also allows the ALU to generate microprogram addresses through the I bus, if required. Moreover, the ALU will operate in either binary or BCD data formats as controlled by the ALU microprogram field.

Unlike most other bipolar bit-slice families, the 10800 family leaves the register file as a separate block. The



4. Interface. Interfacing the bit-slice processor to peripheral equipment through data and address buses, the MC10803 has separate select lines and provides for parallel data transfer and address generation within one microprogram cycle It is expandable as necessary

60



6. Tying it together. This design for a 16-bit minicomputer or signal processor uses only 11 LSI parts from the 10800 all-ECL family plus some ECL MSI parts and memories. Microcode for the processor is a wide word, shown at bottom; broad arrows indicate control fields.

advantages of that are the possibility of file expansion to any size and the flexibility of organization. But most important is the ability of the ALU and memory-interface circuits to share the register file without tying each other up. Some of the devices that may be used for the register-file block include MC10145 16-by-4-bit RAMs and the MC10806 dual-address stack.

The 10803 interfaces to peripheral equipment through data and address ports. The I and O buses can route I/O information directly to or from the required internal processor circuits. An architectural strength of the family is the 10803's ability to transfer data and to generate memory addresses independent of ALU operation. When not used for I/O control, the ALU in the 10803 can even be paralleled with the main ALU for double-precision arithmetic.

Beyond the main components already discussed that are required for processor design, bus-interface parts as well as special-purpose LSI circuits are needed to solve additional system problems. The MC10804 through MC10808 fall into those categories.

More circuits

The 10804 and 10805 are bidirectional level translators for hooking ECL buses to transistor-transistor logic. Many high-speed ECL processors will need to interface to TTL-compatible peripherals. Other translators, such as the 10124 and 10125, handle the interfacing of individual address and control lines. The 10804 and 10805 offer complete interfacing with translation of bidirectional data buses. The 10804 is a 4-bit-wide part in a 16-pin package, while the 10805 handles 5 bits in a 20-pin package. The two can also be combined to permit efficient translation of 9-bit data bytes. Figure 6 shows the internal logic used by both types for each bit.

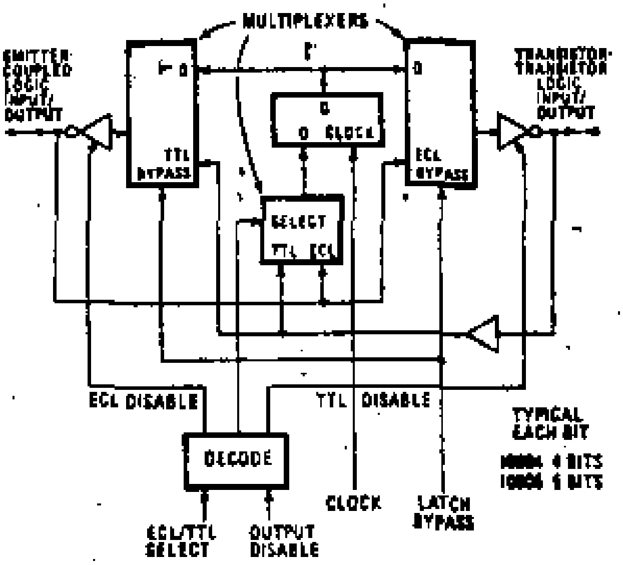
A MECL/TTL select line controls the direction of data through the circuit. That combines with an output disable to force the TTL side into a three-state mode and ECL to an equivalent low-logic-level output. An internal latch holds data in either direction. The latch-bypass input routes data around the latch for faster translation time or to transfer data while holding information in the latch. The 10805 is also designed to drive a heavy capacitive load and, as such, can interface an ECL processor directly to MOS main memory with high speed and a minimum of parts.

A useful variation

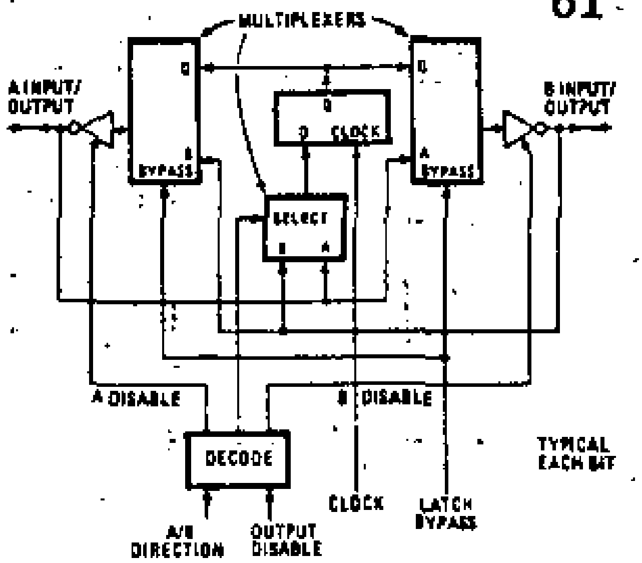
The MC10807 5-bit transceiver was produced when the demand arose for an ECL circuit with 10805 functionality, but without TTL-level translators. This circuit, shown in Fig. 7, is identical with the 10805 but has ECL signals on both ports. The buffered bidirectional ECL driver/receivers allow complex ECL bus networks to maintain a full transmission-line environment.

The MC10806 dual-address stack represents a milestone in standard ECL LSI products. Although equivalent gate count is not entirely pertinent as a description of a memory-type part, circuit complexity can be judged from the chip's large 35,500-mil-square area. Again, experience gained with the 10800 family has allowed circuit complexity so to increase that the 10806 is almost two and a half times larger than the 10800.

The 10806 is intended primarily as a data buffer between the high-speed ECL processor and slower peripherals; however, a versatile dual read/write bus structure also allows the circuit to function as a last-in, first-out or first-in, first-out stack, or as a 10800-system register file. The 10806 has two bidirectional data buses, the A and the B bus, each with access to a 32-by-9-bit memory (Fig. 8). Having output register latches in series wit both data buses gives the circuit a master-slave appearance where the memory cells are master stages and the latches are slaves.



6. Translators. The MC10804 and 10805 bidirectional level translators interface ECL and TTL circuits. The circuit for a single bit is shown; the internal latch can be used to compensate for an ECL bus, which may be much faster than the TTL side.



7. Transceiver. Basically the same logic used in the 10805 level translator, the MC10807 5-bit transceiver has all-ECL inputs and outputs. Having a transceiver in a single package and not several greatly reduces fanout problems and chip count.

The A- and B-address inputs each select a memory word for the corresponding buses. Separate clocks and write- and data-enable lines control read and write operations for each side. Parity checking is automatic on the 9-bit data word and optional on the address inputs. When parity is used, the write operation is prohibited on any address with incorrect parity. Parity can be bypassed in those systems not requiring it.

The ability to write from both data ports could lead to priority problems: writing different data on the A and B data buses into the same word address could cause loss of one or both data words. In order to avoid such an occurrence, address-contention logic in the 10806 looks at write-enables and addresses to detect and indicate address conflicts.

Multibit shifter

The MC10808 programmable multiple-bit shifter is another example of standard LSI that solves a system problem. Using ALU circuits, which shift data only 1 or 2 bits per microinstruction, becomes time-consuming when data must be shifted many places for such jobs as formatting, bit testing, or normalizing floating-point numbers. The 10808 shifts data any number of bits in a single 10-ns pass.

Each circuit is a 16-bit shifter organized as shown in Fig. 9. The algorithm used can be combined with the ECL wired-OR feature for unlimited expansion—4 chips for a 32-bit shifter, 16 for a 64-bit shifter, and so on. Since expansion is in a horizontal manner, only the delay of a single part results, regardless of shifter size.

The number of bits shifted is programmed through scale-factor inputs. Data-shift inputs select one of the eight possible shift types or output controls listed in Table 2. Two of the shift-right and -left instructions (SRC and SLC) program the scale factor as a 2's complement number, controlling both direction and distance of shift. A sign-bit input controls the sign-bit polarity and thus allows the circuit to operate in either positive or negative logic formats.

With the additional 10800 chips, the extremely high-performance processor of Fig. 10 can be built. Translators at interface points give the processor TTL-compatible inputs and outputs. A 10806 data buffer holds 32 data words, allowing the TTL bus to be slower than the ECL system microinstruction cycle time. The 10806 dual-bus structure easily adapts to the bidirectional data bus, so that both incoming and outgoing data can be stored.

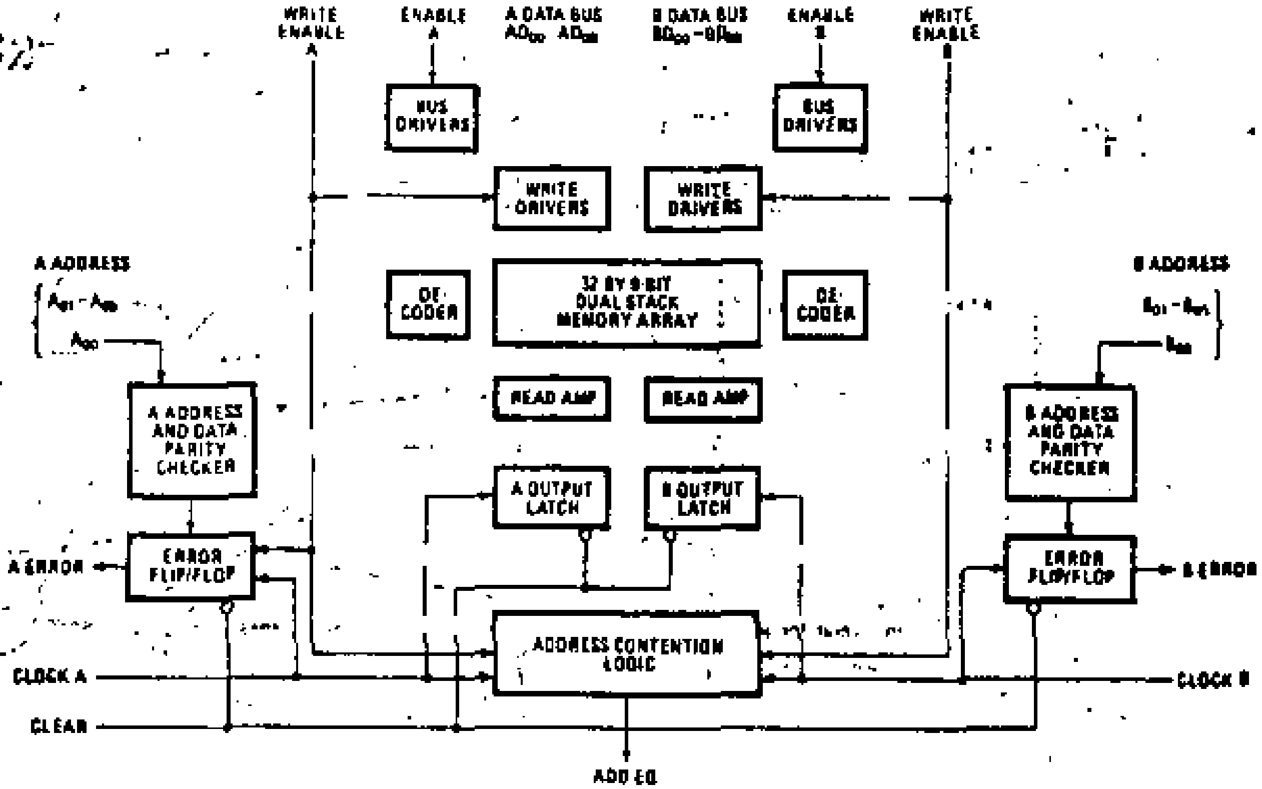
Obtaining higher performance

The data bus following the 10806 buffer is routed directly to the microprogram control. Compared to Fig. 6, this bus structure saves instruction-execution time since a starting address need not go through memory interface, but the ALU no longer has a direct path to microprogram control.

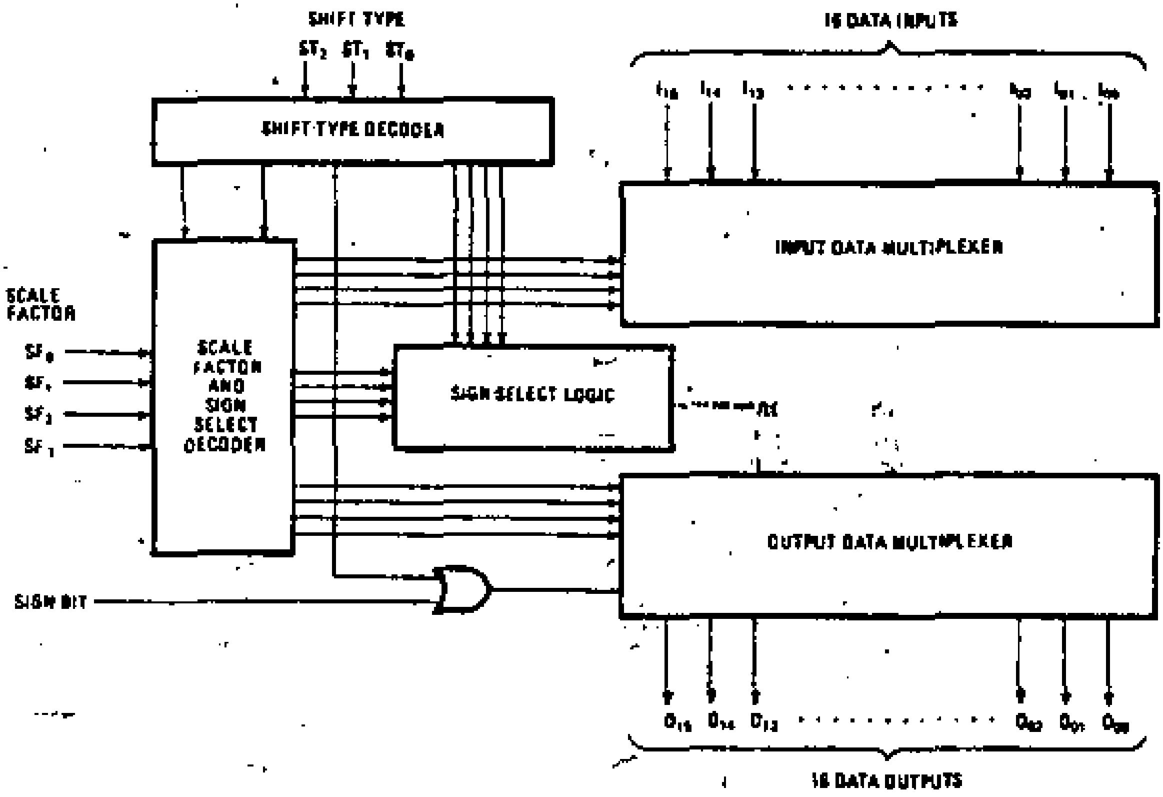
The system also uses the 10806 for a register file. The first half of a microinstruction reads a register file via the O bus, and then the second microinstruction half routes ALU results to memory interface or back to register file with the I bus. O bus latch, internal to the 10800 holds the O bus input during the second to eliminate race conditions within the ALU. The other register file port routes data through the shifter to the ALU A bus, to memory interface, or to microprogram control. The 10808 shifter is placed in front of the ALU for single-pass shift and test. Data shifting through the part only takes about 10 ns, and therefore the series arrangement has little effect on microinstruction cycle time. Moreover, the same microprogram ALU field can control both the 10808 and 10800 for a very powerful ALU function set.

A pipeline register is placed between microprogram memory and the data-handling LSI circuits to reduce microinstruction cycle time. The register permits parallel

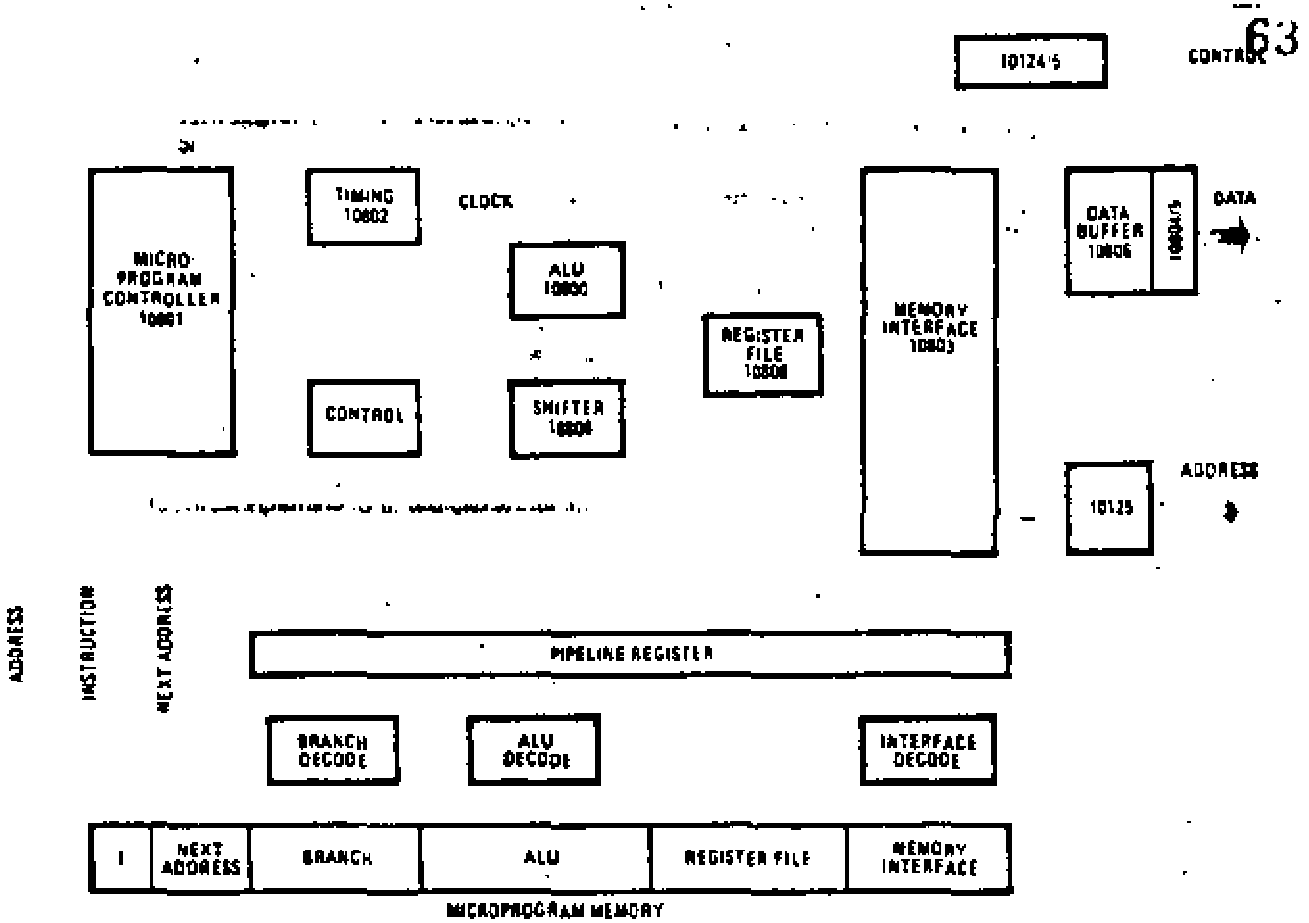
62



8. Stacked. The MC10806 dual-address stack has a 32-by-9-bit memory array with two independent read/write data ports. The dual-port structure combines with internal parity checking (which may be bypassed if unneeded) and solves many buffer-memory design problems.



9. Shifter. A programmable multiple-bit shifter, the MC10808 shifts 16 input bits from 0 to 15 places to the left, to the right, or in rotation. Cascadable like the other bit-slice parts, the circuit can be easily expanded to larger word sizes at no sacrifice in speed.



10. Superprocessor. Hooking the 10808 in series with the 10800 and supporting the 10806 with TTL translators creates an extremely powerful processor. The design also has a pipeline register to reduce cycle time, plus microprogram field decoding to cut word length.

[illegible]

ALS	–	ARITHMETIC SHIFT LEFT
ARS	·	ARITHMETIC SHIFT RIGHT
RLT	–	ROTATE LEFT
RRT	–	ROTATE RIGHT
SRC	–	SHIFT RIGHT 2'S COMPLEMENT
SLC	·	SHIFT LEFT • 2'S COMPLEMENT
ODA		OUTPUT DISABLE
SBO	–	SIGN BIT AT ALL OUTPUTS

operation between microprogram control and the rest of the processor. While the ALU, register file, and memory interface are executing one microinstruction, the 10801 is generating a new microprogram memory address. Pipelining is optional in a system built with the 10800 family—the 10801 interfaces directly to the microprogram in either case.

A final feature of the high-performance processor of Fig. 10 is the use of branch, ALU, and interface-decoding logic. Those blocks allow a relatively wide pipeline register feeding a large number of LSI control inputs to be driven from a narrow microprogram word. For example, a 6-bit microprogram field can select 1 of 64 ALU instructions. The ALU logic, however, may require 12 to 20 control inputs. The fanout is performed in decode logic commonly built with fast 10139 programmable read-only memories (PROMs). In addition to reducing microprogram size for cost reasons, decoding logic allows microprogram fields to be structured for easier programming. The decoding logic does not slow system performance since it is possible to go from clock to the 10801's address output through microprogrammed RAM or PROM and from decoding logic to pipeline register, all within the cycle time of one microinstruction.

Future ECL LSI

Motorola has developed a MECL 10,000 Macrocell-array integrated circuit that is compatible with the 10800 parts and allows rapid development of high-speed LSI circuits with complexities of up to 750 equivalent gates. Although the Macrocell array will be used to develop specialized circuits for specific customers and systems, the advantages of this LSI concept will also lead to new standard products in the 10800 family.

The plans are thus to use new circuit developments to build on the 10800 family rather than around it. New functions under consideration include an advanced 8-bit arithmetic-and-logic and a very high-speed, expandable LSI array multiplier. These circuits, plus the Macrocell array concept, will be featured in an article on ECL LSI in the next issue. □

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

EJEMPLOS DE LENGUAJE ENSAMBLADOR

Julio, 1981

1

```
;          HL = # DE MONITOR(INDICE)
;          IX - DIR. DEL # DE CONTRA-RECIBO
;A LA SALIDA:
;          HL - DIR. DE COLOCACION
;          BC - DIR. APUNTADOR DE INSERCION
;          DE = DIR. DE INSERCION
;          A = FF SI YA SE ENCUENTRA EN MEMORIA
;          A <> 0 SI NO SE ENCUENTRA EN MEMORIA
;          A'=NUMERO DE MONITOR
;          IY=DIR. DEL # DE CONTRA-RECIBO
;
 CR: EQU 0DH
 LF: EQU 0AH
 MEMIM: EQU 1500H
INDICE:     DEFB 0          ;EN ESTA VARIABLE SE ALMACENA EL CAMBIO
                            ;POR MONITOR (BITS 3,2,1 & 0)
;
COLOCA:     LD A,L
            LD (INDICE),A ;A = NUMERO DE MONITOR
            EX AF,AF'
            PUSH HL
            LD DE,IDL
            ADD HL,DE
            EX DE,HL        ;DE = (IDL(I))
            LD BC,INS
            POP HL
            ADD HL,BC       ;HL = (INS(I))
            PUSH HL
            LD C,(HL)
            INC HL
            LD B,(HL)       ;BC=INS(I)
            EX DE,HL
            LD E,(HL)
            INC HL
            LD D,(HL)       ;DE=IDL(I)
            EX DE,HL
C1:         PUSH HL         ;(SP)=IDL(I) NUEVO
            OR A
            SBC HL,BC       ;IDL = INS ?
            POP DE          ;DE=IDL(I)
            JR Z,C3P1       ;FIN
            PUSH DE
            EXX
            POP HL          ;HL'=IDL(I)
            PUSH IX
            POP DE          ;DE'=(NUM)
            LD B,5
            EX DE,HL
C1P5:       LD A,(DE)
            CP (HL)         ;IDL - NUM
            JR Z,C2         ;IDL = NUM
            JR NC,C3        ;IDL > NUM -- FIN
            ;IDL < NUM
            EXX
            LD HL,6
            ADD HL,DE
            LD A,L
            AND 1FH
            CP 1EH
            JR NZ,C1
            INC HL
            INC HL
            JR C1
            EXX
            JR C4
C3:         EXX
C3P1:       XOR A           ;NO EXISTE
C4:         EX DE,HL        ;HL = IDL(I)
            LD D,B
            LD E,C          ;DE = INS(1)
            POP BC          ;BC = (INS(I))
            PUSH IX
            POP IY          ;IY = (NUM)
            RET
;
;SUBRUTINA DE DESPLAZAMIENTO E INSERCION A LA ENTRADA:
;A LA ENTRADA:
;           DE = DIR. ULTIMA
;           BC = DIR. DE APUNTADOR DE INSERCION
;           HL = DIR. PRIMERA
;           IY = DIR DEL # A INSERTAR
;A LA SALIDA:
;           EL NUMERO HA SIDO INSERTADO Y EL BLOQUE SE HA RECORRIDO
;
;
INSERTA:    PUSH BC
            POP IX          ;IX = (INS(I))
            LD BC,6
            PUSH HL         ;(SP1) = IDL(I)
            LD H,D
            LD L,E
            ADD HL,BC       ;HL = INS(I+1)
            LD A,L
            AND 1FH
            CP 1EH          ;30 O 62 ?
            JR NZ,I0
            INC HL
            INC HL
I0:         PUSH HL
            EXX
            LD HL,FDV
            EX AF,AF'       ;A = MONITOR
            LD B,0
            LD C,A          ;BC'= MONITOR
            ADD HL,BC
            LD E,(HL)
            INC HL
            LD D,(HL)       ;DE'=FDV(I)
            LD HL,AREA-400H ;400H = 16 LINEAS
            ADD HL,DE       ;HL'= FDL(I)
            POP DE          ;DE'= INS(LAST)
            SBC HL,DE       ;CY = 0 DE 'ADD'
            EXX
            JR NZ,I0P5      ;SI NO SON IGUALES, CONTINUA
            LD DE,MERR9
            CALL PRINT
            POP AF          ;RECUPERA (SP)
            RET
I0P5:       LD (IX),L
            LD (IX+1),H     ; (INS(I)) NUEVO
            PUSH DE         ; (SP2) = INS(I)
            EXX
            POP HL          ; HL'=INS(I)
            POP DE          ; DE'=IDL(I)
            DEC HL
            DEC DE
            OR A            ;CY=0
```

```
              JR Z,I4          ;IGUALES, SALTA
              EXX
              LD H,D
              LD L,E
              DEC DE
              DEC DE           ; DE = INS(I-1).4 (?)
              LD A,(DE)
              CP 21H           ; SP,CR,LF ?
              JR NC,I1         ; NO, SALTA.
              DEC DE
              DEC DE           ; SI, AJUSTA.
              ;DE = INS(I-1).4
I1:           LD BC,4
              ADD HL,BC        ;HL = INS(I).4
              EX DE,HL         ;HL = INS(I-1).4 L DE=INS(I).4
I2:           LD BC,5
              LDDR
              PUSH HL
              EXX
              POP HL           ;HL'=INS(I) NUEVO
              OR A             ;CY = 0
              SBC HL,DE        ;INS = IDL ?
              JR Z,I4
              EXX
              DEC HL           ;HL = INS(I-1).4
              LD A,(HL)
              CP 21H           ;SP,CR,LF ?
              JR NC,I3
              DEC HL
              DEC HL           ;AJUSTA
I3:           DEC DE           ;DE = INS(I).4
              LD A,(DE)
              CP 21H
              JR NC,I2
              DEC DE
              DEC DE           ;AJUSTA
              JR I2
I4:           INC DE           ;DE = IDL
              PUSH IY
              POP HL           ;HL = (NUM)
              LD BC,5
              LDIR             ;INSERTA
              RET
;
 ;SUBRUTINA DE BORRADO Y DESPLAZAMIENTO,
 ;A LA ENTRADA:
 ;    DE = DIR, ULTIMA
 ;    BC = DIR, POINTER DE INSERCION
 ;    HL = DIR, PRIMERA
 ;A LA SALIDA:
 ;    DEL BLOQUE HA SIDO ELIMINADO EL NUMERO INDICADO
 ;
QUITA:        LD A,E
              AND 1FH
              JR NZ,Q0         ;32 O 64
              DEC DE
              DEC DE
Q0:           PUSH BC
              LD BC,-6
              EX DE,HL
              ADD HL,BC        ;HL = INS(I) NUEVO
              POP IX           ;IX = (INS(I))
              LD (IX+1),H
              LD (IX),L
```

```
          SBC HL,DE      ;INS = IDL ?
          JR Z,Q3
          LD HL,6
          ADD HL,DE      ;DE = IDL(I)
Q1:       LD A,L
          AND 1FH
          CP 1EH
          JR NZ,Q2
          INC HL
          INC HL
          ;HL  = IDL(I+1)
Q2:       PUSH BC        ;(SP) = INS(I)
          LD BC,6
          LDIR
          PUSH HL.
          LD A,E
          AND 1FH
          CP 1EH
          JR NZ,Q2P5
          INC DE
          INC DE
Q2P5:     POP HL
          EX (SP),HL
          LD B,H
          LD C,L
          OR A
          SBC HL,DE
          POP HL
          JR NZ,Q1
Q3:       LD HL,FRAME
          LD BC,6
          LDIR
          RET
 ;
 ;SUBRUTINA DE ESTRUCTURA ---- T↑0.2 SECS.
 ;01234BB56799CRLF
MEMORIA:  EQU MEMIM      ;LOCALIDAD DE TRABAJO
 ;MEMORIA = 50 LINEAS EDITADAS ;4 AREAS
MIMSET:   JR STRSET
SET1:     LD A,5         ;5 CONTRA-RECIBOS
          PUSH DE
          EXX
          POP DE         ;DE'= LOCALIDAD EN DONDE SE COPIA(MEMORIA)
ST1:      LD BC,6        ;6 ESPACIOS POR CONTRA-RECIBO
          LD HL,FRAME    ;APUNTA AL MARCO.DE.REFERENCIA.
          LDIR           ;COPIA EL MARCO A MEMORIA
          DEC A          ;DECREMENTA EL CONTADOR
          JR NZ,ST1      ;SIGUE EN EL LOOP
          PUSH DE
          EXX
          POP DE         ;DE = DE'
          RET
SET2:     CALL SET1
          LD A,' '
          LD (DE),A
          INC DE
          LD (DE),A
          INC DE
          CALL SET1
          EX DE,HL
          LD (HL),CR
          INC HL
          LD (HL),LF
          INC HL
```

```
                JR Z,DD3
                EX DE,HL        ;DE=(RECORD) --HL=(RECORD+1)
                LDIR            ;***COMPACTA***
                EX DE,HL
DD3:            LD (DSCPTR),HL
                LD A,(NUMREC)
                DEC A
                LD (NUMREC),A
                LD DE,INPUT
                CALL PRINT
                SCF                 ;CY=1 - SI SE ENCONTRO
                RET

;A LA ENTRADA:
;               BC = TAMANO DEL BLOQUE
;               HL = DIR. DEL ULTIMO CARACTER DEL BLOQUE
;               IY =  "    "    "      "        DE LA CADENA
;               C' = TAMANO DE LA CADENA
;A LA SALIDA:
;               -PARIDAD PAR SI SE ENCONTRO
;               HL = DIR. DEL PRIMER MATCH (-1)
;               -PARIDAD IMPAR SI NO SE ENCONTRO
;DESTRUYE:
;               BC,DE,HL,BC' Y A
;
BU0:            EXX
BUSCA:          LD B,C
                EXX
                PUSH IY
                POP DE
                LD A,(DE)
                CPDR
BU1:            RET PO
                JR NZ,BU0
                EXX
                DJNZ BU2
                EXX
                RET
BU2:            EXX
                DEC DE
                LD A,(DE)
                CPD
                JR BU1

PUTMNT:         CALL MON1
                CALL MON2
                CALL MON3
                CALL MON4
                RET
;
;--------------MENSAJE-------------------------------------
;RUTINAS USADAS:
;               ACCEPT
;
MENSAJE:        LD A,(INTCNT)
                OR A            ;HAY INTERRUPCION PENDIENTE?
                JR NZ,MENSAJE ;SI: SIGUE CHECANDO
                OUT MASK,A      ;ENMASCARA INTERRUPCIONES
                CALL ACCEPT
                LD DE,CRLF
                CALL PRINT
                LD A,(INPUT)
                CP CTLT
                JR Z,MS1
```

```
           LDIR           ;TRANSFIERE
           LD DE,LLMA     ;OTRA VEZ
           CALL PRINT
           CALL ACCEPT    ;LEE
           LD DE,LLM2
           CALL PRINT     ;BORRA LINEA
           CALL FORMAT
           CALL VALIDA
           JR C,LL2
           LD BC,5
           LD DE,CLAVE
           LD HL,INPUT
           LDIR
           LD DE,LLMB
           CALL PRINT
           CALL PUTKEY    ;PON EN DISCO
           RET
LL2:       LD DE,LLMC
           CALL PRINT
           JR LL1
LLM1:      DEFB CR,LF,'TECLEE LA LLAVE',CR,LF,'$'  ;PARA SUBRE-ESCRIB
LLM2:      DEFB VT,'                  ',CR,LF,'$'       ;VT = VERTICAL TAB
LLM3:      DEFB CR,LF,'DESEA CAMBIAR LA LLAVE(S/N)',CR,LF,'$'
LLM4:      DEFB CR,LF,'TECLEE SU NUEVA LLAVE',CR,LF,'$'
LLM5:      DEFB CR,LF,'VUELVA A TECLEAR SU NUEVA LLAVE',CR,LF,'$
LLM6:      DEFB CR,LF,'LLAVE ACTUALIZADA',CR,LF,'$'
LLM7:      DEFB CR,LF,'NO TECLEO LO MISMO',CR,LF,'$'
LLM8:      DEFB CR,LF,'DESEA ASIGNAR CLAVE?',CR,LF,'$'
LLM9:      DEFB CR,LF,'TECLEE LA NUEVA CLAVE',CR,LF,'$'
LLMA:      DEFB CR,LF,'VUELVA A TECLEAR LA NUEVA CLAVE',CR,LF,'$'
LLMB:      DEFB CR,LF,'NUEVA CLAVE ASIGNADA',CR,LF,'$'
LLMC:      DEFB CR,LF,'NO TECLEO LA MISMA CLAVE',CR,LF,'$'

CONTADOR:  DEFB 255        ;AL PRINCIPIO SON 4 SEGUNDOS
;
INTERVALO: PUSH AF
           PUSH HL
           LD HL,CONTADOR       ;0 - 140
           DEC (HL)
           LD A,255             ;16.33 MILISEGUNDOS
           OUT (TIMER),A
           JR Z,IMAGEN          ; A LA INTERRUPCION BUENA
           POP HL
           POP AF
           EI
           RET
;
;
;SUBRUTINA DE INTERRUPCION
;ESTA RUTINA RECIBE DE LA SELAL DEL TIMER UN COMANDO PARA
;PROCESAR LA INFORMACION DE SUS 4 AREAS DE MEMORIA
HOME:      EQU 0CH         ;CARACTER DE CONTROL(A CASA Y BORRA)
LIMT0:     EQU MEMIM
LIMT1:     EQU MEMIM+AREA
LIMT2:     EQU MEMIM+AREA*2
LIMT3:     EQU MEMIM+AREA*3
LIMT4:     EQU MEMIM+AREA*4
;** DINAMICOS **
IDL:       DEFW LIMT0
FDL:       DEFW LIMT1
           DEFW LIMT2
           DEFW LIMT3
           DEFW LIMT4
```

```
               DEFW LIMT2
               DEFW LIMT3
IDV:           DEFW LIMT0
               DEFW LIMT1
               DEFW LIMT2
               DEFW LIMT3
;** ESTATICOS **
FDV:           DEFW LIMT0+400H
               DEFW LIMT1±400H
               DEFW LIMT2+400H
               DEFW LIMT3+400H
;
MONITOR:       DEFW 0
;
IMAGEN:        LD (HL),140    ;NUEVO INTERVALO (↑2.3 SECS.)
               PUSH BC        ;2.75
               PUSH DE        ;2.75
               EX AF,AF'      ;1
               EXX
               PUSH AF
               PUSH HL        ;2.75
               PUSH BC        ;2.75
               PUSH DE        ;2.75
               PUSH IX        ;3.75
               PUSH IY        ;3.75
;**ALMACENA REGISTROS         ;26 MICROSEGUNDOS
;
IM1:           LD BC,(MONITOR)
               LD IY,INS
               ADD IY,BC      ;INS(I)
               LD D,(IY+1)
               LD E,(IY)      ;DE = DIR. DE INSERCION
               LD IX,IDV
               ADD IX,BC      ;IDV(I) - CY =0
               LD H,(IX+1)
               LD L,(IX)      ;HL = DIR. DE VENTANA
               PUSH HL
               SBC HL,DE      ;IDV - INS
               POP HL
               JR C,IM3       ;OK, SIGUE
               ; CALCULA VUELTA
               LD HL,FDV
               ADD HL,BC      ;FDV(I)
               LD E,(HL)
               INC HL
               LD D,(HL)
               LD HL,-400H
               ADD HL,DE      ;HL = IDV(I)
;
IM3:           CALL ESCLINEA  ;ESCRIBE UNA LINEA
               LD A,(MONITOR)
               CP 6
               JR NZ,IM4
               LD A,-2
IM4:           ADD 2          ;NUEVO MONITOR
               LD (MONITOR),A
;
IM2:           POP IY
               POP IX
               POP DE
               POP BC
               POP HL
               POP AF
```

```
            [illegible]
            POP AF
            EI
            RET
;
ESCLINEA:   LD B,64
            LD C,(HL)       ;BYTE A ESCRIBIR
            PUSH HL         ;SP/II
            LD HL,(MONITOR)
            SRL L           ;*1
            PUSH HL
            ADD HL,HL       ;*2
            ADD HL,HL       ;*4
            POP DE
            ADD HL,DE       ;*5
            LD DE,CASE
            ADD HL,DE
            EX (SP),HL      ;CASE MONITOR
IM4P1:      EX (SP),HL
            JP (HL)
            ;SP/III
CASE:       CALL MON1T      ;CY = 0
            JR IM5
            CALL MON2T      ;CY = 0
            JR IM5
            CALL MON3T      ;CY = 0
            JR IM5
            CALL MON4T      ;CY = 0
IM5:        EX (SP),HL
            INC HL
            LD C,(HL)
            DJNZ IM4P1
            POP DE          ;RECUPERA (SP)
            CALL ANILLO
            LD (IX),L       ;ACTUALIZA IDV(I)
            LD (IX+1),H
            RET
;
;**RUTINA DE CIRCULO EN MEMORIA**
;
ANILLO:     PUSH HL
            PUSH HL         ;(SP1)=(SP2)=APUNTADOR A LA LINEA
            LD HL,(MONITOR)
            LD DE,FDV
            ADD HL,DE       ;CY =0
            LD E,(HL)
            INC HL
            LD D,(HL)
            LD HL,AREA-400H     ;OFFSET
            ADD HL,DE
            LD B,H
            LD C,L          ;BC = FDL(I)
            LD HL,-AREA
            ADD HL,BC       ;HL = IDL(I) - CY = 1
            EX DE,HL        ;DE = IDL(I)
            POP HL
            OR A
            SBC HL,BC       ;IDV = FDL ?
            POP HL          ;RECUPERA APUNTADOR ORIGINAL A LINEA
            RET NZ
            EX DE,HL
            RET
;SUBRUTINA DE ALMACEN DE LLAVE
```

```
            RET
;*****
;
SAR1:       EQU 74H        ;SOFTWARE ADDRESS REVERSE
SAR2:       EQU 94H
CMND:       EQU 62H
MASK:       EQU 63H
CTL1:       EQU 60H
CTL2:       EQU 70H
CTL3:       EQU 80H
CTL4:       EQU 90H
PORT1:      EQU 61H
PORT2:      EQU 71H
PORT3:      EQU 81H
PORT4:      EQU 91H
TXRDY:      EQU 80H
TIMER:      EQU 65H
;
MON1:       IN A,CTL1
            AND TXRDY
            JR Z,MON1
            LD A,C
            OUT PORT1,A
            RET
;
MON2:       IN A,CTL2
            AND TXRDY
            JR Z,MON2
            LD A,C
            OUT PORT2,A
            RET
;
MON3:       IN A,CTL3
            AND TXRDY
            JR Z,MON3
            LD A,C
            OUT PORT3,A
            RET
;
MON4:       IN A,(CTL4)
            AND TXRDY
            JR Z,MON4
            LD A,C
            OUT (PORT4),A
            RET
PERIFERICOS: LD A,0
            OUT SAR1,A
            OUT SAR2,A
            LD A,9         ;RESET L INTA ENABLE
            LD C,CMND
            CALL PUERTOS   ;RESTART A LOS 4 PUERTOS
            XOR A
            LD C,MASK
            CALL PUERTOS   ;SIN INTERRUPCION EN LOS 4 PUERTOS
            LD A,84H       ;300 BAUDS, 1 STOP
            LD C,CTL1      ;BAUD PORT ON OUTPUT
            CALL PUERTOS
            JR PRFCS
PUERTOS:    LD B,4         ;4 PUERTOS
            EX AF,AF'
            LD A,C         ;A'=# DE PUERTO INICIAL
PR1:        EX AF,AF'      ;A = PALABRA DE CONTROL
            OUT (C),A
```

```
                RET
;
PRFCS:          LD C,HOME
                CALL MON1
                CALL MON2
                CALL MON3
                CALL MON4
                LD A,1
                OUT MASK,A      ;HABILITA TIMER1
                NEG             ;A=255
                OUT TIMER,A     ;PRIMER PERIODO
                LD A,12H
                LD I,A          ;I=12H (12XXH)
                IM2             ;MODO Z80
                EI              ;HABILITA INTERRUPCIONES
                RET
;
TSTK:           DEFS 64
STACK:          ;AREA DEL STACK PRINCIPAL
TSTK2:          DEFS 20
STAK2:          ;AREA DE SEGUNDO STACK
I1STACK:        DEFS 2
I2STACK:        DEFW STAK2
INTCNT:         DEFB 0
;
IMONITOR:       LD A,(INTCNT)
                OR A
                JR NZ,MNT
                IN A,63H        ;RESET TBE
                XOR A
                OUT 63H,A       ;DESHABILITA TIMER 1
                LD A,20H
MONCODE:        OUT 63H,A       ;ENABLE TBE, DISABLE TIMER
                LD A,-64
                LD (INTCNT),A
MNT:            LD A,C
                OUT PORT1,A     ;ESCRIBE BYTE
                JR SAVE         ;GUARDA MEDIO AMBIENTE
MON1T:          LD A,63H
                LD (MONCODE+1),A
                LD A,PORT1
                LD (MNT+2),A
                JR IMONITOR
;
MON2T:          LD A,73H
                LD (MONCODE+1),A
                LD A,PORT2
                LD (MNT+2),A
                JR IMONITOR
;
MON3T:          LD A,83H
                LD (MONCODE+1),A
                LD A,PORT3
                LD (MNT+2),A
                JR IMONITOR
;
MON4T:          LD A,93H
                LD (MONCODE+1),A
                LD A,PORT4
                LD (MNT+2),A
                JR IMONITOR
;
```

```
                LD H,10
                LD (CONTADOR),A ;DA UN MARGEN DE 163 MS.
                LD A,1
                OUT 63H,A     ;ENABLE INTERRUPT TIMER
                EXX
                RET
;
;
LAST:           ;**FIN**
FILLER:         DEFS 1260H-103H-LAST
;
;-----------------------
;               ORG 1260H
VECTOR:         DEFW INTERVALO
;
;-----------------------
;
FILL1:          DEFS 8
;ORG 126AH
INT1:           DEFW IOINT
FILL2:          DEFS 14
;ORG 127AH
INT2:           DEFW IOINT
FILL3:          DEFS 14
;ORG 128AH
INT3:           DEFW IOINT
FILL4:          DEFS 14
;ORG 129AH
INT4:           DEFW IOINT
;
FILL5:          DEFS 4
;ORG 12A0H
;----MENSAJES DE ERROR----
;
MERR0:          DEFB CR,LF,'NO $'
MERR1:          DEFB 'EXISTE EL NUMERO DE CONTRA-RECIBO',CR,LF,'$'
CDERR:          DEFB CR,LF,'NO HAY DATOS EN MEMORIA',CR,LF
                DEFB 'COMANDO NO EJECUTADO',CR,LF,'$'
MERR2:          DEFB CR,LF,'NO HAY IMAGEN EN DISCO',CR,LF,'$'
MERR3:          DEFB CR,LF,'DATO MAL TECLEADO',CR,LF,'$'
AERR:           DEFB CR,LF,'SE EXCEDIO EL PERIODO MAXIMO DE PAGO',CR,LF,'$'
MERR5:          DEFB CR,LF,'SE HA EXCEDIDO LA CAPACIDAD MAXIMA'
                DEFB 'DE ALMACENAMIENTO',CR,LF,'$'
DESERR:         DEFB CR,LF,'NO HAY DATOS DE ALTA EN DISCO',CR,LF,'$'
MERR4:          DEFB CR,LF,'FIN DE ARCHIVO DE ALTAS',CR,LF,'$'
MERR7:          DEFB CR,LF,'COMANDO NO RECONOCIDO',CR,LF,'$'
MERR8:          DEFB CR,LF,'CUAL ES SU COMANDO ?',CR,LF,'$'
MERR9:          DEFB CR,LF,'NUMERO MAXIMO DE CONTRA-RECIBOS EXCEDIDO',CR,LF
                DEFB '***NUMERO NO INSERTADO***',CR,LF,'$'
ENCA:           DEFB 'SISTEMA DE NOTIFICACION DE ORDENES DE PAGO'
                DEFB CR,LF,'ADIEL S.C.      1979',CR,LF,LF,LF,'$'
CRLF:           DEFB CR,LF,'$'
 END ORDENES
```

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

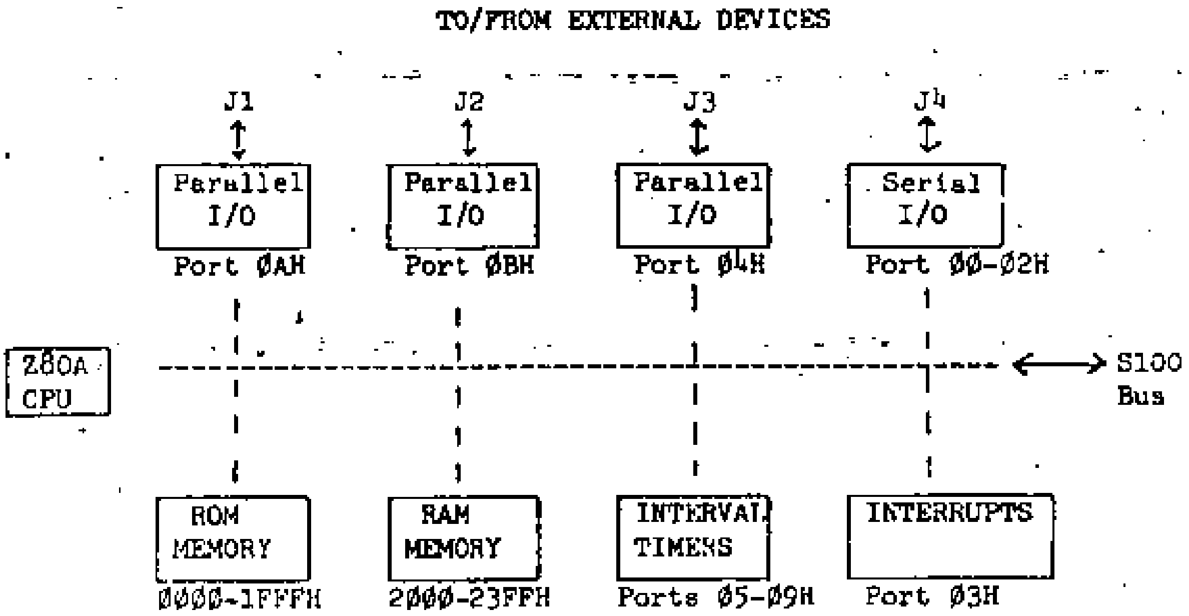
MICROPROCESADORES: TEORIA Y APLICACIONES

SINGLE CARD COMPUTER

Julio, 1981

INTRODUCTION TO THE CROMEMCO SINGLE CARD COMPUTER

The Cromemco Single Card Computer (SCC) is a sophisticated microcomputer with the following layout:



The S-100 bus interface allows convenient expansion of the system if desired. The SCC is guaranteed to work with all of Cromemco's S-100 products, including analog I/O, color graphics; and RAM/ROM memory.

Program execution begins as soon as power is applied (after an automatic power-on-reset).

The SCC is self-contained except for the user-supplied power supplies. These power supplies may be unregulated and should supply +8 volts at 1.2A, +18 volts at 100MA, and -18 volts at 50MA.

The following sections cover programming information, including the Z-80 CPU technical manual and information about the I/O registers on the SCC; a functional description, which explains the operation of each component in the SCC; and the S-100 interface, which describes the bus interface characteristics of the SCC.

SCC INPUT/OUTPUT REGISTER DESCRIPTION

In the following sections, each I/O port (register) will be discussed.

PORT ADDRESS	IN/OUT	DESCRIPTION							
Ø	IN	Status Register:							
		D7	D6	D5	D4	D3	D2	D1	DØ
		Transmit Buffer Empty	Read Data Avail.	Int. Pending	Start Bit Detect	Full Bit Detect	Serial RcV	Over-Run Err	Frame Error

The functions of these flags are indicated in the following sections.

D7 Transmitter Buffer Empty (TBE):

A high in bit 7 indicates that the transmitter data buffer is ready to accept a new byte. TBE goes high as soon as the serial transmitter begins to send the byte currently in the buffer. Since the transmitter is "double-buffered", the user may respond to the TBE signal and load the buffer even before the previous byte has been totally transmitted. TBE also activates interrupt request 5. TBE is cleared when the buffer is loaded and is set by the RESET command.

D6 Receiver Data Available (RDA):

A high in bit 6 indicates that a byte of data is available from the receiver buffer. This flag remains high until the buffer is read. A

RESET command clears the flag. If the buffer is not read by the time the next byte from the receiver is ready, the new byte will write over the old byte and the overrun error flag will be set. RDA also activates interrupt request 4.

D5 Interrupt Pending (IPG):

A high in bit 5 indicates that one or more of the eight interrupt request sources has become active. This flag goes high at the same time as the interrupt request pin of the TMS 5501.

D4 Start Bit Detect (SBD):

A high in bit 4 indicates that the serial receiver has detected a start bit. This bit remains high until the full character has been received. SBD is cleared by RESET command. This bit is provided for test purposes.

D3 Full Bit Detect (FBD):

The FBD flag in bit 3 goes high one full bit time after the start bit has been detected. This bit remains high until the full character has been received. FBD is cleared by a RESET command. This bit is provided for test purposes.

D2 Serial Receive (SRV):

A high in bit 2 indicates high level on the serial data input line. A low in bit 2 indicates a low level on the serial data input line. SRV is high when no data is being received. This bit is provided for break detection and for test purposes.

D1 Overrun Error (ORE):

A high in bit 1 indicates that the receiver has loaded the receiver data buffer before the previous contents were read. ORE is cleared after the status port is read or by the RESET command.

DØ Frame Error (FME):

A high in bit Ø indicates an error in one or both of the stop bits which "framed" the last received data byte. FME remains high until a valid character is received.

Ø OUT Baud Rate Register. Loading this register sets the baud rate and stop bits for serial receive and transmit data. Bit assignment is as follows:

D7	D6	D5	D4	D3	D2	D1	DØ
STOP BITS	96ØØ	48ØØ	24ØØ	12ØØ	3ØØ	15Ø	11Ø

D7 STOP

A high in bit 7 selects one stop bit for serial receive and transmit data. A low in bit 7 selects two stop bits.

D6-DØ BAUD RATE

A high in one of the lower seven bits selects the corresponding baud rate. If more than one bit is high, the highest rate selected will result. If none of the bits are high, the serial transmitter and receiver will be disabled. (For special purposes, these baud rates can be octupled -- see the description of HBD in the command register.)

1 IN Receiver Data. This register contains an assembled byte of data from the serial register.

1 OUT <u>Transmitter Data</u>. This register is loaded with data for the serial transmitter.

2 IN <u>Not Assigned</u>. Reading this port causes no response from the SCC. This address is available for other parts of the computer system.

2 OUT <u>Command Register</u>. The format for the command register is as follows:

--------latched----------

D7	D6	D5	D4	D3	D2	D1	DØ
Not Used	Not Used	Test	HIGH BAUD	INTA Enable	RST7 Sel.	Break	Reset

<u>D5 Test Bit (TB5)</u>:

A high in bit 5 disables the internal interrupt priority logic and then enables the internal clock. Thus, the signal on the INT pin of the 5501 becomes a TTL level clock of 1562.5 Hz (12.5 kHz if HBD is high -- see D4 High Baud below). TB5 should be low for normal operation.

<u>D4 High Baud (HBD)</u>:

A high in bit 4 octuples the rate of the internal clock. This causes the interval timers to count eight times faster and the serial data rates to increase eight-fold. When bit 4 is high, baud rates up to 76.8K are available for high speed data transfers.

<u>D3 INTA Enable (INE)</u>:

A high in bit 3 allows the 5501 to respond to an Interrupt Acknowledge by gating a Restart instruction into the data bus

TIMERS

The SCC provides five internal timers, which are accessed through the five timer command ports and a timer status port which is common to the UART status port. Each timer accepts a byte which represents a delay count (Ø-255) to be counted down by the timer clock. When the count reaches zero, the "Interrupt Pending" status bit is set. Alternatively, the timer may be used to generate an actual interrupt to the CPU if the interrupt system is being used. Maximum delay is 16.32 milliseconds.

INTERRUPTS

The SCC provides two sets of interrupt structures: a maskable set and a single non-maskable interrupt ($\overline{\text{NMI}}$). The NMI signal will always override program execution. This signal is available to the user on S-100 pin 12 as $\overline{\text{NMI}}$.

The maskable interrupt set can interrupt program execution only if the CPU has executed the enable interrupt instruction. Then, program execution may be interrupted under one of eight conditions:

1) Timer 1 timeout
2) Timer 2 timeout
3) $\overline{\text{INT}}$ (external request)
4) Timer 3 timeout
5) UART receiver data available
6) UART transmitter buffer empty
7) Timer 4 timeout
8) Timer 5 timeout

These conditions are listed in the order of priority with the most important first. If two conditions become true together, the condition with the higher priority gets CPU service first. Each of the eight conditions is enabled by a specific bit in the interrupt mask register on the SCC

I/O PORT SUMMARY

Table 3.2 summarizes the allocation of SCC I/O ports.

TABLE 3.2: ALLOCATION OF I/O PORTS

PORT ADDRESS	INPUT FUNCTION	OUTPUT FUNCTION
ØØ	UART Status	Baud Rate
Ø1	UART Rcvr, Data	UART TX, Data
Ø2	Not Used	UART Command
Ø3	Interrupt Address	Interrupt Mask
Ø4	Parallel In (J3)	Parallel Out (J3)
Ø5	Not Used	Timer 1
Ø6	Not Used	Timer 2
Ø7	Not Used	Timer 3
Ø8	Not Used	Timer 4
Ø9	Not Used	Timer 5
ØAH	Parallel In (J1)	Parallel Out (J1)
ØBH	Parallel In (J2)	Parallel Out (J2)

15

THE S-100 BUS INTERFACE

This section deals with the implementation of S-100 signals by the SCC. A series of charts shows the behavior of the bus during each possible type of CPU cycle. These charts are patterned after those in the Z80A Technical Manual (included with this manual) for easy reference. A summary of bus pinout is found in Table 4.1.

One S-100 signal does not occur normally on the Z80, namely PSYNC. This signal, which signals the beginning of a new machine cycle is synthesized from the $\overline{MREQ}$ and $\overline{IORQ}$ signals in the following way: PSYNC is triggered by the falling edge of either signal and stays high until the next rising edge of the clock (Ø2).

Waveforms for an instruction fetch are shown in Figures 1 and 2. Waveforms for other memory cycles are shown in Figures 3 and 4; I/O cycles are shown in Figures 5 and 6. Figure 7 shows a hold request and hold acknowledge sequence. Figure 8 shows an interrupt request followed by an interrupt acknowledge cycle. Figure 9 shows the response to $\overline{NMI}$ and Figure 10 shows an exit from halt.

Cromemco SCC™ System Bus Structure

COMPONENT SIDE OF BOARD

1	+8V
2	+16V
3	XRDY
4	
5	
6	
7	
8	
9	
10	
11	--
12	$\overline{NMI}$
13	
14	
15	
16	
17	
18	$\overline{STAT\ DISABLE}$
19	$\overline{C/C\ DISABLE}$
20	
21	
22	$\overline{ADDR\ DSBL}$
23	$\overline{DO\ DSBL}$
24	Φ2
25	
26	PHLDA
27	
28	
29	A5
30	A4
31	A3
32	A15
33	A12
34	A9
35	DO1
36	DO0
37	A10
38	DO4
39	DO5
40	DO6
41	DI2
42	DI3
43	DI7
44	SM1
45	SOUT
46	SINP
47	SMEMR
48	SHLTA
49	CLOCK (2 MHz)
50	GND

SOLDER SIDE OF BOARD

51	+8V
52	-16V
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	$\overline{MREQ}$
66	$\overline{RFSH}$
67	
68	MWRITE
69	
70	
71	
72	PRDY
73	$\overline{PINT}$
74	$\overline{PHOLD}$
75	$\overline{PRESET}$
76	PSYNC
77	$\overline{PWR}$
78	PDBIN
79	A0
80	A1
81	A2
82	A6
83	A7
84	A8
85	A13
86	A14
87	A11
88	DO2
89	DO3
90	DO7
91	DI4
92	DI5
93	DI6
94	DI1
95	DI0
96	SINTA
97	$\overline{SWO}$
98	4MHz
99	$\overline{POC}$
100	GND

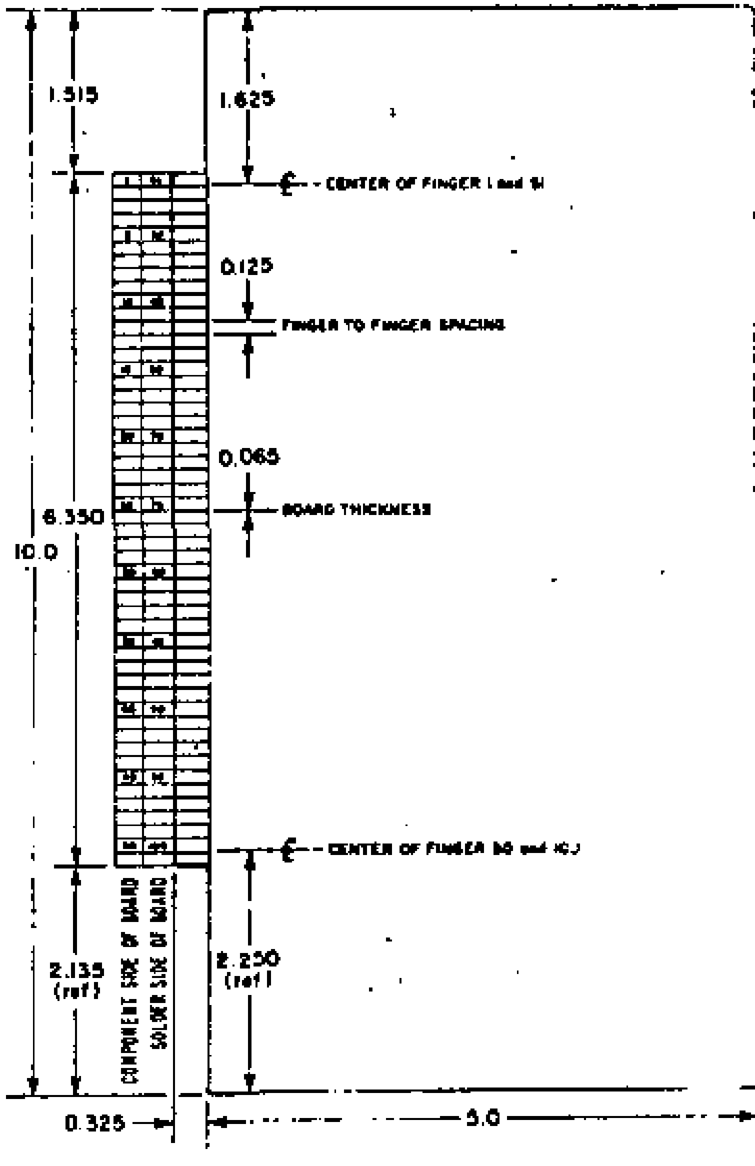


TABLE 4.1:

SCC-040-002

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

TU-ART DIGITAL INTERFACE

Julio, 1981

Introduction

The Cromemco TU-ART (Twin Universal Asynchronous Receiver and Transmitter) provides two channels of duplex serial data exchange; two channels of parallel data exchange; and ten interval timers. Status information is available through polling or by interrupt. In addition, each interval timer activates an interrupt and two interrupt request lines are brought out for the user. The TU-ART has its own crystal-controlled clock and interfaces to the S-100 bus asynchronously so that CPU clock frequency is not critical. The TU-ART incorporates two TMS 5501 NMOS I/O Controller chips.

1.1 Definitions

Throughout this manual the two TMS 5501 chips will be referred to as "Device A" and "Device B." Device A (IC 4) is the leftmost chip. Device B (IC 5) is the rightmost chip. Device A is nearer the heat sink and drives serial connector J4 and parallel connector J2. Device B is located to the right of Device A and drives serial connector J5 and parallel connector J3.

1.2 Switch Selectable Options Addressing The TU-ART

The system CPU views the TU-ART as a dual assembly of input/output ports with interrupt capability. The CPU normally reads data or status from the TU-ART via the S-1ØØ bus by executing an IN A, (port) instruction, and writes data or commands to the TU-ART by executing an OUT (port), A instruction.

There are fourteen I/O ports used for data transfers, commands and status by Device A, and another fourteen by Device B (see Figure 2). The user may independently switch select Device A and Device B I/O Base Addresses (the four most significant I/O address bits); the four least significant bits of the I/O address on the S-1ØØ bus then determine the offset from the selected base address.

The base address of Device A is selected by DIP switch positions 6 thru 3; the base address of Device B is selected by DIP switch positions 1Ø thru 7 (see Figure 1). Notice that positioning a switch ON conditions the TU-ART to respond to a logic Ø on its associated address line; an OFF switch corresponds to logic 1.

For example, if DIP switch positions 6 thru 3 are ON, and positions 1Ø thru 7 are OFF, then the TU-ART Device A Command Register is mapped into output port Ø2H, and the Device B Command Register is mapped into output port ØF2H.

Note that Device A bits A7, A6 and A5 also control D7, D6 and D5 of the TU-ART's Z-8Ø mode 2 Interrupt Acknowledge response vector.

Interrupt Mode

When this switch (position 1) is ON, the TU-ART operates in the 8Ø8Ø interrupt mode: one of eight "Restart" instructions is gated to the data bus during an Interrupt Acknowledge cycle. Since the TU-ART can interrupt from one of 16 different sources, it is necessary to poll the devices if the TU-ART is in 8Ø8Ø mode (see "Operation Using 8Ø8Ø Mode Interrupts").

When switch position 1 is OFF, the TU-ART responds in Z-8Ø mode 2. In this mode, the TU-ART supplies a byte to the data bus during Interrupt Acknowledge that is used as the lower eight bits of a memory address. The Z-8Ø supplies the upper eight bits from the I register and automatically reads the corresponding memory location, as well as the next location, to find the starting location of an interrupt routine. (Refer to Section 3.1 and/or the Z-8Ø CPU Reference Manual, Zilog, 1977, for details.)

Normal/Reverse Address

When this switch (position 2) is ON, it allows Device A and Device B to swap base addresses by means of an output to one of the parallel ports (Software

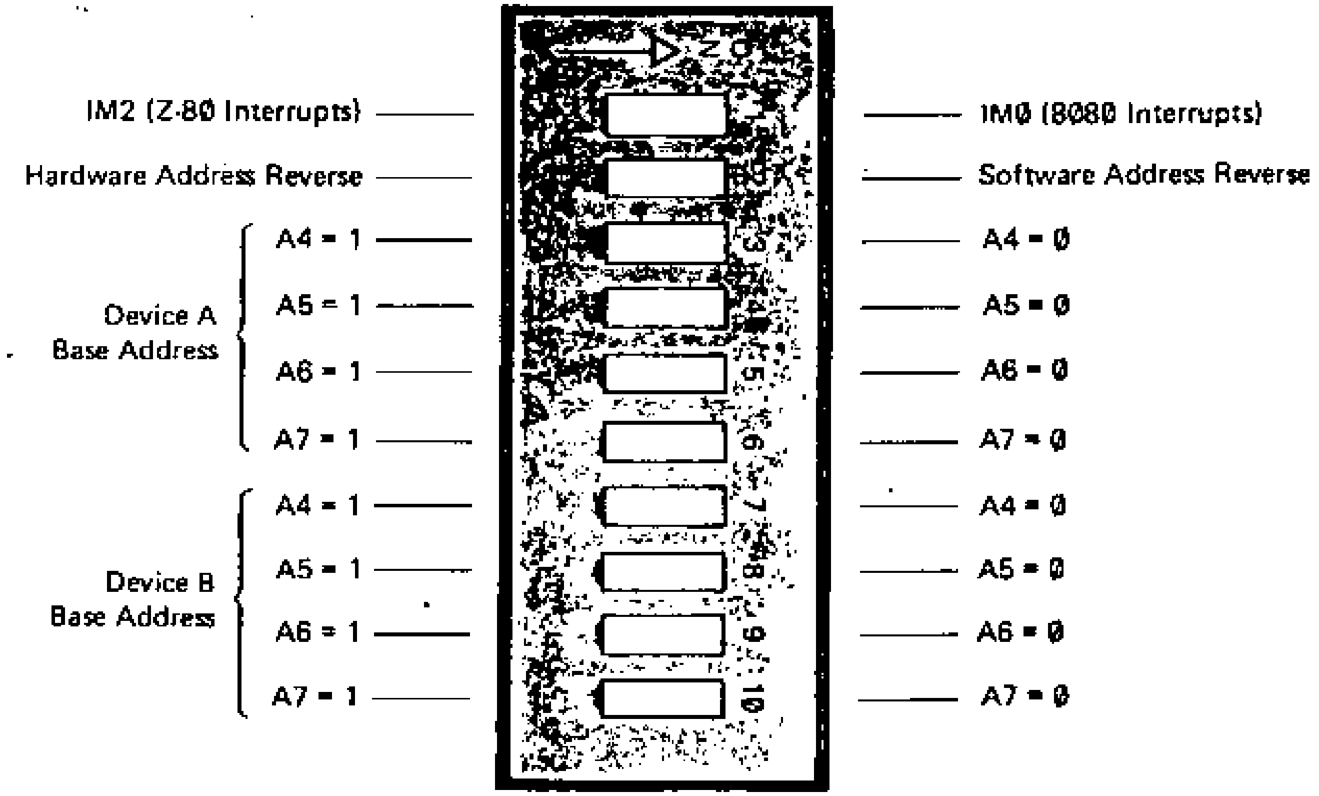
Address-Reverse). This allows either Device A or Device B to be driven by a software driver whose port assignments are frozen in memory. Setting the switch ON connects the MSB of Device A's parallel output port to the Reverse Address control so that addresses may be flipped under software control. To flip addresses, output a byte with D7 high to Device A's parallel output port. To return to normal addressing, output a byte with D7 low to Device B's parallel output port. When switch position 2 is OFF, the Address Reverse switch is disconnected from the parallel port.

The Address Reverse signal is brought out to pin 1 of J2 and J3. When the Address Reverse switch is pin 1 will show the state of the TU-ART:

Pin 1 = 0 means Reverse Mode,
Pin 1 = 1 means Normal Mode.

When the Address Reverse switch is OFF, pin 1 of J2 or J3 may be grounded externally to place the TU-ART in reverse Mode (Hardware Address-Reverse). Do not ground pin 1 of J2 or J3 while the Reverse Address switch is ON as this will conflict with operation of Device A's parallel port.

Figure 1 TU-ART Switch Settings



Caution: Base addresses 00H and 30H are used by Cromemco's 4FDC Floppy Disc Controller; 40H is used by Cromemco memory boards with BANK SELECT; and 50H is used by Cromemco's PRI Printer Interface.

1.3 Interrupt Priority Chain

When more than one TU-ART is used in a system, it is necessary to coordinate the Interrupt Responses in order to prevent bus conflict during Interrupt Acknowledge cycles. This is done by first connecting J1 PRIORITY $\overline{\text{OUT}}$ from the highest priority TU-ART to J1 PRIORITY $\overline{\text{IN}}$ of the next highest priority TU-ART, then connecting J1 PRIORITY $\overline{\text{OUT}}$ of the second TU-ART to J1 PRIORITY $\overline{\text{IN}}$ of the next TU-ART, and so on until all TU-ARTs are connected. The J1 PRIORITY $\overline{\text{IN}}$ pin of the highest priority board is left unconnected. Device A is internally prioritized over Device B on each TU-ART.

1.4 Status Bit Selection

The connection of status flag bits to data bits is done on the PC board at the location of the status socket below J4. Cromemco software conventions assign D6=Receiver Data Available (RDA), and D7=Transmitter Buffer Empty (TBE). For specialized assignments (like more than one bit per flag) see the following "Status Socket" section.

Status Socket

The status flag bits available on input port Ø are connected to the data bits by foil traces in the "status" socket located between IC's 8 and 9.

The flag assignment used by all Cromemco software is discussed in the section entitled "Register Description."

If necessary, the flags may be assigned to different data bits. This may be most easily done as follows:

1. Notice that the flags are arranged along the left row of pads and that the data bits are arranged along the right side row of pads. Note also that only those 8 traces connecting the right and left pads are not covered by the solder mask. There are 5 traces which pass through this area which are covered.

2. Use a razor blade or a sharp knife to cut all 8 of the traces connecting the left and right rows of pads. Be very careful not to cut the traces which are covered by the solder mask.

3. Install and solder a 16 pin IC socket in the 2 rows of pads.

4. Install a 16 pin "component header" in the socket.

5. Using small (24 or 28 Awg) insulated wire connect the flags (on the left) to the desired data bits (on the right) on the component header.

6. The component header is now a "plug" for your particular flag assignment. Several different flag assignment "plugs" can be prepared in the same manner and used at different times to suit the requirements of the software being executed.

Any given flag may be assigned to more than one data bit. However, each data bit can have only one flag assigned to it.

1.5 Interface Options

TTY 20 mA

To drive a Teletype, the following connections should be made (at J4 or J5 for Device A or B respectively):

TU-ART	ASR-33 TTY
J4/J5 PIN 23 connects to	Terminal strip "BL", terminal #7 (current into printer)
J4/J5 PIN 25 connects to	Terminal strip "BL", terminal #6 (return current from printer)
J4/J5 PIN 17 connects to	Terminal strip "BL", terminal #4 (current into keyboard)
J4/J5 PIN 24 connects to	Terminal strip "BL", terminal #3 (return current from keyboard)

Caution: 120 VAC is also present on terminal strip "BL" at terminals #1 and #2.

RS/232C

An RS232 terminal (such as a CRT) may be plugged into an interface cable directly out of J4 or J5. The TU-ART assumes the role of data-set (computer) in this case. See Figure 8: Terminal to TU-ART Cable for this connection.

Parallel I/O

The parallel port output drivers may be tri-stated by grounding pin 8 of the parallel port (J2, J3). A bidirectional bus may be implemented by simply wiring the input and output lines together and using pin 8 to control the direction of data flow. Pin 8 low implies data input to the TU-ART and pin 8 high implies data output from the TU-ART.

Figure 2 Summary Of TU-ART I/O Port Addresses

OFFSET	A7 A6 A5 A4	A3 A2 A1 AØ	FUNCTION
Ø	Device A Base Address (see Fig. 1)	Ø Ø Ø Ø	IN Device A status register
Ø		Ø Ø Ø Ø	OUT Device A baud rate register
1		Ø Ø Ø 1	IN Device A receiver data register
1		Ø Ø Ø 1	OUT Device A transmitter data register
2		Ø Ø 1 Ø	OUT Device A command register
3		Ø Ø 1 1	IN Device A interrupt address register
3		Ø Ø 1 1	OUT Device A interrupt mask register
4		Ø 1 Ø Ø	IN Device A parallel port
4		Ø 1 Ø Ø	OUT Device A parallel port
5		Ø 1 Ø 1	OUT Device A timer 1
6		Ø 1 1 Ø	OUT Device A timer 2
7		Ø 1 1 1	OUT Device A timer 3
8		1 Ø Ø Ø	OUT Device A timer 4
9		1 Ø Ø 1	OUT Device A timer 5
Ø	Device B Base Address (see Fig. 1)	Ø Ø Ø Ø	IN Device B status register
Ø		Ø Ø Ø Ø	OUT Device B baud rate register
1		Ø Ø Ø 1	IN Device B receiver data register
1		Ø Ø Ø 1	OUT Device B transmitter data register
2		Ø Ø 1 Ø	OUT Device B command register
3		Ø Ø 1 1	IN Device B interrupt address register
3		Ø Ø 1 1	OUT Device B interrupt mask register
4		Ø 1 Ø Ø	IN Device B parallel port
4		Ø 1 Ø Ø	OUT Device B parallel port
5		Ø 1 Ø 1	OUT Device B timer 1
6		Ø 1 1 Ø	OUT Device B timer 2
7		Ø 1 1 1	OUT Device B timer 3
8		1 Ø Ø Ø	OUT Device B timer 4
9		1 Ø Ø 1	OUT Device B timer 5

NOTES:

All of the following unassigned ports are free for system use: IN 2, IN 5 through IN 9, IN 1Ø through IN 15 and OUT 1Ø through OUT 15.

If Device A and Device B are set to the same base address, Device A will override.

Device A is IC 4.

Device B is IC 5.

TU-ART Register Descriptions

2.1 Offset IN/OUT Description

Each of the twenty-eight TU-ART registers is viewed as an I/O port by the system CPU. The function of each register is discussed in the following sub-sections. The sub-section headings consist of an I/O port address offset, followed by either "IN" or "OUT," followed by the TU-ART register name. The descriptions given below apply equally to Device A registers and Device B registers. Refer to Figure 3 for a summary of TU-ART register formats.

ØØ IN Status Register

The CPU reads the contents of this register to determine the status of the Device A/Device B serial port. The status bit assignments may be altered by cutting PC foil traces and installing a jumper wire header (see Section 1.4).

D7	D6	D5	D4	D3	D2	D1	DØ
Transmit Buffer Empty	Read Data Avail.	Int Pending	Start Bit Detect	Full Bit Detect	Serial Rcv	Overrun Error	Frame Error

D7 Transmitter Buffer Empty (TBE)

A high in bit 7 indicates that the transmitter data buffer is ready to accept a new byte. TBE goes high as soon as the serial transmitter begins to send the byte currently in the buffer. Since the transmitter is "double-buffered," the user may respond to the TBE signal and load the buffer even before the previous byte has been totally transmitted. TBE also activates interrupt request 5. TBE is cleared when the buffer is loaded and is set by the RESET command.

D6 Receiver Data Available (RDA)

A high in bit 6 indicates that a byte of data is available from the receiver buffer. This flag remains high until the buffer is read. A RESET command clears the flag. If the buffer is not read by the time the next byte from the receiver is ready, the new byte will write over the old byte and the overrun error flag will be set. RDA also activates interrupt request 4.

D5 Interrupt Pending (IPG)

A high in bit 5 indicates that one or more of the eight interrupt request sources has become active. This flag goes high at the same time as the interrupt request pin of the TMS 55Ø1.

D4 Start Bit Detect (SBD)

A high in bit 4 indicates that the serial receiver has detected a start bit. This bit remains high until the full character has been received. SBD is cleared by RESET command. This bit is provided for test purposes.

D3 Full Bit Detect (FBD)

The FBD flag in bit 3 goes high one full bit time after the start bit has been detected. This bit remains high until the full character has been received. FBD is cleared by a RESET command. This bit is provided for test purposes.

D2 Serial Receive (SRV)

A high in bit 2 indicates high level on the serial data input line. A low in bit 2 indicates a low level on the serial data input line. SRV is high when no data is being received. This bit is provided for break detection and for test purposes.

D1 Overrun Error (ORE)

A high in bit 1 indicates that the receiver has loaded the receiver data buffer before the previous contents were read. ORE is cleared after the status port is read or by the RESET command.

DØ Frame Error (FME)

A high in bit Ø indicates an error in one or both of the stop bits which "framed" the last received data byte. FME remains high until a valid character is received.

ØØ OUT Baud Rate Register

The CPU loads this register to set the baud rate and stop bits for serial receive and transmit data. Bit assignments are as follows:

D7	D6	D5	D4	D3	D2	D1	DØ
STOP BITS	96ØØ	48ØØ	24ØØ	12ØØ	3ØØ	15Ø	11Ø

D7 Stop

A high in bit 7 selects one stop bit for serial receive and transmit data. A low in bit 7 selects two stop bits.

D6-DØ Baud Rate

A high in one of the lower seven bits selects the corresponding baud rate. If more than one bit is high, the highest rate selected will result. If none of the bits are high, the serial transmitter and receiver will be disabled. (For special purposes these baud rates can be octupled—see the description of HBD in the command register.)

Ø1 IN Receiver Data

The CPU reads this register to obtain the assembled byte of data from the serial receiver.

Ø1 OUT Transmitter Data

The CPU loads this register with a data byte for serial transmission.

Ø2 IN Not Connected

Reading this port causes no response from the TU-ART. This input port is available to other parts of the computer system.

Ø2 OUT Command Register

The format for the command register is as follows:

		latched					
D7	D6	D5	D4	D3	D2	D1	DØ
Not Used	Not Used	Test	HIGH BAUD	INTA Enable	RST7 Sel.	Break	Reset

D5 Test Bit (TB5)

A high in bit 5 disables the internal interrupt priority logic and then enables the internal clock. Thus, the signal on the INT pin of the 55Ø1 becomes a TTL level clock of 1562.5 Hz (12.5 kHz if HBD is high—see "D4 High Baud" below). TB5 should be low for normal operation.

D4 High Baud (HBD)

A high in bit 4 octuples the rate of the internal clock. This causes the interval timers to count eight times faster and the serial data rates to increase eightfold. When bit 4 is high, baud rates up to 76.8K baud are available for high speed data transfers.

D3 INTA Enable (INE)

A high in bit 3 allows the 55Ø1 to respond to an Interrupt Acknowledge by gating a Restart instruction into the data bus at the correct time and resetting its internal interrupt request latch.

A low in bit 3 prevents the 55Ø1 from detecting an INTA cycle. Bit 3 should be high for normal operation.

D2 RST7 Select (RS7)

A high in bit 2 connects the MSB of the parallel input port to the interrupt request latch for the lowest priority interrupt (interrupt 7). A low-to-high transition on the MSB of the parallel input port (PI7) will activate the interrupt request latch.

A low in bit 2 connects the output of Timer 5 to the interrupt request latch for the lowest priority interrupt (interrupt 7). When the timer count reaches zero, the interrupt request latch will be activated.

D1 Break (BRK)

A high in bit 1 holds the serial transmitter output in the low state (spacing). RES will override (see "DØ Reset" below).

A low in bit 1 allows normal operation. BRK should be low for normal operation.

DØ Reset (RES)

A high in bit Ø causes the following actions:

a) The Serial· Receiver goes into search mode;

RDA, SBO, FBD, and ORE are set to zero. The contents of the receiver buffer are not affected.

b) The Serial transmitter output is set high (marking). If DØ and D1 are both high, the RES function will override. RES sets TBE high.

c) The interrupt register is cleared except for the TBE interrupt request which is set high.

d) The interval timers are cleared.

RES is not latched.

Ø3 IN Interrupt Address

The CPU reads this register to obtain the encoded address of the highest priority interrupt currently requesting service. This address is identical to the "Restart" instruction op-code for the interrupt acknowledge. Thus, the register contents may be (in order of service priority):

CONTENTS	SOURCE
C7	Timer 1
CF	Timer 2
D7	$\overline{\text{Sens}}$
DF	Timer 3
E7	Receiver Data Available
EF	Transmitter Buffer Supply
F7	Timer 4
FF	Timer 5 or PI7

This register is provided for servicing interrupts via polling. After the register is read, the corresponding bit in the interrupt request register is reset. If the register is read when no interrupt is pending, it will read ØFFH.

Ø3 OUT Interrupt Mask

D7	D6	D5	D4	D3	D2	D1	DØ
Timer5 /PI7	Timer4	TBE	RDA	Timer3	$\overline{\text{Sens}}$	Timer2	Timer1

The contents of this register are logically "And"ed with output from the interrupt request register on the 5501. A high bit in the interrupt mask allows the corresponding request to pass on into the priority encoder. A low bit in the interrupt mask inhibits the corresponding interrupt from passing any further. Since the interrupt requests are latched independently of the state of the mask, an interrupt may be requested while the mask bit is low. The request will be retained until the mask is changed and the request allowed to pass on (assuming no RES command in the interim).

Ø4 IN Parallel Input

This register contains the data presented at J2 (Device A) or at J3 (Device B). The input data must be stable 75 ns after Input $\overline{\text{Strobe}}$ goes low. The peripheral supplying data to the TU-ART can indicate data available by activating the $\overline{\text{SENS}}$ line (or by raising the MSB of the parallel input if the RS7 bit in the command register is high).

When using Z-8Ø block input commands, it is not necessary to supply data at full speed. The input peripheral should simply pull down the WAIT line (pin 2 of J1 or J3) whenever Input $\overline{\text{Strobe}}$ goes low and should not let $\overline{\text{WAIT}}$ go high until the next byte is presented to the TU-ART. (The TU-ART will not read this byte until Input $\overline{\text{Strobe}}$ goes low again.)

Ø4 OUT Parallel Output

This register contains the data which drives the parallel output buffers. The output data is guaranteed stable 1.45 μsec after the falling edge of Output $\overline{\text{Strobe}}$. The TTL output buffers which drive J2 (Device A) and J3 (Device B) may be put in a high-impedance state by pulling down on $\overline{\text{Disable}}$ (pin 8).

When using the Z-8Ø block output commands, it is not necessary to receive data at full speed. The output peripheral should simply pull down the WAIT line (pin 21 of J2 or J3) whenever Output $\overline{\text{Strobe}}$ goes low and not let $\overline{\text{WAIT}}$ go high until the output peripheral has had time to "digest" the data.

Ø5 IN Not Connected

Same as Input Ø2.

05 OUT Timer 1

The CPU outputs a "count" to this register to start Timer 1. This count is decremented by 1 every 64 µseconds after initial loading. When the count reaches zero, bit 0 of the interrupt request register is set and the timer disabled. Since the maximum count is 255, the longest interval is 255x64 µsec. = 16.32 msec. Accuracy is plus 0 and minus 64 µsec. Loading a count of zero causes an immediate interrupt request to the interrupt request register. Loading a new count while the timer is counting re-initializes the timer without an interrupt request. If HBD is high in the command register, the timers will count 8 times as fast.

06 IN Not Connected

Same as Input 02.

06 OUT Timer 2

Operates in the same fashion as Timer 1.

07 IN Not Connected

Same as Input 02.

07 OUT Timer 3

Operates in the same fashion as Timer 1.

08 IN Not Connected

Same as Input 02.

08 OUT Timer 4

Operates in the same fashion as Timer 1.

09 IN Not Connected

Same as Input 02.

09 OUT Timer 5

Operates in the same fashion as Timer 1.

0AH-0FFH IN And OUT Not Connected

Addressing these I/O ports causes no response from the TU-ART. These I/O ports are available to other parts of the computer system.

Figure 3 Summary Of Register Formats For TU-ART, Each Device

OFFSET	FUNCTION	D7	D6	D5	D4	D3	D2	D1	D0	REF. PAGE
0	IN STATUS	TBE	RDA	IPG	SBD	FBD	SRV	ORE	FME	6
0	OUT STATUS	STOP	9600	4800	2400	1200	300	150	110	7
1	IN SERIAL	MSB	Direction of shift ——————→						LSB	7
1	OUT SERIAL	MSB	Direction of shift ——————→						LSB	7
2	OUT COMMAND	- - -	- - -	TB5	HBD	INE	RS7	BRK	RES	7-8
3	IN INT ADDR	1	1	I4*	I2*	I0*	1	1	1	8
3	OUT INT MASK	T5/PI7	T4	TBE	RDA	T3	SENS	T2	T1	8
4	IN PARALLEL	MSB							LSB	8
4	OUT PARALLEL	MSB							LSB	8
5-9	OUT Timer 1-5	MSB	(Delay=count x 64 µsec, HBD=0)							9
			(Delay=count x 8 µsec, HBD=1)							

*			
I4	I2	I0	Source of Interrupt
0	0	0	Timer 1
0	0	1	Timer 2
0	1	0	SENS
0	1	1	Timer 3
1	0	0	RDA
1	0	1	TBE
1	1	0	Timer 4
1	1	1	Timer 5/PI7

Interrupt Operation

The TU-ART offers sophisticated interrupt capabilities, including on-board priority encoding, interrupt generation, interrupt acknowledgment, and daisy chain expandability. These features, in conjunction with the Cromemco 4 MHz Z-8Ø processor, make very high throughput possible.

IMPORTANT

Both channels of the TU-ART must be properly initialized. An uninitialized TU-ART may generate spurious interrupts! Further, the rest of the system must be interrupt compatible (all Cromemco boards are, although the 8K Bytesaver requires the interrupt modification shown on the 8K Bytesaver schematic).

A description of interrupt operation follows for both the Z-8Ø and 8Ø8Ø type interrupt modes.

3.1 Operation Using Z8Ø Interrupts

When the TU-ART is used with the Cromemco ZPU, all 16 of the possible interrupt sources on the TU-ART can generate a unique response without the need for chaining the interrupt requests and polling the responses. This means fast response from interrupt request to service routine and, when coupled with the 4MHz Z-8Ø, an extremely powerful realtime system can be implemented.

A "high priority" interrupt request is one which takes precedence over lower priority requests. This is shown in the following table where the interrupts serviced first are at the top.

It is, of course, possible to use the interrupt mask of each Device to selectively enable and disable the sources of interrupts (reference the description of OUT Ø3, Interrupt Mask, in the previous section). Remember that the INE bit in the status register mus be high for correct operation of Interrupt Acknowledge cycles. Also, be sure that the Z-8Ø has executed the interrupt mode setting' command ØED5EH

Table 1 Z80 (Mode 2) Response

Priority	TU-ART's (Hex) Z-8Ø INTA Response								Source of Interrupt
	D7	D6	D5	D4	D3	D2	D1	DØ	
15 (Highest)	Set By Device A Adr. A7	Set B, Device A Adr. A6	Set By Device A Adr. A5	Ø	Ø	Ø	Ø	Ø	Device A, Timer 1
14				Ø	Ø	Ø	1	Ø	Device A, Timer 2
13				Ø	Ø	1	Ø	Ø	Device A, $\overline{\text{SENSA}}$
12				Ø	Ø	1	1	Ø	Device A, Timer 3
11				Ø	1	Ø	Ø	Ø	Device A, RDA
10				Ø	1	Ø	1	Ø	Device A, TBE
9				Ø	1	1	Ø	Ø	Device A, Timer 4
8				Ø	1	1	1	Ø	Device A, Timer 5 (PI7)
7				1	Ø	Ø	Ø	Ø	Device B, Timer 1
6				1	Ø	Ø	1	Ø	Device B, Timer 2
5				1	Ø	1	Ø	Ø	Device B, $\overline{\text{SENSB}}$
4				1	Ø	1	1	Ø	Device B, Timer 3
3				1	1	Ø	Ø	Ø	Device B, RDA
2				1	1	Ø	1	Ø	Device B, TBE
1				1	1	1	Ø	Ø	Device B, Timer 4
Ø (Lowest)				1	1	1	1	Ø	Device B, Timer 5 (PI7)

"IM2") and the interrupt enable command ØFBH 'EI"). Both of these instructions must be executed each time the Z-8Ø is RESET.

Assuming that both the Z-8Ø and the TU-ART have been initialized, the reception of a byte of serial data at Device B would initiate the following sequence of events:

a) The assembled byte is loaded into the receiver data buffer.

b) The RDA status bit is set and the interrupt request register (bit 3) is set.

c) If bit 3 of the interrupt mask of the Device in question is a one, the request passes on to the priority encoder. If bit 3 is a zero, no further action occurs until the mask is changed.

d) The priority encoder compares all incoming interrupt requests and sets its output to the value of the highest priority incoming interrupt. Thus, since Device B receives the serial data byte in our example; the priority encoder will set its output to "priority 3" if Timers 1, 2, 3, and $\overline{\text{SENSB}}$ from Device B are inactive or masked out.

e) Device B's INT pin goes high, which in turn pulls $\overline{\text{PINT}}$ low on the S-1ØØ bus.

f) The Z-8Ø checks the interrupt line at the end of the current instruction, and finding the line active, goes into an Interrupt Acknowledge (INTA) cycle.

g) The occurrence of the INTA cycle is detected by the TU-ART which then transmits PRIORITY $\overline{\text{OUT}}$ = Ø to connector J1. This temporarily disables Interrupt Acknowledge from lower priority boards. If no board with higher priority is holding PRIORITY $\overline{\text{IN}}$ low, and if Device A has no interrupt pending, then Device B gates the proper Z-8Ø INTA response vector onto the data bus. In this example, Device B would place 18H logically ORed with (A7) (A6) (A5) ØØØØØ from Device A's base address on the data bus. The corresponding bit in the interrupt request latch is reset.

h) The Z-8Ø reads the INTA response byte and appends it to the byte in the I register. This then forms a sixteen bit address which points to the first of two sequential bytes in memory which in turn designate the actual starting address of the service routine. The CPU automatically executes a CALL to this starting address.

3.2 Operation Using 8Ø8Ø Mode Interrupts

When the TU-ART is used in Z-8Ø interrupt mode Ø it is necessary to "chain" Device B through the SENS input on Device A. This requires one of the eight INTA responses, RST2 ØD7H), to be serviced by a routine which polls the status and interrupt address registers of Device B. The remaining seven INTA responses are serviced immediately. The resulting priority assignments are shown in Table 2.

Table 2 Z8Ø (Mode Ø) Response

Priority	TU-ART's (Hex) 8Ø8Ø INTA Response	Souce of Interrupt
15 (Highest)	C7 (RSTØ)	Device A, Timer 1
14	CF (RST1)	Device A, Timer 2
13	D7 (RST2)	Device B, Timer 1
12	D7 (RST2)	Device B, Timer 2
11	D7 (RST2)	Device B, $\overline{\text{SENSB}}$
1Ø	D7 (RST2)	Device B, Timer 3
9	D7 (RST2)	Device B, RDA
8	D7 (RST2)	Device B, TBE
7	D7 (RST2)	Device B, Timer 4
6	D7 (RST2)	Device B, Timer 5 (PI7)
5	D7 (RST2)	Device B, $\overline{\text{SENSA}}$
4	DF (RST3)	Device A, Timer 3
3	E7 (RST4)	Device A, RDA
2	EF (RST5)	Device A, TBE
1	F7 (RST6)	Device A, Timer 4
Ø (Lowest)	FF (RST7)	Device A, Timer 5 (PI7)

It is of course, possible to use, the interrupt mask of each Device to selectively enable and disable the sources of interrupts (reference the discussion of OUT Ø3, Interrupt Mask, in the previous section).

It is not necessary to reset the INE status bit of Device B to zero even though Device B can never respond directly to an Interrupt Acknowledge (INTA) cycle. The INTA status information is not fed to Device B if 8Ø8Ø mode INTA has been selected on the Option DIP Switch. Therefore, the 55Ø1 never attempts to drive the bus during INTA.

No wiring changes are necessary to disconnect the INT pin of Device B from the $\overline{\text{PINT}}$ driver and to connect it to the Device A SENS pin. All this is done automatically when 8Ø8Ø mode INTA has been selected on the Option DIP Switch. Note that $\overline{\text{SENSA}}$ at J1 is still connected. Pulling this line low will generate an interrupt request. The Z-8Ø must execute the EI instruction (ØFBH) after resets or interrupts before an interrupt may take place.

The sequence of events corresponding to Device B receiving a byte of serial data are as follows:

a) The assembled byte is loaded into the receiver data buffer.

b) The RDA status bit is set, the interrupt request register bit 3 is set, and the IPG status bit is set in the device which received the character (Device B in this example).

c) If bit 3 of the interrupt mask of the device in question is a one, the interrupt request passes on to the priority encoder. If bit 3 is a zero, no further action occurs until the mask is changed.

d) The priority encoder compares all incoming interrupt requests and sets its output to the value of the highest priority incoming interrupt. Thus, if Device B received the serial data byte in our example, the priority encoder will set its output to priority three if and only if Device B's Timers 1, 2, and 3 and $\overline{\text{SENSB}}$ are inactive or masked out.

e) Device B's INT pin goes high which in turn activates the SENS pin of Device A.

f) If bit 2 of Device A's interrupt mask is a one, the interrupt request will pass on to the priority encoder. If bit 2 is a zero, no further action occurs until the mask is changed.

g) The priority encoder in Device A compares all incoming interrupt requests and sets its output to the value of the highest priority incoming interrupt. In our example, the interrupt from Device B activates the SENS input at Device A. This interrupt will have top priority if and only if Device A's Timers 1 and 2 are inactive or masked out.

h) Device A's INT pin goes high which in turn pulls $\overline{\text{PINT}}$ low on the S-1ØØ bus.

i) The CPU checks the interrupt line at the end of the current instruction and, finding it active, goes into an Interrupt Acknowledge (INTA) cycle.

j) The occurrence of the INTA cycle is detected by the TU-ART which then transmits PRIORITY $\overline{\text{OUT}}$ = Ø to J1. This temporarily disables Interrupt Acknowledge from lower priority boards. If no board with high priority is holding PRIORITY $\overline{\text{IN}}$ low, Device A will gate an 8Ø8Ø INTA response onto the bus. In this example, Device A would place ØD7H on the data bus (RST2). The corresponding bit in Device A's interrupt request register is reset.

k) The Z-8Ø reads the INTA response byte and performs a CALL to location 1ØH, the starting address of the RST2 service routine.

l) The service routine located at starting location 1ØH, reads the status register of Device B. If IPG is zero, no interrupts are pending in Device B so that the interrupt request must have originated from the $\overline{\text{SENSA}}$ line. The service routine branches to the appropriate subroutine.

If IPG is one, Device B has an interrupt pending which must be serviced. The source of the interrupt is determined by reading Device B's Interrupt Address register. In our example, the Interrupt Address register would contain E7H. When this byte is read, the corresponding bit of the interrupt request register will be reset. The service routine has now determined the true cause of the interrupt and branches to the appropriate subroutine.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

BYTESAVER II

Operating Instructions

Julio, 1981

Operating Instructions

Operating the BYTESAVER II board simply involves inserting from one to eight 2708 PROM devices in sockets ROM0 – ROM7 (any sockets may be used or left unused), setting four switch groups to configure the board, plugging the board into a convenient S-100 bus slot, then applying system power. To program a PROM, you will additionally need to run software described in Section 3, PROM PROGRAMMING INSTRUCTIONS.

2.1 Switch Options-An Overview

The BYTESAVER II is configured by setting four switch groups located along the top edge of the board (see Figure 1). To provide an operational overview, and for later quick reference, the function of each switch is briefly explained in this section.

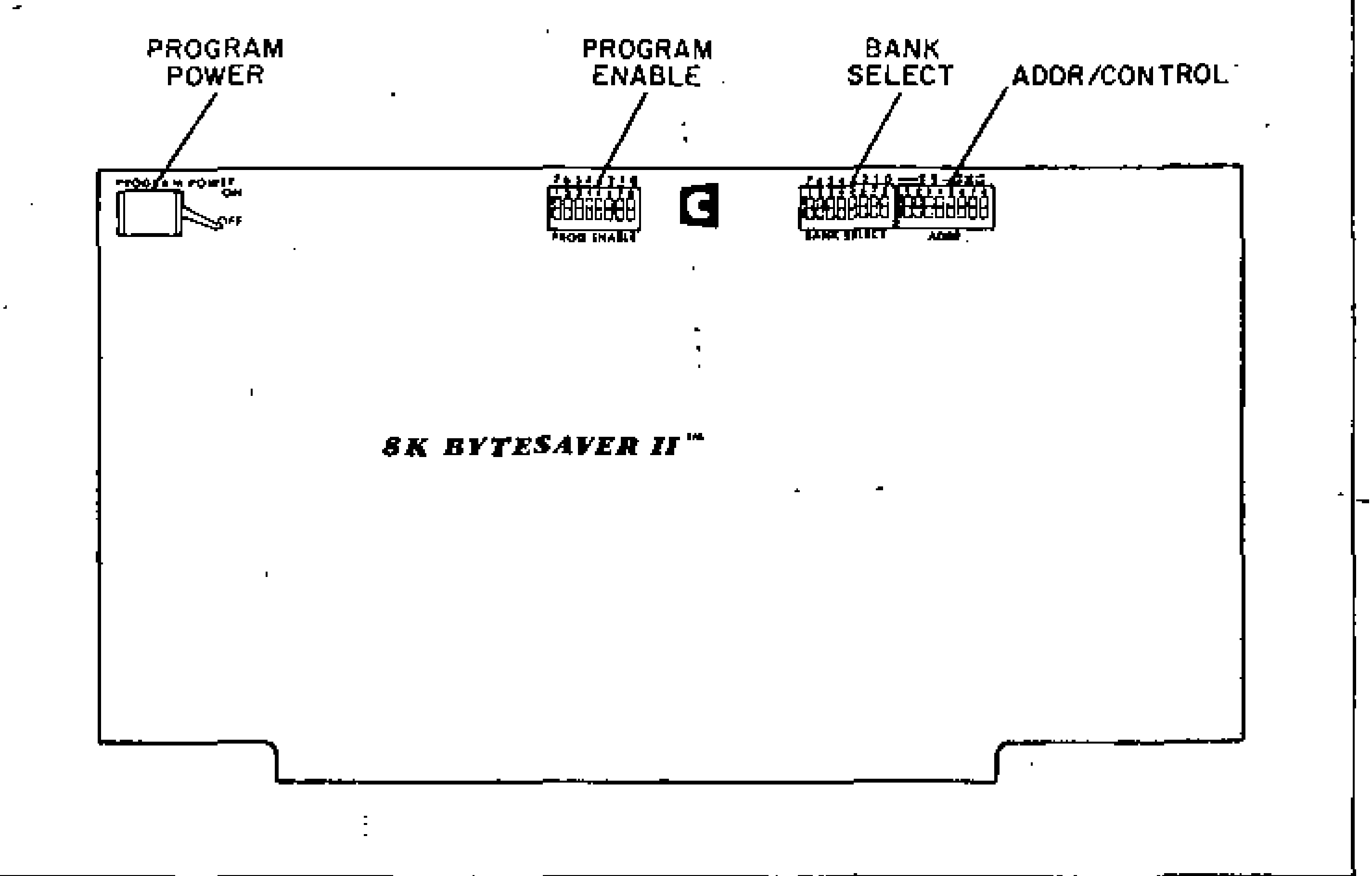
PROGRAM POWER TOGGLE SWITCH

The PROGRAM POWER switch turns the +33.5 volt dc to dc power supply ON and OFF. Position this switch ON before PROM programming; position it OFF when done to prevent inadvertent re-programming.

PROGRAM ENABLE SWITCHES

The eight PROGRAM ENABLE switches individually enable and inhibit programming sockets ROM0 thru ROM7. An ON switch enables programming; an OFF switch inhibits programming. These switches may be alternately viewed as MEMORY PROTECT switches,

Figure 1: SWITCH LOCATIONS



preventing any memory write operations when in the OFF position.

To enable and disable socket programming, associate the board socket numbers (ROM0–ROM7) with the numerals printed **above** the switch DIP (7 to the far left, 0 to the far right).

BANK SELECT SWITCHES

The eight BANK SELECT switches map the BYTESAVER II into any combination of 64K-byte memory banks (bank 0 – bank 7). Setting a BANK SELECT switch ON logically places the board in the correspondingly numbered memory bank; an OFF switch logically removes the board from a bank. Again, associate the bank number with the numerals printed above the BANK SELECT switches, not the numerals on the DIP proper.

ADDR/CONTROL SWITCHES

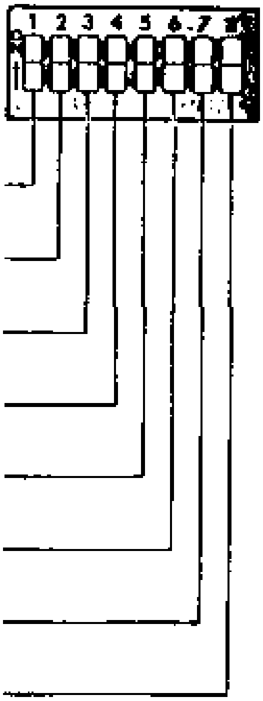
The ADDR/CONTROL switches control several different functions (see Figure 2).

The BANK ENABLE/DISABLE switch enables multiple 64K memory banks (bank 0 – bank 7) when ON, and disables multiple banks when OFF (normal direct 64K addressing).

The DMA ENABLE/DISABLE switch enables DMA OVERRIDE when ON and disables DMA OVERRIDE when OFF. For normal direct 64K DMA addressing, position the switch OFF. When performing DMA with memory banks enabled, turn the switch ON. The DMA IN/OUT switch is active only when DMA OVERRIDE is enabled. With DMA OVERRIDE enabled, the BYTESAVER II will respond directly to a DMA in the board's 16-bit address range by board enabling if DMA is IN, and by board disabling if DMA is OUT, regardless of current active memory bank status at the time. This feature effectively permits the user to define one board out of several stacked in different memory banks as **the** DMA board (the one with DMA IN), and the boards in other memory banks as non-DMA boards (the one with DMA OUT).

The WAIT STATE switch is used to match the CPU cycle time to the 2708 PROM 450ns (max) memory access time. Positioning the WAIT STATE switch ON introduces one wait state during each machine cycle; the OFF position introduces no wait states. When used in a Cromemco system with a ZPU running at 4

Figure 2: ADDR/CONTROL SWITCHES



SWITCH NUMBER		FUNCTION

1.	BANK ENABLE	ON = ENABLED
		OFF = DISABLED

2.	DMA OVERRIDE	ON = ENABLED
		OFF = DISABLED

3.	DMA IN/OUT	ON = DMA OUT
		OFF = DMA IN

4.	WAIT STATE	ON = ONE WAIT STATE
		OFF = NO WAIT STATES

5.	A15 BLOCK SELECT	ON = (A15 = HI)
		OFF = (A15 = LO)

6.	A14 BLOCK SELECT	ON = (A14 = HI)
		OFF = (A14 = LO)

7.	A13 BLOCK SELECT	ON = (A13 = HI)
		OFF = (A13 = LO)

8.	NOT USED	

MHz, position the switch ON. The switch may be left OFF when operating at 2 MHz.

The three high order address select switches A13, A14 and A15 memory map the BYTESAVER II into one-of-eight 8K-byte memory "blocks." Setting all three switches OFF maps the BYTESAVER II into the lowest 8K-byte block of memory (0000H – 1FFFH); setting all switches ON maps the board into the highest 8K-byte block (E000H – FFFFH).

EXAMPLE 1

Suppose you have a 4 MHz Cromemco system, and you want to memory map your BYTESAVER II into the highest 8K-byte memory block (E000H – FFFFH). As a standard practice, you decide to program 2708 PROMs in socket ROM7 only. Also assume there is no other memory overlapping the uppermost 8K of memory, so multiple memory banks are not required.

For memory read operation, the BYTESAVER II switch settings would then be as shown in Figure 3.

To program a 2708 PROM in socket ROM7, all switch settings remain the same except the PROGRAM POWER switch, which must be turned ON. ■

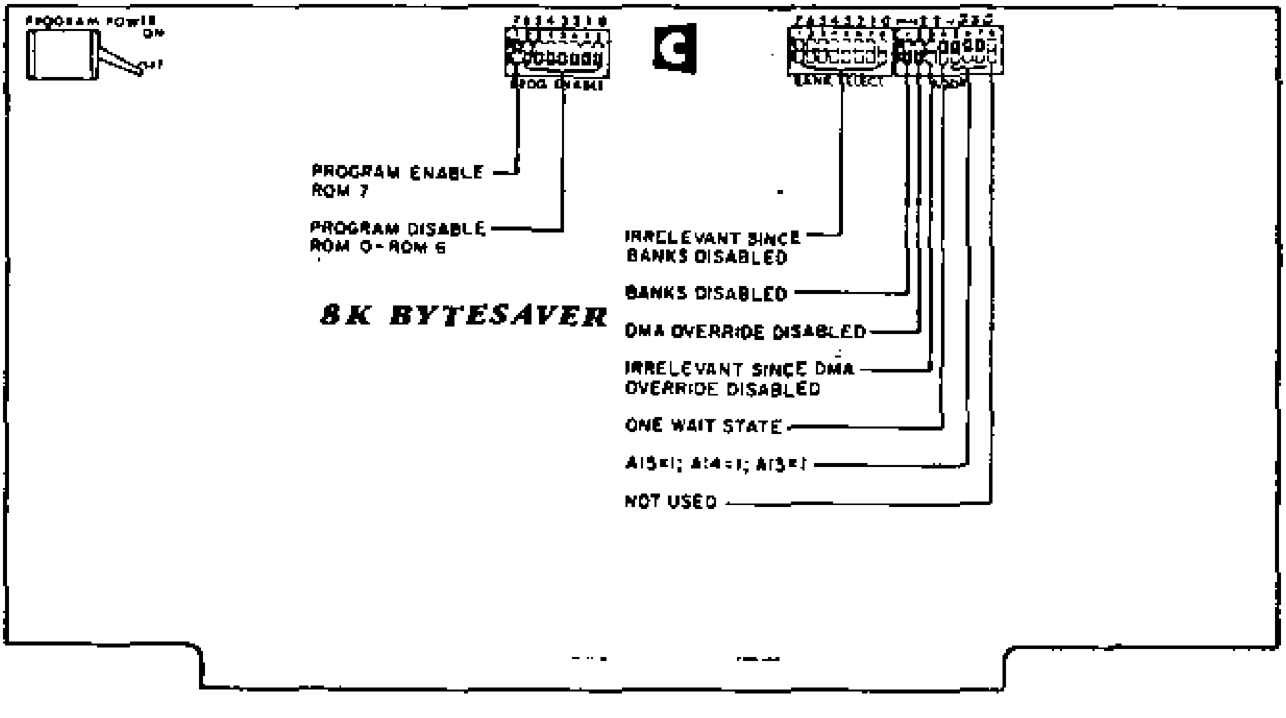
The following example illustrates all of the BYTESAVER II special features.

EXAMPLE 2

Suppose you set your Cromemco ZPU card for 2 MHz operation, and assign your BYTESAVER II to memory block 4 (8000H-9FFFH). Again as a standard practice, you program 2708 PROMs in socket ROM7 only.

Also assume another Cromemco memory board with BANK SELECT exists for DMA transfers only in overlapping memory 8000H – BFFFH, bank 1 (a 16KZ RAM card, for example). You then decide to map the BYTESAVER II into memory bank 0 so that it will be enabled on a system RESET or a Power-ON Clear (see Section 2.5 for details).

Figure 3: EXAMPLE 1 SWITCH SETTINGS



The correct BYTESAVER II switch settings for this configuration are then shown in Figure 4. ■

The sections which immediately follow discuss all of the BYTESAVER II special features and operational modes touched upon in this section in greater detail.

2.2 Addressing The Bytesaver II

Addressing a byte on the BYTESAVER II involves four levels of selection: choosing a memory bank, a memory board, an IC chip, and finally choosing the byte-on-chip.

Memory banks are addressed by the CPU outputting a control word to an integral OUTPUT PORT 40H contained on each BYTESAVER II board. Board, chip and byte-on-chip are all decoded from the sixteen bit address sent out by the CPU on the S-100 bus.

Since the board capacity is 8K bytes, board select is generated by the high order address lines A13, A14 and A15. There are eight ROM sockets, so the next three high order address lines A10, A11 and A12 are used to hardware generate chip enable (selecting ROM0–ROM7), and the remaining ten address lines A0–A9 are used to address one-of-1024 bytes on a 2708 PROM (see Figure 5).

Figure 5 BYTESAVER II ADDRESSING

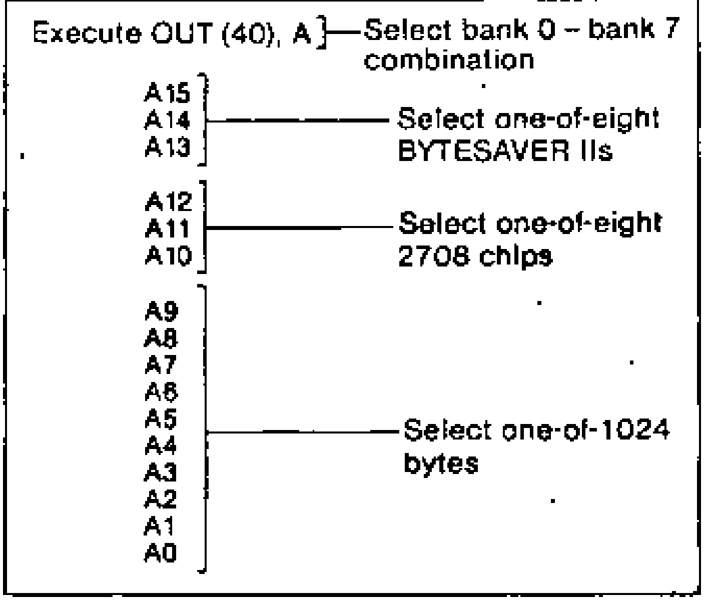
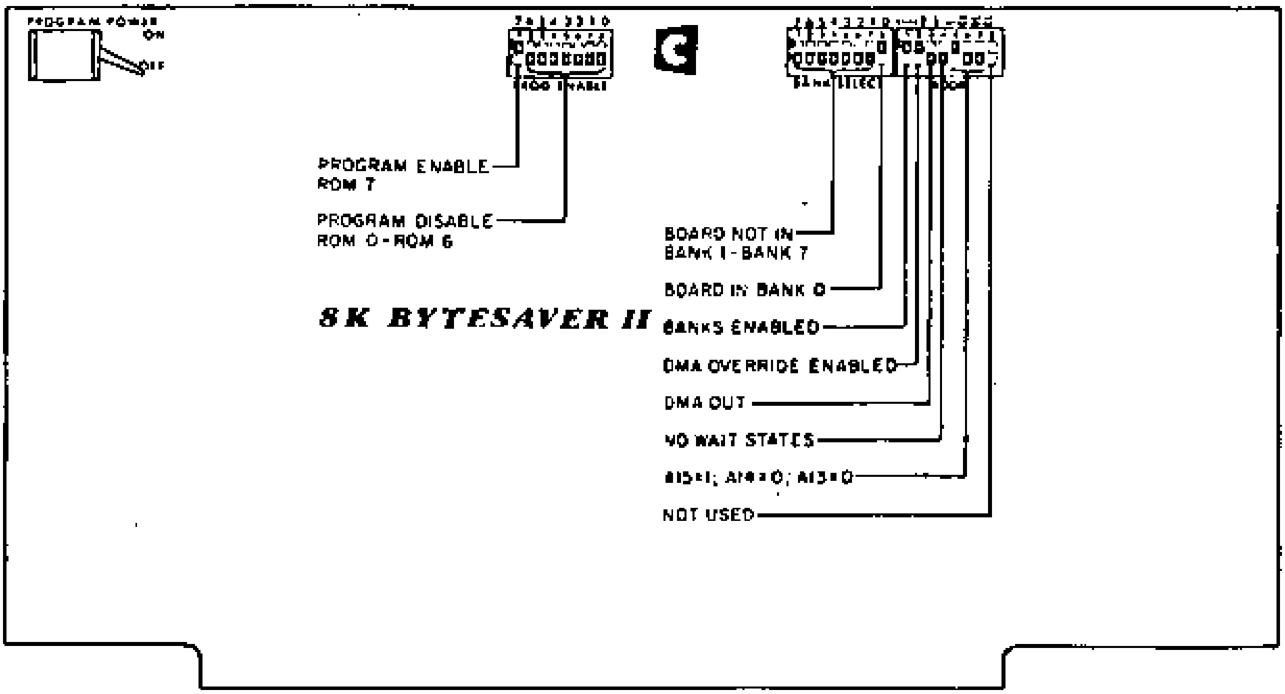

Execute OUT (40), A
Select bank 0 – bank 7 combination
A15
A14
A13
Select one-of-eight BYTESAVER IIs
A12
A11
A10
Select one-of-eight 2708 chips
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
Select one-of-1024 bytes

Figure 4: EXAMPLE 2 SWITCH SETTINGS



2.3 Board SELECT/CHIP Select

The three high order S-100 bus address lines are hardware compared to switches A13, A14 and A15 in the ADDR/CONTROL switch group. Any switch ON corresponds to a selected logic 1 on its corresponding address line; any switch OFF selects a logic 0 on its corresponding address line. The eight switch setting combinations and their corresponding BYTESAVER II memory block assignments are tabulated below.

Table 1

SWITCH			BYTESAVER II MEMORY ASSIGNMENT
A15	A14	A13	
OFF	OFF	OFF	0000 — 1FFF; BLOCK 0
OFF	OFF	ON	2000 — 3FFF; BLOCK 1
OFF	ON	OFF	4000 — 5FFF; BLOCK 2
OFF	ON	ON	6000 — 7FFF; BLOCK 3
ON	OFF	OFF	8000 — 9FFF; BLOCK 4
ON	OFF	ON	A000 — BFFF; BLOCK 5
ON	ON	OFF	C000 — DFFF; BLOCK 6
ON	ON	ON	E000 — FFFF; BLOCK 7

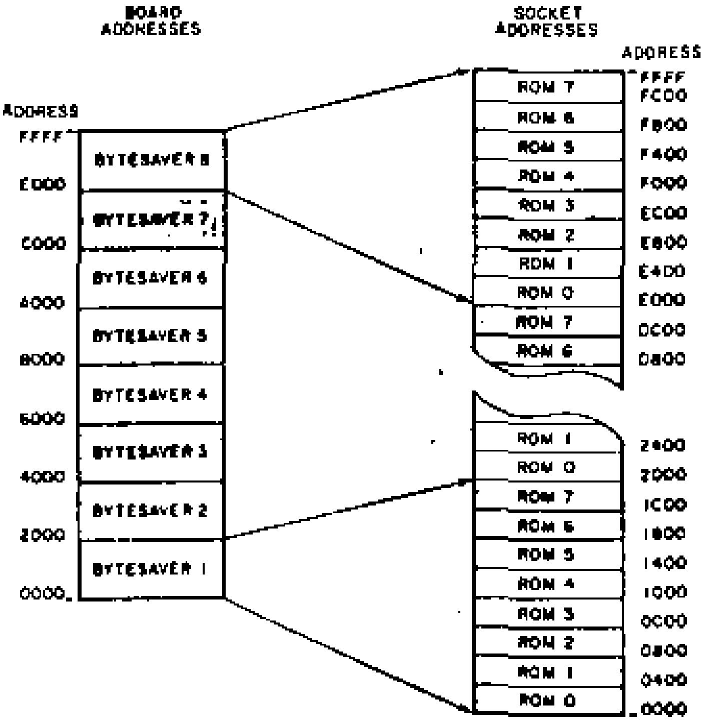
Each ROM socket ROM0 – ROM7 spans a 1K-byte swath of memory. Address lines A10 – A12 feed a one-of-eight decoder (IC19 in the BYTESAVER II Schematic) to generate select signals for each ROM socket. The entire 64K address space may then be spanned by eight BYTESAVER II boards. Figure 6 illustrates such an arrangement along with the address range spanned by each ROM socket.

EXAMPLE 3

Suppose you programmed four 2708 PROMs with Cromemco's Z-80 MONITOR and 3K Control BASIC. The Z-80 MONITOR spans addresses E000H – E3FFH, and Control BASIC spans E400H – EFFFH. To load these programs, you would then place the four programmed PROMs in sockets ROM0, ROM1, ROM2 and ROM3 on a BYTESAVER II assigned to E000H – FFFFH with A13=1, A14=1 and A15=1. ■

Carefully note that another memory module may **not** be mapped into the "hole" created by an empty BYTESAVER II ROM socket. The BYTESAVER II reads an empty ROM socket as memory data 0FFH, and actively drives the S-100 DI bus lines DI0–DI7 at logic

Figure 6: EIGHT BYTESAVER IIs SPANNING THE 64K ADDRESS SPACE



1 levels thereby creating a DI bus conflict when competing with another memory module.

2.4 Memory Banks

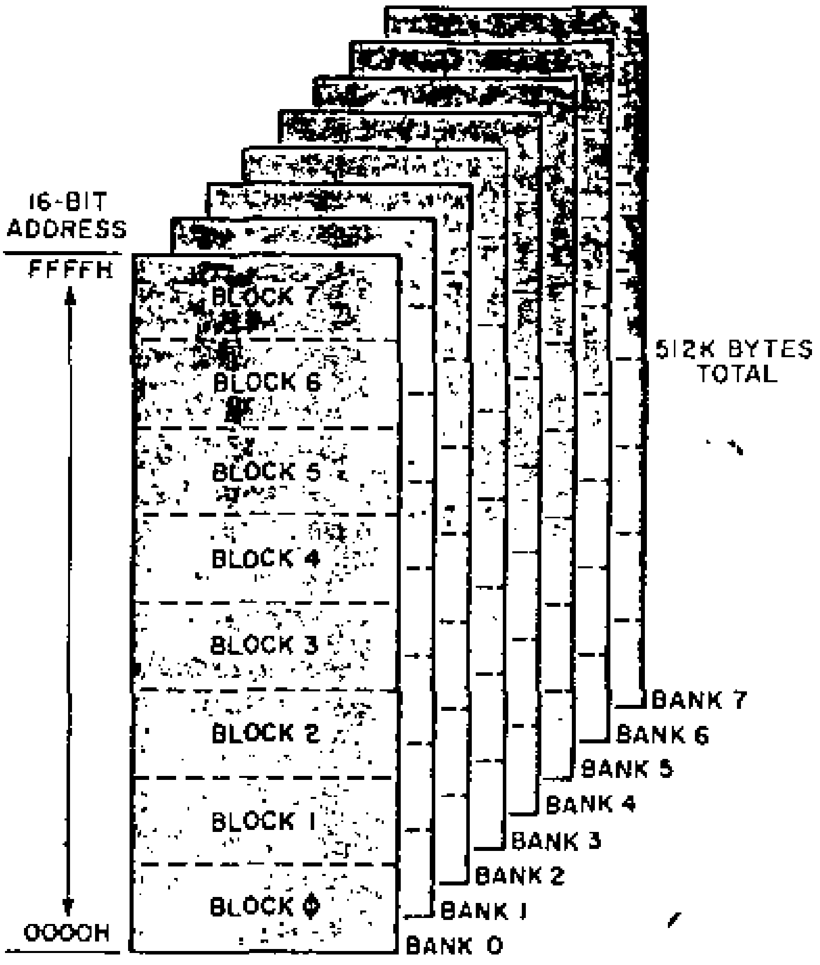
BANK SELECT is an optional board feature which effectively allows memory expansion beyond the CPU's 64K direct addressing range. This feature may be completely disabled by switch selecting BANK DISABLE in the ADDR/CONTROL switch group. When this is done, the eight BANK SELECT switch settings become irrelevant. In this mode the BYTESAVER II exists only in the assigned 8K-byte memory block of the CPU's 64K direct addressing range for memory read, PROM programming and DMA operations.

To enable memory banks, switch select BANK ENABLE in the ADDR/CONTROL switch group. When this is done, the BYTESAVER II is logically placed in one or more 64K-byte memory banks with the eight BAN SELECT switches, and bank addressing is softwar controlled by executing the OUT (40H), A (or equivalent) Z-80 instruction.

Memory may be stacked up to eight banks deep (see Figure 7). Positioning one or more BANK SELECT switches ON places a BYTESAVER II in **each** corresponding memory bank. On the other hand, positioning all switches OFF completely removes the board from the memory map (except possibly for DMA transfers— see Section 2.6).

As stated above, memory banks are activated and de-activated under software control. Each BYTESAVER II contains an integral OUTPUT PORT 40H which latches the bits of the control byte output to it by the CPU. Each set bit (logic 1) enables its corresponding

Figure 7: THE MEMORY MAP WITH MULTIPLE MEMORY BANKS



memory bank, and each reset bit (logic 0) disables its bank. Control byte bit 7 (MSB) controls memory bank 7, bit 6 controls memory bank 6, etc.

If the BYTESAVER II is switch mapped into **any** of the banks activated by the control byte (logical OR), the board responds when addressed and thus is placed "in" the memory map. When this condition occurs, the green LED indicator lights. Conversely, if the BYTESAVER II is switch mapped into **no** bank activated by the output control byte, the board will not respond when addressed and thus is "out" of the memory map. When a control byte inactivates the board, the green LED indicator goes out, and more specifically, the board responds by tri-stating (floating) all of its output lines. This behavior allows two or more memory boards with BANK SELECT to occupy the same or overlapping 16-bit address space but in different memory banks, provided only one board is memory bank active at a time, and all other boards are inactive. Memory bank conflicts may result if:

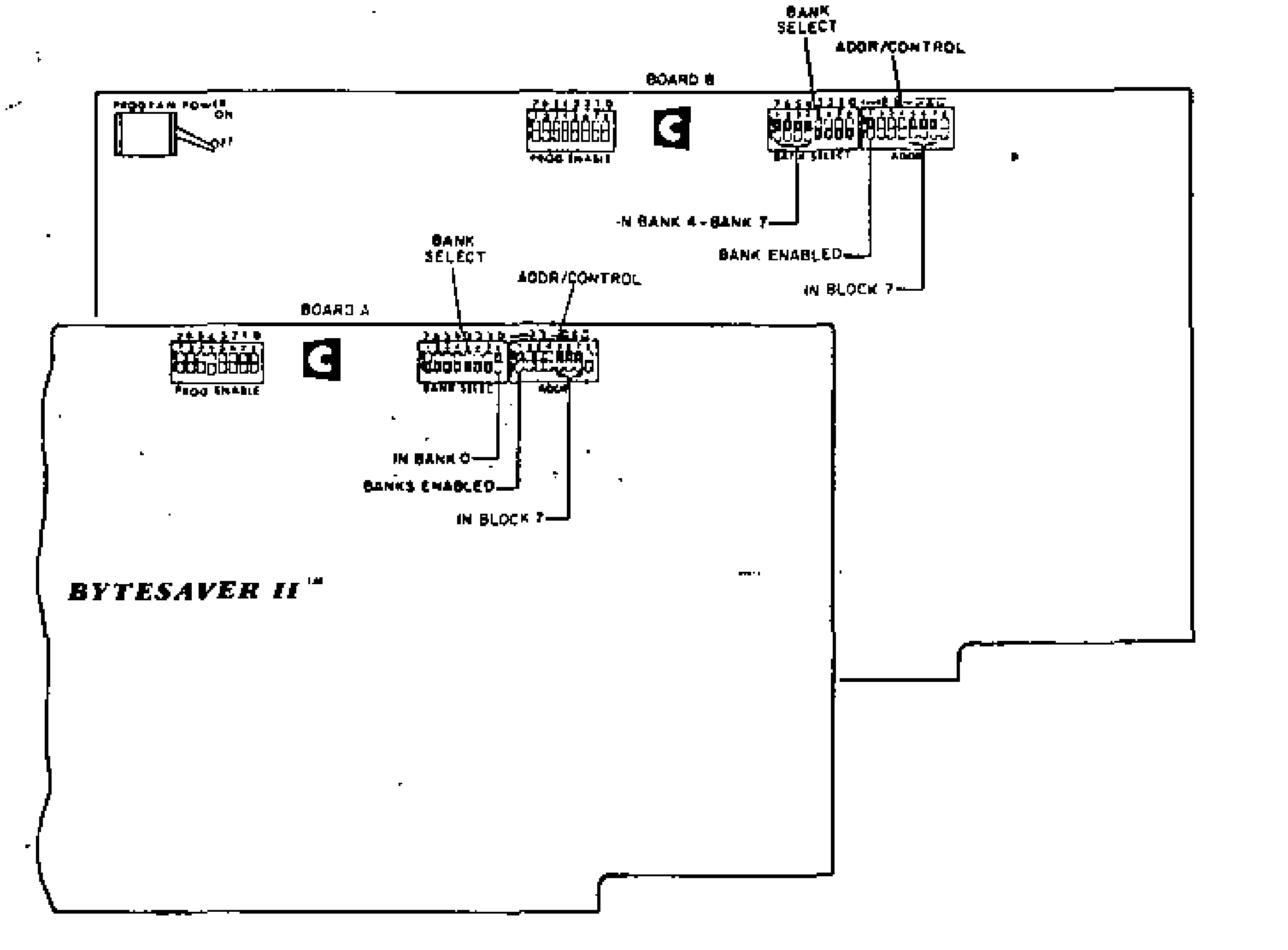
a) Two or more address overlapping memory boards are switch assigned to the same memory bank, or

b) Two or more 16-bit address overlapping memory boards assigned to disjoint memory banks are simultaneously activated by the same control byte.

EXAMPLE 4

Suppose two BYTESAVER IIs are both mapped into the uppermost 8K of memory, and their memory bank switches are set as shown in Figure 8. The resulting memory map is then shown in Figure 9.■

Figure 8: EXAMPLE 4 SWITCH SETTINGS

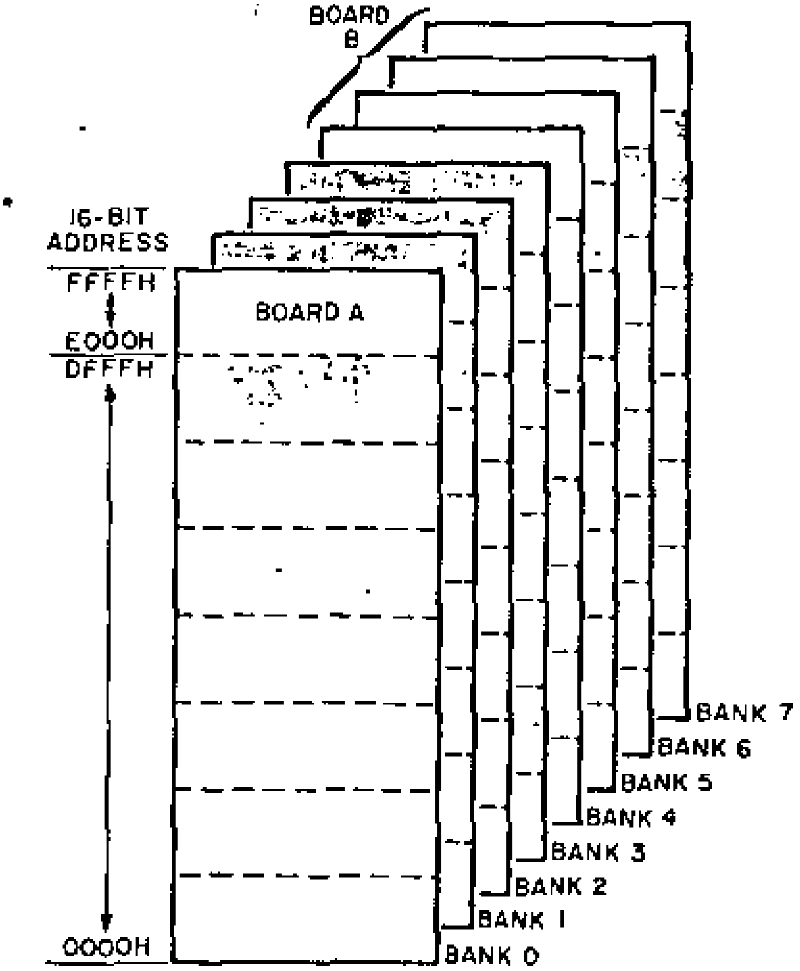


To continue the same sample, the sample programs ahead illustrate how to memory bank enable and disable the two boards.

- Executing the instructions below activates memory banks 2 and 3, and de-activates other memory banks. The instructions then place both board A and board B in inactive memory banks (both boards inaccessible).

ADDR	OBJECT	MNEMONIC	COMMENT
0000 0002 0004	3E0C D340 —	LD A,00001100B OUT (40H),A —	;LOAD 0000 1100 INTO REG. A ;OUTPUT CONTROL BYTE TO PORT 40H ;NEXT INSTRUCTION

Figure 9: EXAMPLE 4 MEMORY MAP



- Executing the instructions below simultaneously activates both boards A and B, and thus is **illegal**.

ADDR	OBJECT	MNEMONIC	COMMENT
0000		LD A,10000001B	;LOAD 1000 0001 INTO REG. A
0002		OUT (40H),A	;OUTPUT CONTROL BYTE TO PORT 40H
0004		—	;NEXT INSTRUCTION

- Executing the instructions below places board A in an active memory bank, and board B in an inactive memory bank (board A available for memory read, PROM programming and DMA transfers; board B inaccessible).

ADDR	OBJECT	MNEMONIC	COMMENT
0000	3E01	LD A,00000001B	;LOAD 0000 0001 INTO REG. A
0002	D340	OUT (40H),A	;OUTPUT CONTROL BYTE TO PORT 40H
0004	—	—	;NEXT INSTRUCTION

- Executing the instructions below places board A in an inactive memory bank and board B in an active memory bank (board A inaccessible; board B available for memory read, PROM programming and DMA transfers).

ADDR	OBJECT	MNEMONIC	COMMENT
0000	3E60	LD A,01100000	;LOAD 0110 0000 INTO REG. A
0002	D340	OUT (40H),A	;OUTPUT CONTROL BYTE TO PORT 40H
0004	—	—	;NEXT INSTRUCTION

2.5 SELECT BANK 0 On RESET Or POWER-ON-CLEAR (POC)

When system power is first applied, or after a subsequent system RESET, the BYTESAVER II will respond in one of two different ways. If multiple memory banks are DISABLED, the board will remain "in" the memory map in the CPU's 64K-byte direct addressing range.

If multiple memory banks are ENABLED, memory bank 0 is **automatically** hardware activated by a system RESET or a POC, and bank 1 – bank 7 are de-activated. Thus, a RESET or a POC to the boards in Example 4 would activate board A, and de-activate board B.

2.6 Direct Memory Access

A device may request direct memory access to the BYTESAVER II by asserting the S-100 bus line pHOLD low. The CPU grants the request by driving line pHLDA (hold acknowledge) high. When control line pHLDA is high, the device then may directly drive the S-100 bus address lines and control lines (which are tri-stated during DMA transfers when pHLDA is high), and use the data bus lines for reading or writing without CPU intervention. The device may then transfer data at a rate limited only by the memory access time.

The general features of a DMA transfer are then:

- Fast asynchronous read or write access to memory.
- The DMA device should **not** be responsible for many overhead tasks (such as memory bank switching) to keep the memory access as quick as possible.
- The access is direct—no CPU intervention to slow the transfer.
- The DMA device must be capable of controlling and driving the tri-stated address, data and control busses.

In line with this general philosophy, the BYTESAVER II's DMA response behavior is controlled by two switches in the ADDR/CONTROL switch group; DMA OVERRIDE and DMA IN/OUT. There are four possible switch setting combinations; each is tabulated and discussed below.

The first table entry indicates the board behav with DMA OVERRIDE DISABLED (note that in this case the DMA IN/OUT switch setting is irrelevant). Here, the key phrase is "correctly addressed;" the BYTESAVER II will respond for memory read, write (PROM programming) or DMA transfers only when it is in an active memory bank (if multiple memory banks are enabled), and the S-100 bus address falls within the board's assigned 8K block of memory. The board in effect **does not** differentiate between a DMA data transfer and a normal read/write cycle in any way.

The BYTESAVER II **does** differentiate between DMA and non-DMA transfers with DMA OVERRIDE ENABLED, as shown in the last two table entries. A typical application demonstrating how DMA OVERRIDE works is shown in Figure 10.

Here, two BYTESAVER IIs are assigned to the same 16-bit address space with switches A13, A14 and A15; board A is assigned to memory bank 0, and board B to memory bank 1 (any other Cromemco memory boards

Table 2

DMA OVERRIDE SWITCH	DMA IN/OUT SWITCH	BYTESAVER II RESPONSE
DISABLED	IN or OUT	Board enables when correctly addressed for either DMA or non-DMA transfers.
ENABLED	OUT	Board enables when correctly addressed for non-DMA transfers (normal operation); board disables during any system DMA transfer.
ENABLED	IN	Board enables when correctly addressed for non-DMA transfers; board enables when the DMA device addresses the board's assigned 8K block of memory, regardless of which banks were active before the DMA request.

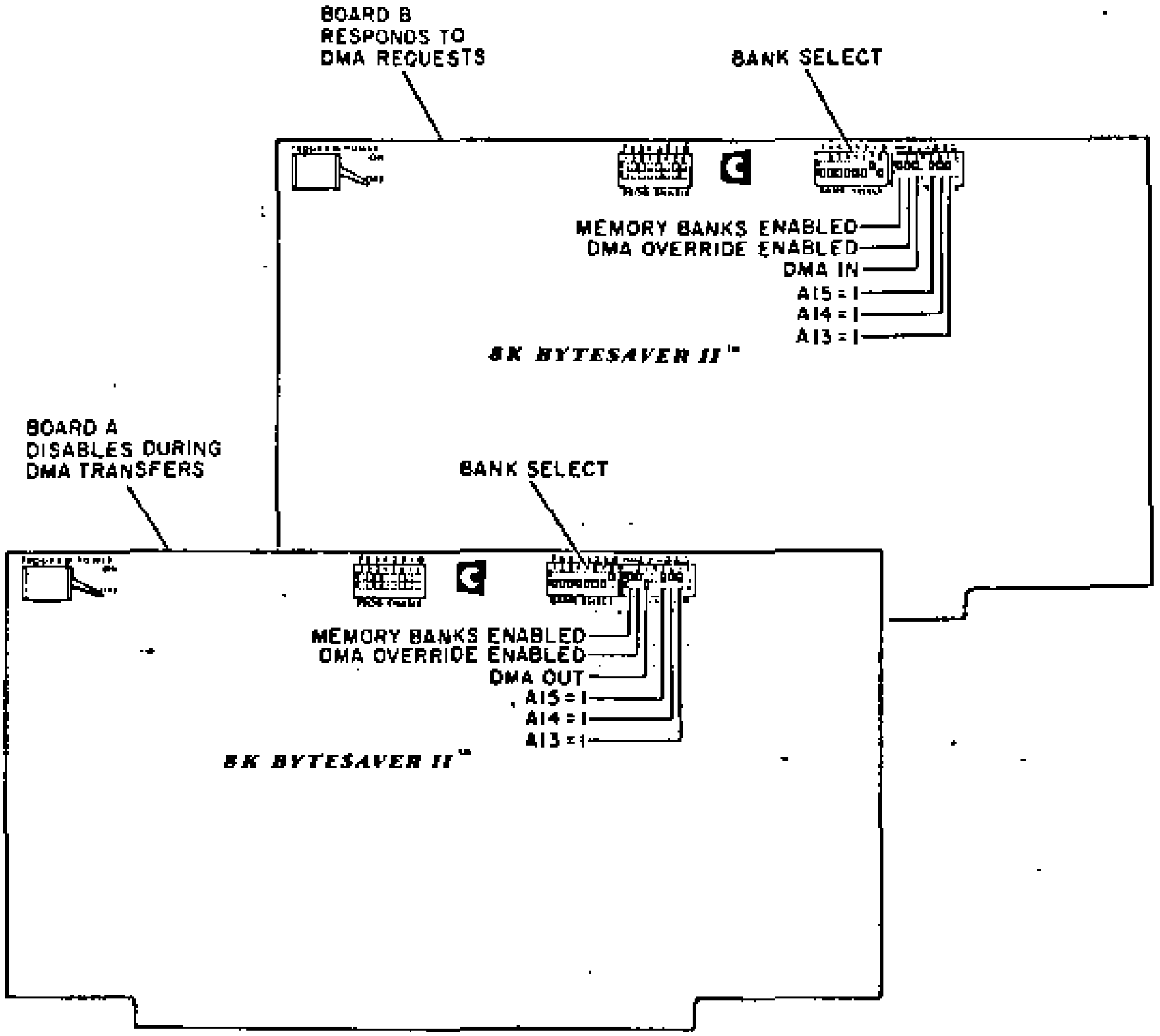
with BANK SELECT and DMA OVERRIDE could also be used in the example). For non-DMA transfers, both boards are available for read/write operations when correctly addressed (board A is in memory bank 0 at E000H – FFFFH and board B is in memory bank 1 at E000H – FFFFH).

When the CPU grants an asynchronous DMA request by driving the pHLDA line high, board A automatically disables and board B enables when the S-100 bus address is in the range E000H – FFFFH, regardless of which board was in an active memory bank before the request.

Thus, the DMA OVERRIDE feature is seen as a means of overriding **logical memory bank boundaries** during a DMA transfer. This provides a fast way of vectoring the DMA device to **the** DMA board (the one with DMA IN) and disabling all non-DMA boards (the ones with DMA OUT) without burdening the DMA device with any overhead memory bank switching responsibilities.

It should be noted that after the DMA transfer is completed, both BYTESAVER IIs revert back to the same memory bank status which existed before the DMA transfer.

Figure 10: DMA OVERRIDE EXAMPLE CONFIGURATION



PROM Programming Instructions

The 2708 is an 8,192-bit ultraviolet light erasable and electrically programmable read-only memory chip. The 2708 is erased, thereby forcing all bits to the logic 1 state, by exposing the chip's transparent quartz window to intense ultraviolet radiation. Consult the 2708 manufacturer's literature for detailed erasure procedures.

To program a 2708 PROM, insert an erased 2708 into a PROGRAM ENABLED BYTESAVER II socket with the system power OFF, turn ON the system power, turn the PROGRAM POWER switch ON, then execute one of the Cromemco system programming commands described in the following sections. If the PROM is to remain in the same socket after programming, the socket would then be PROGRAM DISABLED. The PROGRAM POWER switch should be turned OFF after programming to prevent inadvertent re-programming of other PROMs on the board.

Each 2708 byte is programmed by selectively changing logic 1 (erased) bits to the logic 0 state as required by the pattern being programmed. 2708s may be reprogrammed without intervening erasure provided no attempt is made to change logic 0 bits back to the logic 1 state—only complete EPROM erasure can force this transition.

The Cromemco PROM programming software described below writes a source code byte to each 2708 address in sequence. This process is then repeated until all 1,024 bytes of source code data have been written to the PROM 360 separate times. The BYTESAVER II responds to each memory write cycle by forcing the CPU into an idle state by asserting the CPU pRDY line low, driving the eight 2708 data output pins with the source code byte, and applying a digitally counted 192 usec PROGRAM PULSE (low for 16 usec, high for 176 usec) to the 2708 PROGRAM input pin while the CS/WE line is held at +12 volts. Upon completion of the 192 usec interval, the pRDY line is again asserted high, and program execution resumes. Programming time for one 2708 is then approximately (192 usec/byte) x (1,024 bytes) x (360 programming passes) = 70 seconds.

Specific 2708 programming examples appear in the next three sections. Section 3.1 illustrates how to program 2708s using Cromemco's Z-80 MONITOR, DEBUG and ROS system commands, Section 3.2 discusses programming using 3K Control BASIC, and Section 3.3 illustrates how to program 2708s from Z-80 Assembly Language code.

3.1 Programming From DEBUG, Z-80 Monitor Or ROS

DEBUG (on disc package model FDA-S/L), Z-80 MONITOR (model number ZM-108) and ROS (model number ZA-808) all support a one line 2708 programming command. The respective command formats are illustrated below:

```
-P E000 E3FF FC00<CR>            (DEBUG)
:P E000 E3FF FC00<CR>            (Z-80 MONITOR)
PROM, E000, E3FF, FC00 <CR>      (ROS)
```

where <CR> stands for pressing the RETURN key. Each command would result in programming a PROM located at FC00H – FFFFH with source code located at E000H – E3FFH. Alternatively, these commands could have been entered as:

```
-P E000 S400 FC00 <CR>           (DEBUG)
:P E000 S400 FC00 <CR>           (Z-80 MONITOR)
PROM, E000, S400, FC00<CR>       (ROS)
```

using the swath operator.

The first two arguments in each command define the source code starting location and extent in memory. The source code location may be specified in terms of absolute addresses as in the first three examples, or in terms of a starting address and a swath width as in the last three. The third argument defines the 2708 starting address on the BYTESAVER II. The size of the source file (its swath width) and the 2708 starting address must be an exact multiple of 400H (the addresses must end in either 000H, 400H, 800H or C00H) or the command will be rejected and an error message issued.

After programming the 2708, the source code is compared to the PROM contents, and any discrepancies are printed out according to the format illustrated below:

```
E000 2C 2D FC00
E2BC 03 05 FEBC
E3FF C9 ED FFFF
```

This printout indicates the source code byte 2CH at E000H was incorrectly programmed into the 2708 as 2DH at address FC00H, etc. If there are discrepancies, often reprogramming the 2708 with the same source code will change "stubborn" bits to their proper state. If there are no programming errors, the user is prompted for a new command.

EXAMPLE 5

Assume you have 2K-bytes of development software located at 1000H – 17FFH which you want to store in 2708 PROM. Two erased 2708s occupy sockets ROM6 and ROM7 on a BYTESAVER II assigned to memory area E000H – FFFFH.

To program the PROMs, you would then PROGRAM ENABLE sockets ROM6 and ROM7, turn the PROGRAM POWER switch ON, and issue one of the following commands, depending on which operating system is running:

```
-P 1000 17FF F800<CR>           or    (DEBUG)
-P 1000 S800 F800<CR>

:P 1000 17FF F800<CR>           or    (Z-80
:P 1000 S800 F800<CR>                 .MONITOR)

PROM,1000,17FF,F800<CR>         or    (ROS)
PROM,1000,S800,F800<CR>
```

After programming is complete, the PROGRAM POWER switch should be turned OFF. ■

Storing a disc file program in 2708 PROM is usually accomplished by reading the disc source file into RAM, then executing the DEBUG "P" (Program Proms) command to write the RAM data to PROM located elsewhere in memory. A potential problem exists when programming 2708s with a ".HEX" extension object file using this procedure.

Moving a source file from disc to RAM is accomplished using the "F" (specify file name) and the "R" (read disc file) DEBUG commands. These commands will attempt to load the .HEX file from the disc into RAM beginning at the address specified by the "ORG" statement contained in the source code. The address so defined may not be RAM at all, or the area specified may be inconvenient for other reasons. To circumvent this problem, the .HEX file should be read into a convenient RAM area by specifying an appropriate **displacement** as the argument of the DEBUG "R" command (see next example). For further details, refer to Cromemco's **Macro Assembler Manual**.

EXAMPLE 6

Suppose you have a program named SAMPLE which is "ORGed" at 9200H. You use Cromemco's ASMB program to create a .HEX object file on disc. You would like to read the .HEX file from the disc to RAM starting at 200H, and then to use this data to program a 2708 PROM residing at FC00H – FFFFH.

With DEBUG running, you would then type:

-FSAMPLE.HEX<CR>

This specifies the file name as "SAMPLE.HEX." Then type:

-R7000H

This reads the file from the disc with a displacement of 7000H. The displacement 7000H is added to the ORG operand 9200H to yield the loading point 9200H+7000H= 10200H=0200H when the carry is discarded. Then type:

-P 200 S400 FC00<CR>

This command programs the 2708 at FC00H with the source code located at 200H – 5FFH. ■

3.2 Programming From 3K Control BASIC

3 K Control BASIC (model number CB-308) program text may be stored in 2708 PROM for subsequent loading and execution by issuing an "EPROM" direct command.

To SAVE a Control BASIC (CB) program in a 2708 PROM:

a) Determine the length of the CB program text using the CB SIZE function value.
b) PROGRAM ENABLE sockets containing erased 2708 PROMs.
c) Turn the PROGRAM POWER switch ON.
d) Issue an EPROM ppp command where "ppp" is the 2708 PROM starting "page" address.
e) After receiving a CB message indicating successful programming, turn the PROGRAM POWER switch OFF.

3K Control BASIC logically partitions memory into "pages," where 1 page = 256 bytes. Pages 0 and 1 (0000H – 01FFH) are not used by CB; pages 2 and 3 (0200H – 03FFH) are used for variables, the input buffer and the stack; pages 4 thru 31 (0400H – 1FFFH) are normally used for CB program text and arrays; and pages 32 on (2000H – end of user RAM) are normally used to save CB program files (see Figure 11).

The 'EPROM ppp' command is used to program 2708s with the CB text area for later execution, the 'LOAD ppp' command reads a file from memory back into the text area for editing, and the 'RUN ppp' command initiates execution of the program text located at page 'ppp.'

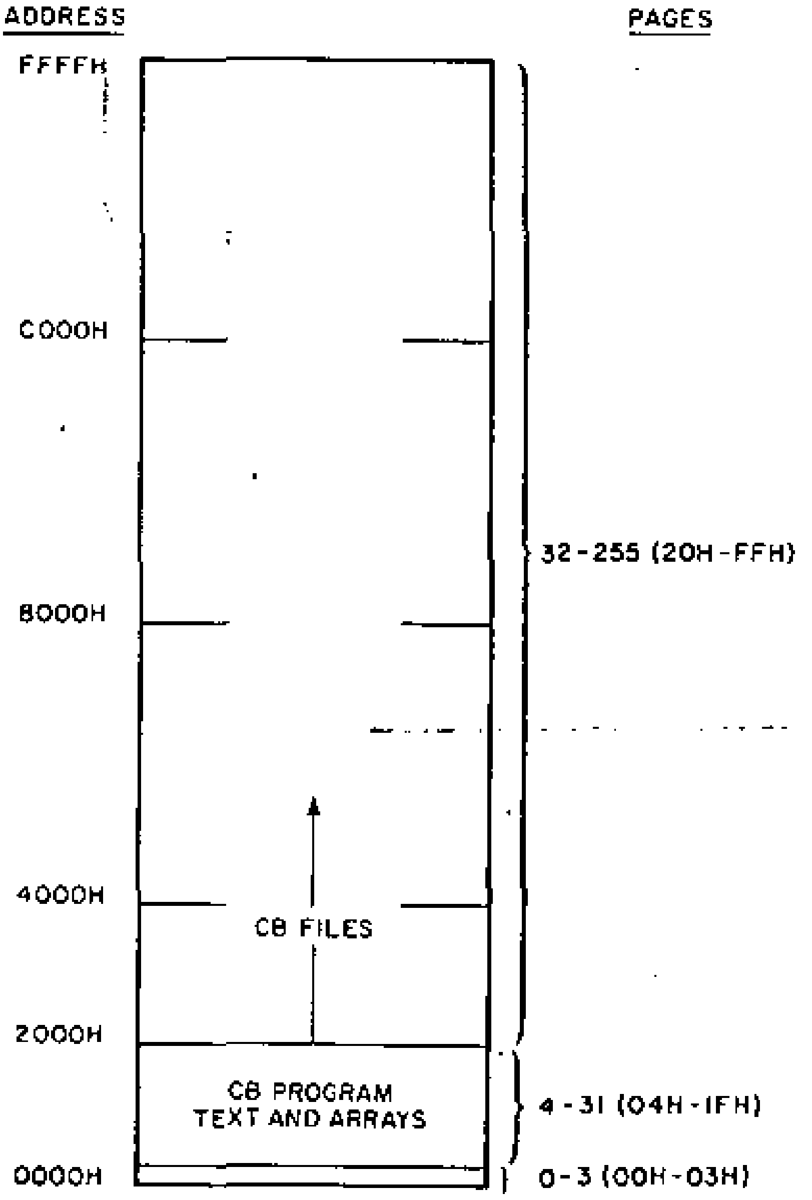
The page number arguments of the EPROM, RUN and LOAD commands are specified in decimal. For the EPROM command, the page argument is the starting address of erased 2708 PROM. This number must be a multiple of 4, and sufficient erased PROM should start at this address to contain all of the CB program text. If the CB text does not completely fill any 2708, the remainder of the PROM will be filled with data 00H, and thus the unused area is **not** available for other data or CB text.

To determine the CB program text length, first clear the text area with the NEW command, then execute the CB program shown below which evaluates and outputs the SIZE function value. The SIZE function evaluates to the number of bytes allocated to, but left unused by the program text.

```
>1 PRINT SIZE
>2 STOP
>RUN
7142
```

The output (7,142 decimal in this example) gives the size of the unfilled CB text buffer. This number should be recorded for later reference. The size of the unfilled

Figure 11: CONTROL BASIC MEMORY MAP



DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

* Del CROMEMCO, TU-ART Digital Interface, Instruction Manual

** Del CROMEMCO, SCC Single Card Computer, Instruction Manual

*** Del CROMEMCO, 3K Control Basic, Instruction Manual

**** Del CROMEMCO, BYTESAVER II, Instruction Manual

***** Del 8080 MICROCOMPUTER SYSTEMS MANUAL

Julio, 1981

Introduction

The Cromemco TU-ART (Twin Universal Asynchronous Receiver and Transmitter) provides two channels of duplex serial data exchange; two channels of parallel data exchange; and ten interval timers. Status information is available through polling or by interrupt. In addition, each interval timer activates an interrupt and two interrupt request lines are brought out for the user. The TU-ART has its own crystal-controlled clock and interfaces to the S-100 bus asynchronously so that CPU clock frequency is not critical. The TU-ART incorporates two TMS 5501 NMOS I/O Controller chips.

1.1 Definitions

Throughout this manual the two TMS 5501 chips will be referred to as "Device A" and "Device B." Device A (IC 4) is the leftmost chip. Device B (IC 5) is the rightmost chip. Device A is nearer the heat sink and drives serial connector J4 and parallel connector J2. Device B is located to the right of Device A and drives serial connector J5 and parallel connector J3.

1.2 Switch Selectable Options Addressing The TU-ART

The system CPU views the TU-ART as a dual assembly of input/output ports with interrupt capability. The CPU normally reads data or status from the TU-ART via the S-100 bus by executing an IN A, (port) instruction, and writes data or commands to the TU-ART by executing an OUT (port), A instruction.

There are fourteen I/O ports used for data transfers, commands and status by Device A, and another fourteen by Device B (see Figure 2). The user may independently switch select Device A and Device B I/O Base Addresses (the four most significant I/O address bits); the four least significant bits of the I/O address on the S-100 bus then determine the offset from the selected base address.

The base address of Device A is selected by DIP switch positions 6 thru 3; the base address of Device B is selected by DIP switch positions 10 thru 7 (see Figure 1). Notice that positioning a switch ON conditions the TU-ART to respond to a logic 0 on its associated address line; an OFF switch corresponds to logic 1.

For example, if DIP switch positions 6 thru 3 are ON, and positions 10 thru 7 are OFF, then the TU-ART Device A Command Register is mapped into output port 02H, and the Device B Command Register is mapped into output port 0F2H.

Note that Device A bits A7, A6 and A5 also control D7, D6 and D5 of the TU-ART's Z-80 mode 2 Interrupt Acknowledge response vector.

Interrupt Mode

When this switch (position 1) is ON, the TU-ART operates in the 8080 interrupt mode: one of eight "Restart" instructions is gated to the data bus during an Interrupt Acknowledge cycle. Since the TU-ART can interrupt from one of 16 different sources, it is necessary to poll the devices if the TU-ART is in 8080 mode (see "Operation Using 8080 Mode Interrupts").

When switch position 1 is OFF, the TU-ART responds in Z-80 mode 2. In this mode, the TU-ART supplies a byte to the data bus during Interrupt Acknowledge that is used as the lower eight bits of a memory address. The Z-80 supplies the upper eight bits from the I register and automatically reads the corresponding memory location, as well as the next location, to find the starting location of an interrupt routine. (Refer to Section 3.1 and/or the Z-80 CPU Reference Manual, Zilog, 1977, for details.)

Normal/Reverse Address

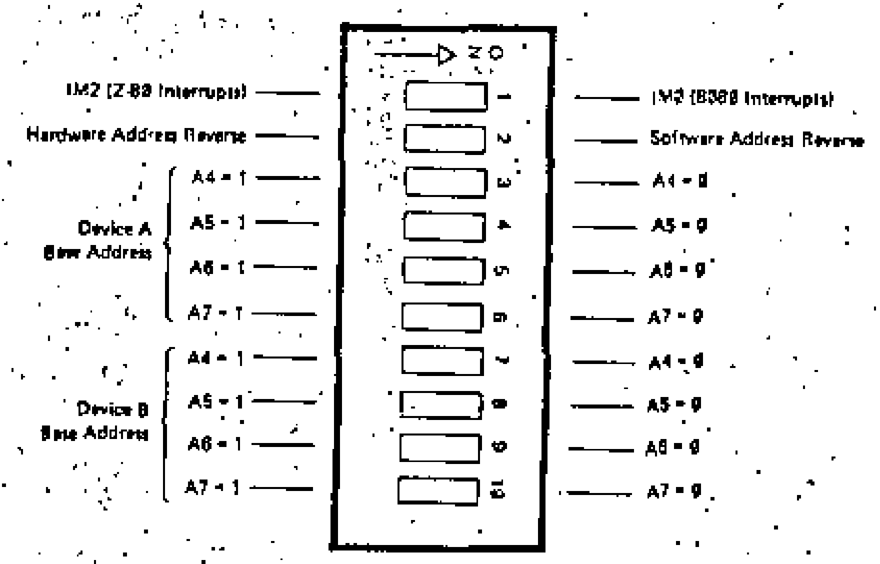
When this switch (position 2) is ON, it allows Device A and Device B to swap base addresses by means of an output to one of the parallel ports (Software Address-Reverse). This allows either Device A or Device B to be driven by a software driver whose port assignments are frozen in memory. Setting the switch ON connects the MSB of Device A's parallel output port to the Reverse Address control so that addresses may be flipped under software control. To flip addresses, output a byte with D7 high to Device A's parallel output port. To return to normal addressing, output a byte with D7 low to Device B's parallel output port. When switch position 2 is OFF, the Address Reverse switch is disconnected from the parallel port.

The Address Reverse signal is brought out to pin 1 of J2 and J3. When the Address Reverse switch is ON, pin 1 will show the state of the TU-ART:

Pin 1 = 0 means Reverse Mode.
Pin 1 = 1 means Normal Mode.

When the Address Reverse switch is OFF, pin 1 of J2 or J3 may be grounded externally to place the TU-ART in reverse Mode (Hardware Address-Reverse). Do not ground pin 1 of J2 or J3 while the Reverse Address switch is ON as this will conflict with operation of Device A's parallel port.

Figure 1 TU-ART Switch Settings



Caution: Base addresses 00H and 30H are used by Cromemco's 4FDC Floppy Disc Controller; 40H is used by Cromemco memory boards with BANX SELECT; and 50H is used by Cromemco's PRI Printer Interface.

1.3 Interrupt Priority Chain

When more than one TU-ART is used in a system, it is necessary to coordinate the Interrupt Responses in order to prevent bus conflict during Interrupt Acknowledge cycles. This is done by first connecting J1 PRIORITY $\overline{\text{OUT}}$ from the highest priority TU-ART to J1 PRIORITY $\overline{\text{IN}}$ of the next highest priority TU-ART, then connecting J1 PRIORITY $\overline{\text{OUT}}$ of the second TU-ART to J1 PRIORITY $\overline{\text{IN}}$ of the next TU-ART, and so on until all TU-ARTs are connected. The J1 PRIORITY $\overline{\text{IN}}$ pin of the highest priority board is left unconnected. Device A is internally prioritized Device B on each TU-ART.

1.4 Status Bit Selection

The connection of status flag bits to data bits is done on the PC board at the location of the status socket below J4. Cromemco software conventions assign D6=Receiver Data Available (RDA), and D7=Transmitter Buffer Empty (TBE). For specialized assignments (like more than one bit per flag) see the following "Status Socket" section.

Status Socket

The status flag bits available on input port 0 are connected to the data bits by foil traces in the "status" socket located between IC's 8 and 9.

The flag assignment used by all Cromemco soft- is discussed in the section entitled "Register Description."

If necessary, the flags may be assigned to different data bits. This may be most easily done as follows.

1. Notice that the flags are arranged along the left row of pads and that the data bits are arranged along the right side row of pads. Note also that only those 8 traces connecting the right and left pads are not covered by the solder mask. There are 5 traces which pass through this area which are covered.

2. Use a razor blade or a sharp knife to cut all 8 of the traces connecting the left and right rows of pads. Be very careful not to cut the traces which are covered by the solder mask.

3. Install and solder a 16 pin IC socket in the 2 rows of pads.

4. Install a 16 pin "component header" in the socket.

5. Using small (24 or 26 Awg) insulated wire connect the flags (on the left) to the desired data bits (on the right) on the component header.

6. The component header is now a "plug" for your particular flag assignment. Several different flag assignment "plugs" can be prepared in the same manner and used at different times to suit the requirements of the software being executed.

Any given flag may be assigned to more than one data bit. However, each data bit can have only one flag assigned to it.

1.5 Interface Options

TTY 20 mA

To drive a Teletype, the following connections should be made (at J4 or J5 for Device A or B respectively):

TU-ART	ASR-33 TTY
J4/J5 PIN 23 connects to	Terminal strip "BL", terminal #7 (current into printer)
J4/J5 PIN 25 connects to	Terminal strip "BL", terminal #6 (return current from printer)
J4/J5 PIN 17 connects to	Terminal strip "BL", terminal #4 (current into keyboard)
J4/J5 PIN 24 connects to	Terminal strip "BL", terminal #3 (return current from keyboard)

Caution: 120 VAC is also present on terminal strip "BL" at terminals #1 and #2.

RS/232C

An RS232 terminal (such as a CRT) may be plugged into an interface cable directly out of J4 or J5. The TU-ART assumes the role of data set (computer) in this case. See Figure B: Terminal to TU-ART Cable for this connection.

Parallel I/O

The parallel port output drivers may be tri-stated by grounding pin 8 of the parallel port (J2, J3). A bidirectional bus may be implemented by simply wiring the input and output lines together and using pin 8 to control the direction of data flow. Pin 8 low implies data input to the TU-ART and pin 8 high implies data output from the TU-ART.

Figure 2 Summary Of TU-ART I/O Port Addresses

OFFSET	A7 A6 A5 A4	A3	A2	A1	AØ	FUNCTION
Ø	Device A Base Address (see Fig. 1)	Ø	Ø	Ø	Ø	IN Device A status register
Ø		Ø	Ø	Ø	Ø	OUT Device A baud rate register
1		Ø	Ø	Ø	1	IN Device A receiver data register
1		Ø	Ø	Ø	1	OUT Device A transmitter data register
2		Ø	Ø	1	Ø	OUT Device A command register
3		Ø	Ø	1	1	IN Device A interrupt address register
3		Ø	Ø	1	1	OUT Device A interrupt mask register
4		Ø	1	Ø	Ø	IN Device A parallel port
4		Ø	1	Ø	Ø	OUT Device A parallel port
5		Ø	1	Ø	1	OUT Device A timer 1
6		Ø	1	1	Ø	OUT Device A timer 2
7		Ø	1	1	1	OUT Device A timer 3
8		1	Ø	Ø	Ø	OUT Device A timer 4
9		1	Ø	Ø	1	OUT Device A timer 5
Ø	Device B Base Address (see Fig. 1)	Ø	Ø	Ø	Ø	IN Device B status register
Ø		Ø	Ø	Ø	Ø	OUT Device B baud rate register
1		Ø	Ø	Ø	1	IN Device B receiver data register
1		Ø	Ø	Ø	1	OUT Device B transmitter data register
2		Ø	Ø	1	Ø	OUT Device B command register
3		Ø	Ø	1	1	IN Device B interrupt address register
3		Ø	Ø	1	1	OUT Device B interrupt mask register
4		Ø	1	Ø	Ø	IN Device B parallel port
4		Ø	1	Ø	Ø	OUT Device B parallel port
5		Ø	1	Ø	1	OUT Device B timer 1
6		Ø	1	1	Ø	OUT Device B timer 2
7		Ø	1	1	1	OUT Device B timer 3
8		1	Ø	Ø	Ø	OUT Device B timer 4
9		1	Ø	Ø	1	OUT Device B timer 5

NOTES:

All of the following unassigned ports are free for system use: IN 2, IN 5 through IN 9, IN 1Ø through IN 15 and OUT 1Ø through OUT 15.

If Device A and Device B are set to the same base address, Device A will override.

Device A is IC 4.

Device B is IC 5.

Interrupt Operation

The TU-ART offers sophisticated interrupt capabilities, including on-board priority encoding, interrupt generation, interrupt acknowledgment, and daisy chain expandability. These features, in conjunction with the Cromemco 4 MHz Z-80 processor, make very high throughput possible.

IMPORTANT

Both channels of the TU-ART must be properly initialized. An uninitialized TU-ART may generate spurious interrupts! Further, the rest of the system must be interrupt compatible (all Cromemco boards are, although the 8K Bytesaver requires the interrupt modification shown on the 8K Bytesaver schematic).

A description of interrupt operation follows for both the Z-80 and 8080 type interrupt modes.

3.1 Operation Using Z80 Interrupts

When the TU-ART is used with the Cromemco ZPU, all 16 of the possible interrupt sources on the TU-ART can generate a unique response without the need for chaining the interrupt requests and polling the responses. This means fast response from interrupt request to service routine and, when coupled with the 4MHz Z-80, an extremely powerful realtime system can be implemented.

A "high priority" interrupt request is one which takes precedence over lower priority requests. This is shown in the following table where the interrupts serviced first are at the top.

It is, of course, possible to use the interrupt mask of each Device to selectively enable and disable the sources of interrupts (reference the description of OUT Ø3, Interrupt Mask, in the previous section). Remember that the INE bit in the status register must be high for correct operation of Interrupt Acknowledge cycles. Also, be sure that the Z-80 has executed the interrupt mode setting command ØED5EH

Table 1 Z80 (Mode 2) Response

Priority	TU-ART's (Hex) Z-80 INTA Response								Source of Interrupt
	D7	D6	D5	D4	D3	D2	D1	DØ	
15 (Highest)	Set By Device A Adr. A7	Set By Device A Adr. A6	Set By Device A Adr. A5	Ø	Ø	Ø	Ø	Ø	Device A, Timer 1
14				Ø	Ø	Ø	1	Ø	Device A, Timer 2
13				Ø	Ø	1	Ø	Ø	Device A, $\overline{\text{SENSA}}$
12				Ø	Ø	1	1	Ø	Device A, Timer 3
11				Ø	1	Ø	Ø	Ø	Device A, RDA
10				Ø	1	Ø	1	Ø	Device A, TBE
9				Ø	1	1	Ø	Ø	Device A, Timer 4
8				Ø	1	1	1	Ø	Device A, Timer 5 (PI7)
7				1	Ø	Ø	Ø	Ø	Device B, Timer 1
6				1	Ø	Ø	1	Ø	Device B, Timer 2
5				1	Ø	1	Ø	Ø	Device B, $\overline{\text{SENSB}}$
4				1	Ø	1	1	Ø	Device B, Timer 3
3				1	1	Ø	Ø	Ø	Device B, RDA
2				1	1	Ø	1	Ø	Device B, TBE
1				1	1	1	Ø	Ø	Device B, Timer 4
Ø (Lowest)				1	1	1	1	Ø	Device B, Timer 5 (PI7)

("IM2") and the interrupt enable command 0FBH ("EI"). Both of these instructions must be executed each time the Z-80 is RESET.

Assuming that both the Z-80 and the TU-ART have been initialized, the reception of a byte of serial data at Device B would initiate the following sequence of events:

a) The assembled byte is loaded into the receiver data buffer.

b) The RDA status bit is set and the interrupt request register (bit 3) is set.

c) If bit 3 of the interrupt mask of the Device in question is a one, the request passes on to the priority encoder. If bit 3 is a zero, no further action occurs until the mask is changed.

d) The priority encoder compares all incoming interrupt requests and sets its output to the value of the highest priority incoming interrupt. Thus, since Device B receives the serial data byte in our example; the priority encoder will set its output to "priority 3" if Timers 1, 2, 3, and $\overline{\text{SENSB}}$ from Device B are inactive or masked out.

e) Device B's INT pin goes high, which in turn pulls $\overline{\text{PINT}}$ low on the S-100 bus.

f) The Z-80 checks the interrupt line at the end of the current instruction, and finding the line active, goes into an Interrupt Acknowledge (INTA) cycle.

g) The occurrence of the INTA cycle is detected by the TU-ART which then transmits PRIORITY $\overline{\text{OUT}}$ = 0 to connector J1. This temporarily disables Interrupt Acknowledge from lower priority boards. If no board with higher priority is holding PRIORITY $\overline{\text{IN}}$ low, and if Device A has no interrupt pending, then Device B gates the proper Z-80 INTA response vector onto the data bus. In this example, Device B would place 18H logically ORed with (A7) (A6) (A5) 00000 from Device A's base address on the data bus. The corresponding bit in the interrupt request latch is reset.

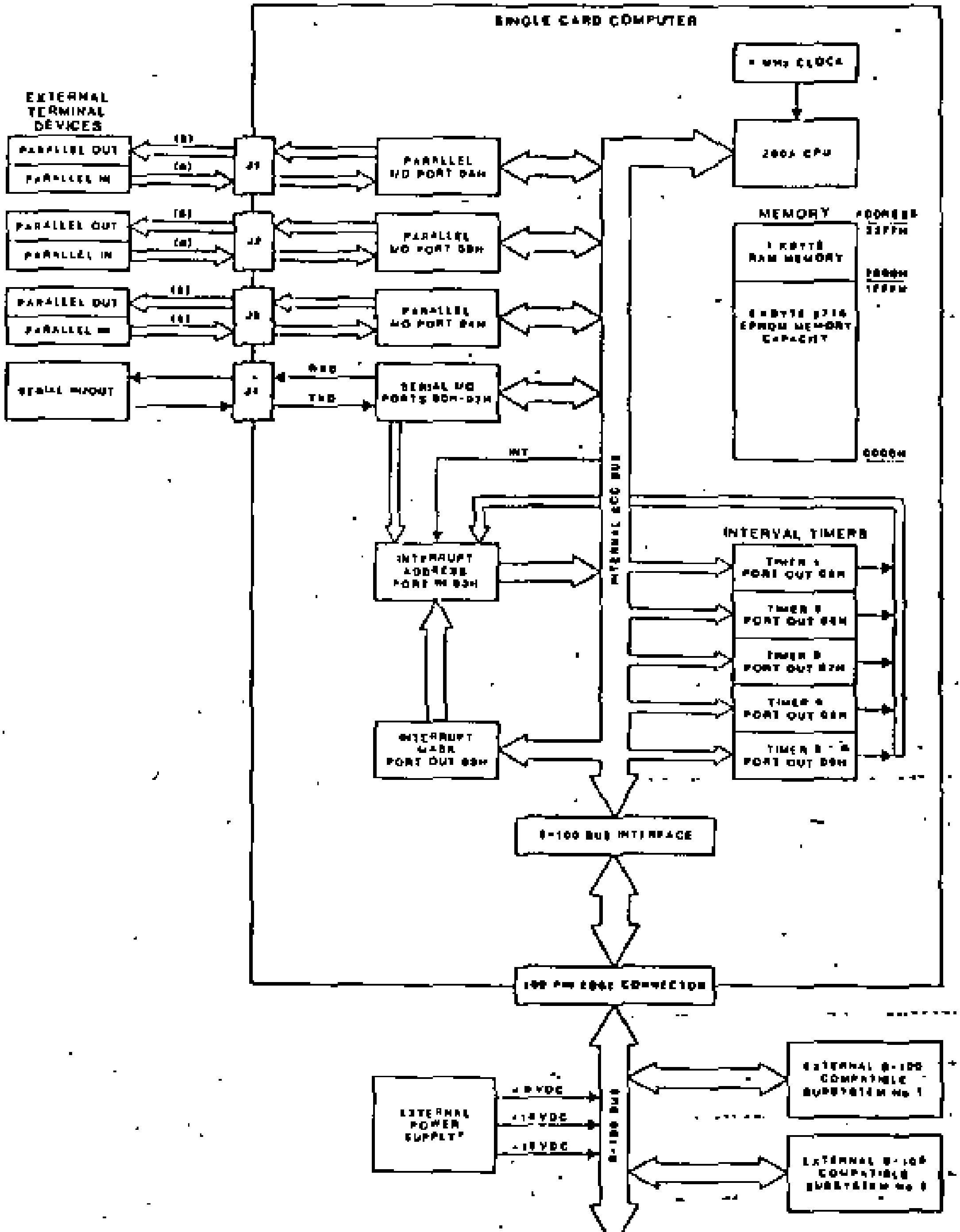
h) The Z-80 reads the INTA response byte and appends it to the byte in the I register. This then forms a sixteen bit address which points to the first of two sequential bytes in memory which in turn designate the actual starting address of the service routine. The CPU automatically executes a CALL to this starting address.

3.2 Operation Using 8080 Mode Interrupts

When the TU-ART is used in Z-80 interrupt mode 0 it is necessary to "chain" Device B through the SENS input on Device A. This requires one of the eight INTA responses, RST2 (D7H), to be serviced by a routine which polls the status and interrupt address registers of Device B. The remaining seven INTA responses are serviced immediately. The resulting priority assignments are shown in Table 2.

Table 2 Z80 (Mode 0) Response

Priority	TU-ART's (Hex) 8080 INTA Response	Source of Interrupt
15 (Highest)	C7 (RST0)	Device A, Timer 1
14	CF (RST1)	Device A, Timer 2
13	D7 (RST2)	Device B, Timer 1
12	D7 (RST2)	Device B, Timer 2
11	D7 (RST2)	Device B, $\overline{\text{SENSB}}$
10	D7 (RST2)	Device B, Timer 3
9	D7 (RST2)	Device B, RDA
8	D7 (RST2)	Device B, TBE
7	D7 (RST2)	Device B, Timer 4
6	D7 (RST2)	Device B, Timer 5 (PI7)
5	D7 (RST2)	Device B, $\overline{\text{SENSA}}$
4	DF (RST3)	Device A, Timer 3
3	E7 (RST4)	Device A, RDA
2	EF (RST5)	Device A, TBE
1	F7 (RST6)	Device A, Timer 4
0 (Lowest)	FF (RST7)	Device A, Timer 5 (PI7)



*REQUIRED

Figure 1: SCC FUNCTIONAL BLOCK DIAGRAM

SCC TECHNICAL SPECIFICATIONS

Processor Type:	Z80-A
Clock Frequency:	4.000 MHz
Instruction Set:	158 instructions including the 78 member 8080A CPU instruction set
ROM Capacity:	8 KBytes spanning address range 0000H - 1FFFH
ROM Type:	Intel 2716 EPROM, TI 2516 EPROM, or equivalent (user supplied)
RAM Capacity:	1 KBytes spanning address range 2000H - 23FFH
RAM Type:	4045 Static (included)
UART Type:	TMS 5501 or equivalent
Serial I/O Port:	RS-232 or 20 mA current loop; software selectable from 110 to 76,800 Baud, one or two stop bits
Parallel Ports:	24 lines of input data organized as three parallel 8-bit ports; 24 lines of output data organized as three parallel 8-bit ports
Interval Timers:	5 independent timers with range 0 - 16.32 mSec and 64 uSec resolution
Interrupts:	All three Z80 interrupt modes supported (IM 0, IM 1 & IM 2); prioritized IM 0 RST vectors may be automatically supplied by eight SCC interrupt sources
Bus Compatibility:	Cromemco Standard-100 (S-100)
Power Requirements:	+8 VDC @ 1.75 Amps (max) +18 VDC @ 100 mA (max) -18 VDC @ 50 mA (max)
Operating Environment:	0 - 55 degrees Celsius

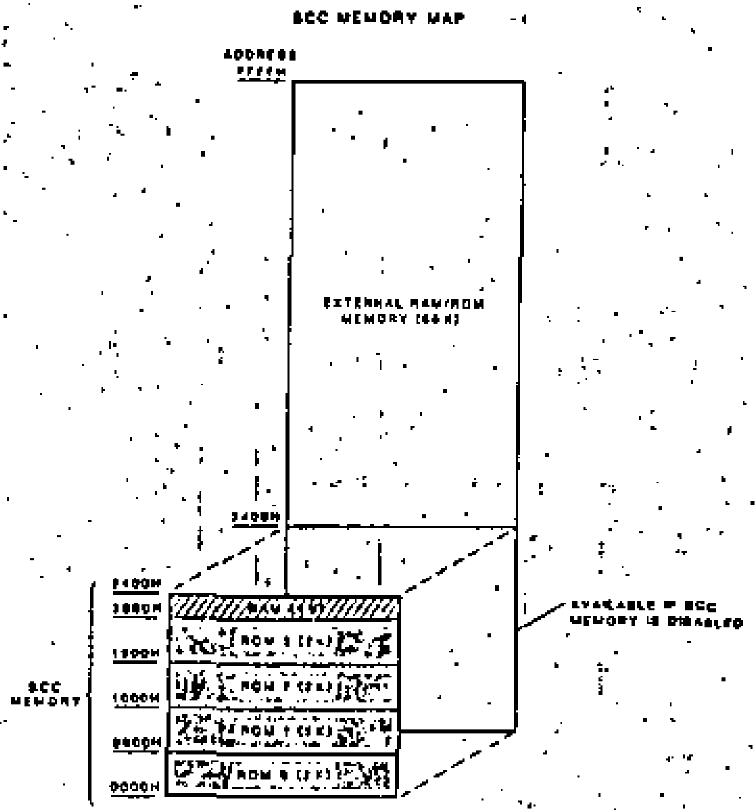


Figure 3: SCC MEMORY MAP

Since the 2716 has a memory access time of 450 nSec while the CPU cycle time is 250 nSec, the SCC automatically inserts one wait state during all memory request machine cycles to the lowest 8 KBytes of memory (0000H - 1FFFH) where 2716 EPROM memory resides. Note that these wait states will not be inserted if the SCC memory

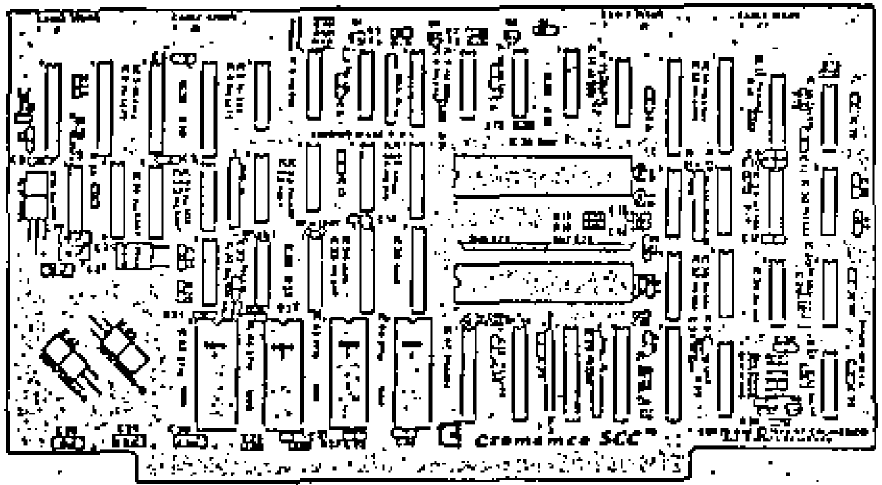


Figure 4: M1 WAITS JUMPER OPTION

The SCC read/write memory physically consists of two 4045 1 K by 4-bit static RAMs occupying sockets IC34 and IC35. Since the 4045 memory access time is 250 nSec (max), no wait states are required for these devices. IC35 contains the high-order nybble of each RAM byte, and IC34 contains all low-order nybbles. When power is first applied to the SCC, the RAM contents should be assumed random.

2.4 MEMORY EXPANSION/SCC MEMORY DISABLE

The amount of memory available to the SCC may be expanded by merely plugging additional RAM or ROM memory cards into the SCC controlled S-100 bus. The additional S-100 bus compatible memory

may overlap, be disjoint with, or be contiguous with the SCC on-board memory. It is the external memory's responsibility to insert wait states by controlling S-100 bus line RDY, if necessary, for 4 MHz operation.

Installing external memory which overlaps the SCC 0000H - 23FFH memory area gives rise to the situation illustrated in Figure 5. For memory reads, the SCC reads data only from its on-board memory in the address range 0000H - 23FFH, and from the external S-100 bus in the address range 2400H - FFFFH; any external memory residing in the 0000H - 23FFH area is simply ignored. For memory writes, the SCC writes data to its on-board memory and to the external S-100 bus in parallel, although writing to the SCC ROM area 0000H - 1FFFH does not alter the 2716 EPROM contents. Stated another way, the SCC 'listens' to only its own memory when the memory address is in the range 0000H - 23FFH, and on-board RAM memory is 'transparent' to all memory write operations (empty on-board ROM sockets are also transparent to memory writes).

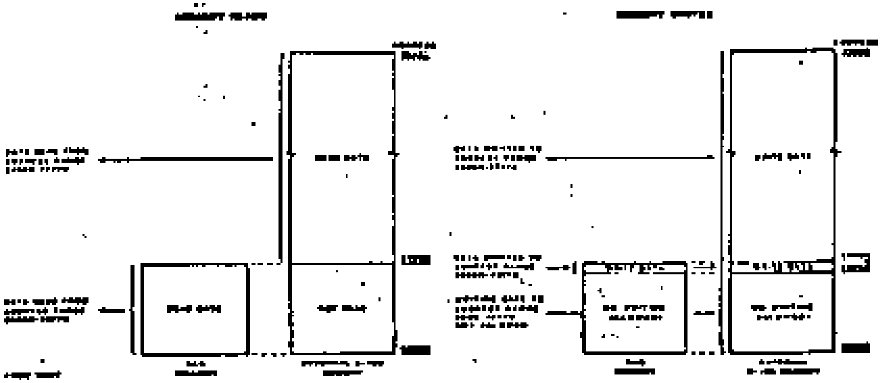


Figure 5: MEMORY OVERLAPPING THE SCC

Additional memory may be added up to the Z80A's 64 KByte direct addressing limit, and by incorporating Cromemco's memory products with <u>Bank Select</u>, memory may be expanded further to 512 KBytes organized as eight 'banks' of 64 KBytes each. Cromemco memory boards with Bank Select (RAM Boards: 4KZ, 16KZ, 64KZ; EPROM Boards: Bytesaver II, 16KPR, 32K Bytesaver) are mapped into any combination of Memory Banks (Bank 0 thru Bank 7) by setting an eight position slide switch on the card. Banks are enabled and disabled under software control by outputting a bank select control byte to port OUT 40H, an integral output port on each Cromemco product with Bank Select. If a logic 1 bit (D0 thru D7)

in the bank select control byte matches any Memory Bank the board is mapped into, the memory board becomes active and is th switched 'into' the memory map. Conversely, if there is no match between any bank activated by the control byte and the banks the board is mapped into, the board goes inactive, and thus is switched 'out of' the memory map. Finally note in connection with the Bank Select feature that SCC on-board memory is <u>not</u> nk Select controlled. An example will illustrate the foregoing ideas.

<u>EXAMPLE 1</u>

Suppose a 16KZ RAM board, with the switch settings shown in Figure 6A, is co-resident with an SCC in an S-100 bus. The switch settings map the 16KZ into the address range 0000H - 3FFFH (the lowest 16 KBytes of memory) in Memory Banks 1 and 2.

(a) When power is first applied, SCC memory unconditionally enables. The 16KZ responds to a POC by enabling only if it is mapped into Bank 0, and disabling otherwise. Thus, the 16KZ is switched out of the memory map in response to a POC. The system memory map immediately following a POC is shown in Figure 6b.

(b) Now suppose the following two instructions, residing in C ROM 0, are executed:

```
ADDR    OBJECT     MNEMONIC              COMMENT

0000     3E82      LD    A,10000010B     ;ENABLE BANK 1 & BANK 7
0002     D340      OUT   40H,A           ;OUTPUT CONTROL BYTE
```

The 16KZ is mapped into one of the banks activated by t bank select control byte (Bank 1), thus it is switched into the memory map (Figure 6C).

(c) Finally assume the following two instructions, also residi in SCC ROM 0, are executed:

```
ADDR   OBJECT      MNEMONIC              COMMENT

0004   3E01        LD    A,00000001B     ;ENABLE ONLY BANK 0
0006   D340        OUT   40H,A           ;OUTPUT CONTROL BYTE
```

Since the 16KZ is not mapped into Bank 0, it is switched

of the memory map in response to the control byte, and the memory map reverts back to that shown in Figure 6b.

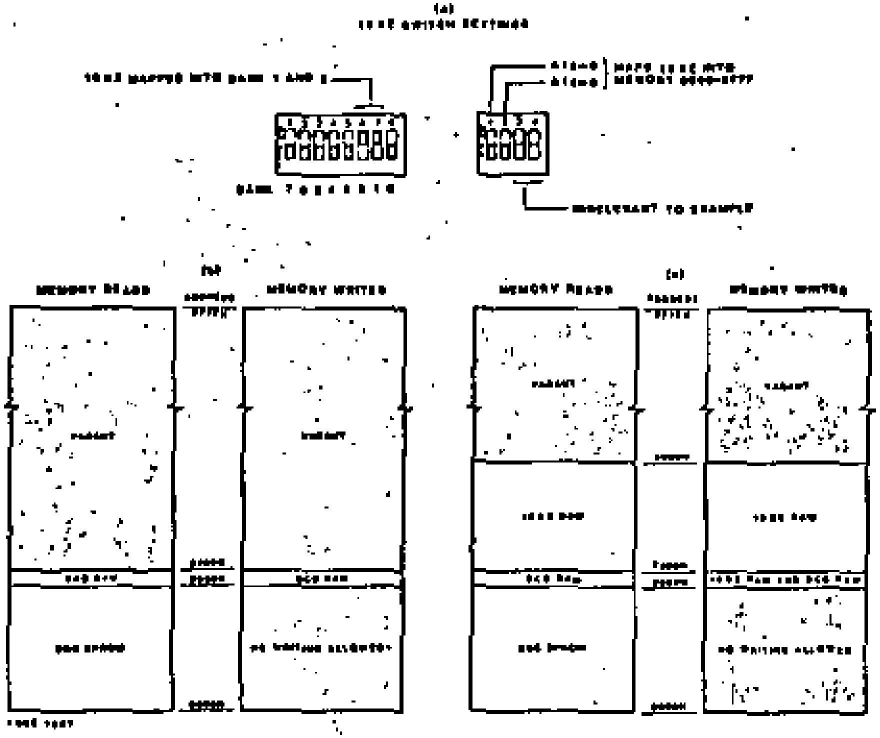


Figure 6: EXAMPLE 1 SWITCH SETTING & MEMORY MAPS

Although the SCC on-board memory does not respond to Bank Select, a functionally equivalent software controlled memory enable/disable option is provided. The Memory Disable option is exercised by cutting the existing foil trace between the two solder pads labeled 'DISABLE' on the SCC board legend (see Figure 7). With the trace cut, high order bit D7 of SCC parallel port

OUT 0AH controls the SCC memory enable/disable function. Setting bit D7 to logic 1 disables SCC memory; resetting bit D7 to logic 0 enables SCC memory. Either a POC or a RESET unconditionally enables the on-board SCC memory, since either a POC or an active low level on S-100 bus line $\overline{\text{RESET}}$ resets all parallel port OUT 0AH bits to logic 0. If the Memory Disable trace is left in its factory shipped closed condition, or if it is later re-installed, bit D7 of port OUT 0AH has no effect on SCC memory enable/disable.

EXAMPLE 2

Assume the same 16KZ RAM and SCC configuration as in Example 1, and also assume the SCC Memory Disable option is selected by cutting the foil trace.

(a) After power is first applied, the 16KZ automatically disables since it is not mapped into Bank 0, and the SCC on-board memory unconditionally enables since all bits of port OUT 0AH are reset to logic 0. The resulting memory map is shown in Figure 7A. Note that an S-100 RESET after power is applied produces the same map.

(b) Now suppose the following four instructions, residing in SCC ROM 0, are executed:

```
ADDR  OBJECT    MNEMONIC                    COMMENT

0000  3E82      LD    A,10000010B          ;ENABLE BANK 7 & BANK 2
0002  D340      OUT   40H,A                ;OUTPUT CONTROL BYTE
0004  3E80      LD    A,10000000B          ;SET BIT D7 TO LOGIC 1
0006  D30A      OUT   0AH,A                ;DISABLE SCC MEMORY
```

The first two instructions activate the 16KZ since it is assumed switch-mapped into Bank 2, and the next two instructions disable all SCC memory by setting bit D7 of Port OUT 0AH to logic 1 (Figure 7B). Note that after the four instructions above are executed, the Z80A will fetch the next opcode from 16 KZ RAM address 0008H (the next sequential address). Thus provisions must be made to maintain program continuity when bank switching in this fashion.

(c) Finally suppose the following two instructions, residing in SCC ROM 0, are executed:

```
ADDR  OBJECT    MNEMONIC                      COMMENT

0008  3E00      LD    A,00000000B             ;RESET BIT D7 TO LOGIC 0
000A  D30A      OUT   0AH,A                   ;ENABLE SCC MEMORY
```

Since no Bank Select control word is output, the 16KZ remains in the map. The two instructions above also enable SCC memory (Figure 7c).

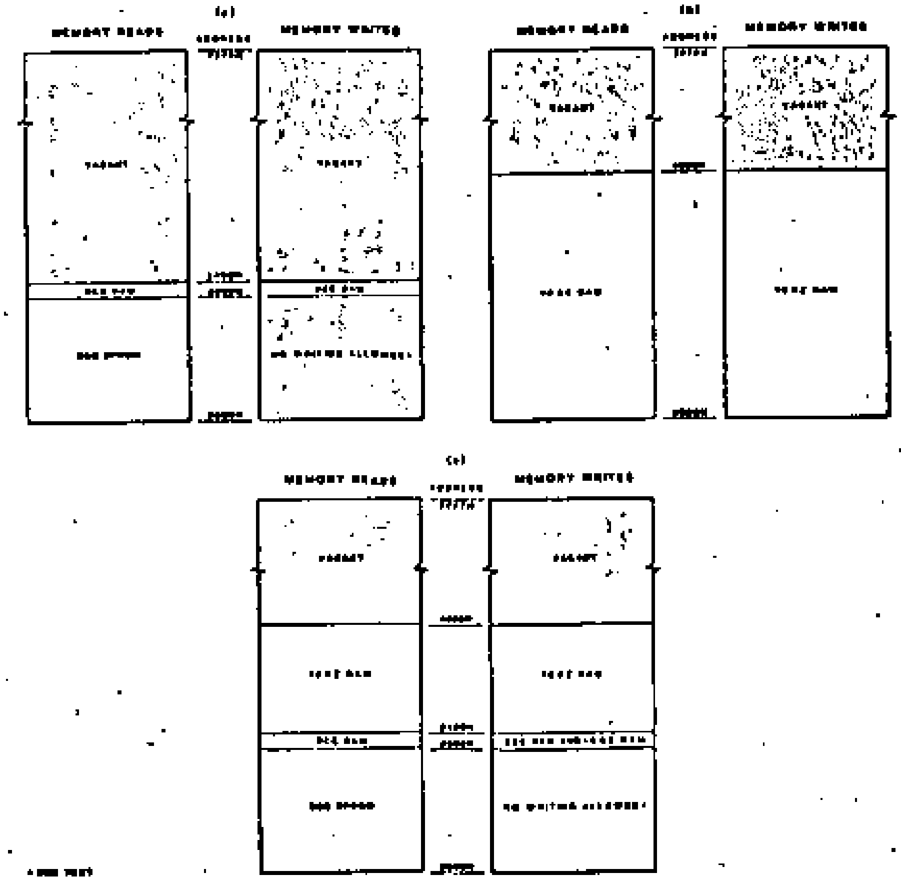


Figure 7: EXAMPLE 2 MEMORY MAPS

CHAPTER 8

Supplementary Information

8.1 Control Basic model CB-308

8.1.1 Memory Allocation (CB-308)

The listing which follows is a detailed outline of memory allocation for Control Basic model CB-308. The table on page 1 maps an entire 64K system. Of this, the minimum required is RAM from 180H to 1FFFH and the Control Basic Interpreter itself in ROM from E400H to EFFFH. The rest of memory (may be either ROM or RAM) except the unused portion from 0 to 17FH may be used to store program files and routines. Page 2 of the listing is a detailed breakdown of the RAM from 200H to 3FDH, which contains Control Basic variables, stack, input buffer, and console parameters.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                       PAGE 0001
3K Control Basic Variables and Memory Allocation

              0003 ;
              0004 ;      Copyright (c) 1978, 1979 Cromemco, Inc.
              0005 ;
              0006 ;
              0007 ;  *** Memory Usage (probably RAM unless otherwise noted) ***
              0008 ;
              0009 ;  0000-017F are not used
              0010 ;  0180-01FF are for the I/O routines
              0011 ;  0200-03FF are for variables, input line, and stack
              0012 ;  0400-1FFF are for Control Basic text & array @()
              0013 ;  2000-DFFF are not used but may be used for saved files
              0014 ;  E000-E3FF are for system monitor and not used here (ROM)
              0015 ;  E400-EFFF are for Control Basic Interpreter (ROM)
              0016 ;  F000-FFFF are not used but may be used for saved files


(0180)        0020 IORAM   EQU    180H            ; START OF I/O ROUTINES IN RAM
(0200)        0021 BOTRAM  EQU    200H            ; "BOTTOM OF RAM" FOR VARIABLES, ETC.
(0400)        0022 TXTRAM  EQU    400H            ; BOTTOM OF "TEXT RAM"
(2000)        0023 TOPRAM  EQU    2000H           ; "TOP OF RAM" USED BY CONTROL BASIC
(E400)        0024 BOTROM  EQU    0E400H          ; "BOTTOM OF ROM" CONTAINING CB
(F000)        0025 TOPROM  EQU    BOTROM+0C00H    ; FIRST FREE LOCATION PAST "TOP OF
              0026                                ;   ROM" (USED BY RND FUNCTION)
(E000)        0027 MONITR  EQU    0E000H          ; MONITOR WARM-START POINT
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                       PAGE 0002
3K Control Basic Variables and Memory Allocation

0000           0030          ORG   BOTRAM
0200  (0068)   0031 VARBGN:  DEFS  2*(26+26)      ; VARIABLE A-Z AND A0-Z0 STORAGE
0268  (0001)   0032          DEFS  1              ; EXTRA BYTE FOR INPUT BUFFER
0269  (0085)   0033 BUFFER:  DEFS  133            ; STORAGE FOR INPUT LINE BUFFER
      (02EE)   0034 BUFEND   EQU   $              ; END OF INPUT BUFFER
02EE  (0028)   0035          DEFS  40             ; EXTRA BYTES FOR STACK
0316  (00C8)   0036 STKLMT:  DEFS  200            ; SOFT LIMIT FOR STACK
      (03DE)   0037 STACK    EQU   $              ; TOP OF STACK
03DE  (0002)   0038 IOINS:   DEFS  2              ; STORAGE FOR AN I/O INSTRUCTION
03E0  (0001)   0039 IORET:   DEFS  1              ; STORAGE FOR THE RETURN INSTRUCT
03E1  (0001)   0040 NULLS:   DEFS  1              ; NUMBER OF NULLS AFTER CR-LF
03E2  (0001)   0041 MARGN:   DEFS  1              ; SOFT TERMINAL WIDTH
03E3  (0001)   0042          DEFS  1              ; HARD TERMINAL WIDTH
03E4  (0001)   0043 OCNG:    DEFS  1              ; COLUMN COUNTER FOR MARGINS
03E5  (0001)   0044 OCSW:    DEFS  1              ; SWITCH FOR VISIBLE OUTPUT
03E6  (0002)   0045 CURRNT:  DEFS  2              ; CURRENT LINE NUMBER STORAGE
03E8  (0002)   0046 STKRUN:  DEFS  2              ; SAVES STACK POINTER DURING 'RUN'
03EA  (0002)   0047 STKGOS:  DEFS  2              ; SAVES STACK POINTER DURING 'GOSUB'
      (03EC)   0048 VARNXT   EQU   $              ; TEMP. STOR. OF CURRENT 'NEXT' VAR.
      (03EC)   0049 CALDE    EQU   $              ; TEMP. STOR. FOR ADDR. FOR 'CALL'
03EC  (0002)   0050 STKINP:  DEFS  2              ; SAVES STACK POINTER DURING 'INPUT'
03EE  (0002)   0051 LOPVAR:  DEFS  2              ; ADDRESS OF 'FOR' LOOP VARIABLE
03F0  (0002)   0052 LOPINC:  DEFS  2              ; 'STEP' SIZE FOR 'FOR' LOOP
03F2  (0002)   0053 LOPLMT:  DEFS  2              ; LIMIT OF 'FOR' LOOP VARIABLE
03F4  (0002)   0054 LOPLN:   DEFS  2              ; TEMP. STOR. OF 'CURRNT' IN 'FOR'
03F6  (0002)   0055 LOPPT:   DEFS  2              ; TEXT POINTER TEMP. STOR. IN 'FOR'
03F8  (0002)   0056 RANPNT:  DEFS  2              ; ADDR. POINTER USED IN RND FUNC.
03FA  (0002)   0057 TXTBGN:  DEFS  2              ; STORAGE OF TEXT POINTER ADDR.
03FC  (0002)   0058 TXTLMT:  DEFS  2              ; TEXT AREA LIMIT (SAME AS 'T
               0059                               ;   UNLESS CHANGED BY 'LOCK')


03FE           0063          ORG   TXTRAM
0400  (0002)   0064 TXTUNF:  DEFS  2              ; POINTER TO UNFILLED TEXT
0402  (0002)   0065 TEXT:    DEFS  2              ; START OF ACTUAL TEXT (STORAGE
               0066                               ;   OF FIRST LINE NUMBER)
```

Also shown on page 2 is the beginning of the text area for the current program. The text area is organized as follows; any stored program (by means of the SAVE or EPROM commands) is organized this same way:

1. The first two bytes of the first page point to the first available byte after the program text relative to 400H.

2. The next two bytes contain the line number of the first line in binary (low byte-high byte). This is followed by the text of the first line and a single 0DH, or CR, at the end of that line.

3. Subsequent lines of text duplicate the form of this first line: 2-byte binary line number, ASCII text, CR-byte.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                     PAGE 0001
3K Control Basic I/O Routines in RAM

0000'              0002           ORG     180H

     (0000)        0004 CSTAT     EQU     0              ; CONSOLE STATUS PORT (IN)
     (0001)        0005 CDATA     EQU     CSTAT+1        ; CONSOLE DATA PORT (IN/OUT)
     (0080)        0006 TBE       EQU     80H            ; TRANSMITTER-BUFFER-EMPTY MASK
     (0040)        0007 RDA       EQU     40H            ; RECEIVER-DATA-AVAILABLE MASK
     (007F)        0008 PSTRIP    EQU     7FH            ; PARITY-STRIP MASK

     (0003)        0010 CTRLC     EQU     03H            ; ASCII CONTROL-C CHARACTER
     (000A)        0011 LF        EQU     0AH            ; ASCII LINE FEED
     (000D)        0012 CR        EQU     0DH            ; ASCII CARRIAGE RETURN
     (000F)        0013 CTRLO     EQU     0FH            ; ASCII CONTROL-O CHARACTER
     (0020)        0014 SPC       EQU     20H            ; ASCII SPACE CHARACTER

     (03E1)        0016 NULLS     EQU     3E1H           ; NO. OF NULLS AFTER CR-LF (1 BYTE)
     (03E2)        0017 MARGN     EQU     3E2H           ; SOFT TERMINAL WIDTH (1 BYTE)
                   0018                                  ; HARD TERMINAL WIDTH (1 BYTE)
     (03E4)        0019 OCNG      EQU     3E4H           ; COLUMN COUNTER FOR MARGINS (1 BYTE)
     (03E5)        0020 OCSW      EQU     3E5H           ; SWITCH FOR VISIBLE OUTPUT (1 BYTE)
                   0021                                  ; (THIS IS CHANGED BY TYPING CTRL-O)
     (E43F)        0022 RSTART    EQU     0E43FH         ; CONTROL BASIC WARM-START LOCATION

180  3E0D          0025 CRLF:     LD      A,CR           ; *** CR-LF ROUTINE ***
                   0026 ;
182  F5            0027 OUTCHR:   PUSH    AF             ; *** OUTPUT CHARACTER ROUTINE ***
183  3AE503        0028           LD      A,(OCSW)       ; CHECK VISIBLE/INVISIBLE SWITCH
186  B7            0029           OR      A              ; IF BIT-7 IS SET, OUTPUT CHARACTER
187  F2D401        0030           JP      P,ENDOUT       ; IF NOT SET, DON'T OUTPUT CHARACTER
18A  DB00          0031 OUTC10:   IN      A,CSTAT        ; CHECK CONSOLE STATUS
18C  E680          0032           AND     TBE            ; GET TRANSMITTER-BUFFER BIT
18E  CA8A01        0033           JP      Z,OUTC10       ; LOOP TO WAIT IF NOT READY
191  F1            0034           POP     AF             ; RETRIEVE CHARACTER
192  E5            0035           PUSH    HL             ; SAVE REGISTERS
193  F5            0036           PUSH    AF             ; /
194  E67F          0037           AND     PSTRIP         ; STRIP PARITY BIT
196  D301          0038           OUT     CDATA,A        ; OUTPUT CHARACTER TO CONSOLE
198  FE0D          0039           CP      CR             ; CHECK FOR CARRIAGE RETURN
19A  21E403        0040           LD      HL,OCNG        ; POINT TO COLUMN-COUNTER
19D  C2A201        0041           JP      NZ,OUTC20      ; SKIP IF CHARACTER IS NOT CR
1A0  3601          0042           LD      (HL),1         ; INIT. COLUMN-COUNTER IF NEW LINE
1A2  FE20          0043 OUTC20:   CP      SPC            ; CHECK FOR PRINTABLE CHARACTER
1A4  DABB01        0044           JP      C,OUTC40       ; SKIP IF NOT PRINTABLE
1A7  7E            0045           LD      A,(HL)         ; GET COLUMN-COUNTER
1A8  34            0046           INC     (HL)           ;   AND INCREMENT IT FOR PRINTABLES
1A9  2AE203        0047           LD      HL,(MARGN)     ; H GETS HARD AND L GETS SOFT MARGIN
1AC  BC            0048           CP      H              ; CHECK FOR HAVING REACHED HARD MARGIN
1AD  CAB801        0049           JP      Z,OUTC30       ; FORCE OUTPUT CR-LF IF SO
1B0  BD            0050           CP      L              ; CHECK FOR HAVING REACHED SOFT MARGIN
1B1  FABB01        0051           JP      M,OUTC40       ; SKIP IF NOT YET REACHED
1B4  F1            0052           POP     AF             ; GET CHARACTER AGAIN
1B5  F5            0053           PUSH    AF             ;   AND PUT BACK
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                     PAGE 0002
3K Control Basic I/O Routines in RAM

01B6  FE20         0054           CP      SPC            ; CHECK FOR SPACE CHARACTER (DO CR-LF
                   0055                                  ;   AT FIRST SPACE AFTER SOFT [illegible])
01B8  CC8001       0056 OUTC30:   CALL    Z,CRLF         ; OUTPUT CR-LF IF FOUND
01BB  F1           0057 OUTC40:   POP     AF             ; RESTORE REGISTERS
01BC  E1           0058           POP     HL             ; /
01BD  FE0D         0059           CP      CR             ; CHECK FOR CARRIAGE RETURN
01BF  C0           0060           RET     NZ             ; RETURN TO CONTROL BASIC IF NOT CR
                   0061 ;
01C0  F5           0062           PUSH    AF             ; SAVE AF-REG.
01C1  3E0A         0063           LD      A,LF           ; IF CR, SEND LINE FEED ALSO
01C3  CD8201       0064 OUTNUL:   CALL    OUTCHR         ; (THIS IS RECURSIVE)
01C6  3AE103       0065           LD      A,(NULLS)      ; GET NUMBER OF NULLS
01C9  C5           0066           PUSH    BC             ; SAVE BC-REG.
01CA  47           0067           LD      B,A            ; PUT NULL-COUNT INTO B-REG.
01CB  97           0068 OUTC50:   SUB     A              ; CLEAR A-REG. TO A NULL
01CC  05           0069           DEC     B              ; DECREMENT NULL-COUNT
01CD  F48201       0070           CALL    P,OUTCHR       ; OUTPUT NULL IF COUNT NOT= 0
01D0  DACB01       0071           JP      C,OUTC50       ; C-FLAG SET BY OUTCHR IF LAST
                   0072                                  ;   CHARACTER WAS NULL
01D3  C1           0073           POP     BC             ; RESTORE REGISTERS
01D4  F1           0074 ENDOUT:   POP     AF             ; /
01D5  C9           0075           RET                    ; RETURN TO CONTROL BASIC


01D6  DB00         0079 CHKINP:   IN      A,CSTAT        ; *** CHECK FOR INPUT ROUTINE ***
01D8  E640         0080           AND     RDA            ; GET RECEIVER-DATA BIT
01DA  C8           0081           RET     Z              ; RETURN TO CONTROL BASIC IF NOT READY
01DB  DB01         0082           IN      A,CDATA        ; READ CHARACTER FROM CONSOLE
01DD  E67F         0083           AND     PSTRIP         ; STRIP PARITY BIT
01DF  FE0F         0084           CP      CTRLO          ; CHECK FOR CONTROL-O CHARACTER
01E1  C2EE01       0085           JP      NZ,CHKC10      ; IF NOT CTRL-O, DO FURTHER CHECKING
01E4  3AE503       0086           LD      A,(OCSW)       ; CONTROL-O TOGGLES VIS./INVIS. SWITCH
01E7  2F           0087           CPL                    ; TOGGLE BIT-7
01E8  32E503       0088           LD      (OCSW),A       ; STORE VISIBLE/INVISIBLE SWITCH BACK
01EB  C3D601       0089           JP      CHKINP         ; INPUT ANOTHER CHARACTER
                   0090 ;
01EE  FE03         0091 CHKC10:   CP      CTRLC          ; CHECK FOR CONTROL-C CHARACTER
01F0  C0           0092           RET     NZ             ; RETURN TO CONTROL BASIC IF NOT CTRL-C
01F1  C33FE4       0093           JP      RSTART         ; IF CTRL-C, ABORT PROGRAM AND WARM-
                   0094                                  ;   START CONTROL BASIC

      (01F4)       0096 FIRSTFREE EQU     $              ; FIRST FREE BYTE OF MEMORY

Errors             0
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                        PAGE 0003
SYMBOL TABLE

CDATA   0001    CHKI1D  01EE    CHKINP  01D6    CR      000D    CRLF    0100    CSTAT   0000
CTRLC   0003    CTRLO   000F    ENDOUT  01D4    FIRSTF  01F4    LF      000A    MARGN   01E2
NULLS   01E1    DCMG    01E4    DCSW    01E5    OUTC10  01BA    OUTC20  01A3    OUTC30  0180
OUTC40  0188    OUTC50  01C0    OUTCHR  0182    OUTNUL  01C3    PSTRIP  003F    RDA     0040
RSTART  242F    SPC     0020    TBE     0080
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                        PAGE 0004
CROSS REFERENCE LISTING

CDATA   0005  0038 0082
CHKI1D  0091  0095
CHKINP  0079  0089
CR      0012  0025 0039 0059
CRLF    0035  0056
CSTAT   0004  0045 0031 0070
CTRLC   0010  0093
CTRLO   0013  0014
ENDOUT  0074  0030
FIRSTF  0096
LF      0011  0063
MARGN   0017  0047
NULLS   0016  0065
DCMG    0019  0040
DCSW    0020  0028 0084 0080
OUTC10  0031  0033
OUTC20  0041  0043
OUTC30  0056  0049
OUTC40  0057  0044 0051
OUTC50  0060  0071
OUTCHR  0027  0064 0070
OUTNUL  0064
PSTRIP  0009  0037 0083
RDA     0007  0060
RSTART  0022  0093
SPC     0014  0043 0054
TBE     0006  0032
```

8.2 Control Basic model MCB-216

8.2.1 Memory Allocation (MCB-216)

The listing which follows is an outline of memory allocation for Control Basic model MCB-216 equivalent to that of section 8.1.1. The minimum memory requirements to run this model are RAM from 2000H to 23FFH and the Control Basic Interpreter and Monitor in ROM from 0 to FFFH, the exact configuration of the Cromemco Single Card Computer. Memory from 1000H to 1FFFH is usually ROM (as on the SCC) and may be used to store previously SAVEd programs. Locations 2400H to the end of addressable memory may be used for SAVEd or EPROMed programs and may be either ROM or RAM. Page 2 of the listing is a detailed breakdown of the RAM from 2000H to 21FEH, which contains Control Basic variables, stack, input buffer, and console parameters.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                        PAGE 0001
3K Control Basic Variables and Memory Allocation (MCB-216)

                    0003 ;
                    0004 ;      Copyright (c) 1976, 1979 Cromemco, Inc.
                    0005 ;
                    0006 ;
                    0007 ;  *** Memory Usage (probably RAM unless otherwise noted) ***
                    0008 ;
                    0009 ;  0000-0422 are for SCC Monitor (ROM)
                    0010 ;  0423-0FFF are for SCC Control Basic Interpreter (ROM)
                    0011 ;  1000-1FFF are for SAVEd user programs (usually ROM)
                    0012 ;  2000-21FE are for variables, input line, and stack
                    0013 ;  21FF-23FF are for Control Basic text & array @(I)
                    0014 ;  2400-FFFF are not used but may be used for saved files
       1

     (2000)         0018 BOTRAM   EQU     2000H          ; "BOTTOM OF RAM" FOR VARIABLES, ETC.
     (2400)         0019 TOPRAM   EQU     2400H          ; "TOP OF RAM" USED BY CONTROL BASIC
     (0423)         0020 BOTROM   EQU     423H           ; "BOTTOM OF ROM" CONTAINING CB
                    0021 ; The following are used for interface to the SCC Monitor!
     (0030)         0022 MONITR   EQU     30H            ; SCC MONITOR WARM-START LOCATION
     (005C)         0023 INITUART EQU     5CH            ; UART INITIALIZATION ROUT. LOCATION
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                        PAGE 0002
3K Control Basic Variables and Memory Allocation (MCB-210)

F00'                    ORG     BOTRAM
F00    (0068)  VARBGN:  DEFS    2*(26*2)     ; VARIABLE A-Z AND A0-Z0 STORAGE
F68    (0001)           DEFS    1            ; EXTRA BYTE FOR INPUT BUFFER
F69    (0085)  BUFFER:  DEFS    133          ; STORAGE FOR INPUT LINE BUFFER
       (1FEE)  BUFEND   EQU     $            ; END OF INPUT BUFFER
FEE    (0028)           DEFS    40           ; EXTRA BYTES FOR STACK
016    (01C4)  STKLMT:  DEFS    456-4        ; SOFT LIMIT FOR STACK
       (21DA)  STACK    EQU     $            ; TOP OF STACK
1DA    (0004)           DEFS    4            ; RESERVED FOR ROCS
1DE    (0001)  BASFLG:  DEFS    1            ; 0 FOR COLD-; NON-0 FOR WARM-START
1DF    (0002)  IOINS:   DEFS    2            ; STORAGE FOR AN I/O INSTRUCTION
1E1    (0001)  IORET:   DEFS    1            ; STORAGE FOR THE RETURN INSTRUCTION
1E2    (0001)  NULLS:   DEFS    1            ; NUMBER OF NULLS AFTER CR-LF
1E3    (0001)  MARGN:   DEFS    1            ; SOFT TERMINAL WIDTH
1E4    (0001)           DEFS    1            ; HARD TERMINAL WIDTH
1E5    (0001)  OCMG:    DEFS    1            ; COLUMN COUNTER FOR MARGINS
1E6    (0001)  OCSW:    DEFS    1            ; SWITCH FOR VISIBLE OUTPUT
1E7    (0002)  CURRNT:  DEFS    2            ; CURRENT LINE NUMBER STORAGE
1E9    (0002)  STKRUN:  DEFS    2            ; SAVES STACK POINTER DURING 'RUN'
1EB    (0002)  STKGOS:  DEFS    2            ; SAVES STACK POINTER DURING 'GOSUB'
       (21ED)  VARNXT   EQU     $            ; TEMP. STOR. OF CURRENT 'NEXT' VAR.
       (21ED)  CALDE    EQU     $            ; TEMP. STOR. FOR ADDR. FOR 'CALL'
1ED    (0002)  STKINP:  DEFS    2            ; SAVES STACK POINTER DURING 'INPUT'
1EF    (0002)  LOPVAR:  DEFS    2            ; ADDRESS OF 'FOR' LOOP VARIABLE
1F1    (0002)  LOPINC:  DEFS    2            ; 'STEP' SIZE FOR 'FOR' LOOP
1F3    (0002)  LOPLMT:  DEFS    2            ; LIMIT OF 'FOR' LOOP VARIABLE
1F5    (0002)  LOPLN:   DEFS    2            ; TEMP. STOR. OF 'CURRNT' IN 'FOR'
1F7    (0002)  LOPPT:   DEFS    2            ; TEXT POINTER TEMP. STOR. IN 'FOR'
1F9    (0002)  RANPNT:  DEFS    2            ; ADDR. POINTER USED IN RND FUNC.
1FB    (0002)  TXTBGN:  DEFS    2            ; STORAGE OF TEXT POINTER ADDR.
1FD    (0002)  TXTLMT:  DEFS    2            ; TEXT AREA LIMIT (SAME AS 'TOPRAM'
                                             ;   UNLESS CHANGED BY 'LOCK')


       (21FF)  TXTRAN   EQU     $
21FF   (0002)  TXTUNF:  DEFS    2            ; POINTER TO UNFILLED TEXT AREA
2201   (0001)  TEXT:    DEFS    1            ; START OF ACTUAL TEXT (STORAGE
                                             ;   OF FIRST LINE NUMBER)
```

Also shown on page 2 of the listing above is the beginning of the text area for the current program. The text area is organized as follows; any stored program (by means of the SAVE or EPROM commands) is organized this same way:

1. The first two bytes of the first page point to the first available byte after the program text relative to 21FFH.

2. The next two bytes contain the line number of the first line in binary (low byte-high byte). This is followed by the text of the first line and a single 0DH, or CR, at the end of that line.

3. Subsequent lines of text duplicate the form of this first line: 2-byte binary line number, ASCII text, CR-byte.

match, the program HALTs with 00H in Reg. A; if any byte does not match, the program HALTs with FFH in Reg. A. In actual use, the HALT instructions would be replaced by a branching instruction to the next code segment. The program may easily be placed elsewhere in memory provided the absolute jump instruction is re-assembled to point to the new EXIT point. It user's responsibility to test the contents of Reg. A for successful verification and take the appropriate action.

```
;
;                    2708 EPROM PROGRAMMER
;
;
;      THIS SAMPLE PROGRAM PROGRAMS TWO 2708 PROMS
;      IN SOCKETS ROM6 AND ROM7 ON A CROMEMCO
;      BYTESAVER II MEMORY BOARD.  THE SOURCE CODE
;      IN THIS EXAMPLE IS ASSUMED LOCATED AT 200H - 9FFH.
;
      (0200)  SOURCE:  EQU   200H          ;SOURCE CODE START AD
      (0800)  SWATH:   EQU   800H          ;SOURCE CODE LENGTH
      (FC00)  EPROM:   EQU   0FC00H        ;2708 PROM START ADDR
0000 210002   PGM2708: LD    HL,SOURCE     ;LOAD SOURCE ADDR,
0003 1100F8            LD    DE,EPROM      ;2708 START ADDR,
0006 010008            LD    BC,SWATH      ;AND SWATH FOR BLOCK XFER
0009 EDB0              LDIR                ;WRITE SOURCE TO 2708
000B E5                PUSH  HL            ;SAVE LAST SOURCE+1 ADDR
000C 2A2B00            LD    HL,(PASSES)   ;GET PROGRAMMING PASS#
000F 2B                DEC   HL            ;DECREMENT IT
0010 7C                LD    A,H           ;TEST FOR PASS#=0000H
0011 B5                OR    L             ;VALID ZERO FLAG
0012 222B00            LD    (PASSES),HL   ;RESTORE PASS#
0015 E1                POP   HL            ;RESTORE LAST SOURCE+1
0016 20E8              JR    NZ,PGM2708    ;PASS# NOT 0000H--GO BACK
0018 010008            LD    BC,SWATH      ;DONE--READY TO VERIFY
001B 2B                DEC   HL            ;POINT TO LAST SOURCE ADDR
001C 1B       VERIFY:  DEC   DE            ;POINT TO LAST 2708 ADDR
001D 1A                LD    A,(DE)        ;GET 2708 BYTE AND
001E EDA9              CPD                 ;COMPARE TO SOURCE BYTE
0020 E22800            JP    PO,EXIT       ;DONE IF (BC)=0000H
0023 28F7              JR    Z,VERIFY      ;BYTES MATCH--NEXT BYTE
0025 3EFF     ERROR:   LD    A,0FFH        ;MISMATCH--LOAD REG. A
0027 76                HALT                ;WITH FFH AND HALT
0028 3E00     EXIT:    LD    A,00H         ;SUCCESS--LOAD REG. A
002A 76                HALT                ;WITH 00H AND HALT
002B 6801     PASSES:  DW    360           ;INITIAL PASS#=360
002D                   END   PGM2708
```

intel Silicon Gate MOS 8708/8704

8192/4096 BIT ERASABLE AND ELECTRICALLY REPROGRAMMABLE READ ONLY MEMORY

- 8708 1024x8 Organization
- 8704 512x8 Organization

- Fast Programming — Typ. 100 sec. For All 8K Bits
- Low Power During Programming
- Access Time — 450 ns
- Standard Power Supplies — +12V, ±5V
- Static — No Clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Modes
- Three-State Output — OR-Tie Capability

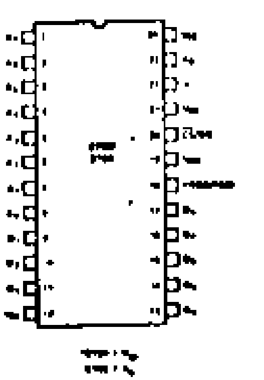
The Intel® 8708/8704 are high speed 8192/4096 bit erasable and electrically reprogrammable ROM's (EPROM) ideally suited where fast turn around and pattern experimentation are important requirements.

The 8708/8704 are packaged in a 24 pin dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the devices.

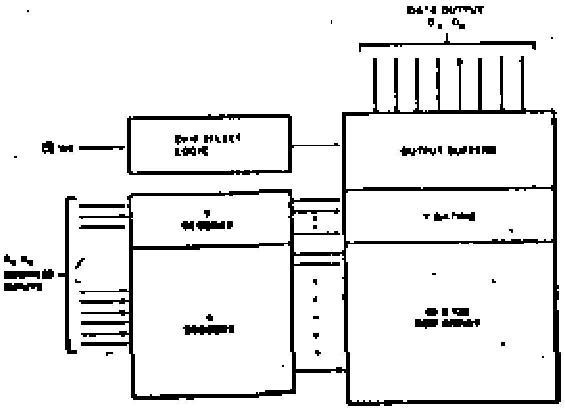
A pin for pin mask programmed ROM, the Intel® 8308, is available for large volume production runs of systems initially using the 8708.

The 8708/8704 is fabricated with the time proven N-channel silicon gate technology.

PIN CONFIGURATIONS



BLOCK DIAGRAM



PIN NAMES

A0-A9	ADDRESS INPUTS
O1-O8	DATA OUTPUTS
CS/WE	CHIP SELECT/WRITE ENABLE INPUT

Absolute Maximum Ratings*

Temperature Under Bias .... -25°C to +85°C
Storage Temperature .... -65°C to +125°C
All Input or Output Voltages with Respect to V_{BB}
(except Program) .... +15V to -0.3V
Program Input to V_{BB} .... +35V to -0.3V
Supply Voltages V_{CC} and V_{SS} with Respect to V_{BB} .... +15V to -0.3V
V_{DD} with Respect to V_{BB} .... +20V to -0.3V
Power Dissipation 1.5W

*COMMENT

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

READ OPERATION

D.C. and Operating Characteristics

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{DD} = +12V ±5%, V_{BB} = -5V ±5%, V_{SS} = 0V, Unless Otherwise Noted.

Symbol	Parameter	Min.	Typ.[1]	Max.	Unit	Conditions
I_{LI}	Address and Chip Select Input Load Current			10	µA	V_{IN} = 5.25V
I_{LO}	Output Leakage Current			10	µA	V_{OUT} = 5.25V, CS/WE = 5V
I_{DD}	V_{DD} Supply Current		60	65	mA	Worst Case Supply Currents:
I_{CC}	V_{CC} Supply Current		6	10	mA	All Inputs High
I_{BB}	V_{BB} Supply Current		30	45	mA	CS/WE = 5V; T_A = 0°C
V_{IL}	Input Low Voltage	V_{SS}		0.65	V	
V_{IH}	Input High Voltage	3.0		V_{CC}+1	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.6mA
V_{OH1}	Output High Voltage	3.7			V	I_{OH} = -100µA
V_{OH2}	Output High Voltage	2.4			V	I_{OH} = -1mA
P_D	Power Dissipation			800	mW	T_A = 70°C

NOTES: 1. Typical values are for T_A = 25°C and nominal supply voltages.
2. The program input (Pin 18) may be tied to V_{SS} or V_{CC} during the read mode.

A.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{DD} = +12V ±5%, V_{BB} = -5V ±5%, V_{SS} = 0V, Unless Otherwise Noted.

Symbol	Parameter	Min.	Typ.	Max.	Unit
t_{ACC}	Address to Output Delay		280	450	ns
t_{CO}	Chip Select to Output Delay			120	ns
t_{DF}	Chip De-Select to Output Float	0		120	ns
t_{OH}	Address to Output Hold	0			ns

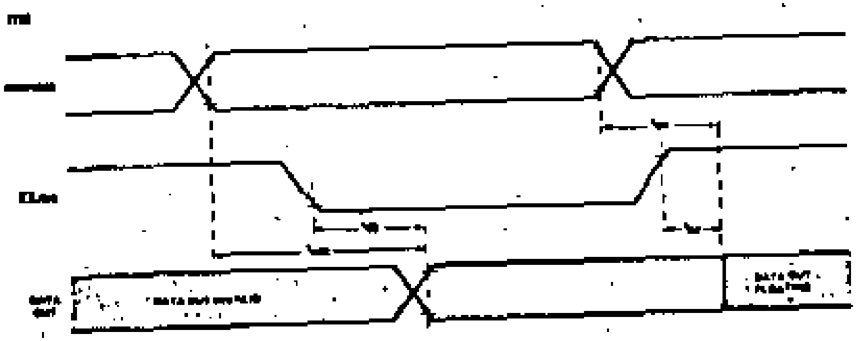
Capacitance[1] T_A = 25°C, f = 1MHz

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN}	Input Capacitance	4	6	pF	V_{IN}=0V
C_{OUT}	Output Capacitance	8	12	pF	V_{OUT}=0V

Note 1. This parameter is periodically sampled and not 100% tested.

A.C. Test Conditions:

Output Load: 1 TTL gate and C_L = 100pF
Input Rise and Fall Times: ≤20ns
Timing Measurement Reference Levels: 0.8V and 2.8V for inputs; 0.8V and 2.4V for outputs
Pulse Levels: 0.65V to 3.0V



PROGRAMMING OPERATION

Description

Initially, and after each erasure, all bits of the 8708/8704 are in the "1" state (Output High). Information is introduced by selectively programming "0" into the desired bit locations.

The circuit is set up for programming operation by raising the CS/WE input (Pin 20) to +12V. The word address is selected in the same manner as in the read mode. Data to be programmed are presented, 8-bits in parallel, to the data output lines (O_1-O_8). Logic levels for address and data lines and the supply voltages are the same as for the read mode. After address and set up one program pulse (V_P) per address is applied to the program input (Pin 18). One pass through all addresses to be programmed is defined as a program loop. The number of loops (N) required is a function of the program pulse width (t_{PW}) according to N x t_{PW} ≥ 100 ms.

For program verification, program loops and read loops may be alternated as shown in waveform B.

Program Characteristics

T_A = 25°C, V_{CC} = +5V ±5%, V_{DD} = +12V ±5%, V_{BB} = -5V ±5%, V_{SS} = 0V, CS/WE = +12V, Unless Otherwise Noted.

Symbol	Parameter	Min.	Typ.	Max.	Units
t_{AS}	Address Setup Time	10			μs
t_{CSS}	CS/WE Setup Time	10			μs
t_{DS}	Data Setup Time	10			μs
t_{AH}	Address Hold Time	1			μs
t_{CH}	CS/WE Hold Time	.5			μs
t_{DH}	Data Hold Time	1			μs
t_{DF}	Chip Deselect to Output Float Delay	0		120	ns
t_{DPR}	Program To Read Delay			10	μs
t_{PW}	Program Pulse Width	.1		1.0	ms
t_{PR}	Program Pulse Rise Time	.5		2.0	μs
t_{PF}	Program Pulse Fall Time	.5		2.0	μs
I_P	Programming Current		10	20	mA
V_P	Program Pulse Amplitude	25		27	V

NOTE: Intel's standard product warranty applies only to devices programmed to specifications described herein.

Erasing Procedure

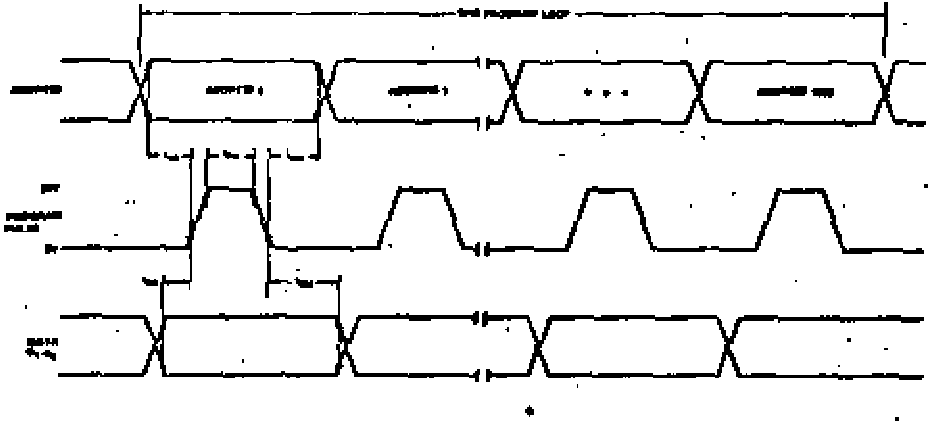
The 8708/8704 may be erased by exposure to high intensity short wave ultraviolet light at a wavelength of 2537Å. The recommended integrated dose (i.e., UV intensity x exposure time) is 10W sec/cm². Examples of ultraviolet sources which can the 8708/8704 in 20 to 30 minutes are the Model UVS-54 and Model S-52 short-wave ultraviolet lamps manufactured by Ultra-Violet Products, Inc. (5114 Walnut Grove Avenue, San Gabriel, California). The lamps should be used without short-wave filters, and the 8708/8704 to be erased should be placed about one inch away from the lamp tubes.

Waveforms

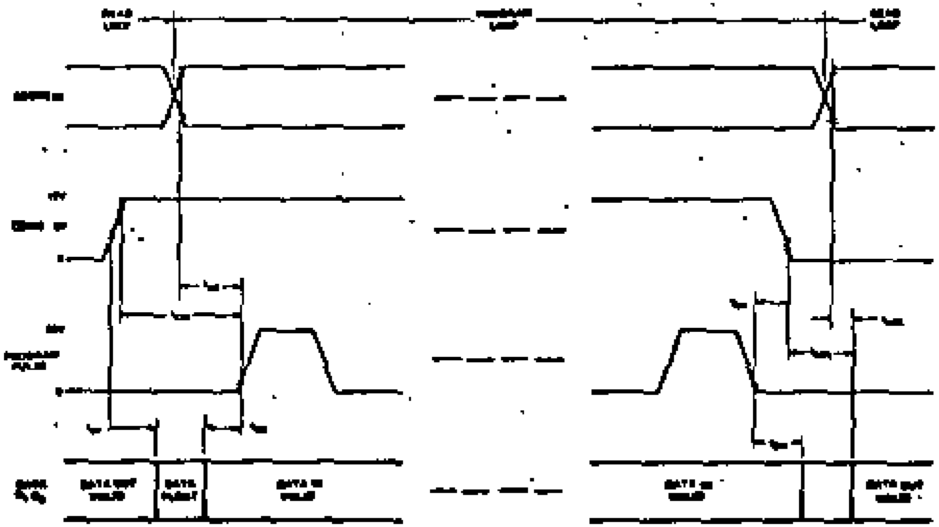
(Logic levels and timing reference levels same as in the Read Mode unless noted otherwise.)

A) Program Mode

$\overline{CS}$/WE = +12V

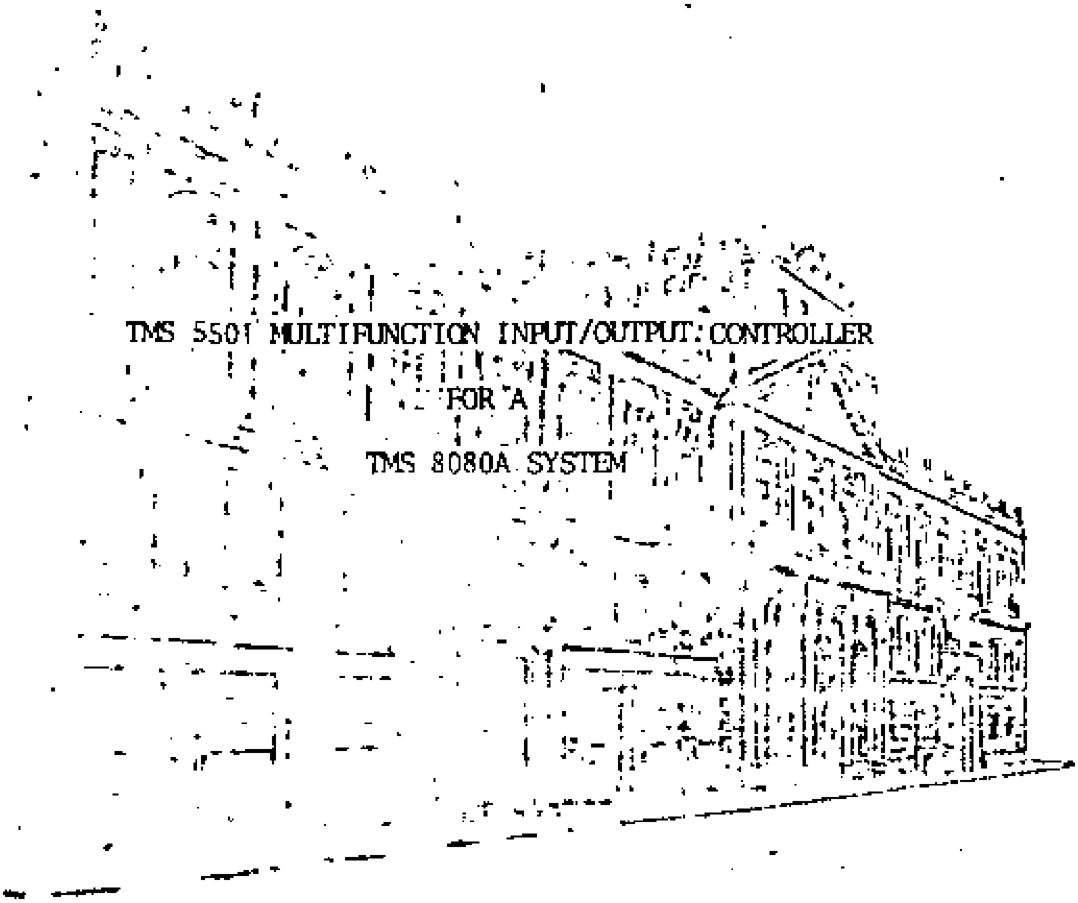


B) Read/Program/Read Transitions



DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIAYY APLICACIONES



TMS 5501 MULTIFUNCTION INPUT/OUTPUT CONTROLLER FOR A TMS 8080A SYSTEM

JULIO, 1981

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Information contained in this publication is believed to be accurate and reliable. However, responsibility is assumed neither for its use nor for any infringement of patents or rights of others that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Texas Instruments or others.

TMS 5501 MULTIFUNCTION INPUT/OUTPUT CONTROLLER

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 5501 is a multifunction input/output circuit for use with TI's TMS 8080A CPU. It is fabricated with the same N-channel silicon-gate process as the TMS 8080A and has compatible timing, signal levels, and power supply requirements. The TMS 5501 provides a TMS 8080A microprocessor system with an asynchronous communications interface, data I/O buffers, interrupt control logic, and interval timers.

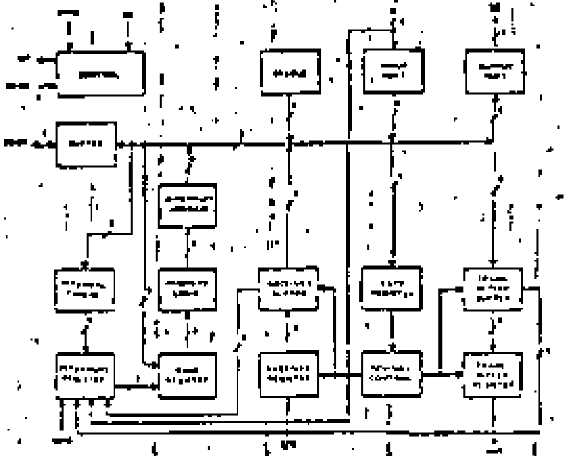


FIGURE 1–TMS 5501 BLOCK DIAGRAM

The I/O section of the TMS 5501 contains an eight-bit parallel input port and a separate eight-bit parallel output port with storage register. Five programmable interval timers provide time intervals from 64 μs to 16.32 ms.

The interrupt system allows the processor to effectively communicate with the interval timers, external signals, and the communications interface by providing TMS 8080A-compatible interrupt logic with masking capability.

Data transfers between the TMS 5501 and the CPU are carried by the data bus and controlled by the interrupt, chip enable, sync, and address lines. The TMS 8080A uses four of its memory-address lines to select one of 14 commands to which the TMS 5501 will respond. These commands allow the CPU to:

- — read the receiver buffer
- — read the input port
- — read the interrupt address
- — read TMS 5501 status
- — issue discrete commands
- — load baud rate register
- — load the transmitter buffer
- — load the output port
- — load the mask register
- — load an interval timer

The commands are generated by executing memory referencing instructions such as MOV (register to memory) with the memory address being the TMS 5501 command. This provides a high degree of flexibility for I/O operations by letting the systems programmer use a variety of instructions.

1.2 SUMMARY OF OPERATION

Addressing the TMS 5501

A convenient method for addressing the TMS 5501 is to tie the chip enable input to the highest order address line of the CPU's 16 bit address bus and the four TMS 5501 address inputs to the four lowest order bits of the bus. This, of course, limits the system to 32,768 words of memory but in many applications the full 65,536 word memory addressing capability of the TMS 8080A is not required.

Communications Functions

The communications section of the TMS 5501 is an asynchronous transmitter and receiver for serial communications and provides the following functions:

Programmable baud rate — A CPU command selects a baud rate of 110, 150, 300, 1200, 2400, 4800, or 9600 baud.

Incoming character detection — The receiver detects the start and stop bits of an incoming character and places the character in the receive buffer.

Character transmission — The transmitter generates start and stop bits for a character received from the CPU and shifts it out.

Status and command signals — Via the data bus, the TMS 5501 signals the status of: framing error and overrun error flags, data in the receiver and transmitter buffers, start and data bit detectors, and end-of-transmission (break) signals from external equipment. It also issues break signals to external equipment.

Data Interface

The TMS 5501 moves data between the CPU and external devices through its internal data bus, input port, and output port. When data is present on the bus that is to be sent to an external device, a Load Output Port (LOP) command from the CPU puts the data on the $\overline{XO}$ pins of the TMS 5501 by latching it in the output port. The data remains in the port until another LOP command is received. When the CPU requires data that is present on the External Input (XI) lines, it issues a command that gates the data onto the internal data bus of the TMS 5501 and consequently onto the CPU's data bus at the correct time during the CPU cycle.

Internal Timers

To start a countdown by any of the five interval timers, the program selects the particular timer by an address to the TMS 5501 and loads the required interval into the timer via the data bus. Loading the timer activates it and it counts down in increments of 64 microseconds. The 8 bit counters provide intervals that vary in duration from 64 to 16,320 microseconds. Much longer intervals can be generated by cascading the timers through software. When a timer reaches zero, it generates an interrupt that typically will be used to point to a subroutine that performs a servicing function such as polling a peripheral or scanning a keyboard. Loading an interval value of zero causes an immediate interrupt. A new value loaded while the interval timer is counting overrides the previous value and the interval timer starts counting down the new interval. When an interval timer reaches zero it remains inactive until a new interval is loaded.

Servicing Interrupts

The TMS 5501 provides a TMS 8080A system with several interrupt control functions by receiving external interrupt signals, generating interrupt signals, masking out undesired interrupts, establishing the priority of interrupts, and generating RST instructions for the TMS 8080A. An external interrupt is received on pin 22, SENS. An additional external interrupt can be received on pin 32, XI7, if selected by a discrete command from the TMS 8080A (See Figure 4). The TMS 5501 generates an interrupt when any of the five interval timers count to zero. Interrupts are also generated when the receiver buffer is loaded and when the transmitter buffer is empty.

When an interrupt signal is received by the interrupt register from a particular source, a corresponding bit is set and gated to the mask register. A pattern will have previously been set in the mask register by a load-mask-register command from the TMS 8080A. This pattern determines which interrupts will pass through to the priority logic. The priority logic allows an interrupt to generate an RST instruction to the TMS 8080A only if there is no higher priority interrupt that has not been accepted by the TMS 8080A. The TMS 5501 prioritizes interrupts in the order shown below:

1st — Internal Timer #1
2nd — Internal Timer #2
3rd — External Sensor
4th — Internal Timer #3
5th — Receiver Buffer Loaded
6th — Transmitter Buffer Emptied
7th — Internal Timer #4
8th — Internal Timer #5 or an External Input (XI 7)

The highest priority interrupt passes through to the interrupt address logic, which generates the RST instruction to be read by the TMS 8080. See Table 3 for relationship of interrupt sources to RST instructions and Figures 6 and 8 for timing relationships.

The TMS 5501 provides two methods of servicing interrupts: an interrupt-driven system or a polled-interrupt system. In an interrupt-driven system, the INT signal of the TMS 5501 is tied to the INT input of the TMS 8080A. The sequence of events will be: (1) The TMS 5501 receives (or generates) an interrupt signal and readies the appropriate RST instruction. (2) The TMS 5501 INT output, tied to the TMS 8080A INT input, goes high signaling the TMS 8080A that an interrupt has occurred. (3) If the TMS 8080A is enabled to accept interrupts, it sets the INTA (interrupt acknowledge) status bit high at SYNC time of the next machine cycle. (4) If the TMS 5501 has previously received an interrupt-acknowledge-enable command from the CPU (see Bit 3, Paragraph 2.2.5), the RST instruction is transferred to the data bus.

In a polled-interrupt system, INT is not used and the sequence of events will be: (1) The TMS 5501 receives (or generates) an interrupt and readies the RST instruction. (2) The TMS 5501 interrupt-pending status bit (see Bit 5, Paragraph 2.2.4) is set high (the interrupt-pending status bit and the INT output go high simultaneously). (3) At the prescribed time, the TMS 8080A polls the TMS 5501 to see if an interrupt has occurred by issuing a "read TMS 5501 status" command and reading the interrupt-pending bit. (4) If the bit is high, the TMS 8080A will then issue a read-interrupt address command, which causes the TMS 5501 to transfer the RST instruction to the data bus as data for the instruction being executed by the TMS 8080A.

APPLICATIONS

Communications Terminals

The functions of the TMS 5501 make it particularly useful in TMS 8080A based communications terminals and generally applicable in systems requiring periodic or random servicing of interrupts, generation of control signals to external devices, buffering of data, and transmission and reception of asynchronous serial data. As an example, a system configuration such as shown in Figure 2 can function as the controller for a terminal that governs employee entrance into a plant or security areas within a plant. Each terminal is identified by a central computer through ID switches. The central system supplies each terminal's RAM with up to 16 employee access categories applicable to that terminal. These categories are compared with an employee's badge character when he inserts his badge into the badge sensor. If a

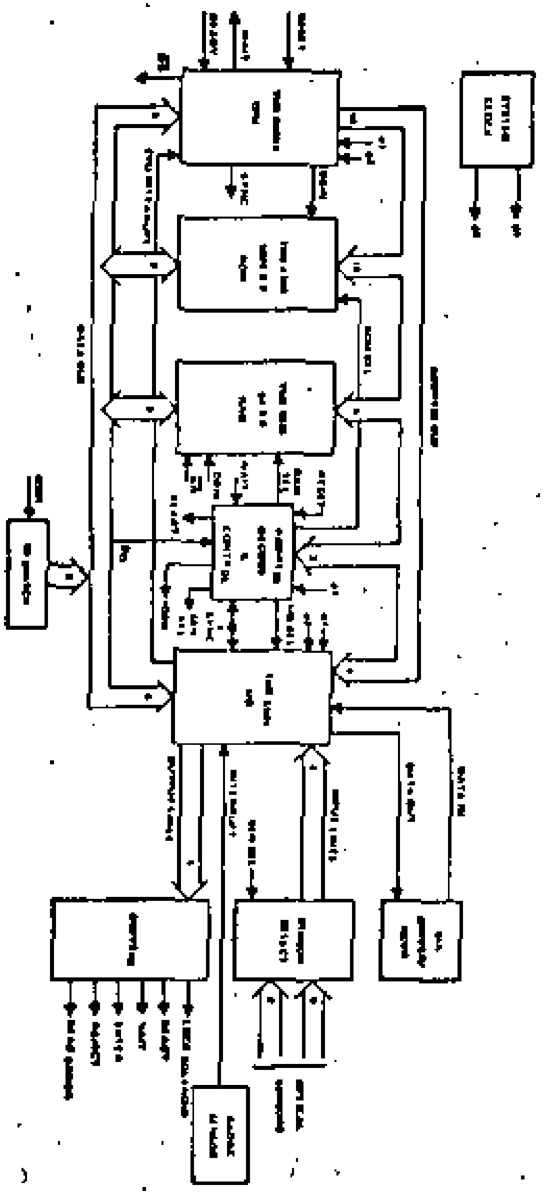


FIGURE 2—ACCESS CONTROL SYSTEM BLOCK DIAGRAM

match is not found, a reject light will be activated. If a match is found, the terminal will transmit the employee's badge number and access category to the central system, and a door unlock solenoid will be activated for 4 seconds. The central computer then may take the transmitted information and record it along with time and date of access.

The TMS 4700 is a 1024 x 8 ROM that contains the system program, and the TMS 4036 is a 64 x 8 RAM that serves as the stack for the TMS 8080A and storage for the access category information. TTL circuits control chip enable information carried by the address bus. Signals from the CPU gate the address bits from the ROM, the RAM, or the TMS 5501 onto the data bus at the correct time in the CPU cycle. The clock generator consists of four TTL circuits along with a crystal, needed to maintain accurate serial data assembly and disassembly with the central computer.

The TMS 5501 handles the asynchronous serial communication between the TMS 8080A and the central system and gates data from the badge reader onto the data bus. It also gates control and status data from the TMS 8080A to the door lock and badge reader and controls the time that the door lock remains open. The TMS 5501 signals the TMS 8080A when the badge reader or the communication lines need service. The functions that the TMS 5501 is to perform are selected by an address from the TMS 8080A with the highest order address line tied to the TMS 5501 chip enable input and the four lowest order lines tied to the address inputs.

2. OPERATIONAL AND FUNCTIONAL DESCRIPTION

This detailed description of the TMS 5501 consists of:

INTERFACE SIGNALS — a definition of each of the circuit's external connections

COMMANDS — the address required to select each of the TMS 5501 commands and a description of the response to the command.

2.1 INTERFACE SIGNALS

The TMS 5501 communicates with the TMS 8080A via four address lines, a chip enable line, an eight-bit bidirectional data bus, an interrupt line, and a sync line. It communicates with system components other than the CPU via eight external inputs, eight external outputs, a serial receiver input, a serial transmitter output, and an external sensor input. Table 1 defines the TMS 5501 pin assignments and describes the function of each pin.

TABLE 1
TMS 5501 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	DESCRIPTION
		INPUTS
CE	18	Chip enable—When CE is low, the TMS 5501 address decoding is inhibited, which prevents execution of any of the TMS 5501 commands.
A3	17	Address bus—A3 through A0 are the lines that are addressed by the TMS 8080A to select a particular TMS 5501 function.
A2	16	
A1	15	
A0	14	
SYNC	19	Synchronizing signal—The SYNC signal is issued by the TMS 8080A and indicates the beginning of a machine cycle and availability of machine status. When the SYNC signal is active (high), the TMS 5501 will monitor the data bus bits D0 (interrupt acknowledge) and D1 ($\overline{WO}$, data output function).
$\overline{RCV}$	5	Receiver serial data input line—$\overline{RCV}$ must be held in the inactive (high) state when not receiving data. A transition from high to low will activate the receive circuitry.

TABLE 1 (continued)
TMS 5501 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	DESCRIPTION
		INPUTS
XI 0	39	External inputs—These eight external inputs are gated to the data bus when the read-external-inputs function is addressed. External input n is gated to data bus bit n without conversion.
XI 1	38	
XI 2	37	
XI 3	36	
XI 4	35	
XI 5	34	
XI 6	33	
XI 7	32	
SENS	22	External interrupt sensing — A transition from low to high at SENS sets a bit in the interrupt register, which, if enabled, generates an interrupt to the TMS 8080A.
		OUTPUTS
$\overline{XO}$ 0	24	External outputs—These eight external outputs are driven by the complement of the output register; i.e., if output register bit n is loaded with a high (low) from data bus bit n by a load-output register command, the external output n will be a low (high). The external outputs change only when a load-output-register function is addressed.
$\overline{XO}$ 1	25	
$\overline{XO}$ 2	26	
$\overline{XO}$ 3	27	
$\overline{XO}$ 4	28	
$\overline{XO}$ 5	29	
$\overline{XO}$ 6	30	
$\overline{XO}$ 7	31	
XMT	40	Transmitter serial data output line—This line remains high when the TMS 5501 is not transmitting.
		DATA BUS INPUT/OUTPUT
D0	13	Data bus—Data transfers between the TMS 5501 and the TMS 8080A are made via the 8-bit bidirectional data bus. D0 is the LSB. D7 is the MSB.
D1	12	
D2	11	
D3	10	
D4	9	
D5	8	
D6	7	
D7	6	
INT	23	Interrupt—When active (high), the INT output indicates that at least one of the interrupt conditions has occurred and that its corresponding mask-register bit is set.
		POWER AND CLOCKS
V_{SS}	4	Ground reference
V_{BB}	1	Supply voltage (−5 V nominal)
V_{CC}	2	Supply voltage (5 V nominal)
V_{DD}	3	Supply voltage (12 V nominal)
$\phi 1$	20	Phase 1 clock
$\phi 2$	21	Phase 2 clock

2.2 TMS 5501 COMMANDS

The TMS 5501 operates as memory device for the TMS 8080A. Functions are initiated via the TMS 8080A address bus and the TMS 5501 address inputs. Address decoding to determine the command function being issued is defined in Table 2.

TABLE 2
COMMAND ADDRESS DECODING
When Chip Enable is High

A3	A2	A1	A0	COMMAND	FUNCTION	PARAGRAPH
L	L	L	L	Read receiver buffer	RBn → Dn	2.2.1
L	L	L	H	Read external inputs	XIn → Dn	2.2.2
L	L	H	L	Read interrupt address	RST → Dn	2.2.3
L	L	H	H	Read TMS 5501 status	(Status) → Dn	2.2.4
L	H	L	L	Issue discrete commands	See Figure 4	2.2.5
L	H	L	H	Load rate register	See Figure 4	2.2.6
L	H	H	L	Load transmitter buffer	Dn → TBn	2.2.7
L	H	H	H	Load output port	Dn → $\overline{XO}$n	2.2.8
H	L	L	L	Load mask register	Dn → MRn	2.2.9
H	L	L	H	Load interval timer 1	Dn → Timer 1	2.2.10
H	L	H	L	Load interval timer 2	Dn → Timer 2	2.2.10
H	L	H	H	Load interval timer 3	Dn → Timer 3	2.2.10
H	H	L	L	Load interval timer 4	Dn → Timer 4	2.2.10
H	H	L	H	Load interval timer 5	Dn → Timer 5	2.2.10
H	H	H	L	No function		
H	H	H	H	No function		

RBn = Receiver buffer bit n
Dn = Data bus I/O terminal n
XIn = External input terminal n
RST = 11 $(IA_2)(IA_1)(IA_0)$ 1 1 1 (see Table 3)
TBn = Transmitter buffer bit n
$\overline{XO}$n = Output register bit n
MRn = Mask register bit n

TABLE 3
RST INSTRUCTIONS

DATA BUS BIT								
0	1	2	3	4	5	6	7	INTERRUPT CAUSED BY
H	H	H	L	L	L	H	H	Interval Timer 1
H	H	H	H	L	L	H	H	Interval Timer 2
H	H	H	L	H	L	H	H	External Sensor
H	H	H	H	H	L	H	H	Interval Timer 3
H	H	H	L	L	H	H	H	Receiver Buffer
H	H	H	H	L	H	H	H	Transmitter Buffer
H	H	H	L	H	H	H	H	Interval Timer 4
H	H	H	H	H	H	H	H	Interval Timer 5 or XI7

The following paragraphs define the functions of the TMS 5501 commands.

2.2.1 Read receiver buffer

Addressing the read-receiver-buffer function causes the receiver buffer contents to be transferred to the TMS 8080A and clears the receiver-buffer-loaded flag.

2.2.2 Read external input lines

Addressing the read-external-inputs function transfers the states of the eight external input lines to the TMS 8080A.

2.2.3 Read interrupt address

Addressing the read interrupt address function transfers the current highest priority interrupt address onto the data bus as read data. After the read operation is completed, the corresponding bit in the interrupt register is reset.

If the read interrupt-address function is addressed when there is no interrupt pending, a false interrupt address will be . TMS 5501 status function should be addressed in order to determine whether or not an interrupt condition is pending.

2.2.4 Read TMS 5501 status

Addressing the read-TMS 5501-status function gates the various status conditions of the TMS 5501 onto the data bus. The status conditions, available as indicated in Figure 3, are described in the following paragraphs.

BIT:	7	6	5	4	3	2	1	0
	START BIT DETECT	FULL BIT DETECT	INTRPT PENDING	XMIT BUFFER EMPTY	RCV BUFFER LOADED	SERIAL RCVD	OVERRUN ERROR	FRAME ERROR

FIGURE 3—DATA BUS ASSIGNMENTS FOR TMS 5501 STATUS

Bit 0, framing error

A high in bit 0 indicates that a framing error was detected on the last character received (either one or both stop bits were in error). The framing error flag is updated at the end of each character. Bit 0 of the TMS 5501 status will remain high until the next valid character is received.

Bit 1, overrun error

A high in bit 1 indicates that a new character was loaded into the receiver buffer before a previous character was read out. The overrun error flag is cleared each time the read-I/O-status function is addressed or a reset command is issued.

Bit 2, serial received data

Bit 2 monitors the receiver serial data input line. This line is provided as a status input for use in detecting a break and for test purposes. Bit 2 is normally high when no data is being received.

Bit 3, receiver buffer loaded

A high in bit 3 indicates that the receiver buffer is loaded with a new character. The receiver-buffer-loaded flag remains high until the read-receiver-buffer function is addressed (at which time the flag is cleared). The reset function also clears the flag.

Bit 4, transmitter buffer empty

A high in bit 4 indicates that the transmitter buffer register is empty and ready to accept a character. Note, however, that the serial transmitter register may be in the process of shifting out a character. The reset function the transmitter-buffer-empty flag high.

Bit 5, interrupt pending

A high in bit 5 indicates that one or more of the interrupt conditions has occurred and the corresponding interrupt is enabled. This bit is the status of the interrupt signal INT.

Bit 6, full bit detected

A high in bit 6 indicates that the first data bit of a receive-data character has been detected. This bit remains high until the entire character has been received or until a reset is issued and is provided for test purposes.

Bit 7, start bit detected

A high in bit 7 indicates that the start bit of an incoming data character has been detected. This bit remains high until the entire character has been received or until a reset is issued and is provided for test purposes.

3 Issue discrete commands

Addressing the discrete command function causes the TMS 5501 to interpret the data bus information according to the following descriptions. See Figure 4 for the discrete command format. Bits 1 through 5 are latched until a different discrete command is received.

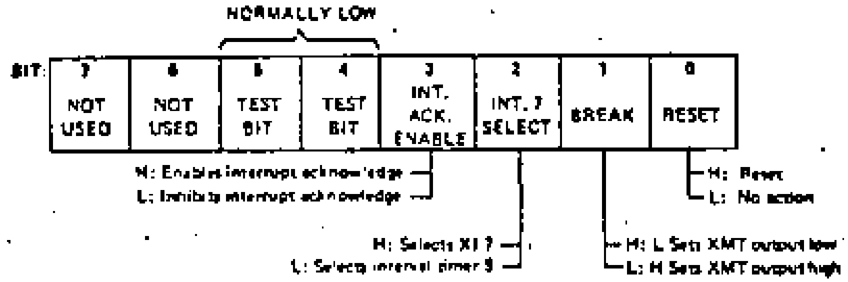


FIGURE 4—DISCRETE COMMAND FORMAT

Bit 0, reset

A high in bit 0 will cause the following:

1) The receiver buffer and register are cleared to the search mode including the receiver-buffer-loaded flag, the start bit-detected flag, the full-bit-detected flag, and the overrun-error flag. The receiver buffer is not and will contain the last character received.

2) The transmitter data output is set high (marking). The transmitter-buffer-empty flag is set high indicating that the transmitter buffer is ready to accept a character from the TMS 8080A.

3) The interrupt register is cleared except for the bit corresponding to the transmitter buffer interrupt, which is set high.

4) The internal timers are inhibited.

A low in bit 0 causes no action. The reset function has no effect on the output port, the external inputs, interrupt acknowledge enable, the mask register, the rate register, the transmitter register, or the transmitter buffer.

Bit 1, break
A low in bit 1 causes the transmitter data output to be reset low (spacing).

If bit 0 and bit 1 are both high, the reset function will override.

Bit 2, interrupt 7 select
Interrupt 7 may be generated either by a low to high transition of external input 7 or by interval timer 5.

A high in bit 2 selects the interrupt 7 source to be the transition of external input 7. A low in bit 2 selects the interrupt 7 source to be interval timer 5.

Bit 3, interrupt acknowledge enable
The TMS 5501 decodes data bus (CPU status) bit 0 at SYNC of each machine cycle to determine if an interrupt acknowledge is being issued.

A high in bit 3 enables the TMS 5501 to accept the interrupt acknowledge decode. A low in bit 3 causes the TMS 5501 to ignore the interrupt acknowledge decode.

Bit 4 and bit 5 are used only during testing of the TMS 5501. For correct system operation both bits must be kept low.

Bit 6 and bit 7 are not used and can assume any value.

2.2.6 Load rate register

Addressing the load rate register function causes the TMS 5501 to load the rate register from the data bus and interpret the data bus (See Figure 5) as follows.

BIT	7	6	5	4	3	2	1	0
	STOP BIT(s)	9600 baud	4800 baud	2400 baud	1200 baud	300 baud	150 baud	110 baud

H: One stop bit
L: Two stop bits

FIGURE 5—DATA BUS ASSIGNMENTS FOR RATE COMMANDS

Bits 0 through 6, rate select

The rate select bits (bits 0 through 6) are mutually exclusive, i.e., only one bit may be high. A high in bits 0 through 6 will select the baud rate for both the transmitter and receiver circuitry as defined below and in Figure 5:

Bit 0	110 baud
Bit 1	150 baud
Bit 2	300 baud
Bit 3	1200 baud
Bit 4	2400 baud
Bit 5	4800 baud
Bit 6	9600 baud

If more than one bit is high, the highest rate indicated will result. If bits 0 through 6 are all low, both the receiver and the transmitter circuitry will be inhibited.

Bit 7, stop bits
Bit 7 determines whether one or two stop bits are to be used by both the transmitter and receiver circuitry. A high in bit 7 selects one stop bit. A low in bit 7 selects two stop bits.

2.2.7 Load transmitter buffer

Addressing the load-transmitter-buffer function transfers the state of the data bus into the transmitter buffer.

2.2.8 Load output port

Addressing the load-output-port function transfers the state of the data bus into the output port. The data is latched and remains on $\overline{XO\ 0}$ through $\overline{XO\ 7}$ as the complement of the data bus until new data is loaded.

2.2.9 Load mask register

Addressing the load-mask-register function loads the contents of the data bus into the mask register. A high in data bus bit n enables interrupt n. A low inhibits the corresponding interrupt.

2.2.10 Load timer n

Addressing the load-timer-n function loads the contents of the data bus into the appropriate interval timer. Time intervals of from 64 μs (data bus = LLLLLLLH) to 16,320 μs (data bus HHHHHHHH) are counted in 64-μs steps. When the count of interval timer n reaches 0, the bit in the interrupt register that corresponds to timer n is set and an interrupt is generated. Loading all lows causes an interrupt immediately.

3. TMS 5501 ELECTRICAL AND MECHANICAL SPECIFICATIONS

3.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	−0.3 V to 20 V
Supply voltage, V_{DD} (see Note 1)	−0.3 V to 20 V
Supply voltage, V_{SS} (see Note 1)	−0.3 V to 20 V
All input and output voltages (see Note 1)	−0.3 V to 20 V
Continuous power dissipation	1.1 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the normally most-negative supply voltage, V_{BB} (substrate). Throughout the remainder of this data sheet, voltage values are with respect to V_{SS} unless otherwise noted.

3.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−4.75	−5	−5.25	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	2.2		V_{CC}+1	V
High-level clock input voltage, $V_{IH(\phi)}$	9		V_{DD}+1	V
Low-level input voltage, V_{IL} (all inputs except clocks) (see Note 2)	−1		0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$ (see Note 2)	−1		0.8	V
Operating free-air temperature, T_A	0		70	°C

NOTE 2: The algebraic convention where the most negative limit is designated as minimum is used in this specification for logic voltage levels only.

3.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITION (UNLESS OTHERWISE NOTED)

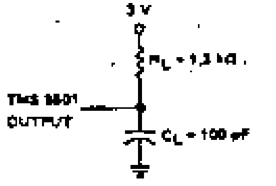
PARAMETER		TEST CONDITIONS		MIN	MAX	UNIT
I_I	Input current any input except clocks and data bus	V_I = 0 V to V_{CC}			±10	µA
$I_{I(\phi)}$	Clock input current	$V_{I(\phi)}$ = 0 V to V_{DD}			±10	µA
$I_{I(DB)}$	Input current, data bus	$V_{I(CE)}$ = 0 V to V_{CC},	CE at 0 V		−50	µA
V_{OH}	High-level output voltage	I_{OH} = −400 µA		2.7		V
V_{OL}	Low-level output voltage	I_{OL} = 1.7 mA,			0.45	V
$I_{BB(av)}$	Average supply current from V_{BB}	Operating at $t_{c(\phi)}$ = 480 ns, T_A = 25°C			−1	mA
$I_{CC(av)}$	Average supply current from V_{CC}				100	
$I_{DD(av)}$	Average supply current from V_{DD}				45	
C_i	Capacitance, any input except clock	V_{CC} = V_{DD} = V_{SS} = 0 V, V_{BB} = −4.75 to −5.25 V, All other pins at 0 V	f = 1 MHz,		10	pF
$C_{i(\phi)}$	Clock input capacitance				15	
C_o	Output capacitance				25	

3.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 5 AND 6)

		MIN	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	480	2000	ns
$t_{r(\phi)}$	Clock rise time	5	50	ns
$t_{f(\phi)}$	Clock fall time	5	50	ns
$t_{w(\phi1)}$	Pulse width, clock 1 high	60		ns
$t_{w(\phi2)}$	Pulse width, clock 2 high	220	350	ns
$t_{d(\phi1L-\phi2)}$	Delay time, clock 1 low to clock 2	0		ns
$t_{d(\phi2-\phi1)}$	Delay time, clock 2 to clock 1	70		ns
$t_{d(\phi1H-\phi2)}$	Delay time, clock 1 high to clock 2 (time between leading edges)	80		ns
$t_{su(ad)}$	Address setup time	50		ns
$t_{su(CE)}$	Chip enable setup time	150		ns
$t_{su(da)}$	Data setup time	150		ns
$t_{su(sync)}$	Sync setup time	50		ns
$t_{su(X1)}$	External input setup time	50		ns
$t_{h(ad)}$	Address hold time	0		ns
$t_{h(CE)}$	Chip enable hold time	10		ns
$t_{h(da)}$	Data hold time	10		ns
$t_{h(sync)}$	Sync hold time	10		ns
$t_{h(X1)}$	External input hold time	40		ns
$t_{w(sens\ H)}$	Pulse width, sensor input high	500		ns
$t_{w(sens\ L)}$	Pulse width, sensor input low	500		ns
$t_{d(sens-int)}$	Delay time, sensor to interrupt (time between leading edges)		2000	ns
$t_{d(rst-int)}$	Delay time, RST instruction to interrupt (time between trailing edges)		500	ns

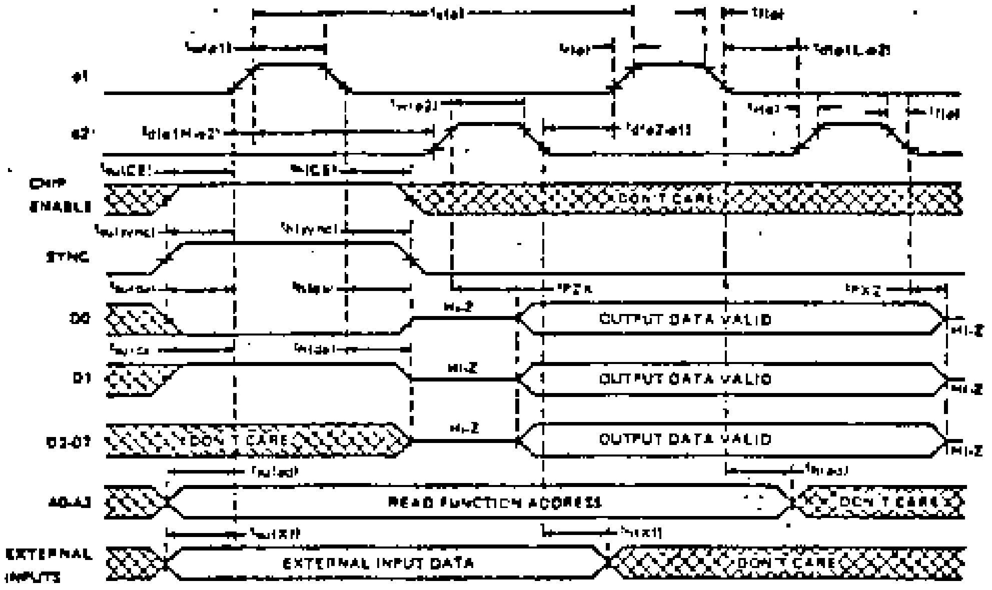
3.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 6 AND 7)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{PZX}	Data bus output enable time	C_L = 100 pF, R_L = 1.3 kΩ		200	ns
t_{PXZ}	Data bus output disable time to high-impedance state			180	ns
t_{PD}	External data output propagation delay time from φ2			300	ns



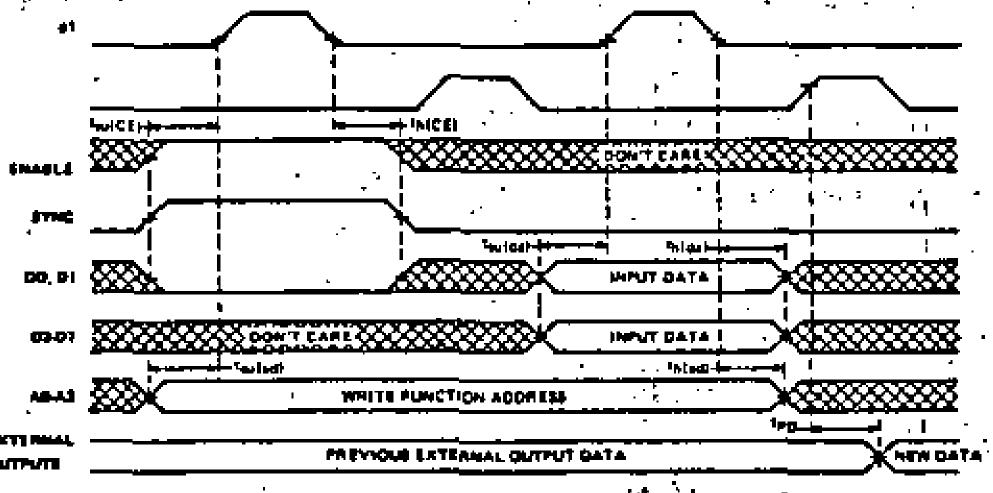
C_L includes probe and jig capacitance

LOAD CIRCUIT



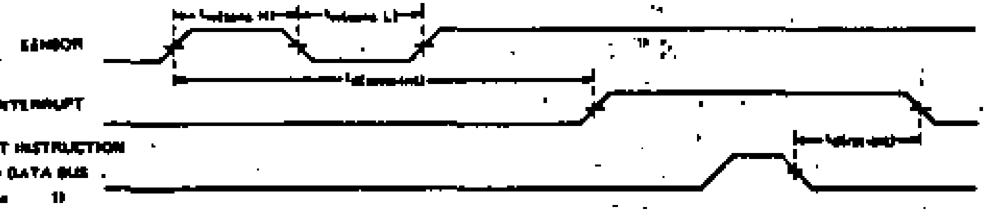
NOTE: For all φ in all figures, high and low timing points are 90% and 10% of $V_{IH(\phi)}$. All other timing points are the 50% level.

FIGURE 6—READ CYCLE TIMING



NOTE: For φ1 and φ2 inputs, high and low timing points are 90% and 10% of $V_{IH(φ)}$. All other timing points are the 50% level.

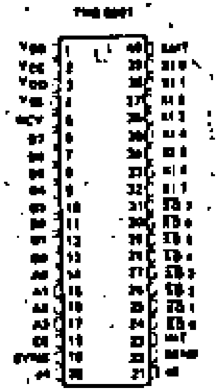
FIGURE 7—WRITE CYCLE TIMING



NOTES 1. The RST instruction occurs during the output data valid time of the read cycle.
2. All timing points are 50% of V_{DD}.

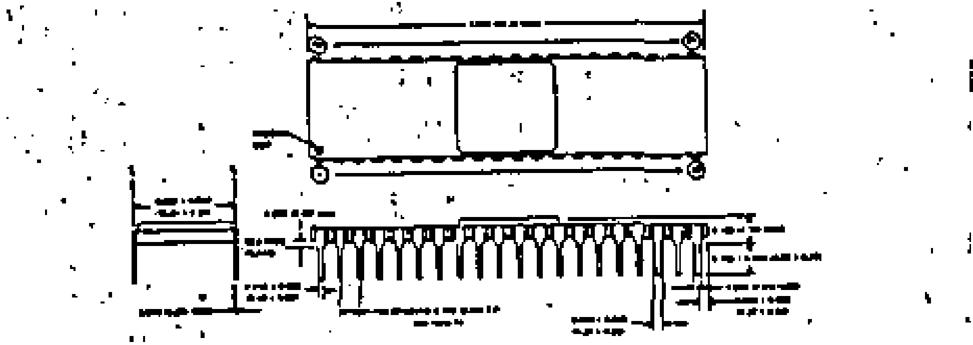
FIGURE 8—SENSOR/INTERRUPT TIMING

2.6 TERMINAL ASSIGNMENTS



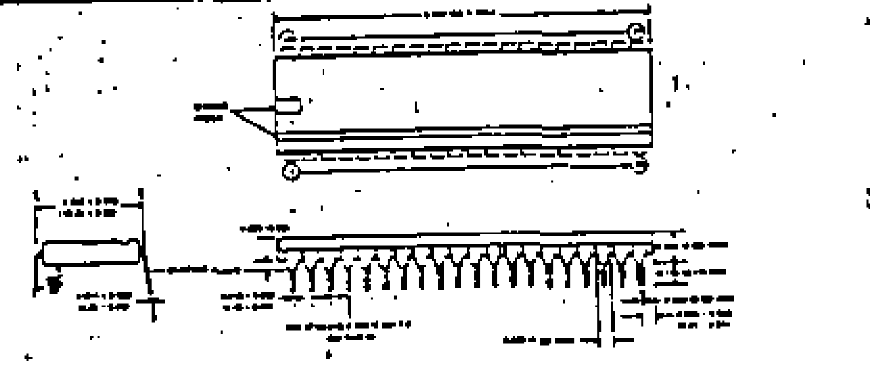
2.7 MECHANICAL DATA

TMS 5501 JL—40-PIN CERAMIC PACKAGE



NOTES: A. Each pin centerline is located within 0.010 inch (0.26 millimeters) of its true longitudinal position.
B. All linear dimensions are shown in inches (and parenthetically in millimeters for reference only). Inch dimensions govern.

TMS 5501 NL—40-PIN PLASTIC PACKAGE



NOTES: A. Each pin centerline is located within 0.010 inch (0.26 millimeters) of its true longitudinal position.
B. All linear dimensions are shown in inches (and parenthetically in millimeters for reference only). Inch dimensions govern.

This proposed standard eliminates many of the problems in the S-100 bus and upgrades it for 16-bit microprocessors. It is offered here for public comment before submission to the IEEE Standards Board.

Standard Specification for S-100 Bus Interface Devices

IEEE Task 696.1 / D2

Kells A. Elmquist, InterSystems Inc.
Howard Fullmer, Parasitic Engineering Inc.
David B. Gustavson, Stanford Linear Accelerator Center
George Morrow, Thinker Toys

Introductory comments by
Robert G. Stewart, Chairman,
IEEE-CS Computer Standards Committee

The following draft of a proposed standard for the S-100 bus is the culmination of over a year and a half of effort to eliminate many of the bus's problems and to upgrade it to be suitable for 16-bit microprocessors. The address bus has been extended to 24 bits, the data in and data out buses ganged to form a 16-bit wide data bus for 16-bit transactions, and two additional handshaking lines added to permit intermixing of 8- and 16-bit memory cards.

A binary encoded multiple master arbitration bus permits up to 16 masters on the bus. The necessary logic can be implemented in one chip. Additional ground lines, a power fail line, and an error line have been added. Three lines termed NDEF—for not to be defined—have been allotted to allow leeway to implementers for specialized use. Such use must be specified in all literature. Five lines are RFU—reserved for future use. Some lines formerly used for front panel purposes have been deleted, with the intention that such lines can best be handled by a jumper cable from the CPU card to the front panel. A DMA protocol is specified which provides overlap of the control lines at the beginning and end of the transition between permanent and temporary masters. This allows the address, data, and control buses to settle before information is transferred.

As a bit of personal testimony, I implemented the new DMA protocol on my own system, which includes a Digital Systems dual floppy disk interfaced to a MITS Altair 6800, using DMA for disk transfers. The soft error rate, presumably due to glitching on the positive true logic lines, dropped from a situation where a file would be seriously munged in a few hours to the present situation where I can work for days on end without an observable error.

We have observed a new typographic convention in publishing the proposed standard. The use of an overbar to denote electrically low active or negative true logic lines has been replaced by a postfix asterisk to avoid confusion with Boolean negation and permit typing on word processing systems. This is verbalized by the word "star," replacing the prior word "bar." The Boolean negation overbar can be optionally replaced by a prefix minus sign, with parentheses if needed.

The named authors of the standard were evenly divided as to whether the asterisk should be included in logic equations and state diagrams as well as in electrical signal names and timing diagrams. Two authors believe that the asterisk, when thought of as a designator rather than as the negation operator, adds clarity and consistency, and lessens the need to remember or look up the electrically active level when converting from logic to electrical representations. Such use makes logic state diagrams more directly useful for interpreting oscilloscope or logic analyzer waveforms.

The other two authors feel that the inclusion of the asterisk in the name of a logic state or variable is likely to carry with it the implication of logical negation, thus causing the logic statements to be interpreted incorrectly. Furthermore, they assert that many designers think mostly in terms of electrical levels, with high being true, which again causes logic statements to be interpreted incorrectly. They propose to resolve this hazard by removing the electrical information, i.e., the asterisk, from the variable name when it is us-

0018-9162/79/0700-0076$00.75 © 1979 IEEE

ed in a logic context as opposed to an electrical or timing context.

A compromise has been reached where the asterisk is not used in the context of logic equations, but is included elsewhere in the document. We solicit feedback from the readers on these two points of view.

The S-100 bus subcommittee has been ably chaired by George Morrow and Howard Fullmer. Both of them provided invaluable technical insights which have been incorporated throughout the draft standard. John Walker of Marinchip Systems suggested the method of using 16-bit memory and interface cards interchangeably with 8-bit cards. David Gustavson and Leo Paffrath of SLAC suggested the bus arbitration scheme which has also been implemented on the Department of Energy's Fastbus. Howard Fullmer suggested the DMA overlap protocol which lowers glitching noise. Kells Elmquist of InterSystems offered a critique of the draft published in May 1978 in *Computer* and provided many useful suggestions for improvement. He carefully investigated numerous timing and electrical alternatives and resolved many open questions relating to the standard. Kells wrote the final version of the draft for submission to and revision by the subcommittee.

The IEEE Computer Society is publishing this standard in draft form to allow you to comment upon it prior to submission to the IEEE Standards Board for adoption as an IEEE standard. For example, should the data bus be extended to 32 bits, and if so, how? Your comments should be sent to George Morrow by August 15, 1979, with copies to Gordon Force. Mr. Morrow's address is:

George Morrow
Thinker Toys
5221 Central Avenue
Richmond, California 94804

If you would like to participate in other standardization efforts of the Microprocessor Standards Committee, please contact its chairman:

Gordon Force
Logical Solutions
1128 Amur Creek Court
San Jose, California 95051

Finally, preparation of this proposed standard has benefited from the contributions of many individuals and companies. We indeed thank them all. ■

The proposed standard

1.0 General

1.1 Scope

This standard applies to interface systems for computer system components interconnected via a 100-line parallel backplane commonly known as the S-100 bus.

It applies to microprocessor computer systems, or portions of them, where

1) Data exchanged among the interconnected devices is digital (as distinct from analog).
2) The total number of interconnected devices is small (22 or fewer).
3) The total transmission path length among interconnected devices is electrically short (25" or less). That is, transmission line propagation delays are not important.
4) The maximum data rate of any signal on the bus is low (less than or equal to 6 MHz).

1.2 Object

This standard is intended:

1) To define a rational, general-purpose interface system for designers of new computer system components that will ensure their compatibility with present and future S-100 computer systems.
2) To provide the microprocessor computer system user with compatible device families which will communicate in an unambiguous way without modification, from which a modularly expandable computer system may be constructed.
3) To enable the interconnection of independently manufactured devices into a single system.
4) To specify terminology and definitions related to the system.
5) To define a system with the minimum number of restrictions on the performance characteristics of devices connected to the system.
6) To define a system that, of itself, is of relatively low cost, and allows the interconnection of low cost devices.
7) To define a system that is easy to use.

1.3 Definitions

The following definitions apply for the purpose of this standard. This section contains only general definitions. Detailed definitions are given in other sections as appropriate.

1.3.1 General system terms

Compatibility. The degree to which devices may be interconnected and used without modification, when designed as defined in Sections 2, 3, and 4 of this standard.

Interface. A shared boundary between parts of a computer system, through which information is conveyed.

Interface system. The device independent functional, electrical, and mechanical elements of an interface necessary to effect unambiguous communication among a set of devices. Driver and receiver circuits, signal line descriptions, timing and control conventions, message transfer protocols, and functional logic circuits are typical interface system elements.

System. A set of interconnected elements constituted to achieve a given objective by performing specified functions.

1.3.2 Signals and paths

Assert. To drive a signal line to the true state. The true state is either a high or low state, as specified for each signal.

Bidirectional bus. A bus used by any individual device, or set of devices, for the two-way transmission of messages, that is, both input and output.

Bit-parallel. A set of concurrent data bits present on a like number of signal lines used to carry information. Bit-parallel data bits may be acted upon concurrently as a group or independently as individual data bits.

Bus. A set of signal lines used by an interface system, to which a number of devices are connected, and over which messages are carried.

Byte. A set of bit-parallel signals corresponding to binary digits operated on as a unit. Connotes a group of eight bits where the most significant bit carries the subscript 7 and the least significant bit carries the subscript 0.

Byte-serial. A sequence of bit-parallel data bytes used to carry information over a common bus.

High state. The electrically more positive signal level used to assert a specific message content associated with one of two binary logic states.

Low state. The electrically less positive signal level used to assert a specific message content associated with one of two binary logic states.

Signal. The physical representation which conveys data from one point to another. For the purpose of this standard, this applies to digital electrical signals only.

Signal level. The magnitude of a signal when considered in relation to an arbitrary reference magnitude (voltage in the case of this standard).

Signal line. One of a set of signal conductors in an interface system used to transfer messages among interconnected devices.

Signal parameter. That parameter of an electrical quantity whose values or sequence of values convey information.

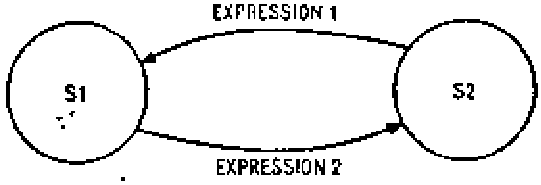
Unidirectional bus. A bus used by a device for one-way transmission of messages, that is, either input only or output only.

Word. A set of bit-parallel signals corresponding to binary digits and operated on as a unit. Usually connotes a group of 16 bits where the most significant bit carries the subscript 15 and the least significant bit carries the subscript 0.

1.4 State diagram notation

Each state that an interface function can assume is represented graphically by a circle. A mnemonic is used within the circle to identify the state.

All permissible transitions between states of an interface function are represented graphically by arrows between them. Each transition between states may be qualified by an expression whose value must be either true or false. If a state transition is not qualified by an expression it is assumed that transition from one state to another will occur after a minimum time period, as indicated in the timing specifications. An interface function must enter the state pointed to if and only if the driving expression becomes true, or in the case of a time dependent transition, as soon as the minimum specified time has passed.



An expression consists of two parts, a driving expression and a driven expression, separated by a slash (/). The driving expression is mandatory and specifies the conditions necessary for the state transition. The driven expression is optional and is used to indicate signal transitions as a result of the state transition. A signal transition is indicated by the signal name followed by an equal sign (=), followed by an indication of the state attained by the signal as a result of the transition. A driving expression consists of one or more messages used in conjunction with the operators AND (a · b), OR (a+b), and NOT (—a). Precedence is defined by parentheses. An example expression is: (driving/driven)

A · (B+C) / D=F(ALSE), E=T(RUE)

If A AND (B OR C) is true, then D is forced false and E is forced true, and the state transition takes place.

1.5 Logical and electrical state relationships

This standard makes a distinction between the logical function of a signal and its electrical implementation. All equations in this standard are logic equations, not electrical equations (unless otherwise stated), and are written in terms of logic states. The use of the term "active" for the purpose of this standard is synonymous with the logic state true.

There are two types of electrical implementation of the logic states:

Active high signals. Active high signals are represented without a suffix after the signal name mnemonic (i.e. ABCD).

LOGIC STATE	BINARY STATE	ELECTRICAL SIGNAL LEVEL	ELECTRICAL STATE
FALSE (F)	0	CORRESPONDS TO ≤.8 V, CALLED THE LOW STATE.	L
TRUE (T)	1	CORRESPONDS TO ≥2.0 V, CALLED THE HIGH STATE.	H

Active low signals. Active low signals are represented with an asterisk suffix after the mnemonic (i.e. ABCD*).

LOGIC STATE	BINARY STATE	ELECTRICAL SIGNAL LEVEL	ELECTRICAL STATE
FALSE (F)	0	CORRESPONDS TO >2.0 V, CALLED THE HIGH STATE.	H
TRUE (T)	1	CORRESPONDS TO <.8 V, CALLED THE LOW STATE.	L

In translating a logic equation into an electrical implementation, care must be taken to account for the active-high or active-low character of the electrical signal. For example, the logic equation

$$\mathrm{MWRT} = \mathrm{pWR} \cdot -\mathrm{sOUT}, \text{ (logic equation)}$$

when implemented electrically, becomes

$$\mathrm{MWRT} = (-\mathrm{pWR}^*) \cdot -\mathrm{sOUT}, \text{ (electrical equation)}$$

since pWR* is the electrical signal carrying the pWR information on the bus.

Note that this is equivalent to

$$\mathrm{MWRT} = -(\mathrm{pWR}^* + \mathrm{sOUT}), \text{ (electrical equation)}$$

by deMorgan's theorem; consequently, a single two-input NOR gate is sufficient to implement MWRT, if it meets the loading and drive requirements.

The edge or change of electrical value of an electrical signal on a timing diagram which causes a transition change of the variable as a logic variable from false to true is:

Signal	*Edge*
active high	rising
active low	falling

Logic equations in state diagrams are written in terms of logic state, not electrical state.

The suffix asterisk "*" is not a negation operator. It is a designator (like a comment or footnote) attached to a name, telling the reader what the relationship is between the truth state and the electrical state. That is, this variable is true when the line on the bus is low.

A prefix minus sign "−" represents the logical negation operator and is equivalent to the use of an overbar. Parentheses are used to enclose the negated variable when required for clarity.

1.6 Interface system overview

1.6.1 Interface system objective

The overall purpose of the interface system is to provide an effective communication link over which messages are carried in an unambiguous way among a group of interconnected devices.

Messages in an interface system belong to either of two broad categories:

1) Messages used to manage the interface system itself, called interface messages.
2) Messages used by the devices interconnected by the interface system, and carried by that system, but not part of the interface system itself (i.e. data). These are called device dependent messages.

The interface system herein described comprises the necessary functional and electrical specifications for interface messages to effect the objective of this standard, but it is beyond the scope of this standard to specify the nature or meaning (other than electrical signal level) of device dependent messages.

1.6.2 Fundamental communication capabilities

An effective communication link requires two basic functional elements to organize and manage the flow of information among devices:

1) A device acting as a bus master.
2) A device acting as a bus slave.

All data transfer communications between a bus master and a bus slave are carried out in terms of a generalized bus cycle generated by the bus master and responded to by the addressed bus slave.

In the context of the interface system described by this standard:

1) A device acting as a bus master has the capability to address all bus slaves, or some portion of them, by generating all interface messages necessary to effect a bus cycle, and has the capability to transfer device dependent messages to or from the addressed slave as a part of that bus cycle.
2) A device acting as a bus slave monitors all bus cycles, and has the capability, thus, to be addressed by the bus master and to transfer device dependent messages to or from the bus master.

Bus master and bus slave capabilities occur both individually and collectively in devices interconnected via the S-100 interface system.

1.6.3 Message paths and bus structure

The S-100 interface system consists of a set of signal lines used to carry all information, interface messages and device dependent messages among interconnected devices.

The bus structure is organized into eight sets of signal lines:

1) Data bus—	16 signal lines.
2) Address bus—	16 or 24 signal lines.
3) Status bus—	8 signal lines.
4) Control output bus—	5 signal lines.
5) Control input bus—	6 signal lines.
6) DMA control bus—	8 signal lines.
7) Vectored interrupt bus—	8 signal lines.
8) Utility bus—	20 signal lines.

2.0 Functional specification

2.1 Functional partition

Functional devices interconnected via the interface system are divided into two broad classifications, bus masters and bus slaves, according to their relationship to the generation and reception of interface messages.

Devices acting as bus masters are responsible for the initiation of all bus cycles, and for the generation of all signals necessary for the conduction of an unambiguous bus cycle. These signals are termed type M signals, and consist of the address, status, and control buses. Device dependent messages are transmitted and received on the data bus.

Bus masters are subdivided into two classifications, permanent masters and temporary masters. A permanent bus master (generally a CPU) is the highest priority master in the interface system. A temporary master may request the bus from the permanent master for an arbitrary number of bus cycles, and then returns control of the bus to the permanent master. The transfer of bus control from a permanent master to a temporary master and back to the permanent master is termed a DMA cycle.

The difference between a permanent bus master and a temporary bus master is that:

1) Only one permanent master may exist within the interface system, whereas up to 16 temporary masters may co-exist in a single system.
2) A temporary master is not subject to a DMA cycle, that is, there are no nested DMA operations.

Devices acting as bus slaves are bus cycle receptors. A bus slave monitors all bus cycles and, if addressed during a particular bus cycle, accepts or sends the requested device dependent message on the data lines. While bus masters must generate a specific set of signals in order to assure an unambiguous bus cycle, a bus slave need only examine and generate that subset of bus signals necessary to communicate with bus masters.

2.2 Signal lines

2.2.1 General

The bus is a collection of message paths defined relative to the current bus master. They are:

1) Address bus.
2) Status bus.
3) Data input/output bus.
4) Control output bus.
5) Control input bus.
6) DMA control bus.
7) Vectored interrupt bus.
8) Utility bus.

The nature and use of each bus is specified in the following sections.

2.2.2 Address bus

The address bus consists of 16 or 24 bit-parallel signal lines used to select a specific location in memory or a specific input/output device for communication during the current bus cycle.

All bus masters must assert at least 16 address bits, but may assert 24 address bits if extended address capability is desired. Validity of the address bus is defined in 2.7.3.

Table 1 summarizes address usage for various bus cycles.

Table 1.
Address usage for different bus cycles.

CYCLE TYPE	STANDARD ADDRESSING	EXTENDED ADDRESSING
MEMORY READ MEMORY WRITE M1 (OP-CODE FETCH)	A0-A15	A0-A23
INPUT OUTPUT	A0-A7†	A0-A15
INTERRUPT ACKNOWLEDGE	NONE	NONE
HALT ACKNOWLEDGE	NONE	NONE

† see 2.2.2.3

2.2.2.1 Standard memory addressing

The standard memory address bus consists of 16 lines specifying 1 of 64K memory locations. These 15 lines are named A0 through A15, where A15 is the most significant bit.

2.2.2.2 Extended memory addressing

The extended memory address bus consists of 24 lines specifying 1 of 16 million memory locations. These 24 lines are named A0 through A23, where A23 is the most significant bit.

2.2.2.3 Standard input/output device addressing

The standard I/O device address bus consists of 8 lines, A0 through A7, specifying 1 of 256 I/O devices. A7 is the most significant bit.

NOTE: The I/O device address has traditionally been duplicated onto the high order address byte, A15-A8. While this is considered acceptable procedure, it is not recommended for new designs as it complicates expansion to extended I/O device addressing.

2.2.2.4 Extended input/output device addressing

The extended I/O device address bus consists of 16 lines, A0 through A15, specifying 1 of 64K devices. A15 is the most significant bit.

2.2.3 Status bus

The status bus consists of eight lines which identify the nature of the bus cycle in progress, and qualify the nature of the address on the address bus.

The mnemonics for status lines always begin with a lower-case s.

The 8 status lines are:

1) Memory read— sMEMR.
2) Op-code fetch— sM1.
3) Input— sINP.
4) Output— sOUT.
5) Write cycle— sWO*.
6) Interrupt acknowledge— sINTA.
7) Halt acknowledge— sHLTA.
8) Sixteen-bit data transfer request— sXTRQ*.

The 8 lines on the status bus must be generated by the current bus master.

Validity of the status bus is given in 2.7.3.

2.2.3.1 Status memory write

One relevant status signal is not directly available on the bus, but may be created by the combination of two others. Status Memory Write is defined as:

sMemory Write =
(−sOUT) · sWO, (logic equation)

that is, status memory write is true when sOUT is false and sWO (write) is true.

2.2.3.2 Status usage chart

Table 2 gives the status word definition for all possible bus cycles. (W) refers to word (16-bit data path) operations; (B) refers to byte (8-bit data path) operations. H=high state. L=low state. X=don't care.

Table 2.
Status usage chart.

STATUS BITS		sMEMR	sM1	sWO*	sOUT	sINP	sINTA	sHLTA	sXTRQ*
CYCLE TYPE									
MEMORY READ	(B)	H	L	H	L	L	L	L	H
	(W)	H	L	H	L	L	L	L	L
OP-CODE FETCH	(B)	H	H	H	L	L	L	L	H
	(W)	H	H	H	L	L	L	L	L
MEMORY WRITE	(B)	L	L	L	L	L	L	L	H
	(W)	L	L	L	L	L	L	L	L
OUTPUT	(B)	L	L	L	H	L	L	L	H
	(W)	L	L	L	H	L	L	L	L
INPUT	(B)	L	L	H	L	H	L	L	H
	(W)	L	L	H	L	H	L	L	L
INTERRUPT ACKNOWLEDGE	(B)	L	X	H	L	L	H	L	H
	(W)	L	X	H	L	L	H	L	L
HALT ACKNOWLEDGE		X	X	H	L	L	L	H	X

WHERE:

H = HIGH STATE
L = LOW STATE
X = DON'T CARE
W = 16-BIT OPERATION
B = 8-BIT OPERATION

2.2.4 Data bus

Data input and data output are always specified relative to the current bus master. Data transmitted by the current bus master to a bus slave is called data output. Data received by the current bus master from a bus slave is called data input.

The data bus consists of 16 lines grouped as two unidirectional 8-bit buses for byte operations and as a single bidirectional bus for 16-bit word operations.

2.2.4.1 Byte operations

Two unidirectional 8-bit buses are used for byte data transfers. Data output appears on the data output bus (DO0-DO7), where DO7 is the most significant bit.

Data input appears on the data input bus (DI0-DI7), where DI7 is the most significant bit.

2.2.4.2 Word operations

For 16-bit data transfers the DI and the DO buses are ganged together, creating a single 16-bit bidirectional bus. Two signal lines control the ganging of the data buses, sixteen request (sXTRQ*) and sixteen acknowledge (SIXTN*). When both of these lines are true (in the low state), the data buses are ganged with DO0 corresponding to DATA 0 and DI7 corresponding to DATA 15, the most significant bit.

Complete specification of the 8/16-bit protocol is given in 2.6.

2.2.5 Control output bus

The 5 lines of the control output bus determine the timing and movement of data during any bus cycle. The mnemonics for the control output lines always begin with a lower-case p.

The five lines are:

1) pSYNC, which indicates the start of a new bus cycle.
2) pSTVAL*, which in conjunction with pSYNC indicates that stable address and status may be sampled from the bus in the current cycle.
3) pDBIN, a generalized read strobe that gates data from an addressed slave onto the data bus.
4) pWR*, a generalized write strobe that writes data from the data bus into an addressed slave.
5) pHLDA, the hold acknowledge signal that indicates to the highest priority temporary master that the permanent master is relinquishing control of the bus.

The control output signals are subject to the functional and timing disciplines given in 2.7, 3.8, and 3.9.

2.2.6 Control input bus

The six lines of the control input bus allow bus slaves to synchronize the operations of bus masters with conditions internal to the bus slave (e.g., data not ready), and to request operations of the permanent master (e.g., interrupt or hold).

The six control input lines are:

1) RDY
2) XRDY

3) INT*
4) NMI*
5) HOLD*
6) SIXTN*

2.2.6.1 Ready lines

The ready lines are used by bus slaves to synchronize bus masters to the response speed of the slave. Thus cycles are suspended and wait states inserted until both ready lines are asserted.

The RDY line is the general ready line for bus slaves. It is specified as an open collector line.

The XRDY line is a special ready line commonly used by front panel devices to stop and single step bus masters. As it is not specified as an open collector line, it should not be used by other bus slaves, since a bus conflict may exist.

2.2.6.2 Interrupt lines

The two interrupt lines, INT* and NMI*, are used to request service from the permanent bus master.

The INT* line may be masked off by the bus master, usually via an internal software operation. If the master accepts the interrupt request on the INT* line, it may respond with an interrupt acknowledge bus cycle, accepting vectoring information from the data bus. The INT* line is often implemented as a "group interrupt" line in conjunction with the vectored interrupt bus. In this case, INT* indicates the presence of one or more vectored interrupt requests.

The NMI* line is a non-maskable interrupt request line, that is, it may not be masked off by the bus master. Accepting an interrupt on the NMI* line need not generate an interrupt acknowledge bus cycle.

An interrupt request on the INT* line is asserted as a level, that is, the line is asserted until interrupt service is received. An interrupt request on the NMI* line, on the other hand, is asserted as a negative going edge, since no interrupt acknowledge cycle need be generated.

Both these lines are specified as open collector lines.

2.2.6.3 Hold request

The hold request line, HOLD*, is used by temporary bus masters to request control of the bus from the permanent bus master. The HOLD* line may be masked by the permanent bus master to prevent temporary masters from gaining bus control.

The HOLD* line is specified as an open collector line, and may only be asserted at certain times. See 2.8.3.

2.2.6.4 Sixteen acknowledge

The sixteen acknowledge line, SIXTN*, is a response to the status signal sixteen request (sXTRQ*), and indicates that the requested 16-bit data transfer is possible.

The SIXTN* line is specified as an open collector line. Detailed specification of the use of this line is given in 2.6.

2.2.7 DMA control bus

The eight lines of the DMA control bus are used in conjunction with control bus signals HOLD* and pHLDA. They arbitrate among simultaneous requests for control of the bus by temporary masters and disable the signal drivers of the permanent bus master, thus effecting an orderly transfer of bus control.

All eight lines of the DMA control bus are specified as open collector lines.

The eight DMA control lines are:

1) DMA0*
2) DMA1*
3) DMA2*
4) DMA3*
5) ADSB*
6) DODSB*
7) SDSB*
8) CDSB*

Detailed specification of the use of these lines is given in 2.8.

2.2.7.1 DMA arbitration

The four lines that arbitrate among simultaneous requests for bus control by temporary masters are DMA0* through DMA3*. The encoded priority of requesters is asserted on these lines and, after settling, they contain the priority number of the highest priority requester.

Detailed specification of this process is given in 2.8.3.

2.2.7.2 Bus transfer signals

Four signals are available on the bus to disable the line drivers of the permanent bus master. They are:

1) ADSB*, address disable.
2) DODSB*, data out disable.
3) SDSB*, status disable.
4) CDSB*, control output disable.

Use of these lines is tightly specified during the transfer of the bus from a permanent master to a temporary master, as given in 2.8.2, and any transfer involving the control output lines should follow a similar protocol.

The address, data, and status signals from the permanent master may be disabled and replaced using these signals as long as the contents of these buses is valid for the current bus cycle as though no replacement had occurred.

2.2.8 Vectored interrupt bus

The eight lines of the vectored interrupt bus are used in conjunction with the generalized vectored in-

terrupt request, INT*, to arbitrate among eight levels of interrupt request priorities. They are typically implemented as inputs to a bus slave which masks and prioritizes the requests, asserts the generalized interrupt request to the permanent bus master, and responds to the interrupt acknowledge bus cycle with appropriate vectoring data.

The eight lines of the vectored interrupt bus are VI0* through VI7*, where VI0* is considered the highest priority interrupt.

The vectored interrupt lines should be implemented as levels, that is, they should be held active until service is received.

2.2.9 System utilities

2.2.9.1 System power

Power in S-100 systems is distributed to bus devices as unregulated voltages. A total of nine bus lines are used:

1) +8 volts, 2 lines.
2) +16 volts, 1 line.
3) −16 volts, 1 line.
4) GROUND, 5 lines.

Ground lines are distributed across the edge connector such that low impedance grounds are available on both sides of the edge connector, and on both sides of the circuit cards.

Power lines are subject to the specifications given in 3.2.

2.2.9.2 System clock

The system clock, Φ, is generated by the permanent master. The control timing for all bus cycles, whether they are cycles of the permanent master or cycles of temporary masters in control of the bus, must be derived from this clock.

This signal is never transferred during a bus exchange operation.

2.2.9.3 CLOCK

This clock is specified as a 2-MHz (0.5 percent tolerance) signal with no relationship to any other bus signal. It is to be used by counters, timers, baud-rate generators, etc.

2.2.9.4 System reset functions

System reset functions are divided into three lines:

1) RESET*, resets all bus masters.
2) SLAVE CLR*, resets all bus slaves.
3) POC*, power-on clear is active only on power-on, and asserts SLAVE CLR* and RESET*.

The POC* signal is specified as having a minimum active period of 10 msec.

RESET* and SLAVE CLR* are specified as open collector lines.

2.2.9.5 Memory write strobe

The memory write strobe, MWRT, must be generated somewhere in the system. It is usually generated by front panel type devices, but is optionally generated by permanent masters or mother boards in systems without front panels. Care must be taken that it is generated at only one point in a given system.

Memory write is defined as:

MWRT = pWR · —sOUT (logic equation)

2.2.9.6 Phantom slaves

A line, PHANTOM*, is provided for overlaying bus slaves at a common address location. When this line is activated phantom bus slaves are enabled and normal bus slaves are disabled.

This line is specified as an open collector line.

2.2.9.7 Error

The line ERROR* is a generalized error line that is asserted when an error of some sort (i.e., parity, write to protected memory) is occurring in the current bus cycle.

This line is specified as an open collector line.

2.2.9.8 Manufacturer specified lines

Three lines which can be specified by individual manufacturers are provided on the bus. These lines, termed NDEF (not to be defined), should only be implemented as options, and shall be provided with jumpers so that possible conflicts may be eliminated.

Any manufacturer MUST specify in detail any use of these lines. Signals on these lines are limited to 5 volt logic levels.

2.2.9.9 Power fail (PWRFAIL)*

The power fail line indicates impending power failure, and remains true until power is restored and POC* is true.

2.2.9.10 Reserved lines (RFU)

The five remaining lines are reserved for future use and may not be used for any purpose.

2.2.10 Pin list

Pin connections to the card edge connector shall conform to the list given in Table 3.

2.3 The permanent master interface

2.3.1 General

The permanent master interface provides the capability to transfer device dependent messages to and from all bus slaves. It is responsible for the genera-

Table 3. S-100 bus pin list.

PIN NO.	SIGNAL & TYPE	ACTIVE LEVEL		DESCRIPTION
1	+8 VOLTS (B)			Instantaneous minimum greater than 7 volts, instantaneous maximum less than 25 volts, average maximum less than 11 volts.
2	+16 VOLTS (B)			Instantaneous minimum greater than 14.5 volts, instantaneous maximum less than 35 volts, average maximum less than 21.5 volts.
3	XRDY (S)	H		One of two ready inputs to the current bus master. The bus is ready when both these ready inputs are true. See pin 72.
4	VI0*(S)	L	O.C.	Vectored interrupt line 0.
5	VI1*(S)	L	O.C.	Vectored interrupt line 1.
6	VI2*(S)	L	O.C.	Vectored interrupt line 2.
7	VI3*(S)	L	O.C.	Vectored interrupt line 3.
8	VI4*(S)	L	O.C.	Vectored interrupt line 4.
9	VI5*(S)	L	O.C.	Vectored interrupt line 5.
10	VI6*(S)	L	O.C.	Vectored interrupt line 6.
11	VI7*(S)	L	O.C.	Vectored interrupt line 7.
12	NMI*(S)	L	O.C.	Non-maskable interrupt.
13	PWRFAIL*(B)	L		Power fail bus signal. (See Section 2.10.1 regarding pseudo open-collector nature)
14	DMA3* (M)	L	O.C.	Temporary master priority bit 3.
15	A18 (M)	H		Extended address bit 18.
16	A16 (M)	H		Extended address bit 16.
17	A17 (M)	H		Extended address bit 17.
18	SDSB* (M)	L	O.C.	The control signal to disable the 8 status signals.
19	CDSB* (M)	L	O.C.	The control signal to disable the 5 control output signals.
20	GND (B)			Common with pin 100.
21	NDEF			Not to be defined. Manufacturer must specify any use in detail.
22	ADSB* (M)	L	O.C.	The control signal to disable the 16 address signals.
23	DODSB* (M)	L	O.C.	The control signal to disable the 8 data output signals.
24	Φ (B)	H		The master timing signal for the bus.
25	pSTVAL*(M)	L		Status valid strobe.
26	pHLDA (M)	H		A control signal used in conjunction with HOLD* to coordinate bus master transfer operations.
27	RFU			Reserved for future use.
28	RFU			Reserved for future use.
29	A5 (M)	H		Address bit 5.
30	A4 (M)	H		Address bit 4.
31	A3 (M)	H		Address bit 3.
32	A15 (M)	H		Address bit 15 (most significant for non-extended addressing.)
33	A12 (M)	H		Address bit 12.
34	A9 (M)	H		Address bit 9.
35	DO1 (M)/DATA1 (M/S)	H		Data out bit 1, bidirectional data bit 1.
36	DO0 (M)/DATA0 (M/S)	H		Data out bit 0, bidirectional data bit 0.
37	A10 (M)	H		Address bit 10.
38	DO4 (M)/DATA4 (M/S)	H		Data out bit 4, bidirectional data bit 4.
39	DO5 (M)/DATA5 (M/S)	H		Data out bit 5, bidirectional data bit 5.
40	DO6 (M)/DATA6 (M/S)	H		Data out bit 6, bidirectional data bit 6.
41	DI2 (S)/DATA10 (M/S)	H		Data in bit 2, bidirectional data bit 10.
42	DI3 (S)/DATA11 (M/S)	H		Data in bit 3, bidirectional data bit 11.
43	DI7 (S)/DATA15 (M/S)	H		Data in bit 7, bidirectional data bit 15.
44	sM1 (M)	H		The status signal which indicates that the current cycle is an op-code fetch.
45	sOUT (M)	H		The status signal identifying the data transfer bus cycle to an output device.
46	sINP (M)	H		The status signal identifying the data transfer bus cycle from an input device.
47	sMEMR (M)	H		The status signal identifying bus cycles which transfer data from memory to a bus master, which are not interrupt acknowledge instruction fetch cycle(s).
48	sHLTA (M)	H		The status signal which acknowledges that a HLT instruction has been executed.
49	CLOCK(B)			2 MHz (0.5%) 40-60% duty cycle. Not required to be synchronous with any other bus signal.
50	GND (B)			Common with pin 100.
51	+8 VOLTS (B)			Common with pin 1.
52	−16 VOLTS (B)			Instantaneous maximum less than −14.5 volts, instantaneous minimum greater than −35 volts, average minimum greater than −21.5 volts.
53	GND (B)			Common with pin 100.
54	SLAVE CLR* (B)	L	O.C.	A reset signal to reset bus slaves. Must be active with POC* and may also be generated by external means.
55	DMA0* (M)	L	O.C.	Temporary master priority bit 0.

PIN NO.	SIGNAL & TYPE	ACTIVE LEVEL		DESCRIPTION
56	DMA1* (M)	L	O.C.	Temporary master priority bit 1.
57	DMA2* (M)	L	O.C.	Temporary master priority bit 2.
58	sXTRQ* (M)	L		The status signal which requests 16-bit slaves to assert SIXTN*.
59	A19 (M)	H		Extended address bit 19.
60	SIXTN* (S)	L	O.C.	The signal generated by 16-bit slaves in response to the 16-bit request signal sXTRQ*.
61	A20 (M)	H		Extended address bit 20.
62	A21 (M)	H		Extended address bit 21.
63	A22 (M)	H		Extended address bit 22.
64	A23 (M)	H		Extended address bit 23.
65	NDEF			Not to be defined signal.
66	NDEF			Not to be defined signal.
67	PHANTOM*(M/S)	L	O.C.	A bus signal which disables normal slave devices and enables phantom slaves—primarily used for bootstrapping systems without hardware front panels.
68	MWRT (B)	H		pWR• — sOUT (logic equation). This signal must follow pWR* by not more than 30 ns. (See note, Section 2.7.5.3)
69	RFU			Reserved for future use.
70	GND (B)			Common with pin 100.
71	RFU			Reserved for future use.
72	RDY (S)	H	O.C.	See comments for pin 3.
73	INT* (S)	L	O.C.	The primary interrupt request bus signal.
74	HOLD* (M)	L	O.C.	The control signal used in conjunction with pHLDA to coordinate bus master transfer operations.
75	RESET*(B)	L	O.C.	The reset signal to reset bus master devices. This signal must be active with POC* and may also be generated by external means.
76	pSYNC (M)	H		The control signal identifying BS$_1$.
77	pWR* (M)	L		The control signal signifying the presence of valid data on DO bus or data bus.
78	pDBIN (M)	H		The control signal that requests data on the DI bus or data bus from the currently addressed slave.
79	A0 (M)	H		Address bit 0 (least significant).
80	A1 (M)	H		Address bit 1.
81	A2 (M)	H		Address bit 2.
82	A6 (M)	H		Address bit 6.
83	A7 (M)	H		Address bit 7.
84	A8 (M)	H		Address bit 8.
85	A13 (M)	H		Address bit 13.
86	A14 (M)	H		Address bit 14.
87	A11 (M)	H		Address bit 11.
88	DO2 (M)/DATA2 (M/S)	H		Data out bit 2, bidirectional data bit 2.
89	DO3 (M)/DATA3 (M/S)	H		Data out bit 3, bidirectional data bit 3.
90	DO7 (M)/DATA7 (M/S)	H		Data out bit 7, bidirectional data bit 7.
91	DI4 (S)/DATA12 (M/S)	H		Data in bit 4 and bidirectional data bit 12.
92	DI5 (S)/DATA13 (M/S)	H		Data in bit 5 and bidirectional data bit 13.
93	DI6 (S)/DATA14 (M/S)	H		Data in bit 6 and bidirectional data bit 14.
94	DI1 (S)/DATA9 (M/S)	H		Data in bit 1 and bidirectional data bit 9.
95	DI0 (S)/DATA8 (M/S)	H		Data in bit 0 (least significant for 8-bit data) and bidirectional data bit 8.
96	sINTA (M)	H		The status signal identifying the bus input cycle(s) that may follow an accepted interrupt request presented on INT*.
97	sWO* (M)	L		The status signal identifying a bus cycle which transfers data from a bus master to a slave.
98	ERROR* (S)	L	O.C.	The bus status signal signifying an error condition during present bus cycle.
99	POC* (B)	L		The power-on clear signal for all bus devices; when this signal goes low, it must stay low for at least 10 msecs.
100	GND (B)			System ground.

tion and timing of all bus cycles while it has control of the bus, and is capable of generating all possible bus cycles.

The permanent master normally has control of the bus. It may relinquish bus control to a temporary bus master via a hold operation for an arbitrary number of cycles. Upon completion of the hold operation control of the bus is always returned to the permanent master.

2.3.2 Permanent master state diagram

The permanent master interface shall be implemented so as to conform to the state diagram given in Figure 1.

2.3.3 Permanent master state descriptions

2.3.3.1 Bus state 1

The initial bus state, BS_1, is the state in which the status and address buses are in transition to their values for the new bus cycle. pSYNC goes true in the middle of the BS_1 state, indicating the beginning of a new bus cycle.

2.3.3.2 Bus state 2

Bus state 2, BS_2, is the state during which the address and status lines become stable. When they are guaranteed stable the pSTVAL*, status valid strobe, is activated.

The ready lines and the sixteen acknowledge line are sampled during the BS_2 state.

2.3.3.3 Wait state

The wait state, BS_w, is entered if the ready line sampled in BS_2 indicates that the addressed bus slave is not ready for data transfer. The ready line is sampled once every clock cycle until a ready condition is indicated. When the ready condition is indicated the BS_2 state is completed and the BS_3 state entered.

The BS_w state is thus used to synchronize bus cycles generated by bus masters with the response speed of assorted bus slaves.

2.3.3.4 Bus state 3

Bus state 3, BS_3, is the bus state during which the data transfer actually takes place between the master and the addressed slave.

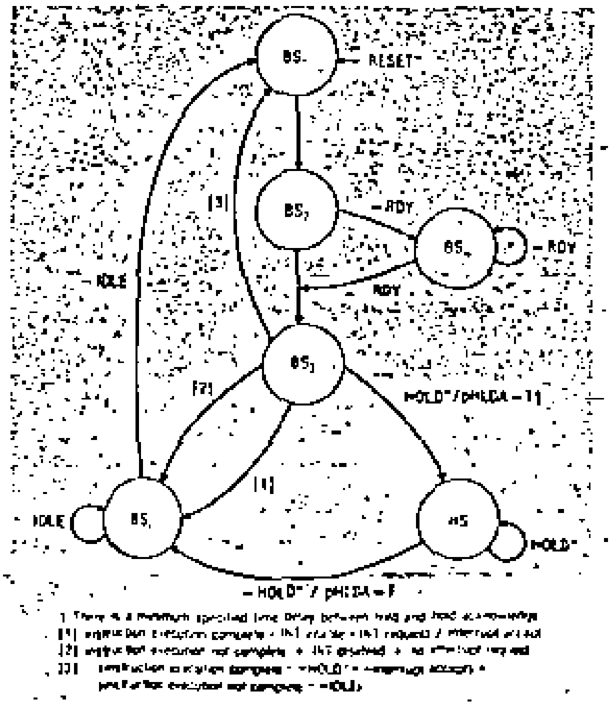

BS.
RESET*
BS₂
[3]
–RDY
BSw
–RDY
RDY
–IDLE
BS₃
[2]
HOLD*/pHLDA–1¹
[1]
IDLE
BS,
HS
HOLD*
–HOLD*/pHLDA–F
1 There is a minimum specified time delay between hold and hold acknowledge
[1] instruction execution complete • INT enable • INT request / interrupt accepted
[2] instruction execution not complete + INT disabled + no interrupt request
[3] instruction execution complete • ~HOLD* • ~interrupt accept + instruction execution not complete • ~IDLE

Figure 1. Permanent master state diagram.

2.3.3.5 Idle bus states

After completion of the BS_3 state, the master may enter one or more idle bus states.

While in an idle bus state the generalized data strobes, pWR* and pDBIN, must not be active, and pSTVAL* must not be asserted in conjunction with pSYNC active.

2.3.3.6 Hold accept

Permanent masters must be configured to conditionally accept hold operations from temporary masters. This function may be gated off under hardware or software control, to allow indivisible test and set operations. If hold is enabled and active, the permanent master will enter the hold state HS following a BS_3 state, and pHLDA will be asserted.

The permanent master remains in the hold state until the hold request HOLD* becomes false.

Hold operations always take priority over interrupt operations.

2.3.3.7 Interrupt accept

If hold request is not active, if execution of the current instruction is complete, and if interrupts are enabled and an interrupt is being requested, then the permanent master accepts the interrupt request at the end of the BS_3 state. In the case of a vectored interrupt, the next bus cycle may be an interrupt acknowledge bus cycle. In the case of a non-maskable interrupt, the response is usually a transfer to a predetermined location.

2.3.4 Required signals for permanent masters

2.3.4.1 Output signals

The following signals are output signals from permanent masters to bus slaves:

1) A0-A23†.
2) All status signals.
3) All control output signals.
4) Data output signals (8 or 16 depending on processor type).
5) Φ, the system clock.

2.3.4.2 Input signals

The following signals are required input signals to permanent masters:

1) The control input signals, except NMI* and SIXTN*.
2) Data input signals (8 or 16 depending on processor type).
3) The four disable signals ADSB*, DODSB*, SDSB*, CDSB*.
4) RESET*.

† A16 through A23 are optional on permanent masters.

2.3.5 Dummy mastering

In cases where a number of processors co-exist in a single system as temporary masters, it may prove inefficient from a systems point-of view to implement a permanent master.

In such a case it is permissible that the permanent master be implemented as a dummy, that is, as a device that conducts no bus cycles, but only supplies an arbitration interval so that the DMA control bus may settle.

The dummy master takes control of the bus between temporary masters, asserting the control output bus in the null state, and passes the bus to the next requester after an arbitration interval of one clock cycle.

Required output signals for dummy masters are the control output signals, and the system clock Φ. Input signals are HOLD* and CDSB*.

2.4 The temporary master interface

2.4.1 General

The temporary master interface provides the capability to transfer device dependent messages to and from a selected set of bus slaves. The temporary master thus differs from the permanent master in that it need not generate all possible bus cycles.

The temporary master requests control of the bus from the permanent master. If the bus is granted, the temporary master is responsible for the generation and timing of all bus cycles until it returns control to the permanent master.

Since up to 16 temporary masters may co-exist in a single system, a protocol has been developed to arbitrate among simultaneous bus requests. Detailed specification of this protocol is given in 2.8.3.

2.4.2 Temporary master state diagram

The temporary master interface shall be implemented so as to conform to the state diagram given in Figure 2.

2.4.3 Temporary master state descriptions

2.4.3.1 Arbitration (ARB)

If more than one temporary master is present in the system, bus requesters must arbitrate for the bus as given in 2.8.3.

During the arbitration sequence, bus requesters try to assert their priorities on the arbitration bus, and the contents of the arbitration bus are compared with each requester's priority.

If the contents of the arbitration bus is of higher priority than the locally attempted priority assertion, then a higher priority requester is present in the system, and the low priority requester removes its low order bits from the arbitration bus. Thus, after some settling time, the priority of the highest priority requester is present on the arbitration bus. This requester is granted the bus on the rising edge of hold acknowledge.

2.4.3.2 Bus transfer states (XS I and XS II)

Since the bus has positive polarity control signals, extreme care must be taken in bus transfer operations to avoid erroneous pulses on the control lines.

In general terms, this is accomplished by specifying that both the permanent master and the temporary master drive the control lines in specified logic states during the bus transfer.

Detailed specification of this operation is given in 2.8.2.

Proposed S-100 bus layout—Quick reference

pin 1	+8 Volts (B)		pin 51	+8 Volts (B)	
pin 2	+16 Volts (B)		pin 52	−16 Volts (B)	
pin 3	XRDY (S)	H	pin 53	GND	
pin 4	VI0* (S)	L	pin 54	SLAVE CLR* (B)	L
pin 5	VI1* (S)	L	pin 55	DMA0* (M)	L
pin 6	VI2* (S)	L	pin 56	DMA1* (M)	L
pin 7	VI3* (S)	L	pin 57	DMA2* (M)	L
pin 8	VI4* (S)	L	pin 58	sXTRQ* (M)	L
pin 9	VI5* (S)	L	pin 59	A19	H
pin 10	VI6* (S)	L	pin 60	SIXTN* (S)	L
pin 11	VI7* (S)	L	pin 61	A20 (M)	H
pin 12	NMI* (S)	L	pin 62	A21 (M)	H
pin 13	PWRFAIL* (B)	L	pin 63	A22 (M)	H
pin 14	DMA3* (M)	L	pin 64	A23 (M)	H
pin 15	A18 (M)	H	pin 65	NDEF	
pin 16	A16 (M)	H	pin 66	NDEF	
pin 17	A17 (M)	H	pin 67	PHANTOM* (M/S)	L
pin 18	SDSB* (M)	L	pin 68	MWRT (B)	H
pin 19	CDSB* (M)	L	pin 69	RFU	
pin 20	GND		pin 70	GND	
pin 21	RFU		pin 71	NDEF	
pin 22	ADSB* (M)	L	pin 72	RDY (S)	H
pin 23	DODSB* (M)	L	pin 73	INT* (S)	L
pin 24	Φ (B)	H	pin 74	HOLD* (M)	L
pin 25	pSTVAL* (M)	L	pin 75	RESET* (B)	L
pin 26	pHLDA (M)	H	pin 76	pSYNC (M)	H
pin 27	RFU		pin 77	pWR* (M)	L
pin 28	RFU		pin 78	pDBIN (M)	H
pin 29	A5 (M)	H	pin 79	A0 (M)	H
pin 30	A4 (M)	H	pin 80	A1 (M)	H
pin 31	A3 (M)	H	pin 81	A2 (M)	H
pin 32	A15 (M)	H	pin 82	A6 (M)	H
pin 33	A12 (M)	H	pin 83	A7 (M)	H
pin 34	A9 (M)	H	pin 84	A8 (M)	H
pin 35	DO1 (M)/DATA1 (M/S)	H	pin 85	A13 (M)	H
pin 36	DO0 (M)/DATA0 (M/S)	H	pin 86	A14 (M)	H
pin 37	A10 (M)	H	pin 87	A11 (M)	H
pin 38	DO4 (M)/DATA4 (M/S)	H	pin 88	DO2 (M)/DATA2 (M/S)	H
pin 39	DO5 (M)/DATA5 (M/S)	H	pin 89	DO3 (M)/DATA3 (M/S)	H
pin 40	DO6 (M)/DATA6 (M/S)	H	pin 90	DO7 (M)/DATA7 (M/S)	H
pin 41	DI2 (S)/DATA10 (M/S)	H	pin 91	DI4 (S)/DATA12 (M/S)	H
pin 42	DI3 (S)/DATA11 (M/S)	H	pin 92	DI5 (S)/DATA13 (M/S)	H
pin 43	DI7 (S)/DATA15 (M/S)	H	pin 93	DI6 (S)/DATA14 (M/S)	H
pin 44	sM1 (M)	H	pin 94	DI1 (S)/DATA9 (M/S)	H
pin 45	sOUT (M)	H	pin 95	DI0 (S)/DATA8 (M/S)	H
pin 46	sINP (M)	H	pin 96	sINTA (M)	H
pin 47	sMEMR (M)	H	pin 97	sWO* (M)	L
pin 48	sHLTA (M)	H	pin 98	ERROR* (S)	L
pin 49	CLOCK (B)		pin 99	POC* (B)	L
pin 50	GND		pin 100	GND	

2.4.3.3 Bus cycle

The definition of bus cycle states is the same as that for the permanent master interface, given in 2.3.3.1 through 2.3.3.5.

An arbitrary number of bus cycles may be performed by the temporary master before returning control to the permanent master.

2.4.4 Required signals for temporary masters

2.4.4.1 Output signals

The following are required output signals for a temporary master interface:

1) Address lines A0-A23†.
2) All status signals.
3) All control output signals ††.
4) Data output lines.
5) DMA arbitration lines DMA0*-DMA3*.
6) Hold request, HOLD*.

2.4.4.2 Input signals

The following are required input signals for a temporary master interface:

1) The ready lines, RDY and XRDY.
2) Hold acknowledge, pHLDA.
3) Data input lines.
4) The system clock, Φ.

2.5 The slave interface

A slave device responds to a bus cycle initiated by a bus master. Memory and input/output devices are examples of bus slaves.

A slave device may request service by a bus master by generating an interrupt request.

2.5.1 Slave interface state diagram

The slave interface shall conform, in general, to the state diagram given in Figure 3. Slave interfaces need not have both read and write capability.

2.5.2 Slave state definitions

2.5.2.1 Slave idle state

The slave idle state, S_I, is a passive state with respect to the bus.

The slave monitors the stream of bus cycles to determine if it is selected for the current bus cycle.

The slave may be performing internal operations while in the idle state.

The assertion of SLAVE CLR* forces all slaves into the idle state.

2.5.2.2 Slave setup

A slave moves from the slave idle state to the setup state, S_S, when it has been addressed by the current bus cycle. This is an operation internal to the slave which sets up a data transfer with a bus master. If a

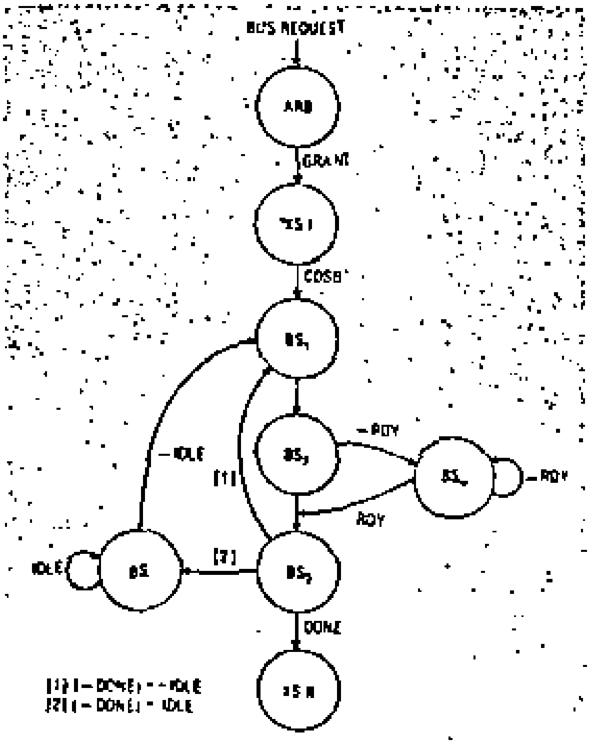


Figure 2. Temporary master state diagram.

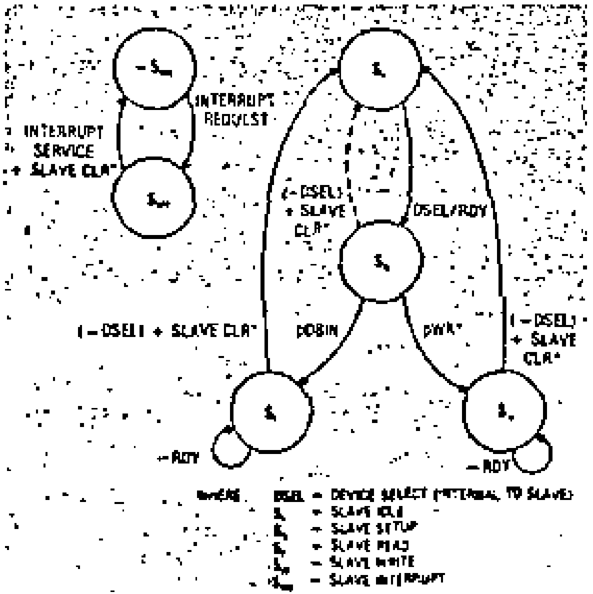


Figure 3. Slave interface state diagram.

† *Note:* Temporary masters must generate A16-A23; they need only generate falses or lows on these 8 lines, however.

†† *Note:* Temporary masters should provide a jumper on the pSTVAL* line, as 8080 CPUs of old design do not transfer this line with the control output lines. In this case all bus masters use the same pSTVAL* signal.

slave can tolerate spurious transitions from the idle state to the setup state, then the device select signal may be decoded statically from the address and status buses. If a device cannot tolerate spurious transitions, the device select line should be decoded in conjunction with the status valid strobe, pSTVAL*.

If synchronization is required by the slave before the data transfer may take place, the ready line is asserted false during this state until the device is ready for data transfer.

2.5.2.3 Slave read

Data from the addressed slave is gated onto the data bus during the slave read state, S_r. The generalized read strobe governs the transition to this state.

When device select becomes false the slave returns to the idle state.

2.5.2.4 Slave write

Data from the current bus master is written into the slave during the active period of the generalized write strobe, pWR*.

When device select becomes false the slave returns to the idle state.

2.5.2.5 Interrupt request state

If a slave requires service by a bus master, an interrupt request may be generated by the slave. The interrupt should be held active until the slave is serviced, or until SLAVE CLR* is asserted.

2.5.3 Required signals for slave interfaces

Slave interfaces need only receive and generate that subset of bus signals necessary for communication with masters.

2.6 8/16-bit data transfer protocol

2.6.1 General

Implementation of the 8/16-bit data transfer protocol allows both 8-bit and 16-bit parallel data transfers over the bus, and hence allows both 8-bit masters and 16-bit masters and slaves to co-exist in a single system. For 16-bit transfers the two unidirectional 8-bit data buses are ganged to form a single 16-bit bidirectional data bus.

Two lines are assigned to control the ganging of the data bus:

1) sXTRQ*, status output from the master, which indicates a request for a 16-bit data transfer.
2) SIXTN*, an acknowledge input to the master, which indicates that a 16-bit data transfer is possible.

Use of the sixteen acknowledge line SIXTN* permits the use of current design 8-bit memory boards without modification. When SIXTN* is false, a 16-bit transfer may be accomplished by two sequential single-byte transfers.

2.6.2 8-bit data paths

The current bus master requests an 8-bit transfer by not asserting sXTRQ*.

Byte data output from the master to the addressed slave is asserted on the data output bus, DO0 through DO7.

Byte data input from the addressed slave to the current bus master is asserted on the data input bus, DI0 through DI7.

2.6.3 16-bit data paths

The current bus master requests a 16-bit transfer by asserting sXTRQ*.

If the addressed slave is capable of a 16-bit parallel data transfer, it asserts SIXTN*, as shown in the timing diagram (see page 52).

Sixteen-bit data transfer is then conducted via the ganged data buses, where DO0 = DATA0, and DI7 = DATA15.

2.6.4 Memory organization

Memory devices capable of both 8-bit and 16-bit parallel data transfers are organized, as shown in Figure 4, as two banks of 8-bit memory, a high-byte bank and a low-byte bank. These data banks may be activated either together or separately, depending on the condition of the sixteen request status line, sXTRQ*.

2.6.4.1 Byte references

When sXTRQ* is not asserted, memory references are single-byte transfers.

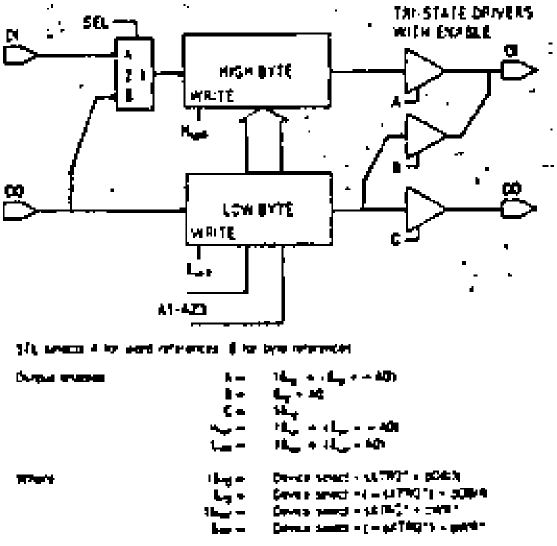


Figure 4. 8/16-bit memory organization.

The proper location in memory is selected by the address output on address lines A1 through A15 (A23 for extended addressing systems), while the A0 line selects the high byte or the low byte. A0 equals 0 selects the high byte of the 16-bit word, while A0 equals 1 selects the low byte of the word.

See Figure 5 for address usage.

In the 8-bit mode, data output from the master, on the DO bus, is connected to the data input lines of both memory banks; the low-byte data input lines are connected directly to the DO bus, and the high-byte data input lines are connected to the DO bus via a two-to-one multiplexer controlled by sXTRQ*.

Data output from the memory banks is routed to Tri-State† bus drivers A and B in Figure 4. One of these drivers is enabled when the read strobe is activated, depending on the condition of A0. The selected byte is thus available to the master on the DI bus.

2.6.4.2 Word references

When sXTRQ* is asserted by the master, and SIXTN* is asserted by the slave, memory references are double-byte transfers.

Address lines A1 through A15 (A23 in extended address systems) select the proper word from memory. The condition of the A0 bit does not enter into the decoding or addressing for word references.

See Figure 5 for address usage.

In the 16-bit mode, data output from the bus master is asserted on the 16 signal lines of the DO bus and the DI bus. The multiplexer on the data input lines now routes the high-byte data, on the DI bus, to the data input lines of the high-byte bank. Low-byte data, on the DO bus, is connected to the data input lines of the low-byte bank.

Data output from the memory banks is routed through buffers A and C to their respective data paths. Both A and C will be enabled by the read strobe.

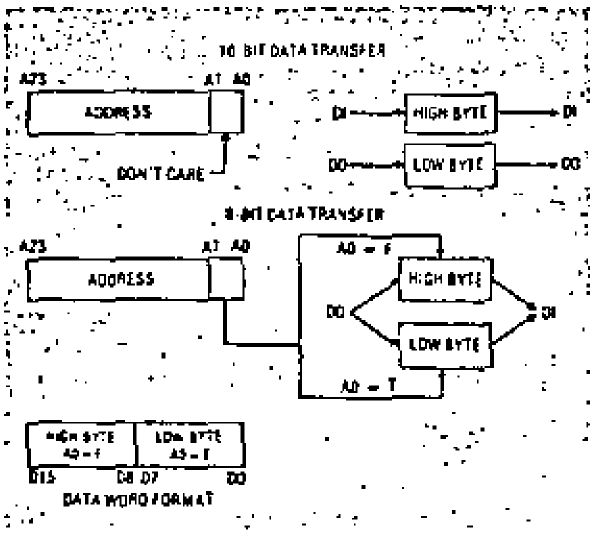


Figure 5. 8/16-bit address and data usage.

2.6.5 Sixteen acknowledge (SIXTN)*

Implementation of the sixteen acknowledge line allows the use of 8-bit memory boards in a 16-bit system without modification, but with a reduction in maximum system bandwidth.

If a 16-bit master requests a 16-bit transfer, but the addressed slave is not capable of such a transfer, the sixteen acknowledge lines will not be asserted.

The master will respond in one of two ways, by generating an error trap or by conducting the transfer in byte-serial fashion.

2.6.5.1 Byte-serial response

If the sixteen acknowledge line is not activated after a specified period, circuitry may be included on bus masters to conduct the requested 16-bit transfer as two consecutive byte operations, thus assembling the requested 16-bit word while holding the master in a wait state.

For this process to occur, the sixteen acknowledge line must meet the timing specifications for the ready line inputs.

2.6.5.2 Error response

If circuitry does not exist on the master to conduct the requested 16-bit transfer as two consecutive byte operations, an error condition shall result immediately, with ERROR* asserted.

2.7 Fundamental bus cycle timing

2.7.1 General

This section deals with the fundamental timing concepts involved in the standard bus cycle. Detailed specification of the timing parameters discussed in this section is given in 3.8 and 3.9.

The standard bus cycle is a pseudo-synchronous cycle, that is, the timing of the control signals bears a specified relationship to the master system clock Φ.

All data transfers, including read or write cycles, 8- or 16-bit transfers, memory or input/output device transfers, and interrupt acknowledge are conducted on the bus as a standard bus cycle.†

Figure 6 shows the fundamental timing for a standard bus cycle, with a single wait state inserted by the addressed slave.

2.7.2 Address and status buses

The beginning of a new bus cycle is indicated by the rising edge of the pSYNC signal, which closely follows the rising edge of the system clock, Φ.

The address and status buses are changing to their values for the new cycle during the beginning of the

†Tri-State is a trademark of National Semiconductor.

† See possible exception, section 2.7.5.3.

pSYNC interval. Shortly after they can be guaranteed stable on the bus, the status valid strobe, pSTVAL*, is asserted. pSTVAL*, decoded in conjunction with pSYNC, indicates to all bus slaves that stable address and status may be sampled from the bus.

The position of the status valid strobe within the pSYNC interval is independent of the system clock, Φ. This affords the designer of bus masters considerable flexibility in interfacing different processors to the bus. The status valid strobe should be positioned within the pSYNC interval such that the delay between guaranteed status on the bus and the activation of the status valid strobe is as close to the minimum specification as possible, thus maximizing memory and device access time.†

In order to prevent false cycle starts in bus slaves, only one negative edge of the status valid strobe may occur while pSYNC is asserted.

Address and status information is thus stable on the bus from the negative transition of the status valid strobe during pSYNC, and is held stable until a specified period after the trailing edge of the data strobe (pDBIN in the read case, and pWR* in the write case). This hold time ensures that false decoding of the address and status information will not occur at the end of the bus cycle.

2.7.3 Ready and sixteen acknowledge lines

The sixteen acknowledge line, since it may be used to place the bus master in a wait state while a requested 16-bit transfer is conducted in byte-serial fashion, is subject to the same timing constraints as the ready lines.

The ready lines are first sampled by the bus master on the rising edge of the system clock during the BS2 state, and if active, the master enters a wait state, sampling the ready line once every clock cycle on the rising edge of the system clock until the slave is ready for data transfer.

A minimum setup time before the rising edge of the system clock, and a minimum hold time after sampling must be met for the proper operation of the ready lines.

The time between the active edge of the status valid strobe and the sampling of the ready line may be very short. Hence, it is recommended practice not to make assertion of the ready line dependent on pSTVAL*.

Data output, address, and status are held stable during wait states.

2.7.4 Read cycles

2.7.4.1 General

There are four types of read cycles: op-code fetch (M1), memory read, input, and interrupt acknowledge. These cycles are all similar with respect to timing, but make different use of the status bits and the address bus. See Tables 1 and 2.

2.7.4.2 The read strobe

The generalized read strobe pDBIN is used to gate data from an addressed slave onto the data bus during a read operation. The read strobe is asserted true by the bus master after a minimum specified time from the assertion of the status valid strobe.

It is held true during any inserted wait states, and returns to the false state, returning the data bus to the high impedance state, shortly before the address and status buses are allowed to change.

2.7.5 Write cycles

2.7.5.1 General

There are two possible types of write cycles on the bus, a memory write cycle and an output cycle.

These two cycles are similar with respect to timing, but make different use of the status bits and address bus. A special write strobe, MWRT, is generated for memory cycles.

2.7.5.2 The write strobe

The generalized write strobe, pWR*, is used to write data from the data bus into the addressed bus slave. The write strobe may be asserted by the master after the completion of the pSYNC interval.

Data out on the data bus must be guaranteed valid for a specified period both before and after the activation of the write strobe. Hence, either the leading or the trailing edge of the write strobe may be used to strobe data into the addressed slave.

Address and status information must be held valid for a specified period of time from the trailing edge of the write strobe.*

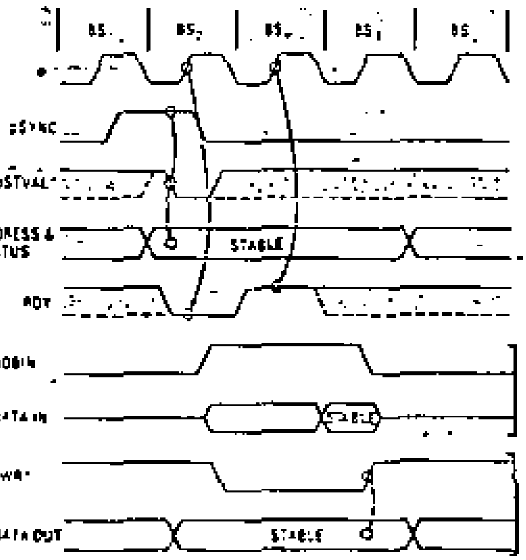


Figure 6. Bus cycle fundamental timing relationships.

*Note: The Φ1 signal output from current 8080 processor boards meets all the specifications as a status valid strobe. Hence, these boards meet the bus cycle specification without modification.

2.7.5.3 Memory write strobe

While the generalized write strobe is activated for all write cycles, the memory write strobe is activated for memory write cycles only. The memory write strobe is usually generated by front-panel devices, if they exist in the system, as a function of bus memory write or a front-panel deposit. If front-panel devices do not exist the memory write strobe must be generated somewhere in the system, but at only one point. This circuit should be designed such that it generates the memory write strobe for all bus masters. Jumpers shall be provided to allow extra circuits to be disabled.

The memory write strobe, MWRT, is defined as:

$$\mathrm{MWRT} = \overline{\mathrm{pWR}} \cdot -\mathrm{sOUT}, \text{ (logic equation)}$$

that is, memory write is true when pWR is true and sOUT is false.

The memory write strobe must follow the pWR* strobe by not more than a specified period.†

2.8 Special bus operations

2.8.1 General

This section describes two special bus operations related to DMA operations, that is, the transfer of bus control from the permanent bus master to a temporary bus master for an arbitrary number of bus cycles, and the return of control to the permanent master.

These two operations are:

1) The bus transfer protocol.
2) The arbitration protocol among simultaneous bus requesters.

2.8.2 Bus transfer protocol

2.8.2.1 General

When a temporary bus master has been gra[illegible] the bus by the permanent bus master, control must be transferred to the temporary master in such a way that spurious signals are not generated on the control output lines, causing false bus cycles. Since some of the control output signals are of positive polarity, extreme care must be taken in this operation. In general, the specified bus transfer protocol accomplishes this by having the permanent master and the temporary master drive the control output lines simultaneously at specified levels during the bus transfer.

2.8.2.2 Bus transfer state diagram

The bus transfer operation shall be implemented so as to conform to the bus transfer state diagram given in Figure 7.

2.8.2.3 Bus transfer state definitions

2.8.2.3.1 Idle

The idle state signifies that the temporary master is either involved in internal operations, and does require the bus, or that it is waiting for the bus to become free so that it may assert its bus request.

2.8.2.3.2 Arbitration

If a temporary master desires the bus, and HOLD is false and pHLDA is false, the temporary master enters the arbitration sequence, where it contests with other bus requesters for control of the bus.

Detailed specification of this process is given in 2.8.3.

2.8.2.3.3 Bus grant

Priority assertions on the arbitration bus settle in the interval between the assertion of a hold rec, and a hold acknowledge. At the rising edge of the hold acknowledge signal the bus is granted to the highest priority requester, enabling the bus transfer operation for that requester.

If the bus is not granted to a requester, that requester returns to the idle state.

The bus grant state is termed MINE.

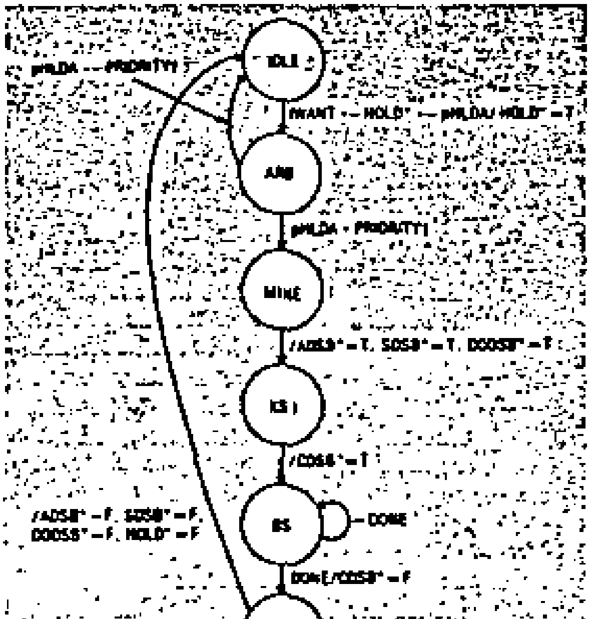


Figure 7. Bus transfer state diagram.

†Note: Historically the MWRT strobe has been generated by front-panel devices to accomplish the deposit function. In such a case, the MWRT strobe is asserted while the front-panel holds the CPU in a wait state during a memory read cycle. Note that the status will indicate a read cycle and the pDBIN strobe will be active.

While this is acceptable procedure for 8-bit systems, and 8-bit memories should respond to MWRT as well as pWR*, it is not permissible in 16-bit systems as a conflict will exist on the bidirectional data bus. A front-panel could be implemented either as a temporary master, or integrated into the CPU.

2.8.2.3.4 Transfer state one, XS I

The bus transfer sequence begins with transfer state one, XS I. The bus transfer control circuit asserts the following signals together:

1) ADSB*
2) SDSB*
3) DODSB*

disabling the address, status, and data output drivers of the permanent bus master and enabling the control output drivers of the temporary master. Both the permanent master and the temporary master are now driving the control output lines. These lines are required to have the following levels during this time.

Signal	*Logic state*	*Electrical level*
1) pSYNC	F	L
2) pSTVAL*	F	H†
3) pDBIN	F	L
4) pWR*	F	H
5) pHLDA	T	H

The transfer state is terminated by the assertion (by the bus transfer control circuit) of the CDSB* line, disabling the control drivers of the permanent master and enabling the address, status, and data out drivers of the temporary master. The temporary master now has complete control of the bus and begins its first bus cycle.

2.8.2.3.5 Bus cycles

Any number of standard bus cycles are then conducted by the temporary bus master. Bus control is never transferred between cycles. When the temporary master is done, the process proceeds to XS II, transfer state two.

2.8.2.3.6 Transfer state two

Transfer state two, XS II, is the mirror image of the sequence in XS I. The state begins with the release of the CDSB* signal, enabling the control output drivers of the permanent master and disabling the address, status, and data output drivers of the temporary master.

Both the temporary master and the permanent master drive the control output lines for the remainder of XS II at the levels prescribed for XS I.

The state is ended by the release of other disable signals and HOLD*, enabling the address, status, and data out drivers on the permanent master, and disabling the control output drivers of the temporary master. The permanent master now has complete control of the bus and the temporary master returns to the idle state.

2.8.2.4 Bus transfer timing relationships

2.8.2.4.1 General

The fundamental timing relationships for a bus transfer and a single DMA bus cycle are given in Figure 8.

Relationship to the bus transfer states is shown in boxes at the bottom of the figure.

Detailed specification of these times is given in 3.10 and Table 5.

2.8.2.4.2 T_{set}

A minimum time between the rising edge of the hold acknowledge signal and the assertion of the disable signals in XS I allows time for completion of the preceding bus cycle.

2.8.2.4.3 T_{ov}

The time that both the temporary master and the permanent master must drive the control output signals has a specified minimum to assure a smooth bus transfer.

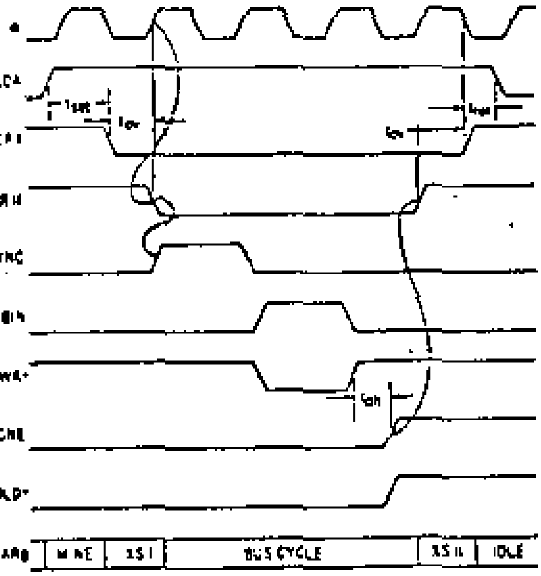
Assertion of the XFER II signal, or CDSB*, is specified relative to the rising edge of the system clock, Φ, so that the assertion of this signal may be used by the temporary master as a cycle start signal.

2.8.2.4.4 T_{dh}

The "done" signal is a signal internal to the temporary master. This signal should not be asserted until the hold time for data output, status, and address signals in the standard bus cycle has been met.

2.8.2.4.5 T_{rel}

Completion of the reverse bus transfer XFER II shall precede the release of the pHLDA signal by a minimum specified time.



NOTE.
XFER I — ADSB*, SDSB*, DODSB*
XFER II — CDSB*

Figure 8. Bus transfer and single bus cycle.

†See note in section 2.4.4.1.

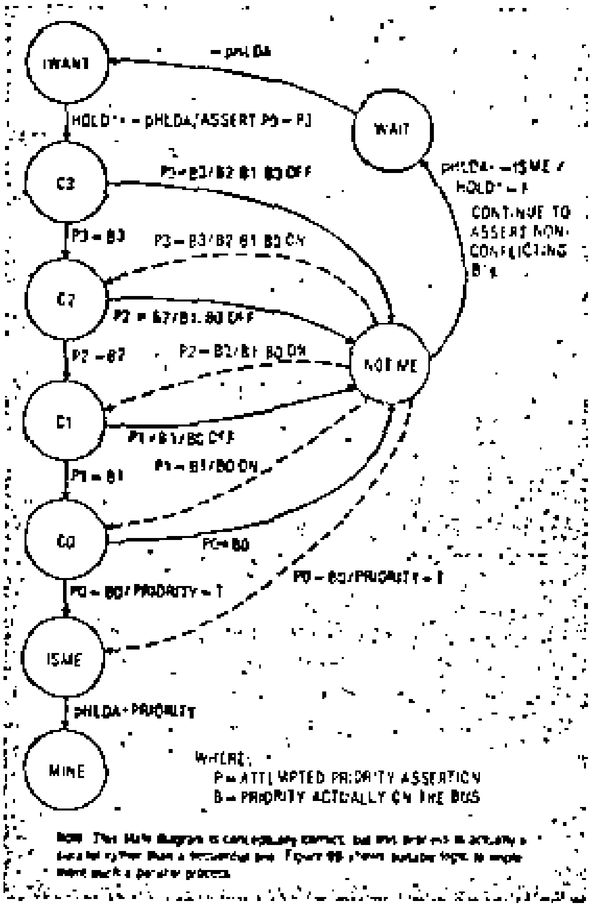


Figure 9a. Bus arbitration diagram.

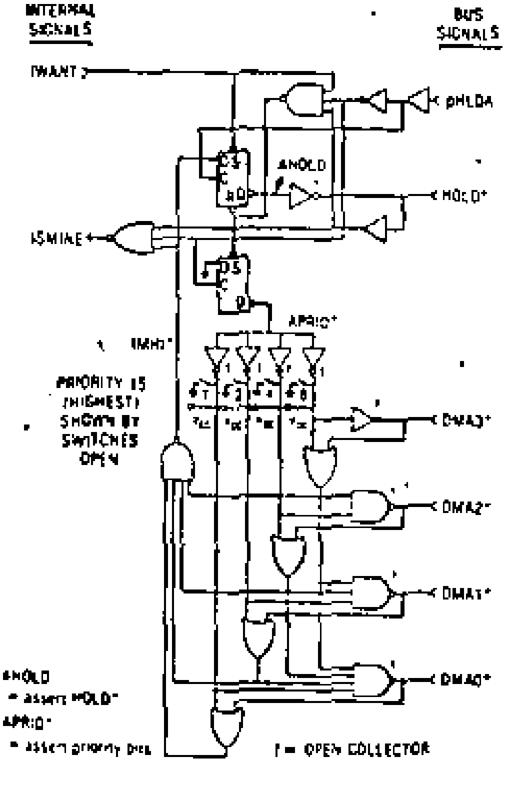


Figure 9b. Bus arbitration example.

2.8.3 Bus arbitration protocol

In a system which allows more than one master to use the system bus, for example a CPU permanent master and several temporary masters such as DMA controllers or multiple CPUs, some means must be provided to determine which device will be allowed to control the bus at any given time.

The bus arbitration system uses four bus lines for arbitrating among 16 temporary masters. These lines are driven by open collector drivers, and are pulled high by pullup resistors. Each temporary master has a unique priority number which it asserts on the arbitration bus at an appropriate time. A higher binary number indicates a higher priority.

The temporary masters compare the priority appearing on the active-low open-collector bus with the priority they are asserting, starting with the most significant bit. If disagreement is detected by any temporary master at any given bit position, then another temporary master must be asserting that priority bit and thus must have a higher priority. In that case all less significant bits are removed by the detecting temporary master. All more significant bits agree, and thus need not be removed, and the bit which disagreed must have been a 0 and thus was not asserted. Leaving the agreeing bits asserted reduces system noise caused by the redistribution of driving currents in the bus, and speeds settling of the correct priority on the arbitration bus. This process is a continuous asynchronous parallel process, not a sequential bit-by-bit process as it may seem from the above description. Incorrect comparisons will occur and be removed as the bus lines settle for as long as four bus delays (not related to the choice of four bus lines) plus logic delays.

The four lines which comprise the arbitration bus are DMA0* through DMA3*, where DMA3* is the most significant bit. These lines, in conjunction with HOLD* and pHLDA, control the bus arbitration process.

2.8.3.1 Bus arbitration implementation

An implementation of the bus arbitration protocol is shown in Figures 9a and 9b.

Any implementation shall obey the rules summarized in section 2.8.4.

2.8.3.2 Bus arbitration state definitions

2.8.3.2.1 IWANT

The IWANT state is an internal state for a temporary master which has determined that a bus access is necessary and thus wishes to arbitrate for bus control.

Temporary masters may not assert their priorities nor remove them at arbitrary times, or the arbitration bus may be in transition when the result is needed. A temporary master may assert its priority and the HOLD* bus request only if (1) pHLDA is not asserted (the permanent master has the bus), and (2)

HOLD* is not already asserted. This guarantees an ample time to settle the arbitration bus before the granting of the bus on the rising edge of pHLDA.

This scheme usually results in the first requester winning the bus. Only if simultaneous bus requests occur will the arbitration have any effect. This, however, is not improbable, since multiple unsuccessful requesters will become synchronized by waiting for the falling edge of pHLDA.

2.8.3.2.2 Priority compare states

The priority comparison states, C3 through C0, are the states where each requester compares the priority it is attempting to assert on the arbitration bus with the priority actually on the arbitration bus. Though C3 through C0 are shown and described as sequential, they are actually parallel processes. While disagreement occurs at any bit position, less significant bits are removed from the arbitration bus. If no disagreement persists after the settling time, the requester has the highest priority and will be granted the bus on the rising edge of pHLDA, proceeding to the state "MINE", where the bus transfer begins. All requesters continue to assert their priorities on the arbitration bus until the falling edge of pHLDA. Thus the priority number of the current bus master is available on the DMA bus while pHLDA is true. If the permanent master has the bus, pHLDA will be false.

A temporary master that wins the bus continues to assert its priority and HOLD* until its bus cycles are complete. A temporary master that loses the bus continues to assert its priority bits not turned off by the arbitration process, but must remove its assertion of the HOLD* line, so that the winner may indicate that it is finished by releasing HOLD*. A losing requester in this state is said to be in the "WAIT" state.

2.8.3.3 Bus arbitration timing relationships

Figure 10 shows two possible cases of the bus arbitration procedure. The first of these is a case where the requester has no competition; it requests the bus and the bus is granted. The second case shows the requester waiting for the bus to be free, arbitrating for the bus and losing, and arbitrating for the bus and winning.

2.8.3.3.1 No competition

When the temporary master determines that it requires the bus, it raises the internal signal IWANT. In this case, the rising edge of IWANT finds the pHLDA signal unasserted, meaning the permanent master has the bus, and the HOLD* signal unasserted, meaning that no other devices are requesting the bus. The temporary master may then assert the HOLD* signal and assert its priority on the arbitration bus. The ISME signal is the result of the arbitration process, and is asserted if none of the bit-wise comparisons on the arbitration bus fail. This arbitration result is clocked by the rising edge of the pHLDA signal, creating the bus grant signal MINE.

When the temporary master is finished with the bus, the IWANT signal is released, releasing the HOLD* signal and resetting the bus grant signal, MINE. The permanent master releases the pHLDA signal, and all assertions are removed from the arbitration bus.

2.8.3.3.2 Wait-lose-win

In this example the requester raises its IWANT signal, but finds the bus already busy and must wait to assert its bus request and priority until the falling edge of pHLDA.

The requester arbitrates for the bus during try 1, but another requester has a higher priority and the arbitration result ISME is low at the rising edge of pHLDA, indicating a loss in the arbitration process. The losing requester removes its assertion of the HOLD* signal, but continues to assert the non-conflicting high-order bits of its losing priority until the falling edge of pHLDA. At the falling edge of pHLDA, the process repeats, but this time results in a win for the requester.

2.8.4 Summary of arbitration protocol

Figures 9A and 9B represent an example, not a required implementation. Any implementation which obeys the rules may be used. The rules which must be obeyed by a temporary master are:

1) HOLD* may be asserted only when it is not already asserted and pHLDA is low.
2) HOLD* must be removed when pHLDA rises if another controller has asserted higher priority.
3) HOLD* must be removed when the controller no longer needs the bus.
4) Priority must be asserted whenever HOLD* is asserted, and must remain asserted until the next falling edge of pHLDA.

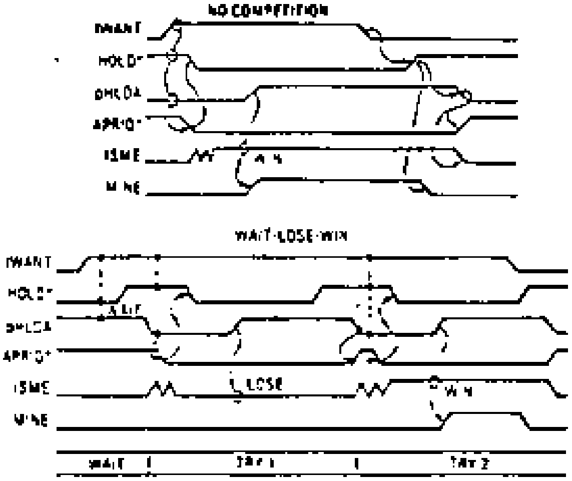


Figure 10. Bus arbitration timing diagrams.

5) The priority level must be user-selectable by switches and asserted by open-collector drivers on bus lines DMA3*-DMA0*.
6) The most significant bit of the priority level (appearing on DMA3*) must be compared with the priority asserted. If the line is asserted low but not by this temporary master, all less significant priority bit assertions must be removed. Similarly, bits DMA2*, DMA1*, and DMA0* must be examined and possible less significant conflicting bits removed.
7) If no lines are asserted low except those asserted by this temporary master after sufficient settling time, this temporary master has highest priority and may take the bus when pHLDA rises.
8) Logic implementations must be such that settling of the arbitration circuitry and bus will be completed between the assertion of HOLD* and the rise of pHLDA.

2.9 Interrupt protocol

The purpose of an interrupt system is to allow peripheral devices to suspend the operation of a bus master in an orderly way and to request that the master service the requesting peripheral. When service is complete, the bus master returns to the operation from which it was interrupted.

The interrupt protocol is comprised of an 8-level vectored interrupt system and a non-maskable interrupt. A complying master need only implement INT*.

2.9.1 Vectored interrupts

2.9.1.1 Vectored interrupt requests

Eight levels of vectored interrupt requests are issued on the vectored interrupt lines, VI0* thru VI7*, where VI0* is the most significant interrupt priority level. Vectored interrupt requests, however, may be rotated, masked individually, or "fenced out" by the interrupt control slave, and hence the priority levels are not fixed. Requests on the VI lines should be asserted as levels, that is, they should be held active until service is received. A slave which asserts a VI line need take no further action to generate an interrupt. It is assumed that if interrupt acknowledge cycles occur, an interrupt controller somewhere in the system will respond appropriately.

The generalized interrupt request line, INT*, is implemented as a communication line between the interrupt controller and an interruptable master. Any slave or interrupt controller, using the INT* line, must respond appropriately to any interrupt acknowledge cycles. The interrupt controller is not required to use INT*. A vectored interrupt may occur without INT* ever being asserted.

2.9.1.2 Interrupt acknowledge

The interrupt acknowledge cycle is a standard bus read cycle. The interrupt acknowledge cycle requests vectoring information from the interrupt controller to be asserted on the data bus during pDBIN.

Since no address information is asserted during an interrupt acknowlege cycle, only one interrupt controller may exist on the bus. If multiple interrupt controllers exist, they must either be "daisy chained" to avoid possible bus conflicts, or polled by the bus master.

2.9.2 Non-maskable interrupt (NMI)*

The non-maskable interrupt is an optional control input to bus masters. This interrupt is not maskable by a software instruction, and takes priority over other interrupt requests. The NMI* line may be used in the implementation of the special condition lines, ERROR* and PWRFAIL*.

NMI* is an open collector line. The bus master shall respond to negative going transitions on the NMI* line.

2.10 Special condition lines

Two special condition lines, PWRFAIL* and ERROR*, are available on the bus. Their use is optional.

2.10.1 Power-fail pending (PWRFAIL)*

This line indicates an impending system power failure. It is specified that this line shall be activated at least 50 msecs before the local voltage regulators drift out of specification.

The line stays low until the power-on clear signal is activated. This implies that either a normally closed relay or a battery powered circuit drive the power fail line. The circuit driving this line must meet the electrical specifications for an open collector line.

*2.10.2 ERROR**

This is a generalized error line that indicates that the current bus operation is producing an error of some sort (i.e., memory parity error, write to protected memory, inability to accommodate 8-bit slaves, etc.)

The ERROR* line should be implemented as a trap. All relevant information about the error-causing cycle—address, data, status, device number (for temporary masters)—should be latched on the falling edge of ERROR*.

ERROR* is implemented as an open collector line.

3.0 Electrical specifications

3.1 Application

This section defines the electrical specifications for interface devices to be used in S-100 bus systems. Proper operation of these devices also depends on two other factors:

1) Short physical distance between devices.
2) Relatively low electrical noise.

The electrical specifications for the bus driver and receiver circuits do not imply a particular technology, unless otherwise noted.

All specifications apply over the temperature range $T_a = 0°C$ to $70°C$.

3.2 Power distribution

Power in S-100 systems is distributed as unregulated DC power at three voltages, +8 volts, +16 volts, and −16 volts. Because these voltages are on adjacent lines it is relatively easy to short these lines on card removal. Therefore, bleeder resistors or other constant loads sufficient to discharge all three supplies rapidly are recommended.

3.2.1 +8 volt specification

Instantaneous minimum must be greater than +7 volts, instantaneous maximum less than 25 volts, and average maximum less than 11 volts.

3.2.2 +16 volt specification

Instantaneous minimum must be greater than 14.5 volts, instantaneous maximum less than 35 volts, and average maximum less than 21.5 volts.

3.2.3 −16 volt specification

Instantaneous maximum must be less than −14.5 volts, instantaneous minimum greater than −35 volts, and average minimum greater than −21.5 volts.

3.3 General signal discipline

Other than the power lines noted above, all signals on the bus are limited to positive signal levels between 0 volts and +5 volts, and may not have loaded rise or fall times less than 5 nsecs.

3.4 Driver requirements

3.4.1 Driver types

Three types of bus drivers are defined:

1) An active driver, either in the high state or in the low state or in transition, which has the capability to accept current in the low state and to provide current in the high state.
2) An open collector driver, which will not accept or provide current in the high state. A 1000Ω ± 5% pullup resistor to +5 volt or equivalent must be provided somewhere in the system for open collector lines. It is recommended that these pullup resistors be provided on the bus. However, implementation on the permanent master is also acceptable.
3) A Tri-State driver, which has the capability to be in the high-impedance state as well as in the high and low states.

3.4.2 Driver specifications

Specifications for bus drivers shall be as follows:

Low state (V_{OL}): Output voltage less than or equal to +0.5 volts at 24 mA sink current.

High state (V_{OH}): Output voltage (for active and Tri-State drivers) greater than or equal to +2.4 volts at 2 mA.

The leakage current for Tri-State drivers in the high-impedance state is specified as not greater than ±25 μA.

The internal capacitive load of a driver shall not exceed 15 pF at 25°C whether in the active or the high-impedance state.

The rise and fall times of bus drivers should be minimized, subject to 3.3. In no case should the rise or fall times exceed 50 nsec at rated capacitive load.

3.5 Receiver specifications

The specifications for receivers on the bus shall be as follows:

Low state: A voltage less than or equal to +0.8 volts shall be recognized as a low state.

High state: A voltage greater than or equal to +2.0 volts shall be recognized as a high state.

Bus receivers shall source no more than 0.5 mA at 0.5 volts and sink no more than 50 μA at 2.4 volts.

Bus receivers shall have diode clamp circuits to prevent excessive negative voltage excursions.

Additional noise immunity is afforded by the use of Schmitt-type receiver circuits. Recommended hysteresis for such receivers should be greater than or equal to 0.4 volts.

3.6 Bidirectional signals

Some interface signals, such as the data bus, are combined Tri-State drivers and receivers. For each function these devices must meet the same specifications as separate drivers and receivers.

The total internal capacitive load for a line transceiver shall not exceed 20 pF at 25°C.

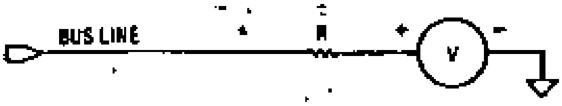
3.7 Card-level bus loading

At the card level, the following specifications apply:

1) The total capacitive load on any bus input shall not exceed 25 pF.
2) A card may not source more than 0.5 mA at 0.5 volts nor sink more than 60 μA at 2.4 volts on any signal line except for DMA0*, DMA1*, DMA2*, DMA3*, PHANTOM*, and PWRFAIL*. On these lines a card may not source more than 0.4 mA at 0.5 volts.

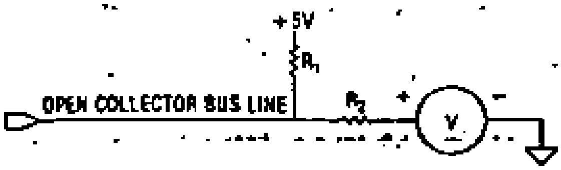
3.7.1 Bus termination

All bus lines except the power and ground lines may be terminated to reduce bus noise using a circuit equivalent to



where V = 2.6 volts ± 0.2 volts and R is no less than 180Ω(± 5%).

Open collector lines may have a combination pullup and termination scheme using a circuit equivalent to



where V = 2.6 volts ± 0.2 volts, R_1 = 1.5KΩ ± 5%, and R_2 should be no less than 180Ω (± 5%).

3.8 Read cycle timing specification

Figure 11a depicts the read cycle timing waveforms with the pertinent timing parameters shown. Table 4 specifies these parameters.

3.9 Write cycle timing specification

Figure 11b depicts the write cycle timing waveforms with the pertinent timing parameters shown. Table 4 specifies these parameters.

3.10 Ready and sixteen request timing specification

Figure 12 depicts RDY, XRDY, and SIXTN* timing waveforms during read and write cycles, with pertinent timing parameters shown. Table 4 specifies these parameters.

3.11 Bus transfer timing specification

Figure 8 depicts bus transfer timing waveforms with the pertinent timing parameters shown. Table 5 specifies these parameters.

4.0 Mechanical specifications

4.1 Application

This section defines the mechanical specifications for standard interface systems.

4.2 Connector type

The card edge connector is a 100-pin (dual 50) connector with contacts spaced on 0.125" centers. It is nominally designed for printed circuit boards 0.062" thick.

The connector is subject to the specifications in 4.2.1 and 4.2.2.

4.2.1 Electrical considerations

1) Voltage rating: 200 volts DC, minimum pin to pin.
2) Current rating: 2.5A per contact.
3) Contact resistance: 50 mΩ maximum at rated current after 100 insertions.
4) Insulation resistance: 1000 MΩ minimum.

4.2.2 Connector spacing

Connectors should be spaced 0.75 inches −0.01 inches center to center.

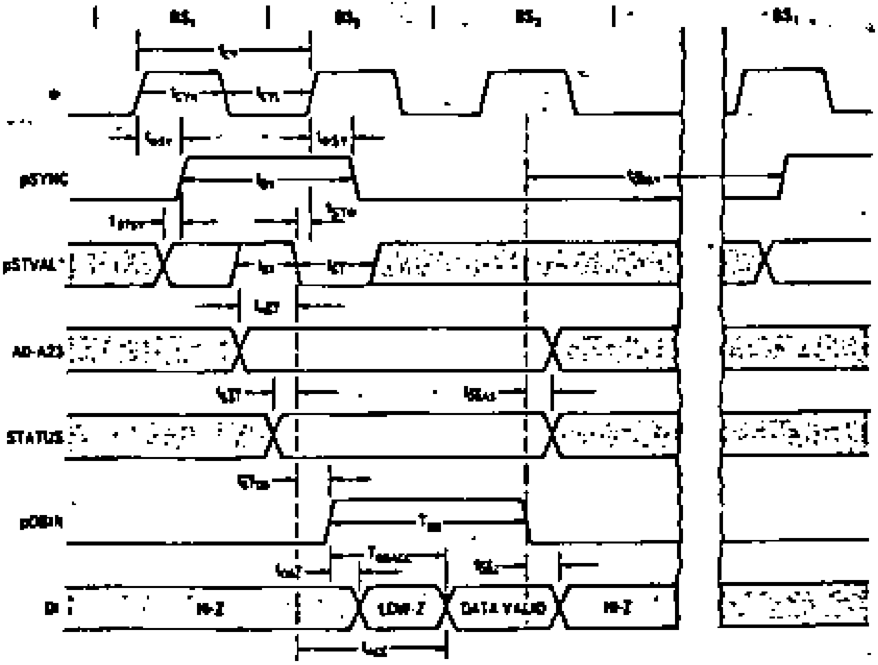


Figure 11a. Read cycle timing diagram.

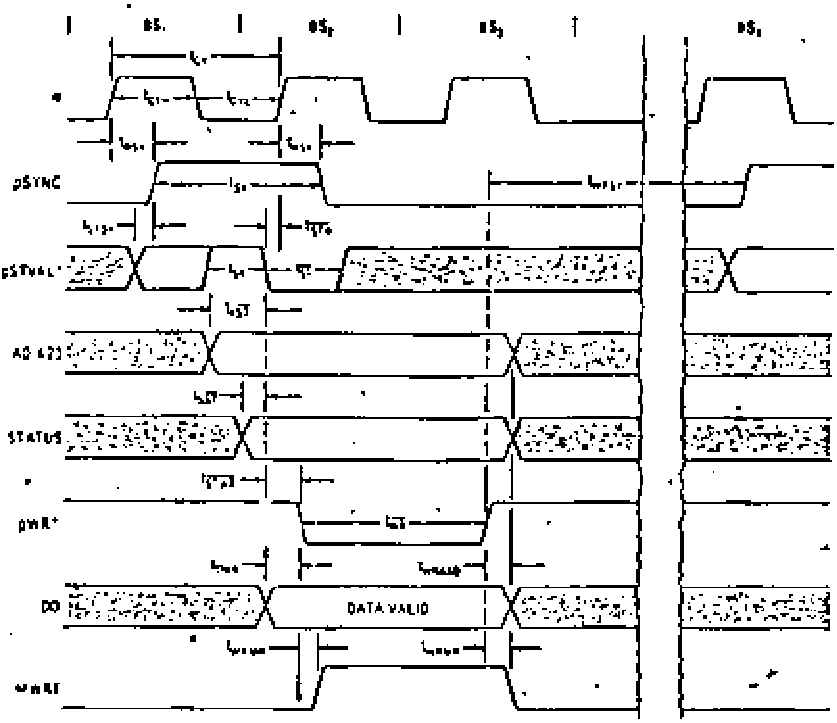


Figure 11b. Write cycle timing diagram.

Table 4. Read/write cycle timing parameters.

		MIN.(NSECS)	MAX.(NSECS)
t_{CY}	Φ PERIOD	166	2000
t_{CYH}	Φ PULSE WIDTH HIGH	$0.4t_{CY}$	
t_{CYL}	Φ PULSE WIDTH LOW	$0.4t_{CY}$	
$t_{\Phi SY}$	DELAY Φ HIGH TO pSYNC HIGH; DELAY Φ LOW TO pSYNC LOW	10	$0.4t_{CY}$
t_{SY}	pSYNC PULSE WIDTH HIGH	$0.7t_{CY}$	
$t_{\overline{ST\Phi}}$	pSTVAL* LOW PRIOR TO Φ LOW DURING pSYNC	0	
t_{ST}	pSTVAL* PULSE WIDTH HIGH	50	
$t_{\overline{ST}}$	pSTVAL* PULSE WIDTH LOW	50	
t_{STSY}	pSTVAL* FALLING EDGE PRIOR TO pSYNC HIGH	0	
$t_{\overline{AST}}$	ADDRESSES STABLE PRIOR TO pSTVAL* LOW DURING pSYNC HIGH	70	
$t_{\overline{SST}}$	STATUS STABLE PRIOR TO pSTVAL* LOW DURING pSYNC HIGH	40	
t_{DB}	pDBIN PULSE WIDTH HIGH	$0.9t_{CY}$	
$t_{\overline{ST}DB}$	DELAY pSTVAL* LOW TO pDBIN HIGH.	20	
$t_{\overline{DB}SY}$	DELAY pDBIN LOW TO pSYNC HIGH	0	
$t_{\overline{DB}AS}$	HOLD TIME FOR ADDRESSES AND STATUS AFTER pDBIN LOW	50	
$t_{\overline{DB}Z}$	DELAY pDBIN LOW TO SLAVE DI DRIVERS HI-Z		$75 + 0.1t_{CY}$
$t_{DB\overline{Z}}$	DELAY pDBIN HIGH TO SLAVE DI DRIVERS ACTIVE	10	$25 + 0.1t_{CY}$
t_{ACC}	DELAY pSTVAL* LOW TO DATA VALID	SPECIFIED BY MANUFACTURER. WORST CASE MAXIMUM FOR ALL SLAVES AND WORST CASE MINIMUM FOR ALL MASTERS.	
$t_{S\overline{DB}}$	DATA VALID SETUP TIME TO pDBIN LOW		
$t_{\overline{WR}}$	pWR* PULSE WIDTH LOW	$0.9t_{CY}$	
$t_{\overline{STWR}}$	DELAY pSTVAL* LOW TO pWR* LOW	30	
t_{WRSY}	DELAY pWR* HIGH TO pSYNC HIGH	0	
$t_{D\overline{WR}}$	SETUP TIME DO VALID TO pWR* LOW	$0.1t_{CY}$	
t_{WRASD}	HOLD TIME ADDRESSES, STATUS, AND DO FROM pWR* HIGH	$0.2t_{CY}$	
t_{WRMW}	DELAY pWR* LOW TO MWRT HIGH; DELAY pWR* HIGH TO MWRT LOW		30
$t_{RDY\Phi}$	SETUP TIME RDY, XRDY, SIXTN* TO Φ RISING	60	
$t_{\Phi RDY}$	HOLD TIME RDY, XRDY, SIXTN* AFTER Φ RISING	70	

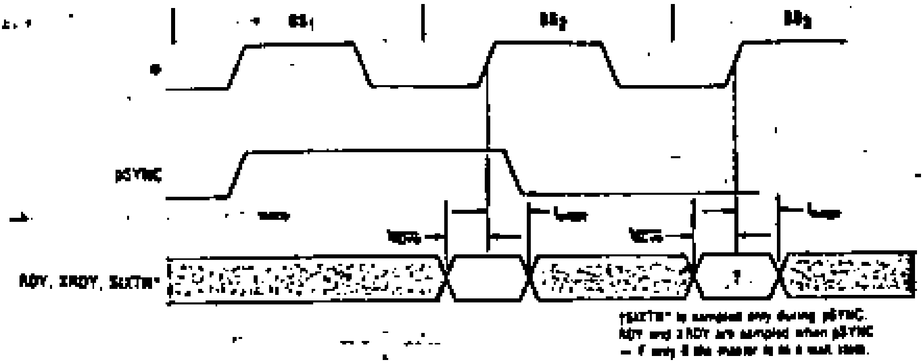

BS₁
BS₂
BS₃
pSYNC
RDY, XRDY, SIXTN*
†SIXTN* is sampled only during pSYNC.
RDY and XRDY are sampled when pSYNC
— if only if the master is in a wait state.

Figure 12. Timing of RDY, XRDY, and SIXTN* during read and write cycles.

Table 5.
Bus transfer timing parameters.

t_{SET}	DELAY pHLDA TO ADSB*, SDSB*, DODSB*	30
t_{DV}	TIME BOTH TEMPORARY AND PERMANENT MASTER DRIVE THE CONTROL OUTPUT LINES	$0.5t_{CY}$
t_{DH}	HOLD TIME ADDRESS, STATUS, AND DATA OUT DURING DMA CYCLE	$0.2t_{CY}$
t_{REL}	SETUP TIME, END OF BUS TRANSFER TO pHLDA RISING EDGE	20

4.3 Board size specification

Circuit boards shall conform to the board size specifications given in Figure 13. The edge connector pin shown in the figure is pin 50. Pin 100 opposes pin 50 on the back side of the board.

Total board depth shall not exceed 0.65".

Nominal board thickness is 0.062".

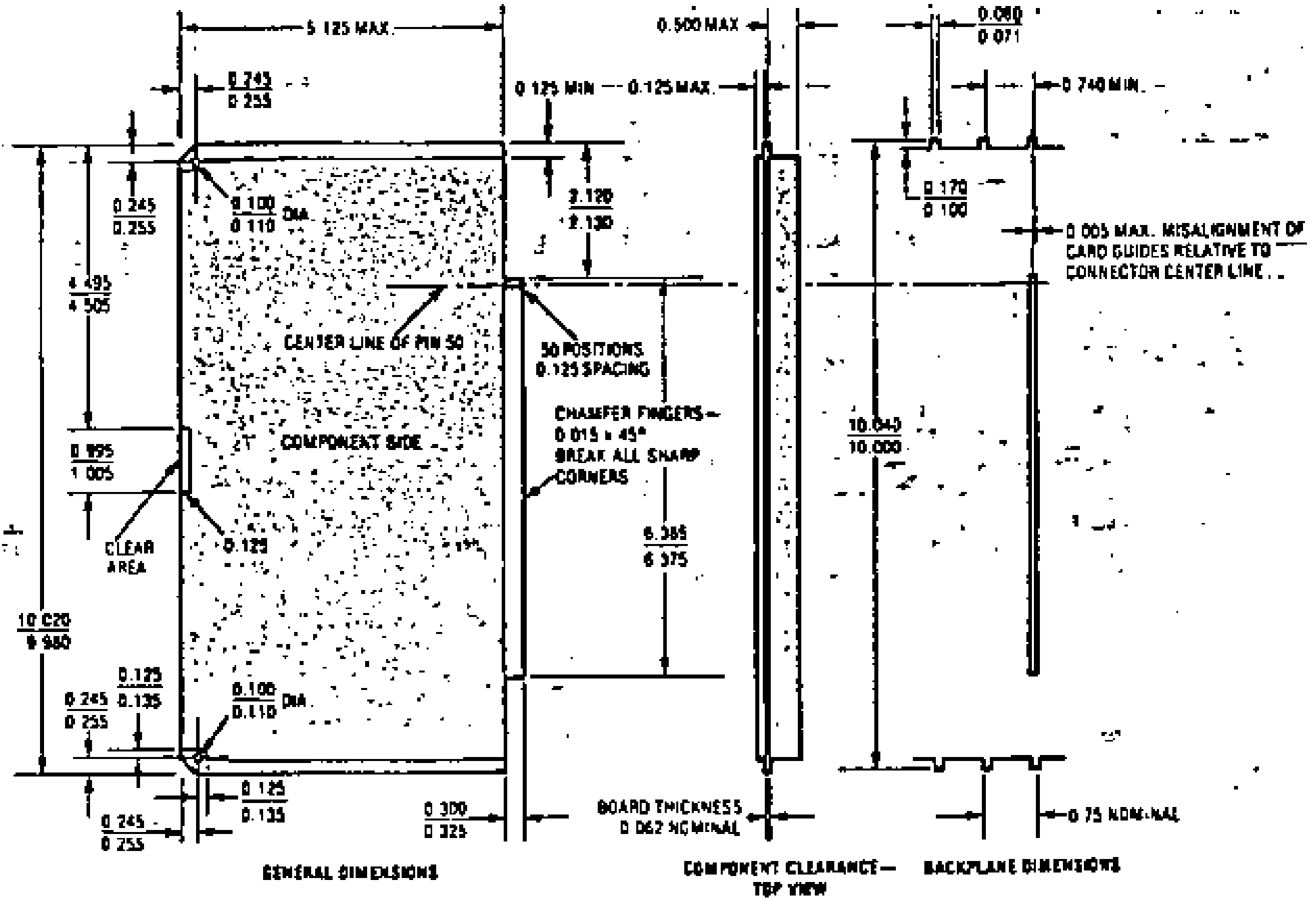


Figure 13. S-100 board mechanical parameters.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROSESADORES: TEORIA Y APLICACIONES

* AN ARCHITECTURAL COMPARISON OF CONTEMPORARY 16-BIT MICROPROCESSORS

* SINGLE-CHIP N-MOS MICROCOMPUTER PROCESSES SIGNALS IN REAL TIME

JULIO,1981

Today's microprocessors exhibit powerful computing capabilities. Their characteristic differences favor each machine for a distinct portion of the applications spectrum.

An Architectural Comparison of Contemporary 16-Bit Microprocessors

Hoo-min D. Toong and Amar Gupta
Massachusetts Institute of Technology

The evolution of microprocessor architecture during the past decade has progressed at an incredible pace. From the primitive 4004, introduced in 1971, to the present spectrum of sophisticated microprocessors, the growth has been swift, dramatic, and almost revolutionary. Today's products possess astonishing computational capabilities a nd support primary memories of up to 64M bytes. They incorporate high-level languages and technical innovations only recently introduced on larger mainframes. Microprocessor-based systems now offer facilities for direct support of multiuser/multitask environments and sophisticated operating system implementations.

The current single-chip, 16-bit microprocessor market has three major contenders:

- the 8086 (iAPX 86): designed by Intel; second-sourced by Mostek in the US and by Siemens in Europe;
- the Z8000: designed by Zilog; second-sourced by AMD in the US, by SGS/ATES in Europe, and by Sharp in Asia;
- the MC68000: designed by Motorola; second-sourced by AMD and Rockwell in the US, by EFCIS in Europe, and by Hitachi in Asia.

Several of these devices are also being produced by major systems houses for internal use. In addition, National Semiconductor has recently announced the NS16000 microprocessor chip family, scheduled to be introduced during 1981. In the following paragraphs, we examine and compare the architectures of the three processors. A section is devoted to the preliminary data on the National 16000 series.

General characteristics

The characteristics of the microprocessors being discussed here are summarized in Table 1.

The 8086. Of the three processors, the Intel 8086 is the oldest and simplest. Basically, it is an improved, 16-bit version of the 8080; an 8080-type multiplexed bus is expanded to a 16 bit external bus. As in the 8080, the instructions are byte-oriented. One of the major enhancements is a six-byte instruction prefetch queue. This buffer feeds instructions to the execution unit in eight-bit segments. The queue decreases address bus/data bus idle times by prefetching data, thus increasing processor speed. The 8086 register structure is very similar to the 8080's. Registers in both machines are special-purpose. Rooted in a basic design philosophy requiring storage efficiency, special-purpose registers allow implied register addressing in most instructions and permit shortened instructions. Most addressing modes are the same in both machines. An address space of one megabyte is implemented in the 8086 through a memory segmentation scheme using 64K segments. Memory segments of up to 64K bytes can be placed on an eight-bit boundary, allowing a maximum of one megabyte to be addressed. By basing the segmentation and addressing mechanisms on 16 bits, Intel has preserved close compatibility between the 8086 and the 8080.

The 8086 has 95 basic instructions, of which a substantial number are only eight bits long. In the few 16-bit instructions, only the first eight bits are used for operation codes; the additional byte specifies data displacement. Instructions longer than two bytes use the remaining bytes for specifying data. The 8086 instruction set is an expanded version of the 8080 instruction set. Hence 8080 code can be converted easily to 8086 code. Many enhanced programming features are available in the 8086 instruction set. Base segment registers have been added to provide software support for certain operating system functions and for extended addressing range. Changing these segment registers allows the programmer to do process swaps with relative ease. Internally, the 8086 retains an eight-bit instruction path similar to the 8080's. The ALU is 16 bits wide, like the 8080's ALU. Thus, the 8086

0272-1732/81/0500-0026$00.75 © 1981 IEEE

The machine's 110 basic instructions are either 16 or 32 bits long, and the instruction set is word-oriented with vector operations strongly represented. These instructions are the basis of the performance of the Z8000, as is most evident in the block operations. A single 32-bit instruction is used for set and move operations, while in the 8086, six single-byte instructions must be used. For future expansion, the designers of the Z8000 have left one register unused and unassigned. With the present 4-MHz clock speed, the fastest and the slowest instruction times are 0.75 microseconds and 90 microseconds, respectively. A 6-MHz version is being made available.

The MC68000. The designers of this microprogrammed machine have chosen to implement a very wide engine. The external 16-bit bus is multiplexed from the 32 bits inside the engine. A wide, 32-bit ALU has been coded as the user machine. Unlike the general-purpose registers of the Z8000, the 16- by 32-bit registers of the 68000 are partitioned into eight address registers and eight data registers. Motorola has made a great effort to design the engine with a very regular instruction set, making available several general addressing modes for most instructions. This design permits easy implementation of stacks and queues without special instructions. Two 32-bit stack pointers are provided for aiding in systems calls. A special flag register can be set to move the machine into a debugging, single-step mode for program development. Traps for illegal instructions can be used for software extension of the basic instruction set and for floating-point operations. Besides reducing software development problems, the traps allow software compatibility with future hardware improvements.

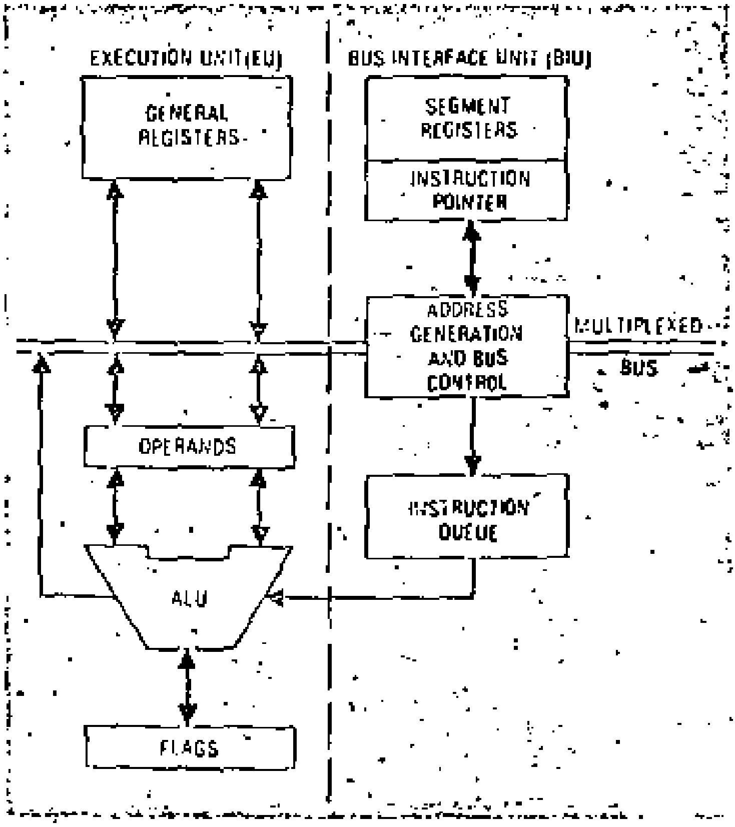


Figure 1. Basic structure of the Intel 8086.

The MC68000 supports 56 basic instructions and 14 addressing modes. The total number of instructions is misleading, however, because many instructions perform triple functions and are encoded differently. The number 56 is an artifact of the assembler. Instruction sizes vary from one to five words. The address bus uses 23 bits for word addressing, providing an addressing capability of 16M bytes.

Architectural details

Basic principles of operation. The basic structure of the Intel 8086 is shown in Figure 1. The 8086 (and the 8088) CPU consists of two separate processing units, the execution unit, or EU, and the bus interface unit, or BIU, connected by a 16-bit ALU data bus and an eight-bit Q bus. The EU obtains instructions from the instruction prefetch queue, IQ, maintained by the BIU, and executes instructions using the 16-bit ALU. Execution of instructions involves maintenance of CPU status and control logic, manipulation of general registers and instruction operands, and manipulation of segment offset addresses within 16-bit limits. The EU accesses memory and peripheral devices through requests to the BIU, which is the second processing unit, performing all bus operations for the EU on a demand basis. This involves generating physical addresses from segment register and offset values, reading operands, and writing results. The BIU is also responsible for prefetching instructions from the IQ whenever possible, to keep the EU busy with prefetched instructions under normal conditions, and for resetting the IQ when the EU transfers control to another location. The execution unit and the bus interface unit operate independently of each other, enabling the 8086 to overlap instruction fetch and execution.

The Z8000 is a random-logic-based CPU; its basic structure is shown in Figure 2. The internal 16-bit data bus is used for internal addressing and data communication. The instructions are fetched through the Z-bus interface and executed by the instruction execute control unit. Throughput is enhanced through "limited" pipelining, which allows prefetching of the next single-word instruction (or the first word of the next multiword instruction) from the memory into the instruction buffer. This occurs only during execution of the current instruction, provided the current instruction does not require the bus to complete the execution cycle. No instruction prefetching occurs when the bus is assigned to another bus master. The 16-bit ALU manipulates data and generates logical offset addresses in the general-purpose register block in accordance with the instruction executed. The CPU status and control flags are maintained in the program status registers, and the CPU, which can operate in a system mode or normal mode, can execute privileged instructions only in the system mode. Interrupts and traps are handled by the exception-handling control unit, and there is provision for multiple interrupt tables. A refresh

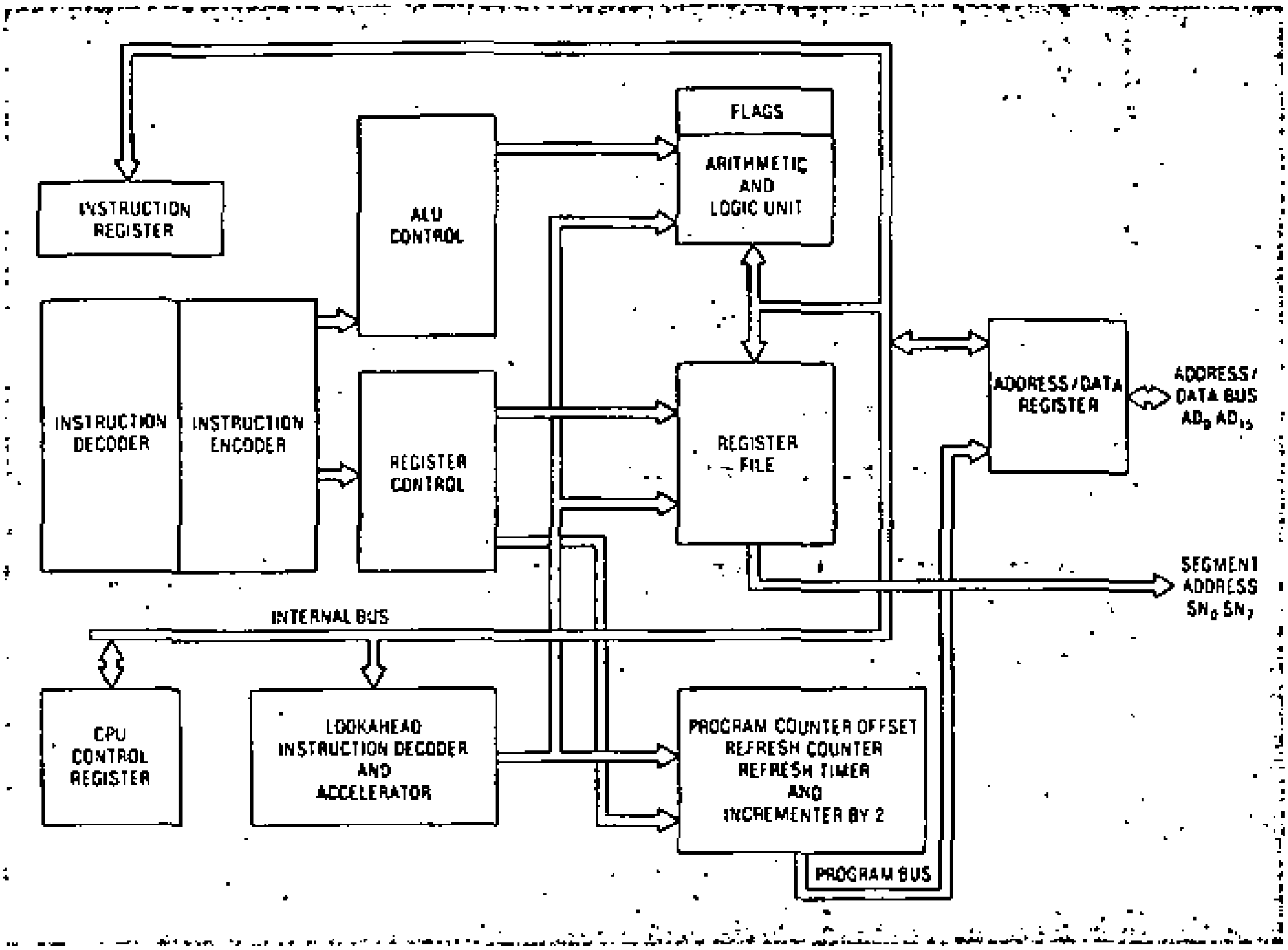

INSTRUCTION REGISTER
ALU CONTROL
FLAGS
ARITHMETIC AND LOGIC UNIT
INSTRUCTION DECODER
INSTRUCTION ENCODER
REGISTER CONTROL
REGISTER FILE
ADDRESS/DATA REGISTER
ADDRESS/ DATA BUS AD_0 AD_{15}
SEGMENT ADDRESS SN_0 SN_7
INTERNAL BUS
CPU CONTROL REGISTER
LOOKAHEAD INSTRUCTION DECODER AND ACCELERATOR
PROGRAM COUNTER OFFSET REFRESH COUNTER REFRESH TIMER AND INCREMENTER BY 2
PROGRAM BUS

Figure 2. Basic structure of the Z8000.

counter provides the refresh control logic with timing information for CPU-driven memory refresh operation.

The MC68000 is architecturally quite distinct from the others. Its block structure is shown in Figure 3. The microcode-based CPU is centered around a microprogram-controlled execution unit. The control store area size is minimized through the use of a two-level control structure. At level one, the machine instructions are produced by sequences of micro-instructions in the micro-control store. These micro-instructions are actually pointers (addresses) to nano-instructions in the nanostore at level two. The nano-control store contains an arbitrarily ordered set of unduplicated machine-state control words, which control the execution unit. All information that is machine instruction static (timing-independent) bypasses the control store and is transmitted directly to the execution unit. In all, about 22.5K bits of control store is used, 50 percent less than the control store required for a single-level implementation. However, the two-level structure increases total access time. An attempt has been made to overcome this by means of a pipelined architecture, in which the instruction fetch, instruction decode, and instruction execute cycles are fully overlapped across every macro-instruction boundary. An attempt has also been made to minimize delays in looping (branching) by prefetching instructions associated with the most likely branch condition. The MC68000 execution unit is a dual bus structure that performs both address and data processing. The CPU may run in either a

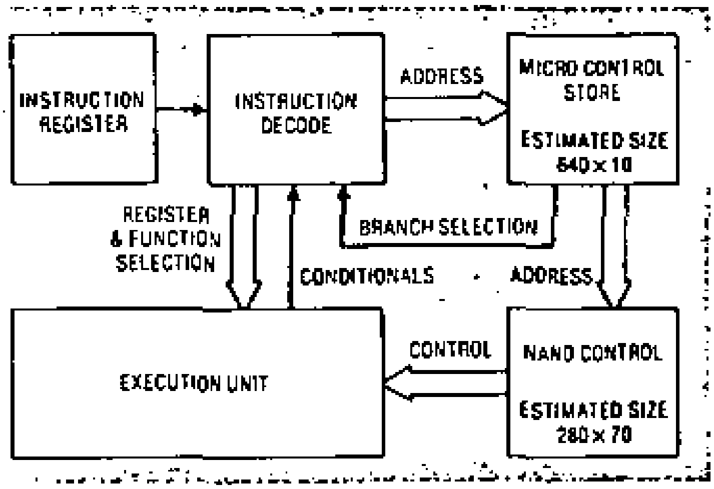


Figure 3. Basic structure of the MC68000.

supervisor mode with privileged instructions or in a user mode. Although internal data paths are all 32 bits wide, the packaging limitation on the number of pins constrains data paths to and from memory to be only 16 bits wide. The 64 pins are comprised of 23 pins for address bus, 16 for data bus, five for asynchronous bus control, three for bus arbitration control, three for interrupt control, three for system control, three for peripheral control, three for processor status, and the remaining five for power supply ground and for clock. The chip specifications provide for floating-point and string operations, but current versions do not have these features because of technological limitations on circuit density and size. At present, these unimplemented instructions cause traps for software emulation. Overall, the MC68000 CPU implements as large a subset of the complete 68000 system architecture as is feasible under current technology. It is expected that on-chip memory, faster clock speeds, custom microprogramming, and facilities for run-time changes in microprogram will be provided by Motorola during the lifetime of the 68000 architecture.

Register organization. The Intel 8086 execution unit contains four 16-bit pointer and index registers and four 16-bit data registers addressable on an individual byte basis. These eight registers are used implicitly by the instruction set, providing compact encoding at the cost of reduced flexibility. The BIU contains one 16-bit instruction pointer, which contains the offset of the next instruction to be fetched. This pointer is updated by the BIU but cannot be directly accessed by programs. The BIU also contains four 16-bit dedicated segment registers for segment base addressing, which enables programs to access up to four 64K-byte segments at a time. Finally, the EU contains six one-bit status flags and three one-bit control flags.

The Z8000 family is characterized by sixteen 16-bit general-purpose registers. All can be used as accumulators, and all but one can be used as index pointers or memory pointers. The one exception is an escape mechanism for address changes. The general register architecture avoids bottlenecks inherent in dedicated or implied registers. Register grouping and overlapping provide for byte, double-word, and 64-bit registers. Two registers are used as implied stack pointers for system mode and normal mode. All Z8000 family chips contain one 16-bit segment, offset register, and the Z8001 also contains one 16-bit segment number register. The Z8001 also contains a 16-bit reserved register, two 16-bit program status area pointers, one 16-bit flag and control register, and one 16-bit refresh counter.

The MC68000 has eight 32-bit data registers, seven 32-bit address registers, and two implied 32-bit stack pointers. The data registers can be addressed as byte registers, word registers, or double-word registers. The address registers are used for 32-bit base addressing, 32-bit software stack operations, and word and long word address operations. The implied stack pointers are used for 32-bit base addressing and for word and long word address operations. The MC68000 also contains a 32-bit program counter and a 16-bit status register. The program counter addresses one large linear address space from a full 32-bit address, but only 24 bits are available in the present version. There is no segmentation at the CPU level. The status register contains one user byte and one system byte. The user byte contains five control bits, including the extend bit for extended operations. The system byte contains a trade mode bit, a supervisor state bit, and a three-bit interrupt mask.

System structure. The 8086 utilizes two types of multimaster buses, the local bus and the system bus. Microprocessors are always connected to a local bus, and memory and I/O usually reside on a system bus. The two buses are linked by interface components, the number of which depends on the size and complexity of the system. On the local bus, the address and data lines are multiplexed to reduce the number of processor pins. Signals coordinate up to three processors with an implied priority structure. On-chip arbitration logic enables both independent processors and coprocessors to share the bus. The system bus is functionally and electrically compatible with the Intel Multibus and provides for interconnection of multiple processing modules. This bus is composed of address lines, data lines, control lines, interrupt lines, and arbitration lines. Because it is modular in design, only a subset may be implemented, according to the needs of the application. All memory and I/O modules on the system bus are accessible to all the processing modules.

The Z-bus interconnects Z8000 family components in a master/slave fashion. The CPUs obey the Z-bus protocol directly at the chip level, and no extra circuitry is needed to generate bus signals. Multiplexing of address and data minimizes pin count without significant performance loss in "read-oriented" applications. Demultiplexing is performed when necessary within the individual modules. The daisy chain serial priority philosophy resolves interrupts/traps, bus requests, and requests for shared resources. The Z-bus is always controlled by one of the devices, which, on request, grants control to legal bus masters for bus transactions. Multiple CPUs communicate on a bus-to-bus basis, using the FIFO input/output interface units, or FIOs. Multiple FIOs may be connected to one Z-bus.

The MC68000 family is supported by two different master/slave-based multimaster buses for interconnection of components—the local bus and the global bus. The local bus connects microprocessor, memory, and I/O devices to form individual microcomputer modules, and the global bus interfaces to various local buses through bus arbitration modules. In the minimum version, the local bus uses transmission and control lines as they appear directly at the CPU. The extended version of the local bus, the Versabus, designed to support all future versions of the 68000 family, utilizes additional control lines.

Memory. The one megabyte of real addressing space in the 8086 is treated as a group of segments, each segment 64K bytes in size. Four segments are addressable at one time, providing up to 64K bytes of code, 64K bytes for stack, and 128K bytes for data. The starting address is obtained through segment registers. Segmentation permits writing of position-independent programs. Two portions

of the memory are dedicated and reserved. Physical addresses are generated by shifting the segment base value four bits to the left and adding the offset. In the case of programming code, the offset is obtained from the instruction pointer, and, in the case of operands, it is the result of calculation based on the addressing mode. This organization of memory does not provide easy management and protection.

On the Z8000 chips, addresses are always expressed in bytes. Single bytes can be read and written using the byte/word output line. The eight megabytes of directly addressable memory is split up as 128 segments, each of 64K bytes. The 23 address lines (on the Z8001) provide a seven-bit segment number and a 16-bit segment offset pointer. The two address parts can be manipulated separately or together by all the available word and long word operations. The CPU generates processor status information, which enables the address range to be increased beyond its nominal limits by physically separating code, data, and stack spaces in system and normal modes (6 × 8M bytes = 48M bytes). External logic is needed for this memory extension. The Z8010 memory management unit, or MMU, can be used with the Z8001 microprocessor to improve and expand memory addressing capabilities, randomly relocating all 128 segments in the six address spaces with translation tables for each space.

The MC68000 has 23 address lines, providing a 16M-byte addressing capability. The address space is linear with no internal segmentation. Although words are normally addressed, single bytes can be read or written using upper and lower data strobes. Instructions and multibyte data are always aligned on even byte boundaries. Similar to that of the Z8000, the processor status information separates address space into four areas: the supervisor program, the supervisor data, the user program, and the user data. The proposed memory management unit, MC68451, would support sophisticated management and protection of 32 variable-sized segments, ranging from 256 bytes to 16M bytes in increments of 256 bytes, and would allow trapping of unauthorized accesses. Without the MMU, it is possible to equip the MC68000 with a simple memory protection mechanism by separating user and supervisor space into high and low memory.

Stack organization. While Intel 8086 systems can have many stacks, each less than or equal to 64K bytes, only the current stack is directly addressable. Other stack pointers are located in memory and are implemented through the stack segment register, SS. Thus, multiple concurrent stacks are not feasible on Intel systems.

In contrast, the Z8000 can have multiple concurrent stacks; stacks can be located anywhere in memory and are addressed via stack pointer registers. Any register except RO can serve as a stack pointer by means of PUSH and POP. Call return, interrupts, and traps use implied stack. The system stack can be accessed only in system mode, whereas the normal stack can be accessed in both modes.

The MC68000 has two implied stack pointers for use in user mode and supervisor mode. Multiple concurrent user stacks and queues can be created and maintained by employing the address register indirectly with postincrement and predecrement addressing modes.

I/O mechanisms. The Intel 8086 has a 64K-byte (32K words) separate I/O space. A memory-mapped I/O capability that can respond like a memory device is available for linking I/O devices, but Intel does not recommend its use for the Multibus. Any memory reference instruction can be used to access an I/O device, providing additional programming flexibility. Word-based devices should use even addresses for maximum throughput. Intel has reserved eight locations for future products. High-speed I/O operations can be carried out with traditional DMA controllers. Intel also offers the 8089 IOP, an independent processor with two DMA channels and an instruction set tailored for I/O operations.

The Z8000 family CPUs support two different I/O address spaces of 64K bytes through special I/O instructions, which can be executed only in the system mode. Standard I/O instructions transfer data between the CPU and peripherals, and special I/O instructions transfer data to and from external CPU support chips. Processor status information enables separation of address spaces. The I/O addressing scheme is identical to the basic memory addressing scheme. For DMA operations, two signals, bus request and bus acknowledge, are available. Inhibited from controlling the bus during DMA operations, the CPU must wait for the bus to be given up by the DMA controller.

The MC68000 possesses no separate I/O space. All I/O is memory-mapped and all I/O protection must occur at the memory protection level. Three signals, bus request, bus grant, and bus grant acknowledge, allow master devices to get control of the bus for DMA operations. The three signals are used by potential bus masters to decide who will be the next bus master. The actual arbitration protocol handles overlapped arbitration and data transfer and resolves multiple simultaneous bus requests. The CPU has been designed to operate in conjunction with the MC68450, a direct memory access controller scheduled to be available during 1981, which will allow block transfer rates of up to four megabytes per second.

Software. As mentioned previously, the 8086 is an improved and expanded version of the 8080. The 8080's basic eight-bit instructions have been retained, and expanded with extended instruction lengths when necessary. For efficient code, the instructions most often executed are only a single byte long. Implied register addressing also reduces code size. To allow for expansion of the instruction set, an "escape" facility is available for transferring control to a coprocessor. The 8086 instruction set provides automatic repetition of many nondecision-making instructions, large I/O space with register indirect addressing, decimal operations, error traps, and software traps. The addressing highlights of the 8086 include the ability to finely segment memory, and the facilities for indexing with displacement and without displacement.

The Z8000 achieves high speeds through random logic encoding. Code is space-efficient because the instructions most often executed are shortest in length, and because it distinguishes between long branches and short branches. The Z8000 also has an expandable instruction set. Unlike the 8086, it does not use implied registers. Zilog provides

16 completely general registers and consciously avoids specialized ones. The instruction set facilitates multiprogramming through a context switching facility. Other instruction highlights include signed 32-bit multiply and divide, decimal operations, multiple load, vector-based instructions, and the test and set instruction, which is especially valuable in multiprocessor applications. Addressing schemes include indexing, with and without displacement, and multiple increment indexing. Multiple stacks, segmented memory, and the very large address space (48M bytes) ease programming effort. Finally, the user/supervisor stacks are all hardwired.

The MC68000 has a regular instruction set and provides multiuser support. It emphasizes space-efficient code through "quick" instructions and short jumps on loops. The MC68000 offers the advantage of excellent debug tools like single-step execution, traps on illegal instructions, and debug mode. The instruction set can be expanded by remasking the microcode or by traps. Context switching facilitates multiprogramming, and the test and set instruction aids in multiprocessor and data-base applications. Other advantages indude complex push and pop capabilities, the 32-bit internal structure, and instructions for multiple load and signed multiply and divide. It is possible to address 16M bytes directly and 64M bytes through functional segmentation. Post/pre increment/decrement facilities are available for most instructions. Real-time control applications are aided by multilevel interrupt and seven auto-vector interrupt capabilities.

National 16000. National Semiconductor has announced a family of 16-bit microprocessors, and sample production is expected to begin in 1981. The 16000 series consists of the NS16008, NS16016, and NS16032 processors. Of these, the NS16008 and the NS16016 are very similar, each offering an internal data ALU bus 16 bits wide and a direct addressing range of 64K bytes. Further, either of these two chips can operate in two distinct modes:

(1) native mode, in which the two processors have 100 basic instructions and are directly compatible with the NS16032;
(2) 8080 compatibility mode, which permits direct emulation of the 8080, with a speed four times that of the 8080.

Transfer from one mode to another within a program is implemented with an ESCAPE instruction. No separate translator and assembly programs are needed.

The NS16008 and NS16016 processors are designed to bridge the gap between the 8080 and the high-end members of the NS16000 family. The NS16008 and 16016 have 16-bit address pointers that are upwardly compatible through software to the 16032 address space. The primary difference between them is that the NS16016 has a 16-bit data bus, whereas the NS16008 has only an eight-bit data bus and is primarily suitable for use in systems with eight-bit-wide memory and peripherals.

The NS16032 achieves an address range of 32M bytes by means of a memory management unit, or MMU. However, it does not have an 8080 compatibility mode. The NS16032 has an internal data ALU bus that is 32-bits wide and a direct address range of 16M bytes using 24-bit address pointers. Unlike the 16032, the 16008 cannot be supplemented with an MMU to increase the address space.

All the National microprocessors have eight general-purpose registers that can be used (without any restrictions) as base registers and index registers. Instructions are not register-specific and can make use of every relevant addressing mode, including scale index (powerful when using high-level languages), external address (used to construct modular software), and memory relative. Furthermore, the symmetry between registers and memory means that each memory location can serve as an accumulator or base register as needed. The NS16081 floating-point unit, or FPU, has an additional set of eight general-purpose registers, supplementing the GPRs on the master processor. The MMU, NS16082, can serve as a second slave processor.

The main CPU has eight dedicated registers: program counter register, processor status register, user stack pointer, interrupt stack pointer, frame pointer, static base register, mod register (for module map), and interrupt base register. The MMU provides eight dedicated registers, and the FPU provides one floating-point status register. The NS16000 family offers several symmetric addressing modes, including top of stack addressing, memory relative addressing, external addressing, and scaled indexing. National is unique in providing modular software capabilities for the new microprocessors, permitting a user to develop a software package independent of all other packages and without regard to individual addressing. This provides flexibility in system design and lower programming costs. The ROM code is totally relocatable and easy to access. Within the system, a module consists of three components: a code component (contains the code that the processor executes in a given module), a static data component (contains local variables and data for the particular module), and a linkage component (contains all information required to link references from one module to another).

National has attempted to provide as much compatibility as feasible. The floating-point unit, the NS16081, is compatible with the proposed IEEE floating-point formats by means of its hardware and software features. It can be driven not only by National microprocessors but also by any Microbus-compatible CPU.

Microcomputers

Several microcomputers configured around 16-bit microprocessors are now available. Intel Corporation offers the SDK-86 based on the 8086, Zilog offers the Z8000 Development Module based on the Z8002, and Motorola offers the MEX-68-KDM based on the MC68000. Several independent system houses offer equivalent systems. The broad features of various systems are summarized in Table2.

Multiprocessor capabilities

To increase computational bandwidth and/or system resilience, integration of several microprocessors in a

single system frequently becomes necessary. The overall throughput and efficiency of such systems is directly dependent on the hardware and software interconnection mechanisms supported by the basic microprocessor chips. Many different interconnection systems have evolved over the years, but the single timeshared bus offers distinct advantages as an interconnection mechanism for multimicroprocessor systems. Under such a scheme, different modules can share the bus resource equally on a time-multiplexed or demand-multiplexed basis. However, the internal design of the present 16-bit microprocessors does not facilitate efficient concurrent operation of a large number of processors on such a bus.

Intel 8086. The Multibus is the structure for interfacing Intel's 8080/85/86 products. It supports a one-megabyte address space. The 8289 bus arbiter controls Multibus accesses by multiple masters. The control lines are designed according to a master-slave concept: a master (processor) in the system takes control of the Multibus; then the slave device (I/O or memory), upon recognizing its address, acts upon the command provided by the master. An asynchronous handshaking protocol allows modules of different speeds to use the bus. Although the basic definition in the bus standard specifies only two types of units—bus masters and bus slaves—the system also can include "intelligent" slaves, which cannot control the bus, but put more processing power into the bus slave. Multiple masters can be connected in either a daisy chain priority scheme or in a parallel priority scheme.

Coordination features of the 8086 multiprocessor include

- the 8289 bus arbiter, which decides which master may use the bus during the next cycle;
- the bus lock signal, activated on execution of lock prefix instructions, blocking interrupts and requests by other processors until the lock sequence is completed;
- semaphore using the lock prefix in conjunction with the XCHG instruction;
- synchronization to an external event using a WAIT instruction and the test input signal;
- escape instruction allowing other processors to obtain an instruction and/or a memory operand from the host;
- two bidirectional request/grant lines, used to share the local bus between one host and two other pro-

Table 2.
Microcomputer characteristics.

	ADVANCED MICRO COMPUTERS 96/4016	ADVANCED MICRO COMPUTERS 96/4116	INTEL ISBC 86/12A	INTEL SDK-86	MICRODA-SYS MD-68K	MOTOROLA MEX-68-KDM	ZILOG 05-6101-01
GENERAL							
PROCESSOR USED	AMZ8002	Z8000	8086	8086	68000	68000	Z8002
WORD SIZE (BITS)	16	16	16	16	16	16	16
ADDRESSING							
ADDRESS SIZE (BITS)	16	16	20	20	24	24	16
TOTAL MEMORY ADDRESSABLE (BYTES)	64K	160K	1M	1M	4M	16M	64K
AMT. OF RAM ON CARD (BYTES)	8K	32K	32-64K	2-4K	128K	32K	32-48K
AMT. OF ROM ON CARD (BYTES)	0-12K	0-8K	0-32K	8K	0-16K	8-64K	4-16K
DMA CAPABILITY	NO	YES	YES	YES	YES	NO	YES
FREQUENCY, ETC.							
CLOCK FREQUENCY (MHz)	4	4	5	2.5 or 5.0	?	8	2.5 or 3.9
SUPPLY VOLTAGES	+5, +12	+5, +12	+5, +12	+5	+5, +12	+5, +12	+5, +12
BOARD SIZE (IN)	6.75×12	6.75×12	6.75×12	12×13.5	12×15	9.75×14	11×14
I/O CAPABILITY							
BUS TYPE	SPECIAL	MULTIBUS	MULTIBUS	SPECIAL	SPECIAL	EXORCISER	SPECIAL
PARALLEL I/O LINES	24+	24+	24	48+	32+	32+	32+
NUMBER OF I/O PORTS	2	2	1	1	4	2	2
MAX I/O RATE (K BAUD)	38.4	19.2	38.4	4.8	300	9.6	19.2
ADDTL. H'WARE DETAILS							
INTERRUPT PROVISIONS	YES	YES	YES	YES	YES	YES	YES
MULTIPROCESSING CAPABILITY	NO	YES	YES	NO	YES	YES	NO
NO. OF TIMERS	3	5	2	?	4	3	5
BITS PER TIMER	16	16	16	?	16	16	8
SOFTWARE							
OPERATING SYSTEM	YES	YES	YES	YES	NO	YES	YES
HIGH-LEVEL LANGUAGE(S)	YES	YES	YES	YES	NO	YES	YES
ASSEMBLER	YES	YES	YES	YES	YES	YES	YES
DEBUGGING AIDS	YES	YES	YES	YES	YES	YES	YES
APPLICATION PACKAGES	NO	NO	YES	NO	NO	NO	YES

cessors via a handshake sequence—request, grant, release;
- the 8288 bus controller that outputs system bus signals compatible with Multibus.

Z8000. Two different multimicroprocessor mechanisms are possible on the Z8000. Zilog has designed a FIFO buffer communication module, which can run each processor as a separate system and pass messages back and forth through buffers to achieve total system communication. The processors are very loosely coupled, and any high-speed resource sharing is virtually impossible.

The second multiprocessor mechanism employs two signal pins called micro-in and micro-out (MI and MO) for implementation of a daisy-chained, software-controlled, global priority scheme. A processor examines the chain for busy condition (global resource allocation locked). If the bus is not busy, the processor places a request into the chain and then re-examines it after a settling delay (to prevent races). The result of the operation is reported with a flag handled in software. Thus, with an appropriate software driver, a single global locking scheme can be implemented. However, the time required to operate this locking mechanism rules out any high-speed communication.

Multimicroprocessor operation with the Z8000 is facilitated by the following features:

- four special, privileged "multimicro" instructions—MBIT, MREQ, MRES, and MSET;
- pins for bus request, bus acknowledge, multimicro in, multimicro out, and segment trap;
- test and set instructions, TSET and TSEB;
- special output instructions;
- bus arbitration mechanisms;
- normal and system modes;
- provision for asynchronous Z-bus to Z-bus communication using the Z8038 FIO;
- simple external SSI logic to establish actual daisy-chain;
- semaphore using TSET (test and set) to synchronize software processes that require exclusive access to certain data or instructions at one time;
- sharing of large memory by various processors under the memory management scheme.

On the Z8000, six op-codes have been reserved for extended instructions to be used in conjunction with extended processing units (coprocessors).

MC68000. In a Motorola environment, each processor has a local bus with local memory and peripherals. A global bus connects all local buses together through bus arbitration modules (BAMs). A processor is free to execute at full speed in its own bus space until it needs something from another processor's area, or until another device needs something from the former's domain. This is not a true multiprocessing system, but rather a connected group of individual microcomputer systems. Resources are not equally available to each processor. Any access involving the global bus takes longer than a simple local access. Access from the global bus back to a local bus is obtained through a DMA operation. There are no strictly global, shared resources, and the mechanism is suitable only for low, nonlocal access rates. Also, there is nothing to prevent several processors from making continuous accesses into one processor, effectively stopping that processor entirely. With the priority on the global bus fixed, a processor with the lowest priority may never get a global transaction started or completed.

MC68000 multimicroprocessor operation is facilitated by

- bus arbitration modules (BAMs), which provide support in global bus multiprocessor design;
- the TAS (test and set) instruction;
- signals for bus request, bus grant, and bus grant acknowledge, which provide necessary input signals for arbitration purposes. Such arbitration requires some external hardware.

Interlocked multiprocessor communication is achieved through an indivisible read-modify-write cycle. For this purpose, the TAS instruction is used, and the address strobe is asserted throughout the cycle to inhibit other bus members from accessing the bus. The bus arbitration handles overlapped bus arbitration and transmission; however, it is not very powerful for multiple CPUs. The extended bus arbitration provided by Versabus is more powerful, but the inherent master/slave nature of its protocol presents a major bottleneck as the number of processors increases.

NS16000. The NS16000 series uses local buses and system buses. The local bus can connect the NS16032 CPU to the NS16081 floating-point unit, the NS16082 memory management unit, and the NS16203 DMA controller. The system bus is used for communications to other processors and global memory, and also to the bus arbiter and the interrupt control unit. The two buses communicate through "drivers" and "address latch" circuitry. It is too early to comment on specific system capabilities and potential bottlenecks of the National bus protocols.

Multiprocessing overview. In all the 16-bit chips, support for multiprocessing is rather primitive, and one must consciously avoid the various pitfalls mentioned above. National Semiconductor still has to make known the details of its more sophisticated mechanisms. Among the other multiprocessors, the amount of resource sharing in the 8086 Multibus design is more restricted than that in the MC68000 local/global bus structure. Although all resources on the system bus can be accessed by any master, local bus resources are directly accessible only by the resident 8086. A further constraint imposed by Multibus is the fixed master-slave relationship of devices on the system bus, limiting interprocessor communication to the level of mailbox messages via global memory. Multibus, like the MC68000 local/global structure, is subject to saturation by high-priority devices. Individual transactions on Multibus are much faster than those on the MC68000 bus for two reasons: fetches from MC68000 local memory involve contention with the local processor, while Multibus global memory fetches do not; MC68000 inter-BAM communication adds two additional steps to

the global memory access procedure. The Z8000 offers special signal pins, MI and MO, and four special instructions to support multiprocessing.

Selection strategy

In selecting a microprocessor for a particular application, one must analyze a spectrum of issues, both technical and nontechnical. Let us consider the relevant technical issues first.

Technical issues. The operational speeds of all the 16-bit microprocessors have improved over the previous generation 8080, Z80, and MC6800 processors. The shortest execution (assuming sufficiently fast memory) is 400 ns for the 8086, 750 ns for the Z8000, and 500 ns for the MC68000. In all three microprocessors, extended address ranges allow large memory sizes to be directly accessed. The upper limit on directly addressable memory is one megabyte on the 8086, 48M bytes on the Z8000, and 64M bytes on the MC68000. Such large memory space requires some form of management. The internal segment registers of the 8086 provide internally controlled memory management via relocation. Both the Z8000 and the MC68000 are designed to be used with an external memory management chip, which allows increased function by increasing silicon area. These management units can relocate, check bounds, and check functions of all references to support very sophisticated memory mapping and protection facilities.

The 8086 and Z8000 have separate I/O addressing facilities, while the MC68000 uses memory-mapped I/O. Separate I/O space makes system memory design and management easier. Memory-mapped I/O allows all memory referencing instructions to also be I/O-referenced. This saves instructions, but the I/O cannot be protected at the instruction level; it can be protected only at the memory level.

A valuable feature of the Z8000 and MC68000 processors is the implementation of supervisor/user mode separations, allowing the protection of certain instructions and separate system/user stack pointers. The 8086 does not offer such facilities. Also, the Z8000 and the MC68000 can handle 32-bit operands.

The basic speed of instruction execution is an important selection criterion. Available independent benchmark studies do not cover the NS16000 series, which is not yet commercially available. Hence, we must use the figures published by National Semiconductor. The speed data, listed in Table 3 for the four microprocessors, must be interpreted with caution. Actual throughput is a function of the exact instruction sequence, displacements, data lengths, clock frequency, and other factors. Also, the numbers may represent a slight positive bias in favor of National. Overall, the MC68000 is the best on the various branch operations. For simple data transfer operations, the NS16032 and the MC68000 are superior to the 8086 and the Z8000.

The direct address sizes supported by the various microprocessors are considerably different, making the machines suitable for different application areas. Simple text editing, for example, generally requires less memory than data-base management, and memory requirements increase in direct proportion to the number of users simultaneously on-line. Thus, each processor has its own application niche (see Figure 4). The sophisticated addressing modes and segmentation schemes used in both the MC68000 and the Z8000 families simplify the implementation of large programs. Conversely, the small address space of the NS16008 and the NS16016 prevents their use for any large-scale programs.

Software is another factor determining application suitability of the various microprocessors. The Z8000 is

Table 3.
Execution speeds (in microseconds) of 16-bit microprocessors.

OPERATION	DATA TYPE	8086	Z8000	MC68000	NS16032
REGISTER-TO- REGISTER MOVE	BYTE/WORD DOUBLE-WORD	0.40 0.80	0.75 1.25	0.50 0.50	0.30 0.30
MEMORY-TO- REGISTER MOVE	BYTE/WORD DOUBLE-WORD	3.40 6.80	3.50 4.25	1.50 2.00	1.00 1.40
MEMORY-TO- MEMORY MOVE	BYTE/WORD DOUBLE-WORD	7.00 14.00	7.00 8.50	2.50 3.75	1.60 2.40
ADD MEMORY TO REGISTER	BYTE/WORD DOUBLE-WORD	3.60 7.20	3.75 5.25	1.50 2.25	1.10 1.50
COMPARE MEMORY TO MEMORY	BYTE/WORD DOUBLE-WORD	7.00 14.00	7.25 9.50	3.00 4.00	1.80 2.60
MULTIPLY MEMORY TO MEMORY	BYTE WORD DOUBLE-WORD	13.00 23.00 115.20	20.25 16.00 85.75	N/A 8.75 43.00	2.80 4.60 7.60
CONDITIONAL BRANCH	BRANCH TAKEN BRANCH NOT TAKEN	1.60 0.80	1.50 1.50	1.25 1.00	1.40 0.70
MODIFY INDEX BRANCH IF ZERO	BRANCH TAKEN	2.20	2.75	1.25	1.30
BRANCH TO SUBROUTINE		3.80	3.75	2.25	2.50

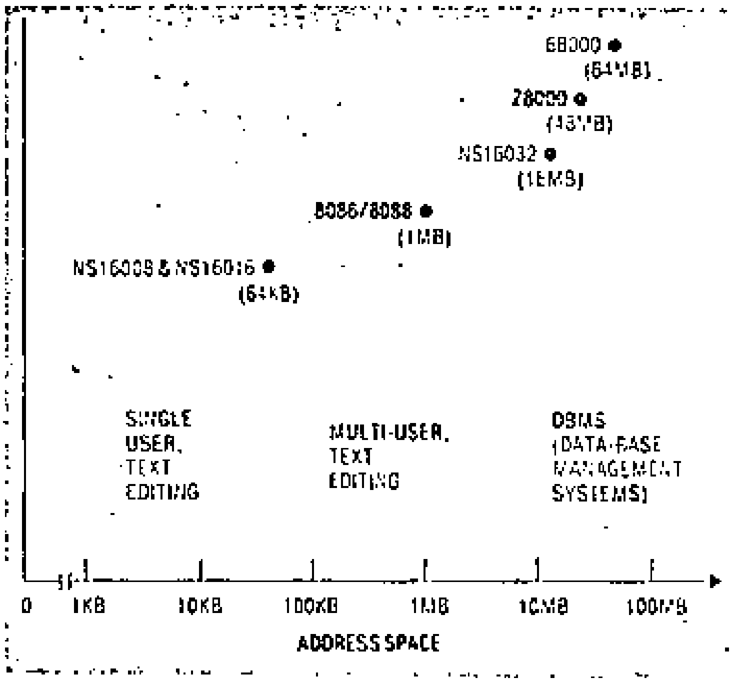


Figure 4. Application niches for 16-bit microprocessors.

best suited for word processing and text editing applications because of its sophisticated string instruction repertoire. On the other hand, the MC68000 lacks this capability but offers excellent support for handling interrupts and thus is suitable for real-time and control systems and also for multitasking. The perceived market for Intel 8086 systems is largely to upgrade earlier Intel products and to serve as a steppingstone to 16-bit applications. The National products, according to current specifications, are attractive across the range because of their direct upward compatibility with the Intel 8080, as well as their string-processing and interrupt-handling capabilities.

Another major evaluation dimension is the availability and type of support chips. Such chips greatly facilitate particular functions—e.g., memory management, bus arbitration, floating-point operations, array processing, and a whole spectrum of I/O operations. Although all chip vendors are involved in the development of "coprocessor," "slave," or support chips, only a few such chips are commercially available. Since the 8086 has been in existence much longer than the other processors, it possesses a distinct advantage in this realm. Table 4 presents a listing of the various support chips. In most cases, the new 16-bit microprocessors will interface with support chips designed earlier for eight-bit microprocessors. This may result, however, in a substantial performance loss, as the earlier chips use relatively obsolete technology, operate at lower speeds, and use fewer data lines. The new chips cost more than the earlier ones—thus the price-performance trade-offs of the two options must be considered in the design of any new system.

The software conversion costs of upgrading to a 16-bit microprocessor are not insignificant. A system presently using an Intel 8080 must convert either to an Intel 8086 or a National Semiconductor 16008/16016. These alternatives will yield a speed increase of a factor of 4-6. It is possible to obtain higher speeds with an NS16032 or a Motorola 68000, but the costs of rewriting software plus the associated costs of debugging can more than offset the gain. Likewise, the only appropriate upward path from an MC6800 is to an MC68000. The Z8000 has a basic structure different from the previous Z80; hence, one must fully analyze present programs before making a final choice.

The major features discussed in this section are listed and evaluated in Table 5 (A is excellent; B is good; C is fair; D is poor).

Commercial issues. The availability of vendor support and second sourcing is critical, especially for high-volume systems. (Second sources for the 16-bit microprocessor-based systems were listed in the second paragraph of this article.) It is unwise to depend on a single source for the supply of all chips. So far, no second source has been identified for the National microprocessors, and this constraint must be kept in mind.

We come finally to the pertinent issue of new product versus old product. The Intel 8086 has been available since 1978, the Zilog 8000 since 1979, and the Motorola

Table 4.
Support chips for 16-bit microprocessors.

TYPE OF CHIP	INTEL 8086	Z8000	MC68000	NS16000
MEMORY MANAGEMENT		Z8010	68451	NS16082 (NOT USEABLE W/NS16008)
BUS ARBITER	8289	Z8001/8002 MAY BE USED		NS16024
FLOATING-POINT	8087		68000X	NS16081
DMA CONTROLLER	8089/8237	Z8016	68450	NS16023
INTERRUPT CONTROL UNIT	8259A			NS16202
I/O PROCESSOR/INTERFACE	8089	Z8038		
PERIPHERAL CONTROLLER	8041A/8741A	Z8034	68120	
FLOPPY DISK CONTROLLER	8271/8271-6/8771-8			
CRT CONTROLLER	8275	Z8052		
ARRAY PROCESSOR				
BUBBLE MEMORY CONTROLLER			68453	

68000 since January 1980. The number of Intel users is the largest. Hence, the company's spectrum of support chips is also the largest, and there is a much lower probability of a bug in the software. The MC68000 can use the large number of support chips designed earlier for the MC6800 family. However, very few support chips designed exclusively for the MC68000 are presently available. National Semiconductor support products will be available this year. But by that time, Intel will also have new iAPX series products, which will offer higher speeds, wider data paths, and superior addressing facilities. Microprocessing is a dynamic world, and we can always expect newer and more powerful chips. ◘

Table 5.
Ranking of 16-bit microprocessors.

	INTEL 8086	Z8000	MC68000	NS16000
SPEED	C	B	A	A
NUMBER OF REGISTERS	B	A	A	C
ADDRESS RANGE	D	A	A	B
COMPATIBILITY W/EARLIER MICROPROCESSORS	A	B	B	B
SUPPORT CHIPS	A	B	C	D
MULTIPROCESSING CAPABILITY	B	B	B	C
SECOND SOURCE	A	A	A	D

Acknowledgment

The authors sincerely thank John-Francis Mergen and Svein Ove Strommen for their valuable assistance in the preparation of this article.

Bibliography

Brooks, F. P., "An Overview of Microcomputer Architecture and Software," *Micro Architecture*, EUROMICRO 1976 Proceedings, pp. 1-3a.

Childs, R. E., "Multiple Microprocessor Systems: Goals, Limitations and Alternatives," *Digest of Papers COMPCON Spring 79*, pp. 94-97.

Enslow, P. H., Jr., ed., *Multiprocessors and Parallel Processing*, John Wiley, New York, 1974.

Franklin, M. A., S. A. Kahn, and M. J. Stucki, "Design Issues in the Development of a Modular Multiprocessor Communications Network," *Sixth Ann. Symp. Computer Architecture*, Apr. 23-25, 1979, pp. 182-187.

Fung, K. T., and H. C. Torng, "On the Analysis of Memory Conflicts and Bus Contentions in a Multiple-Microprocessor System," *IEEE Trans. Computers*, Vol. C-27, No. 1, Jan. 1979, pp. 28-37.

Harris, J. A., and D. R. Smith, "Hierarchical Multiprocessor Organizations," *Fourth Ann. Symp. Computer Architecture*, Mar. 23-25, 1977, pp. 41-48.

Intel Corp., "8086 User's Guide," and other 8086 technical publications, 3065 Bowers Ave., Santa Clara, CA 95051.

Lipovski, G. J., "On Virtual Memories and Micronetworks," *Proc. Fourth Ann. Symp. Computer Architecture*, Mar. 23-25, 1977, pp. 125-134.

Motorola Semiconductor, "MC68000 Microprocessor User's Manual," and other MC68000 technical publications, Motorola IC Division, 3501 Ed Bluestein Blvd., Austin, TX 78721.

Myers, G., *Advances in Computer Architecture*, John Wiley, New York, 1978.

National Semiconductor Corp., "The NS16000 Family of 16-Bit Microprocessors," and other NS16000 technical publications, 2900 Semiconductor Dr., Santa Clara, CA 95051.

Patel, J. H., "Processor-Memory Interconnections for Multiprocessors," *Proc. Sixth Ann. Symp. Computer Architecture*, Apr. 23-25, 1979, pp. 168-177.

Thurber, K. J., and G. M. Masson, *Distributed Processor Communication Architecture*, Lexington Books, Lexington, MA, 1979.

Toong, H. D., J. F. Mergen, and C. J. Smith, "Issues of Advanced Microprocessor Architecture," Technical Report #4, M.I.T. internal monograph, July 1979.

Toong, H. D., S. O. Strommen, and E. R. Goodrich II, "A General Multimicroprocessor Interconnection Mechanism for Non-Numeric Processing," *Proc. Fifth Workshop on Computer Architecture for Non-Numeric Processing*, 1980, pp. 115-123.

Zilog Corp., "Z8000 User's Guide," and other Z8000 technical publications, 10460 Bubb Rd., Cupertino, CA 95014.

Hoo-min D. Toong is an assistant professor of management in the Sloan School of Management at MIT, which he joined in 1978. He is in charge of the Digital Systems Laboratory of the Center for Information Systems Research of MIT. Before joining the Sloan School, he was a faculty member of the Electrical Engineering and Computer Science Department of MIT. His research interests are VLDB architectures, multiprocessors, distributed operating systems, and the organizational impact of such systems.

Toong received the BS in 1967, the MS and EE in 1969, and the PhD in electrical engineering and computer science in 1974 from MIT. He is a member of Tau Beta Pi, Eta Kappa Nu, Sigma Xi, IEEE, and ACM. .

Amar Gupta is a visiting research fellow in the Sloan School of Management of MIT. His research interests include multiprocessor architectures, performance measurement, and office automation. He has participated in research dealing with tightly coupled SIMD and MIMD machines.

Gupta received his B.Tech in 1974 from the Indian Institute of Technology, Kanpur, his MS in 1980 from MIT's Sloan School, and his PhD in 1980 from the Indian Institute of Technology, Delhi.

The authors' address is Massachusetts Institute of Technology, Sloan School of Management, 50 Memorial Dr., Cambridge, MA 02139.

Single-chip n-MOS microcomputer processes signals in real time

Analog inputs and outputs surround high-speed pipelining processor; user builds filters, oscillators, other analog systems with E-PROM software

by M. E. Hoff and Matt Townsend, *Intel Corp., Santa Clara, Calif.*

□ Digital processing of real-time analog signals, attractive because the waveforms can be manipulated like numbers under program control, has been confined to especially configured high-speed processors that can handle the calculations necessary. Now, just as the microprocessor put low-cost computing power into the hands of the logic designers, a device from Intel Corp. is offering analog system designers a new tool: programmability, in the form of its single-chip real-time signal processor, the 2920.

The chip was conceived out of a pressing need for a programmable circuit to serve the various protocols and modulating techniques used in telecommunications circuits. It combines analog-to-digital and digital-to-analog converters with a specially configured microcomputer to build a device with a unique instruction set, one capable of programming entire analog subsystems.

The kind of functions the 2920 can perform—filtering, modulating, detecting, limiting, mixing, and more—usually require lots of passive components, operational amplifiers, and other such discrete and linear devices. With few outboard components, the 2920 can build such complex devices as modems, equalizers, tone sources, and tone receivers; the 2920 can also be used for such nontelephonic applications as process controllers and motor or servomotor drivers.

Programmability is what gives the chip its great advantage: the contents of its on-board erasable programmable read-only memory (E-PROM) customizes it for each application.

Significant differences

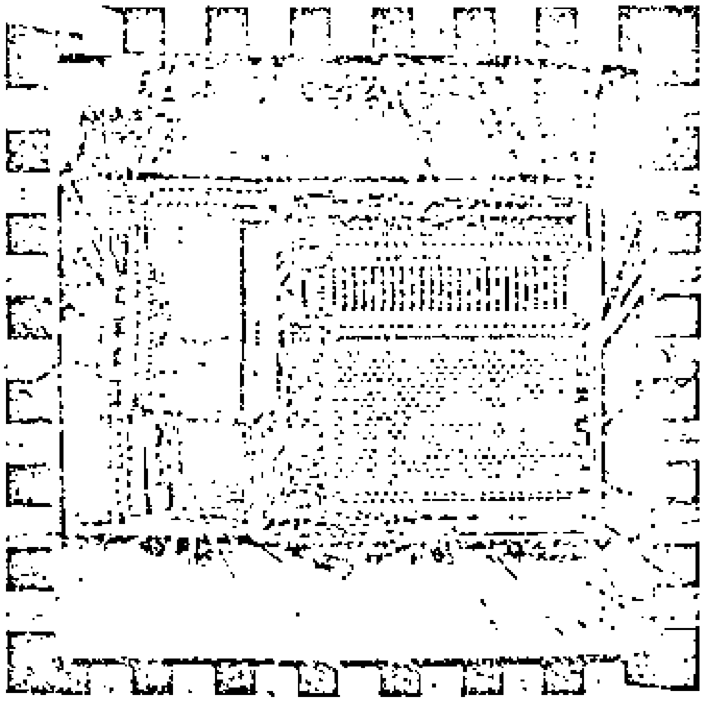
The processor in the 2920 differs significantly from a conventional general-purpose microprocessor. There is some resemblance: it processes digital data from a conventional a-d converter at the chip's input and feeds the result to an output d-a converter. However, the kinds of calculations needed for signal processing differ greatly from those of data processing.

Moreover, since the operation is based on a real-time sampling system, the 2920 must run through its entire program each time it receives a data sample from the input a-d converter. The program execution time determines the sampling rate, and that in turn restricts the operation of the microprocessor in certain ways.

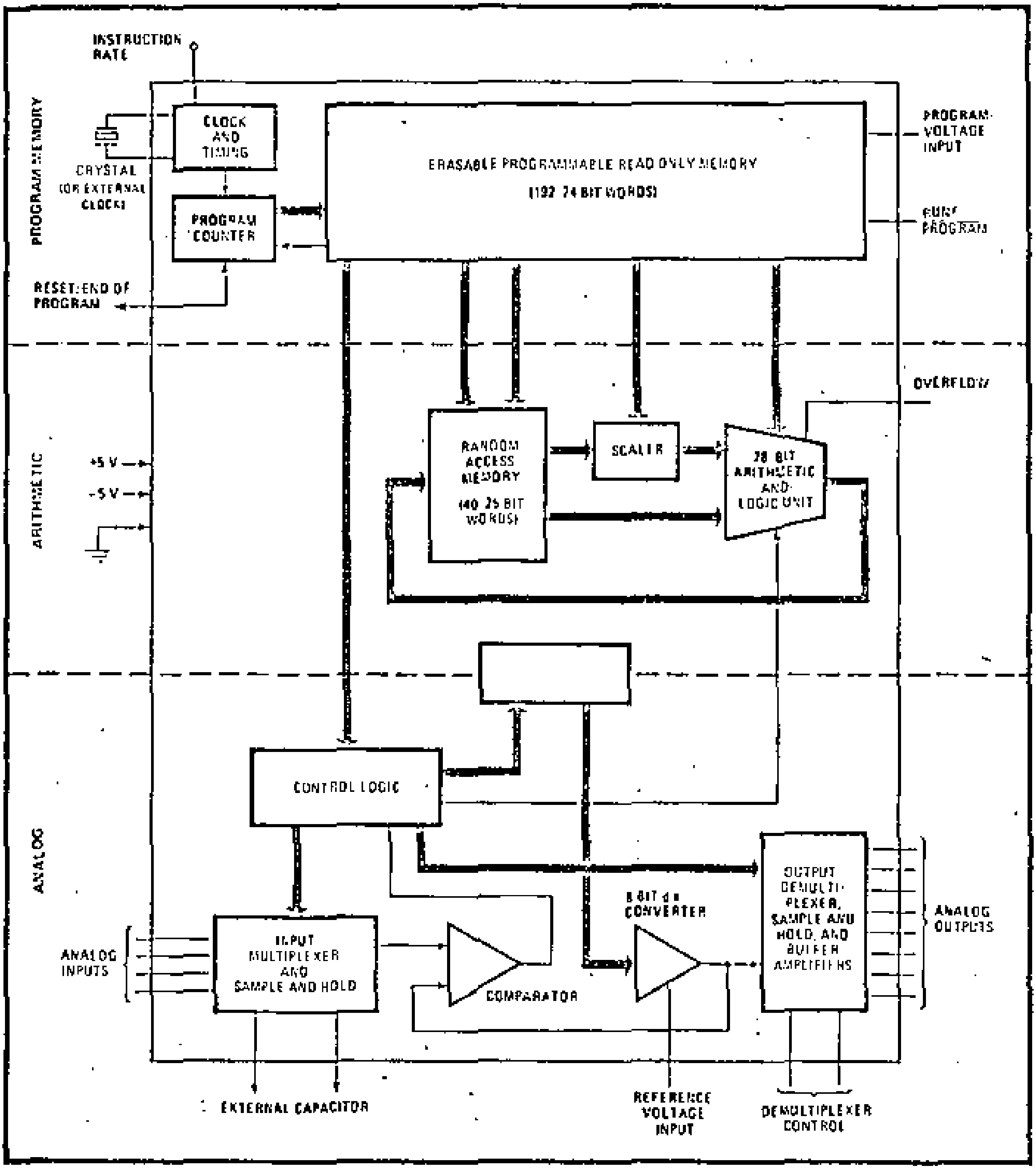
In general, a fixed rate is necessary because the characteristics of any simulated analog system are affected greatly by it—in fact, even small perturbations in sample rate can add intolerable noise to a system. Therefore, the time in which the 2920 runs through its program has perforce to be fixed.

Cut to fit

To achieve that, the 2920 executes each instruction in the same amount of time, regardless of the operation. Also, to ensure that the total program execution time will never vary, no conditional-jump instructions are in the instruction repertoire (save for the jump from the end of the program back to the beginning). Conditional



1. Analog microcomputer. Entire analog subsystems can be programmed with Intel's 2920 real-time digital signal processor. The 47,000-mil^2 device surrounds a special central processing unit with a-d and d-a converters, all under E-PROM program control.



2. Architecture. The 2920 is programmed with 24-bit instructions, up to 192 of which are stored in E-PROM. The digital section pipelines instructions to boost throughput and interfaces with the analog section through the data register, which appears as memory location.

operators, however, do allow logical operations to be carried out without any variation occurring in the program execution time.

Although these techniques do somewhat restrict programming, they establish a fixed sample rate, based on the time needed to execute one pass through the program. The rate can thus be computed simply as the instruction execution rate divided by the number of instructions in the program.

For digital processing of analog signals on a sampled basis, the microprocessor must be extremely fast if any decent bandwidth is to be attained. Nyquist's theory dictates that a continuous system can be accurately simulated on a sampled basis only if the sampling rate exceeds twice the highest frequency present in the signals being processed—and most practical situations call for an even higher sampling rate.

On top of that, since all the calculations associated

Simulating with sampling systems

The operation of the 2920 finds its basis in sampling theory, which states that a continuous function (or waveform) can be accurately represented by a train of periodic samples, as long as the sampling is done with high enough frequency. It is the high sampling rate required—and more difficult still, the immense computational grind between samples—that has restricted the digital processing of analog signals to full-blown machines. The 2920 solves the number-crunching problem with a pipelined architecture and a clever multiplication algorithm that requires many fewer circuits and steps than the shift-add algorithms with which most computers multiply numbers.

The RLC active filter shown below in A has a frequency response that produces a complex-conjugate pair of poles. The characteristics of the filter—its transfer function, gain, and pole locations—are given by the equations.

The continuous filter can be simulated by the sampling system in B. The circled Xs represent multipliers, the circled Σ is an adder, and the blocks with z^{-1} represent timing delays of one sample period. Coefficients β_1 and β_2 control the filter's frequency parameters, while coefficient G adjusts its gain.

The equations to the right of the figure give the simulated filter's response. Although a sampling system, the filter would simulate exactly the continuous function in A, were the sampling done at infinite frequency: coefficient $\beta_1 = 2e^{-aT}$ approaches the value 2 as the sample period tends to zero; similarly, coefficient $\beta_2 = -e^{-2aT}$ approaches −1. However, at finite sampling frequencies, small errors in those coefficients can cause significant changes in the characteristics of the filter, and all arithmetic must therefore be performed with high precision.

A general-purpose digital computer is able to simulate the sampling stage in B with three equations:

$$y_2 = y_1$$
$$y_1 = y_0$$
$$y_0 = \beta_1 y_1 + \beta_2 y_2 + Gx$$

where the variables to the left of each equals sign are assigned the values to its right. Note that each multiplication involves one variable and one value that is fixed by the design of the filter. Instructions in the 2920 are set up to add or subtract a variable from another where the first variable is scaled by a power of two. Thus a single instruction could take any of the following forms:

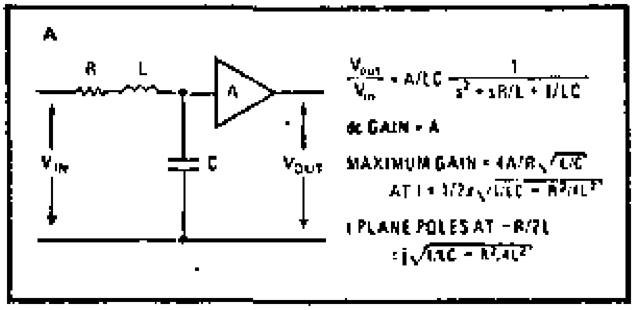
$$x = y(2^n)$$
$$x = x + y(2^n)$$
$$x = x - y(2^n)$$

The usefulness of scaling by powers of two soon becomes clear. The coefficients can be expressed in sums and differences of powers of two, as in the examples:

$$\beta_1 = 1.7656 = 2^1 - 2^{-2} + 2^{-6}$$
$$\beta_2 = -0.99414 = -2^0 + 2^{-7} - 2^{-9}$$
$$G = 0.00293 = 2^{-8} - 2^{-10}$$

which the 2920 can perform quickly and with a minimum of circuits. The filter stage of B is carried out directly with 2920 instruction in table D. A left-shift of 1 is equivalent to multiplying by 2^1, a right-shift of 6 multiplies by 2^{-6}, and so on. The mnemonics LDA, ADD, and SUB represent load, add, and subtract operating codes.

This method permits much faster multiplications by constants than a conventional shift-add multiplier could achieve. It is especially effective here because such multiplications dominate digital-filter calculations.

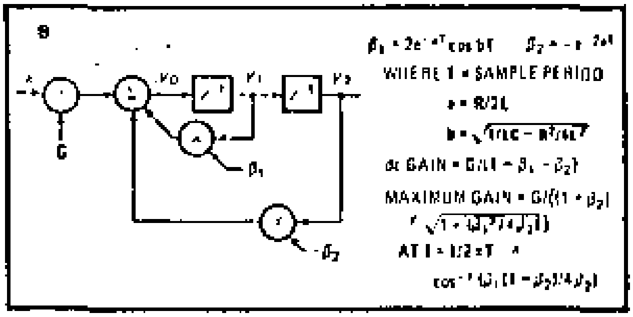


$$\frac{V_{out}}{V_{in}} = A/LC \frac{1}{s^2 + sR/L + 1/LC}$$

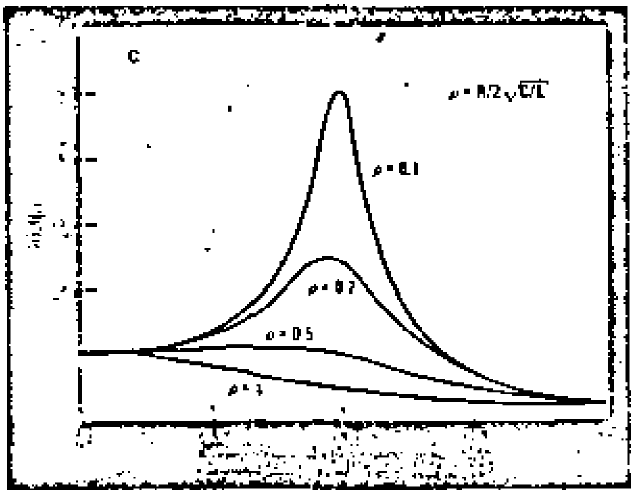
dc GAIN = A

MAXIMUM GAIN = $4A/R\sqrt{L/C}$
AT $f = 1/2\pi\sqrt{1/LC - R^2/4L^2}$

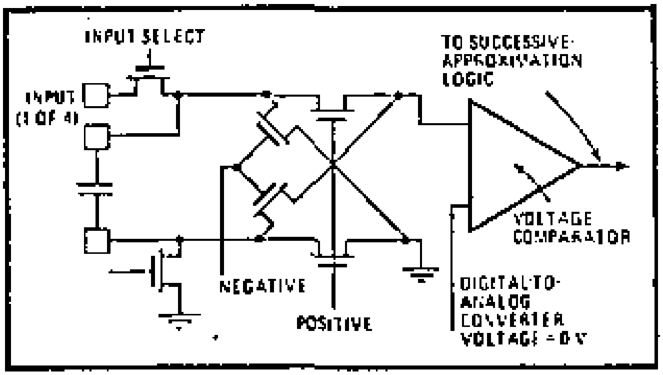
s PLANE POLES AT $-R/2L$
$\pm j\sqrt{1/LC - R^2/4L^2}$



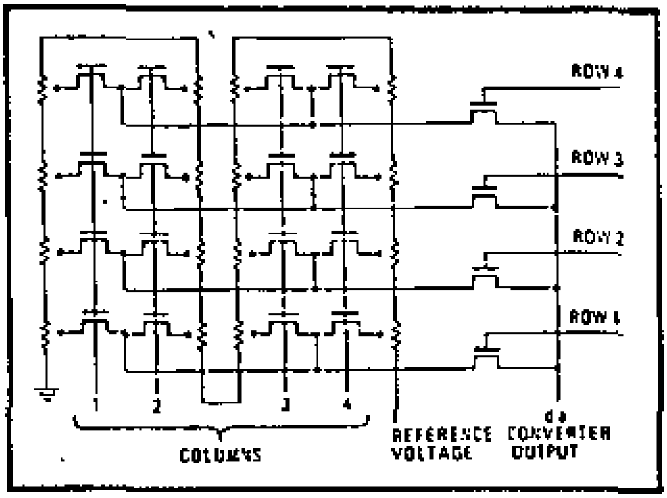
$\beta_1 = 2e^{-aT}\cos bT$ $\beta_2 = -e^{-2aT}$
WHERE T = SAMPLE PERIOD
$a = R/2L$
$b = \sqrt{1/LC - R^2/4L^2}$
dc GAIN = $G/(1 - \beta_1 - \beta_2)$
MAXIMUM GAIN = $G/((1 + \beta_2)$
$\times \sqrt{1 - \beta_1^2/4\beta_2})$
AT $f = 1/2\pi T \times$
$\cos^{-1}(\beta_1(1 - \beta_2)/4\beta_2)$



Instruction				Comments
Operation (3 bits)	Source (6 bits)	Destination (6 bits)	Shift (4 bits)	
LDA	y_1	y_2	none	equivalent to $y_2 = y_1$
LDA	y_0	y_1	none	equivalent to $y_1 = y_0$
LDA	y_2	y_0	left 1	
SUB	y_1	y_0	right 2	
ADD	y_1	y_0	right 6	now have $y_0 = \beta_1 y_1$
SUB	y_2	y_0	none	
ADD	y_2	y_0	right 7	
SUB	y_2	y_0	right 9	now have $y_0 = \beta_1 y_1 + \beta_2 y_2$
ADD	x	y_0	right 8	
SUB	x	y_0	right 10	now have $y_0 = \beta_1 y_1 + \beta_2 y_2 + G x$


INPUT SELECT
INPUT (1 OF 4)
TO SUCCESSIVE-APPROXIMATION LOGIC
VOLTAGE COMPARATOR
NEGATIVE
POSITIVE
DIGITAL-TO-ANALOG CONVERTER VOLTAGE = 0 V

3. Sign bit. Adding a sign bit to basic 8-bit conversions yields 9-bit precision. The 2920 uses a flip switch to supply a correct-polarity signal from the sample-and-hold capacitor (left) to the comparator used for successive-approximation analog-to-digital conversion.



4. Folded ladder. The 2920's analog-to-digital converter uses a resistive ladder, folded into a square array. That configuration minimizes the effects of process inconsistencies and temperature across the surface of the chip, thus improving overall accuracy.

with the production of an output sample must be performed with each sample taken, the microprocessor must have an extremely high-speed number-crunching capability. Even a 10-kilohertz bandwidth, for example would require sampling at least at a 20-kHz rate, or once every 50 microseconds. A program of, say, 100 instructions, which was executed at a rate of 50 microseconds per sample, would require an average instruction-cycle time of 500 ns—a speed that few minicomputers can boast.

But the 2920 can do it all on a 47,000-square-mil chip (Fig. 1), which, moreover, is built with a standard n-channel metal-oxide-semiconductor process.

Divided in three parts

The 2920, as diagrammed in Fig. 2, divides into three major subsections: program memory, the arithmetic (or digital-processing) portion, and the analog input and output conversion section. The E-PROM program memory controls both the analog conversion and digital sections of the chip.

Four analog inputs enter and eight analog outputs leave the 2920. A multiplexer allows the four analog inputs to share the input sample-and-hold circuit. Analog-to-digital conversion is performed by successive approximation with the output d-a converter, which is the resistor-ladder type. A demultiplexer provides eight buffered outputs, each of which has its own sample-and-hold circuit. The data register links the analog sections to the digital portion of the chip.

Unusual arithmetic

The microprocessor in the 2920 comprises a two-port scratchpad random-access memory, a binary scaler, and an arithmetic-and-logic unit. (The data register used by the analog portion of the chip is actually part of the RAM, so that the processor sees the inputs and outputs as an address location in memory.)

Although the precision of the a-d and d-a converters is 9 bits, internal arithmetic is with 25-bit precision, since accommodating the buildup of small round-off error over time requires much higher precision in the intermediate calculations than for the final value. Should an arithmetic overflow occur, the processor saturates: in other words, it automatically and instantaneously replaces the result with the largest storable value.

To handle the arithmetic, each of the 40 locations in scratchpad RAM is 25 bits wide. Addressing, however, is with a 6-bit word; the additional 24 addresses select predetermined constants and the analog section's data register. To boost throughput, the RAM was designed with dual-port cells that can be addressed through either of their ports.

The E-PROM can store up to 192 instructions of 24 bits each. The instruction format has five contiguous fields: the digital operator, the source address, the destination address, the extent of shifting, and the analog operator. Such a wide word may be likened to a microprogram word in a computer with a control store, for it performs several operations at once. In this case, they are complete memory-to-memory operations.

The lower limit

At the 2920's fastest operating speed, each instruction executes in 400 nanoseconds. The largest program the 2920 can handle—192 instructions—executes in 76.8 μs, and thus yields a minimum sampling rate of approximately 13 kHz. The worst-case bandwidth of the chip, according to Nyquist then, is about 6.5 kHz.

Shorter programs will, of course, provide higher sampling rates. Also, techniques like stacking multiple copies of a routine can boost the sampling frequency and hence the possible bandwidth.

To maximize speed of calculation, the 2920 signal processor uses pipelining techniques. Four instructions are fetched from E-PROM at once, and the fetch operation for the next four overlaps execution of the previous four instructions. Moreover, the arithmetic operations are equipped with carry-lookahead across the full width of the accumulator.

The arithmetic-and-logic unit (ALU) carries out such basic operations as data movement, addition, subtraction, absolute magnitude, and several logical operations. Each elementary machine instruction fetches two operands from the scratchpad RAM, passes the first through the binary scaler, performs the selected arithmetic func-

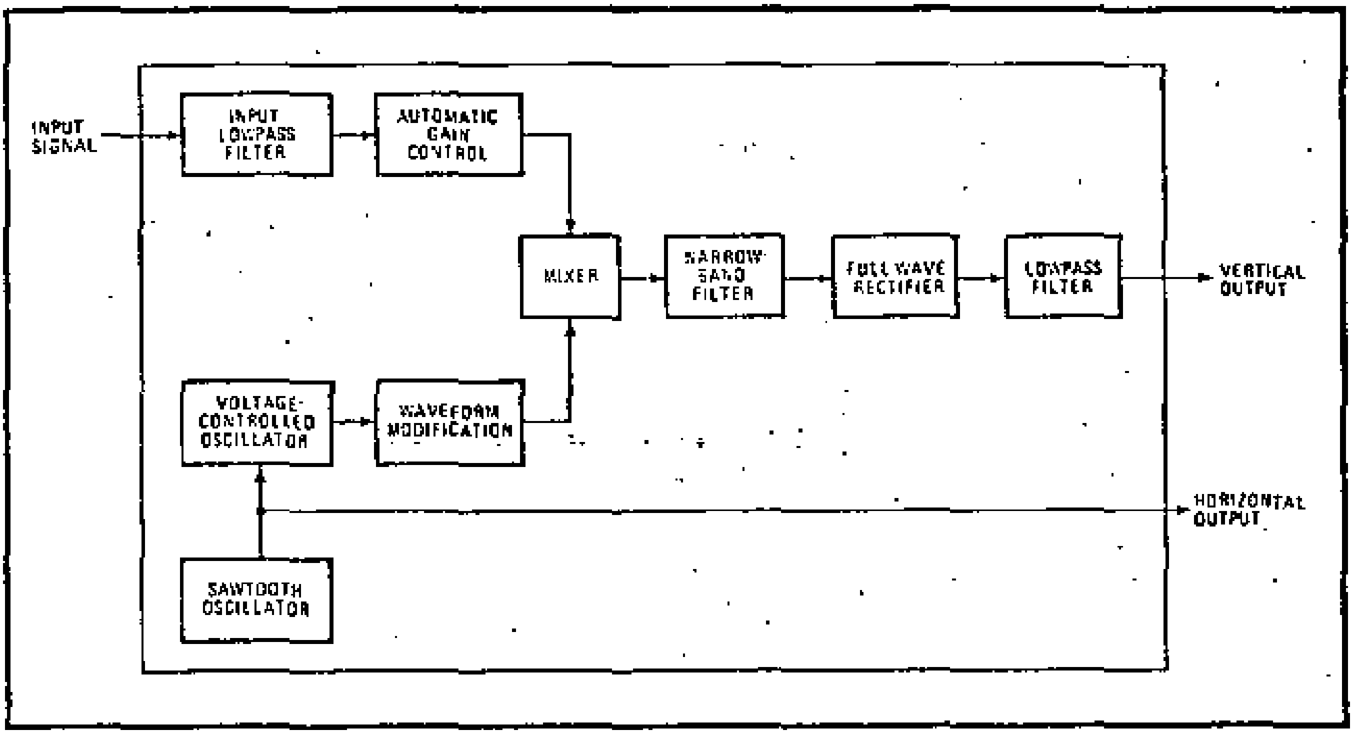
is a widened 8080 with enhanced addressing and instruction prefetch. In all, 24 addressing modes are supported. With a clock frequency of 5 MHz, the fastest instruction time is 0.4 microseconds.

The Z8000. A register-rich 16-bit processor, the Z8000 is not an enhancement of Zilog's Z80 family and has a different internal structure. The Z8000 is based on a regular register use and a symmetric instruction set. Its operating system support is far more sophisticated than that available on eight-bit machines. An entire set of registers controls systems calls and manages process swaps. Internal registers allow 32-bit double-word operations. Traps of illegal addresses and illegal instructions serve as debugging tools through the use of an expanded flag register, also permitting software expansion of the instruction set. The Z8000 is a true 16-bit machine, as data and instruction paths are 16 bits wide.

Table 1.
Specifications of 16-bit microprocessors.

	8086	Z8000	68000	16008/16016	16032
YEAR OF COMMERCIAL INTRODUCTION	1978	1979	1980	1981	1981
NO. OF BASIC INSTRUCTIONS	95	110	61	100	100
NO. OF GENERAL-PURPOSE REGISTERS	14	16	16	8	8
PIN COUNT	40	48/40	64	40	48
DIRECT ADDRESS RANGE (BYTES)	1M	48M*	16M/64M	64K/16M	16M
NUMBER OF ADDRESSING MODES	24	6	14	9	9
BASIC CLOCK FREQUENCY	5MHz (4-8MHz)	2.5-3.9MHz	5-8MHz	10MHz	10MHz
SYSTEM STRUCTURES					
UNIFORM ADDRESSABILITY			•	•	•
MODULE MAP AND MODULES				•	•
VIRTUAL					•
PRIMITIVE DATA TYPES					
BITS		•	•	•	•
INTEGER BYTE OR WORD	•	•	•	•	•
INTEGER DOUBLE-WORD		•	•		•
LOGICAL BYTE OR WORD	•	•	•	•	•
LOGICAL DOUBLE-WORD			•		•
CHARACTER STRINGS (BYTE, WORD)	•	•		•	•
CHARACTER STRINGS (DOUBLE-WORD)					•
BCD BYTE	•	•	•	•	•
BCD WORD				•	•
BCD DOUBLE-WORD					•
FLOATING-POINT				•	•
DATA STRUCTURES					
STACKS	•	•	•	•	•
ARRAYS				•	•
PACKED ARRAYS				•	•
RECORDS	•	•	•	•	•
PACKED RECORDS				•	•
STRINGS	•	•		•	•
PRIMITIVE CONTROL OPERATIONS					
CONDITION CODE PRIMITIVES		•	•	•	•
JUMP	•	•	•	•	•
CONDITIONAL BRANCH	•	•	•	•	•
SIMPLE ITERATIVE LOOP CONTROL	•	•	•	•	•
SUBROUTINE CALL	•	•	•	•	•
MULTIWAY BRANCH				•	•
CONTROL STRUCTURE					
EXTERNAL PROCEDURE CALL				•	•
SEMAPHORES	•	•	•	•	•
TRAPS	•	•	•	•	•
INTERRUPTS	•	•	•	•	•
SUPERVISOR CALL		•	•		•
OTHERS					
USER MICROCODE			•		
DEBUG MODE			•		

*6 SEGMENTS OF 8M EACH



5. One-chip spectrum analyzer. The 2920's resources are sufficient for building an audio spectrum analyzer. The input signal is gain-controlled, then heterodyned with a swept oscillator; the detected sum signal drives the vertical input of an oscilloscope.

tion, and replaces the second operand with the results of the operation.

The key to the 2920's high-speed, high-precision arithmetic is the binary scaler. Unlike the usual shift-add multipliers, which require a cycle per bit, the 2920 uses sequences of scaled additions and subtractions (see "Realizing a complex-conjugate pole pair," p. 107) in multiplying variables by constants to reduce the number of cycles to about a third. (Since most filters used in analog applications are fixed, multiplications are usually by constants.)

The binary scaler modifies a value passing through it by in effect multiplying it by a power of two, or 2^k, where k ranges from +2 to −13. If, as a result, the product is greater than 25 bits (and hence too large for the RAM), the ALU saturates and provides a signal on its overflow output pin. The overflow output is useful during system debugging for determining when scaled variables are out of range.

Conditional options

Some of the ALU's basic operations can operate conditionally, using selected bits of the data register normally associated with a-d and d-a conversions. The multiplication or division of one variable by another is made possible by conditional addition and subtraction, respectively. Finally, conditional operations can perform logic and can generate discontinuous transfer functions.

The a-d and d-a conversions are given 9-bit precision with the addition of a sign bit. A flip-switch circuit, which is shown in Fig. 3, provides the double economy of appending the sign bit while at the same time allowing the 2920 signal processor to use only a single positive-voltage reference.

The d-a converter is built around the folded resistor string and switch array shown in Fig. 4. Folding the resistor string lessens the converter's sensitivity to temperature and process variations across the surface of the chip. That, coupled with the facts that processor timing is crystal-controlled, that the converter's accuracy is established by an external reference voltage, and that all internal calculations are digital, adds up to an analog subsystem that is far stabler than fully analog counterparts.

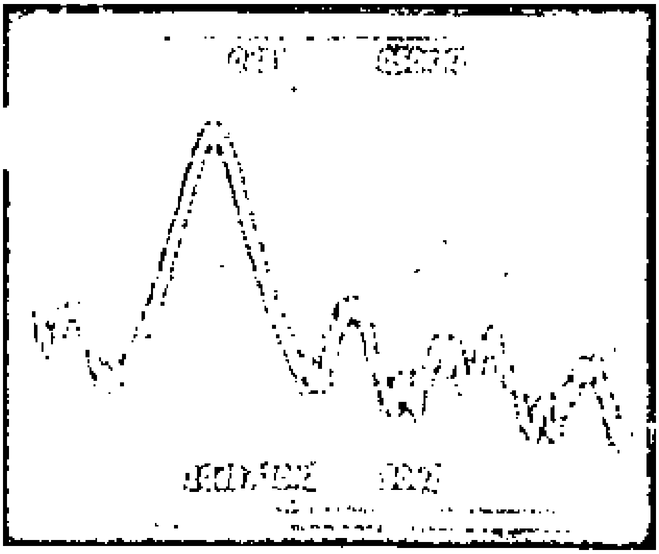
The resources of the 2920 are sufficient to provide the equivalent of up to 40 poles of filtering, or 20 complex-conjugate pole pairs. That amount is enough to put many complex analog systems, including a dual-tone multifrequency (DTMF), or Touch-tone, receiver on just a single chip.

The power of the 2920 is dramatized by an audio spectrum analyzer, which can display the frequency response of a circuit under test on an oscilloscope. One such analyzer is diagrammed in Fig. 5. It generates vertical and horizontal outputs for direct connection to the oscilloscope and uses heterodyning to mix the input signal with the signal from a swept oscillator.

Sequence of events

The input signal to be analyzed is first filtered with a simple external network to remove high-frequency components that could cause aliasing, or the generation of spurious signals. Once in the 2920, the signal passes through the equivalent of a two-pole low-pass filter for further band-limiting. Next it is modulated, prior to mixing, by an automatic gain control—simply a division algorithm whose divisor is derived by passing the absolute magnitude of the signal through a lowpass filter (to get the weighted average).

A second portion of the program simulates a pair of



6. Two poles. Spectral analysis of a simple second-order filter section centered at 400 Hz displays sharp bandpass characteristics. The two-pole filter uses 10 instructions of the 2920's E-PROM program memory, leaving 182 instructions for other filter functions.

oscillators, the first of which determines the spectrum scan rate—hence, the horizontal sweep frequency of the oscilloscope—and the second of which simulates a voltage-controlled oscillator (VCO) driven by the first. It is the second oscillator, swept through the range of interest, that beats against the input signal in the mixer.

Dealing with aliasing

'oth oscillators produce linear sawtooth waveforms; . because the program computes sampled waveforms for these simulated oscillators, aliasing distortion could be a problem if harmonics in the sawtooth were to interact with the sampled frequency and produce spurious frequency components.

For that reason, the output of the second oscillator undergoes a nonlinear transformation in order to approximate a sinusoidal waveform. The 2920 performs the nonlinear transformation with a piecewise-linear approximation that combines absolute-magnitude functions and the effects of overflow saturation in the ALU. The first oscillator, it may be noted, needs no such transformation because it operates at such a low frequency that any harmonics high enough to react with the sample rate are insignificant.

Good mixer

The mixer uses a multiplication routine to combine the output of the second oscillator with the filtered, amplitude-controlled signal under analysis. The output of the multiplier contains sum and difference frequencies. Only the sum frequency is of interest for the analysis, and a narrow-band filter extracts it from the mixer's composite signal. The amplitude of the sum frequency corresponds exactly to the input signal level at a frequency determined by the difference between the filter frequency of the narrow-band filter and the frequency of the VCO.

The 2920's absolute-magnitude algorithm performs in effect a full-wave rectification of the narrow-band filter's output. Finally, the signal is lowpass-filtered to drive the vertical display. Since both horizontal and vertical display outputs of the 2920 are delivered as sampled-and-held signals, some simple external filters may be used to smooth the display.

The filters used in the spectrum analyzer are single-pole or complex-conjugate-pole recursive sections. Multiple pole filters are simply cascades of basic sections. The 2920 can also simulate finite impulse-response filters, and can implement zeros either independently or between pole sections.

The analyzer has a frequency range of 300 Hz to 3 kilohertz, swept 10 times per second, with a resolution of about 100 Hz. The narrowband filter is centered at 4.5 kHz. The sampling rate is about 13 kHz.

Room to spare

The table shows the amount of program and scratch-pad memory allocated to each block for the given parameters. (Different parameters for the functional blocks in the analyzer might change the amount of 2920 memory used in each, however.)

[illegible]

Module	Read only memory words	Random access memory words
Input filter	20	5
Automatic gain control	18	1
Sweep oscillator	5	1
Voltage controlled oscillator	7	1
Waveform modifier	10	1
Mixer	12	0
Narrow band filter	30	6
Rectifier	1	0
Lowpass filter	10	2
Total	113	17

Since the program does not occupy all the RAM and E-PROM available, more functions could be added or, alternatively, the sampling rate could be raised to as high as 20 kHz. As another option, multiple copies of input sampling and filtering algorithms could be inserted to increase the effective sampling rate, thereby allowing use of a simpler anti-aliasing filter external to the 2920 signal processor's front end.

The 2920 is scheduled for production in the latter part of this year. Since the chip is a microprocessor and then some, it must be supported at least as adequately as current digital microprocessors. Support plans call for an assembler and simulator that will be resident on the Intellec microcomputer development system.

Because the arithmetic associated with the design and optimization of digital filters is extremely complex, a design-aid software package capable of interactive filter design and automatic compilation into 2920 assembly language is also planned. Those packages will equip analog system designers with all the conveniences users of conventional microprocessors are by now accustomed to enjoy. □

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MIRCOPROCESADORES: TEORIA Y APLICACIONES

Projecting VLSI's Impact on Microprocessors

Julio, 1981

Eltavez
ene 80

Projecting VLSI's impact on microprocessors

Single-chip microcomputers with 1 million bits of information, English-language programming, and canned software are seen ahead

One million devices on a single silicon chip by 1985? RAMS of 1024 kbits with prices as low as 0.002 cent per bit? Both developments will be possible in the next decade with a new generation of very-large-scale integrated circuits. Microprocessors and related components will be built on these ICs with new circuit-design techniques and advanced fabrication processes. Among the likely products are single-chip microcomputers with as much as 256 kbits of electrically erasable memory.

Contrast this progress with present silicon chip densities of less than 100 000 devices and prices at least an order of magnitude higher than 0.002 cent per ~~chip~~. bit.

Future ICs of high complexity will be intimately linked to the microprocessor. This is because this device, with its flexibility, opens up large-volume applications (and thus low costs per function). Until the microprocessor's advent, IC manufacturers were in a dilemma: they needed to increase IC densities to lower function costs, but in doing so, they started creating complex ICs that were limited in applications (and thus restricted in volume and costly).

Radical changes are ahead

Straight-line projections of IC complexities have been accurately forecast (see Fig. 1, based largely on a similar figure presented by H. W. Spence and R. W. Rozeboom at the 1977 International Conference on Microlithography). Recent trends in IC design and processing (Figs. 2, 3) suggest that these increases in complexity will continue. This gives rise to interesting speculations about the shape of future ICs.

For example, the present IC power dissipation of 1 to 2 W, imposed by conventional packaging, is bound to more than double to handle the higher densities. Thus radically new packaging ideas will evolve, which in turn will dictate new design standards.

One possibility is the gradual abandonment of the 5-V standard for TTL ICs. Since the demand for lower power dissipation and closer on-chip line spacing is tied to lower operating voltages, future high-density ICs are likely to operate at new standards of 3 or 1.5 V. Such voltages are compatible with those of batteries, which are likely to be used increasingly in future IC systems.

Microwatt power-dissipation levels will be necessary for each gate to achieve the high densities. In addition circuit designers will demand internal gate speeds below 1 ns.

But achieving million-component densities will require more than just lower operating voltages. With advances in fabrication and equipment, it is certain that the present barrier of 1-μm IC line widths and spacings will be overcome in production. Two-layer metal interconnections on the chip will become the rule rather than the exception.

In IC processing, plasmas are now replacing wet chemicals for deposition, oxidation, and other steps. This is making possible finer on-chip line geometries and lower processing costs. This trend will intensify.

NMOS technology indicated

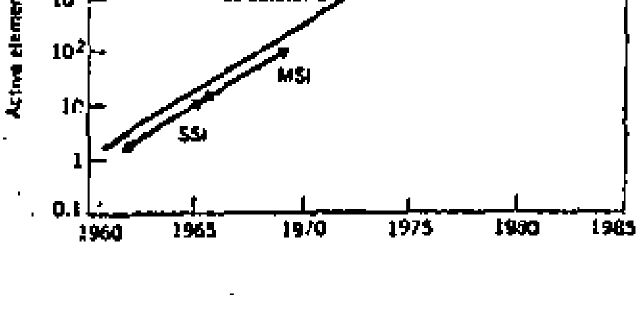
NMOS technology can be the vehicle for future ICs. And CMOS is a likely supplement, more so than now. CMOS's major drawbacks of slower operating speeds and larger cell sizes, compared with NMOS, are being resolved.

Speeds from 20 to 40 MHz (depending on power-supply voltage) are possible if silicon-gate techniques, instead of metal-gate, are used. Up to 100-MHz operation has been achieved in the laboratory with isolated-substrate or dielectric-isolation designs. ICs with such speeds should become commercially available by the early 1980s. And CMOS cell sizes can be reduced by such design innovations as the use of a single MOS element in the output stage, instead of the classical complementary transistor pair.

The future architecture

Present IC complexities of 50 000 to 80 000 devices per chip can support, besides memory, a full range of very powerful, 16-bit peripheral chips. The peripheral chips of the 1980s could contain components for direct memory access, the bus arbitrator, memory segmentation, and the peripheral processor. The peripheral-processor components are likely to be programmed by the user for application in floppy-disk or CRT controllers, dedicated processors, memory sharing, and special communication protocols.

[1] By 1985, ICs with complexities of one million devices per chip will be possible. Microcomputers handling 32-bit operations are one manifestation.



When we consider that the most powerful central processing unit available uses the equivalent of six million devices, the possibility of building single-chip CPUs of one million bits can be fully appreciated. More likely, future microcomputers will be apportioned as follows:

- 100 000 devices for the CPU.
- 400 000 devices for I/O adapters.
- 512 000 devices for an electrically erasable ROM (no RAM/ROM splitting).

Resident in memory will be powerful *firmware* in the form of a high-level-language compiler/interpreter and, probably, the most universal technical language: English. The CPU could have an 800-to-2000-word vocabulary, making it possible to instruct it directly in English. Recent speech-synthesizer ICs point the way.

The CPU will probably be 16 bits wide and feature a 20-ns cycle time. The I/O area will provide many direct inputs and outputs for interfacing to the outside world. With the type of organization projected, such a microcomputer system could have artificial intelligence.

How to reduce software costs

Producing more powerful computing machines does not, in itself, create a market for them, even if the cost does not increase in proportion with the increased capability. This is clearly indicated by sales of 16-bit microcomputers, which are only starting to be used by select customers for sophisticated applications. Less-powerful 8-bit microcomputer applications, on the other hand, have been growing significantly, and 4-bit microcomputer sales are literally exploding.

A major reason for such hesitant expansion of 16-bit microcomputers is the cost of software. As a rule of thumb, the generation of an average line of software costs approximately $10. One way to reduce software costs is to simplify programming.

From a hardware standpoint, the cost of computing power is dropping so drastically that by 1984 the capability of an IBM 1800 mainframe computer will be available for about $100 (Fig. 4). This creates an irresistible incentive for semiconductor manufacturers to plunge ahead with ever more sophisticated designs, so that new markets for microcomputers can open up. This is not an attempt to replace larger computers with 16-, 32-, and 64-bit microcomputers; rather it is a well-researched plan for taking advantage of the rapidly decreasing price/performance ratio of ICs to anticipate and stimulate the microcomputer market. It calls for using the increased device complexities to simplify the operation of powerful CPUs.

A few years ago the trend was to build more powerful CPUs every year, often in big mainframes. Then suddenly the microcomputer appeared with a downgraded CPU but many I/O and memory capabilities on the chip. Semiconductor manufacturers turned to this market, leaving 16-bit CPUs to a comparatively small market. The 4-bit and 8-bit microcomputer market began to take off because of easier use and lower costs.

The third-generation microcomputer

The next generation of microcomputers—the third generation—will be 8-, 16-, and 32-bit systems with architectures structured for programming simplicity instead of program length. Programmed with high-level languages, they will feature minicomputer-like architectures and powerful instruction sets. (Present single-chip microcomputers are second-generation systems.)

Two harbingers of third-generation microcomputers are the Motorola M6809, now available, and M68 000, which will be available shortly. The M6809 is an 8-bit microcomputer with 16-bit internal organization, and the M68 000 is a 16-bit microcomputer with 32-bit internal organization. The major features of both are architectures that support position-independent code and re-entrant programming. Such features will have a far-reaching impact on the way microcomputers are used.

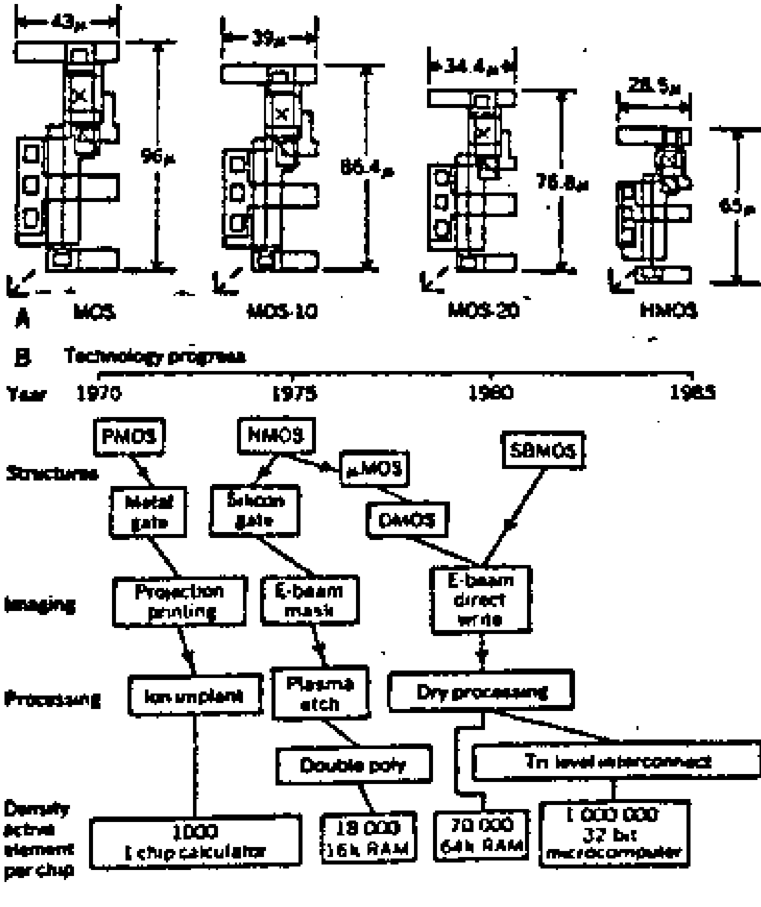
The position-independent code allows written software to be executed in any location of the microcomputer's address map. The re-entrant feature permits interruption of a given program location in the address map, and the return to the proper address from a subroutine. Together these two design features lend themselves to modular programming, thus making possible "canned software."

With canned software, a large range of subprograms, 4 to 8 kbytes long, will become available from semiconductor manufacturers and other sources at very low prices. An applications program can then be developed by selection of the desired software. Program continuity can be provided with short, simple "patch" programs of high-level instructions. This technique can cut the cost of programming by an order of magnitude.

The availability of standard firmware in the form of fully debugged ROM-resident programs will enable users to build inexpensive software libraries and to split programming tasks into small, manageable modules. This, in turn, will reduce programming time and the knowledge needed for microcomputer applications. The microcomputer will then reach millions more of potential users.

To increase flexibility and expand throughput, third-

[2] Design innovations resulting in small IC cell sizes (A) and processing innovations for finer line-width and space control (B) are the tools of future high-density ICs.



generation microcomputers will be supported by language-compatible software that permits movement from 8-bit operation to 16 bits and back again. Since the microcomputer will handle high-level languages efficiently, the user's software library is not compromised by a move from an 8-bit to a 16-bit microcomputer, or even to a 32-bit system.

Will this compatibility apply across the board from 4 to 32 bits? Probably not. Low-end, 4-bit microcomputers, with mass use, will tend to be more hardware-sensitive. Since they will be used in applications where a single program will serve millions of parts, a small savings on hardware can quickly offset program development costs. At this point such microcomputers will sell below $1 in large quantities.

Improvements in processing needed

In view of the far-reaching nature of the microcomputer revolution, progress must be made in processing technology to reach the expected goals. Table I compares the progress in processing in the past few years with what will be needed to put a million devices on a chip by 1985. Similarly Table II shows IC electrical specifications and the improvements expected in emerging processing technologies.

The market inroads resulting from exploding microcomputer applications are listed in Table III. By 1983 the microcomputer market will be worth well over $2 billion in the United States and will represent nearly one-third of the domestic IC market. Microcomputers will account for over 20 percent of the world semiconductor market, compared with only 9 percent in 1978. A valid question is how will the semiconductor industry respond to this explosive demand? By spectacular increases in productivities. Let us illustrate:

In 1976 the number of good dies per 3-inch wafer (*GD/WF*) was around 200 for an average chip surface of 160 × 160 mils. Three basic considerations limit the possible good dies from what is obtained at the end of the production line:

1. Wafer yield (*Wy*).
2. Assembly yield (*Ay*).
3. Final test yield (*FTy*).

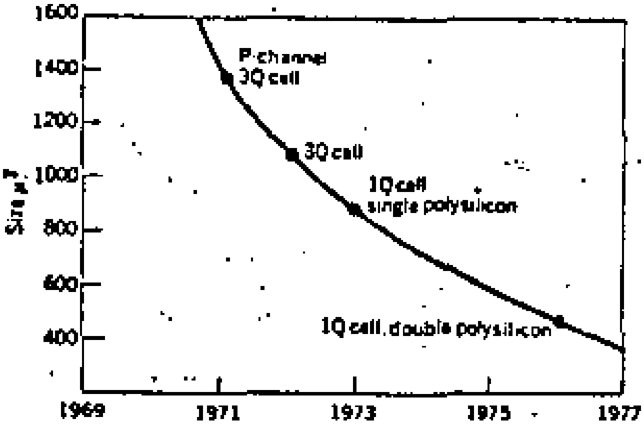
In 1976 the average values of microprocessor yields were: *Wy*, 40 percent; *Ay*, 80 percent; and *FTy*, 85 percent. By the formula

$$\text{Good finished products } (GFP) = (GF/WF)\,(Wy)\,(Ay)\,(FTy)$$

it is possible to calculate the approximate number of good finished products per wafer. Thus the 1976 overall yield per wafer was

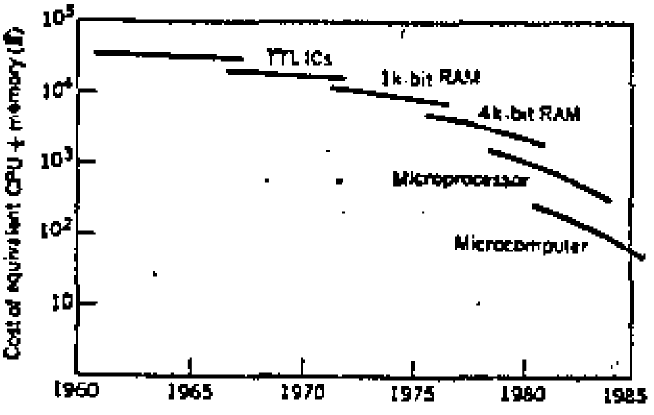
$$(200)\ (0.4)\ (0.8)\ (0.85) = 54 \text{ units}$$

This means that with 10 000 wafer starts, it was possible to manufacture 540 000 microprocessors.



[3] The evolution of RAM cell sizes from 1400 μm² to 400 μm² in the last eight years is an example of how IC design innovations are bound to increase future chip densities.

[4] The cost of computing power (hardware) is dropping so drastically that by 1984 the capabilities of a much larger IBM 1800 computer will be available in a $100 IC chip.



I. Comparative IC processing data

Parameter	1976	1979	1985
Die size (mil²) (production level, mm²)	160 x 160 4.0 x 4.0	250 x 250 6.35 x 6.35	450 x 450 11.4 x 11.4
Resolution (μm) (production level)	6	3	1
Device per die (nonmemory)	10 000	60 000	>1 000 000
Wafer size (inches)	3	4	5 (ribbon technology)

II. Comparative IC performance data

Parameter	1976 (NMOS process)	1980 (HMOS II process)	1985 (SBMOS/DIMOS processes)*
Speed, ns (internal gate)	6	2	1
Power dissipation, mW (per gate)	3	0.5	0.25
Speed-power product, pJ	18	1.0	0.25
Power dissipation per package, watts	1	1.5	5

* Schottky barrier MOS/dielectric-isolation MOS

III. The world market value for microcomputers

	1978	1979	1980	1981	1982	1983
Percentage of total semiconductor market	8.8	12.8	14.4	15.5	17	20.2
Percentage of IC market	14.9	20.6	22.5	23.1	24.4	28.5

Let's see what the situation will be in 1979 for the same type of product (160 × 160 mils). Manufacturing by then will be done on 4-inch wafers. Since the ratio of 4-inch to 3-inch wafers is approximately 1.8, the 200-chip maximum per wafer will increase to 360. At the same time we can expect the following yield values:

$$Wy = 50 \text{ percent}$$
$$Ay = 90 \text{ percent}$$
$$FTy = 90 \text{ percent}$$

This results in (360) (0.5) (0.9) (0.90) = 145 good finished products per wafer.

With the same wafer-start capability, it is now possible to manufacture 1 450 000 microprocessors, or nearly three times more than in 1976, without extra production capacity.

If we apply the above calculation to the 250 × 250-mil chip sizes expected for 1979 products, we find:

$$\text{Die surface ratio} = \frac{250 \times 250}{160 \times 160} = 2.44$$

$$\text{Wafer surface ratio} = \frac{4^2}{3^2} = 1.77 \text{ (from 3 to 4 inch)}$$

$$\text{Possible good dies per wafer} = \frac{200 \times 1.77}{2.44} = 145$$

Using 145 as the value of yield, we have:

$$GFP = (145)\ (0.5)\ (0.9)\ (0.90) = 58$$

Therefore even if we mulitply the circuit complexity by 8 (an increase from 10 000 to 80 000 devices per chip), the same 10 000 wafer starts will yield 580 000 pieces—more than the 540 000 pieces available from today's less-complex 160 × 160 mil circuits.

This same progression should apply for the years beyond 1979, so that by 1985 it is reasonable to expect square 5-inch wafers with a new method of generating monocrystal silicon—ribbon technology. Here, once a laser remelts polycrystalline silicon after the silicon has been grown, the silicon is drawn out in ribbon form. At this point, with a small increase in yield, it is still possible to expect more than 100 good finished dies per wafer for systems of million devices-per-chip complexity, or 350 good finished dies for microprocessors of 1979 complexity.

Therefore productivity increases, coupled with straight-line production expansion, should provide the quantity of microcomputers needed by the market.

IV. Failure rates and mean time between failure (MTBF) for Motorola MC6800 microprocessor (ceramic package, 70°C ambient temperature)

Year	Failure rate (percent/1000 hours)	MTBF, hours (years)
1974	1.27	78 000 (9)
1975	0.5	200 000 (23)
1976	0.12	833 000 (95)
1977	0.06	1 700 000 (194)
1978	0.013	7 700 000 (877)
1979	0.006	16 666 666 (1901)

Reliability on the upswing

One final facet important to the expanding world of IC chip complexity is reliability. Just as power dissipation must be greatly reduced to accommodate one million devices on a single chip, so must reliability be improved by several orders of magnitude.

If one were to build a microprocessor of modern complexity using simple, plastic-packaged TTL gates, it would have a mean time between failure of only a few months. By contrast, the mean time between failure of a commercially available Motorola MC6800 CPU, a device with 3000-gate complexity and packaged in ceramic, was 877 years last year and is projected to reach 1900 this year. This clearly shows the reliability advantage of LSI.

As shown in Table IV, reliability is evolving asymptotically, with a failure rate of 10^{-8} as a trend. But even if we cannot realistically expect an improvement in this value for all new ICs, we can at least expect to see this value reached by new microprocessors of increased complexity. With a million devices per chip (300 000 equivalent gates, and a failure rate of 10^{-8}, the corresponding fa[illegible] is 3×10^{-14} per gate.

Self-diagnostics for greater reliability

The impressiveness of these reliability figures is further magnified by the fact that the price will be low enough to allow the use of redundant systems equipped with self-diagnostics. Should a failure occur, it will be invisible.

Self-diagnostic features will become necessary for two major reasons: manufacturing costs and end-system reliability.

The increasing complexity of IC products makes their final testing very difficult. The necessarily large number of package leads will not allow convenient internal accessing, and this makes testing costly. One solution is to have internal circuitry, conveniently located on the chip, devoted solely to functional testing of all or part of the circuit. The testing of the product then becomes faster and more reliable.

The next logical step is to use this test function on the chip more than once in final testing and to make it available to the end user. At a given time interval, a test sequence of all microprocessors used in end equipment can be initiated and a go/no-go message transmitted to the user, who can immediately know if the equipment is fully operational. Another alternative is to design the system with redundancy of functions (a definite possibility in view of the evolution of the price/performance ratio of microprocessors). As soon as a defect is detected, another pathway will be used. ✦

Daniel Queyssac is world market development manager for microsystems and microcomponents at Motorola Semiconductor Products, Phoenix, Ariz. He joined Motorola in 1965, becoming marketing manager for France in 1972, then European microprocessor marketing manager in 1975 in Geneva, Switzerland. Before going to work for Motorola, he was employed by a French company that designed digital equipment for the French Atomic Commission. Mr. Queyssac received an electronics degree from Collège D'enseignment Technique in Paris in 1959.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

MICROPROCESADORES: TEORIA Y APLICACIONES

PRACTICA

SABADO 18, 1981

PROGRAMA QUE DEMUESTRA EL USO DE UN PUERTO EN PARALELO PROGRAMABLE -P10- Y LA GENERACION DE INTERRUPCIONES POR EL PUERTO, USANDO EL MODO 2 DE INTERRUPCION.

PROBLEMA:

Consiste en leer por el puerto A del P10 4 switches que dan un valor binario y compactado a un dígito BCD, este valor BCD se - procesa a través de una tabla para sacar el código de 7 segmentos que deberá prender el display por medio del puerto B. Todo esto solo sucede si hay una petición de interrupción en el puerto A del P10.

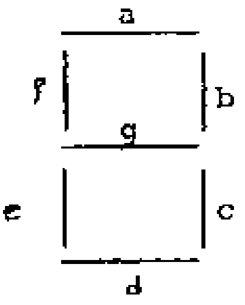
```
                                    ; * Programa para convertir un
                                    ; * dígito "BCD" en 7 segmentos.


                   PIOADR   EQU  80H        ; Registro de datos -PA-
                   PIOACR   EQU  82H        ; Registro de control -PA-
                   PIOBDR   EQU  81H        ; Registro de datos -PB-
                   PIOBCR   EQU  83H        ; Registro de control -PB-
                   PIOAIV   EQU  00H        ; Vector de interrupción -PA-
                   TABLA    EQU  2240H      ; Tabla de BCD A 7 Seg.
                   TEMP     EQU  2250H      ; Apunta (IX)Temporal
                   SUBINT   EQU  2200H      ; Inicio subrutina Interrup.
                   TABINT   EQU  2100       ; Tabla vector de Interrup.
                            ORG  2000H      ; Origen del programa
20 00    3E 21              LD A, 21H
20 02    ED 47              LD I, A         ; Inicializa Vec.Int.
20 04    3E 00              LD A, 00H
20 06    D3 82              OUT (PIOACR),A; Vector Int.(PIOA)
20 08    3E 4F              LD A, 4FH
20 0A    D3 82              OUT (PIOACR),A; PIOA Entrada
20 0C    3E 87              LD A, 87H
20 0E    D3 82              OUT (PIOACR),A; PIOA Habilita Int.
20 10    3E OF              LD A, OFH
20 12    D3 83              OUT (PIOBCR),A; PIOB Salida
```

```
20 14   21 00 22            LD HL, SUBINT ; Direc.subrutina de servi
                                            cio.
20 17   22 00 21            LD (TABINT),HL; Almacena la Direcc. en la
                                            tabla.
20 1A   ED 5E               IM 2          ; Modo 2 de Interr.
20 1C   FB                  EI            ; Habilita Interr.
20 1D   3E 7F               LD A, 7FH
20 1F   D3 81               OUT (PIOBDR),A; Prende solo el LED
20 21   00               L1 NOP
        C3 21 20            JP LI         ; Programa principal


                              * ; Subrutina de servicio de inte-
                                  rrupción para el P1O A

                            ORG 2200H

22 00   21 50 22     SUBINT LD HL, TEMP
22 03   DD21 40 22          LD IX, TABLA  ; Regi de indice con Inic.
                                            tabla
22 07   DD22 50 22          LD (TEMP),IX  ; Lo guarda temporalmente
22 OB   DB 80               IN A,(P1OADR) ; Lee el "BCD"
22 0D   E6 OF               AND OFH       ; Lo enmascara
22 OF   86                  ADD A,(HL)    ; Se lo suma al Reg. Temp.
22 10   77                  LD (HL),A     ; Carga esta suma
22 11   DD2A 5022           LD IX, (TEMP) ; El IX toma el valor de
                                            Temp.
22 15   DD 7E 00            LD A, (IX + 0); Se carga el valor en for
                                            ma indezada
22 18   D3 81               OUT (PIOBDR),A; Se manda el dato al dis-
                                            play.
22 1A   FB                  EI            ; Habilita Interr.
22 1B   ED 4D               RTI           ; Regreso de interrupción
```

....

```
                                    * ; Tabla de conversión BCD a
                                        7 segmentos

                              ORG 2240 H

2240     81                   81H     ; 0
  41     CF                   CFH     ; 1
  42     92                   92H     ; 2
  43     86                   86H     ; 3
  44     CC                   CCH     ; 4
  45     A4                   A4H     ; 5
  46     AO                   AOH     ; 6
  47     8F                   8FH     ; 7
  48     80                   80H     ; 8
  49     84                   84H     ; 9
  4A     B0                   BOH     ; Entradas no válidas
  4B     BO                   BOH
  4C     BO                   BOH
  4D     BO                   BOH
  4E     BO                   BOH
  4$     80                   BOH


                           ; Tabla de servicio de interrupción

21 00    00 22
```

Despliegue de 7 segmentos:



O   LED

P1O BDR

LED

MSB	a	b	c	d	e	f	g
7	6	5	4	3	2	1	0

0 = Prende segmento
1 = Apaga segmento

	1 000,0001	=	81	=	0
	1100,1111	=	CF	=	1
	1001,0010	=	92	=	2
	1000,0110	=	86	=	3
	1100,1100	=	CC	=	4
	1010,0100	=	A4	=	5
	1010,0000	=	AO	=	6
	1000,1111	=	8F	=	7
	1000,0000	=	80	=	8
	1000,0100	=	84	=	9