INTRODUCCION A LOS MICROPROCESADORES (Z-80)

DIRECTORIO DE PROFESORES

ING. LUIS CORDERO BORBOA
JEFE DEL DEPARTAMENTO DE COMPUTACION
DIVISIONDE INGENIERIA MECANICA Y ELECTRICA
FACULTAD DE INGENIERIA, UNAM
CIUDAD UNIVERSITARIA
MEXICO 20, D.F.
TEL: 550.52.15 ext. 3750

M. EN C. MANUEL ESTEVEZ KUBLI
SECRETARIO ACADEMICO
CENTRO DE INSTRUMENTOS, UNAM
CIRCUITO EXTERIOR
MEXICO 20, D.F.
TEL: 550.04.16 y 550.52.15 ext. 5013

DR. DANIEL PAUL PETERSEN
PROFESOR VISITANTE
DEPARTAMENTO DE COMPUTACION
DIVISION DE INGENIERIA MECANICA Y ELECTRICA
FACULTAD DE INGENIERIA, UNAM
CIUDAD UNIVERSITARIA
MEXICO 20, D.F.
TEL: 550.52.15 ext. 3750

M. EN C. MARCIAL PORTILLA ROBERTSON
INVESTIGADOR
CENTRO NACIONAL DE CONTROL DE ENERGIA
CALLE DON MANUELITO S/N ESQ. AV. TOLUCA piso 1
MEXICO 20, D.F.
TEL: 595. 55. 44 y 595.55.33 ext. 240

M. EN C. ARMANDO TORRES FENTANES
INVESTIGADOR
CENTRO NACIONAL DE CONTROL DE ENERGIA
CALLE DON MANUELITO S/N ESQ. AV. TOLUCA piso 1
MEXICO 20, D.F.
TEL: 595.55.44 y 595.55.33.ext. 240

'pmc.

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

(3,4 y 5 de marzo de 1981)

F E C H A	HORARIO	T E M A S	P R O F E S O R E S
3 de marzo	9 a 13	CONCEPTOS BASICOS Aritmética Compuertas Memorias	M. EN C. ARMANDO TORRES FENTANI
	13 a 15	C O M I D A	
	15 a 19	ESTRUCTURAS FUNDAMENTALES DEL Z-80 C.P.U. y registros Memoria Canales (buses) Puertos e interfasas Registros especiales D M A	M. EN C. MARCIAL PORTILLA / ROBERTSON
4 de marzo	9 a 13	MODOS DE DIRECCIONAMIENTO Inmediato Directo Indirecto CONJUNTO BASICO DE INSTRUCCIONES Aritméticas Lógicas Desplazamientos y rotaciones	ING. LUIS CORDERO BORBOA
	13 a 15	C O M I D A	
	15 a 19	SEÑALES DE CONTROL Señales de dirección Señales de memoria Ciclos de E/L Solicitud de buses Interrupciones	M. EN C. MANUEL ESTEVEZ KUBLI

... 2

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

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

CONCEPTOS BASICOS

M en C Armando Torres Fentanes

Marzo, 1981

CURSO MICROS

- CONCEPTOS BASICOS DE ELECTRONICA
- CONCEPTOS BASICOS DE ELECTRONICA DIGITAL
- CONCEPTOS DE PROGRAMACION
- LENGUAJE DE PROGRAMACION
- µPROCESADOR M6800
- PERIFERICOS
- APLICACIONES
- LABORATORIO

CONCEPTOS BÁSICOS

VOLTAJE – POTENCIAL QUE GENERA LA CORRIENTE
UNIDAD: [Volts] [V]
Ej.: BATERIAS, ACUMULADORES

BATERIAS (CARACT.)
- POTENCIAL
- BORNES + y –
- ENERGIA QUE ENTREGA
- SIMBOLO $\frac{1+}{T-}$
- PUEDEN CONECTARSE EN SERIE

RESISTENCIA (R): OPOSICION AL FLUJO DE ELECTRONES
UNIDAD: [OHMS] [Ω]

$1000\,\Omega = 1\,K\Omega$
$1000\,K\Omega = 1\,M\Omega$

SIMBOLO GRAFICO

RESIST. CARBON

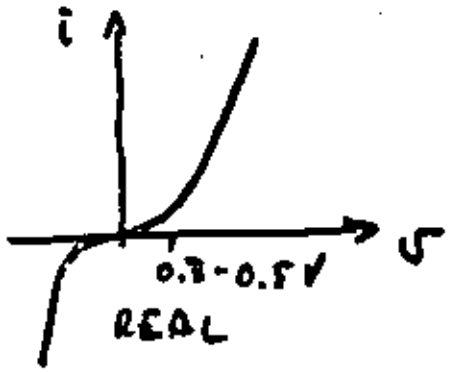
A: 1er DIGITO
B: 2° DIGITO
C: POTENCIA DE DIEZ X
D: TOLERANCIA

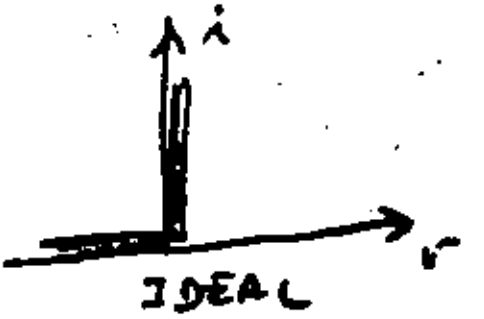
COLORES:

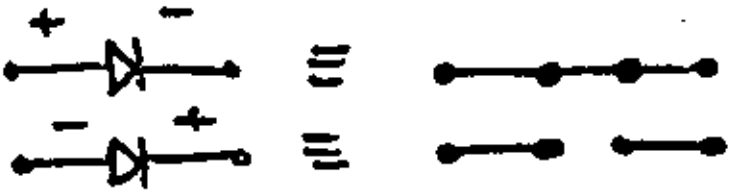
NEGRO	0	ORO 5%
CAFE	1	PLATA 10%
ROJO	2	
NARANJA	3	
AMAR.	4	
VERDE	5	
AZUL	6	
VIOLETA	7	
GRIS	8	

DIODOS: PERMITEN EL FLUJO DE CORRIENTE EN UN SOLO SENTIDO (SWITCH)

SIMBOLO GRAFICO





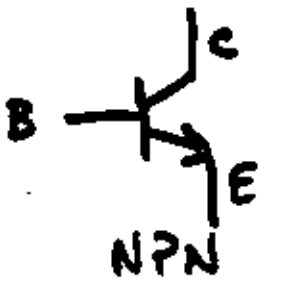


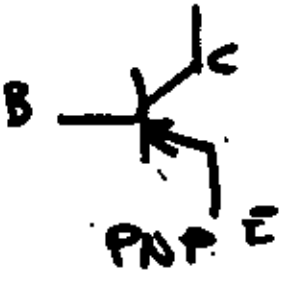
TRANSISTOR: DISPOSITIVO CREADO CON 3 CAPAS DE SEMICONDUCTORES

USOS: - FTE DE CORRIENTE
- AMPLIF. DE CORR.
- INVERSOR
- OTROS

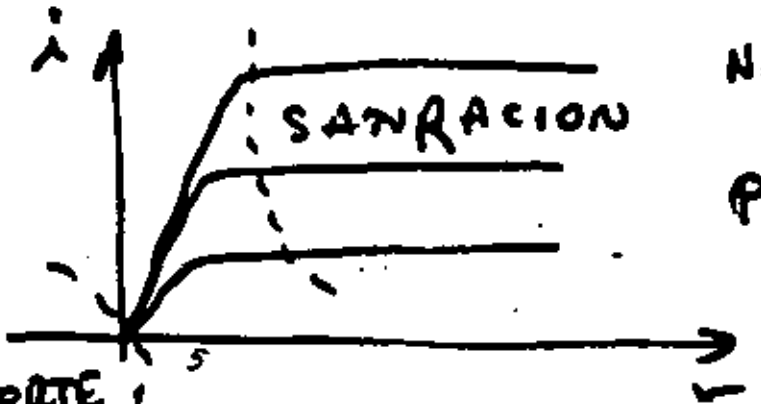
APLIC. SIST. DIGITALES: SWITCH

SIMBOLO GRAFICO



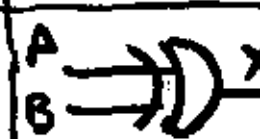


$i_C = \beta i_B$

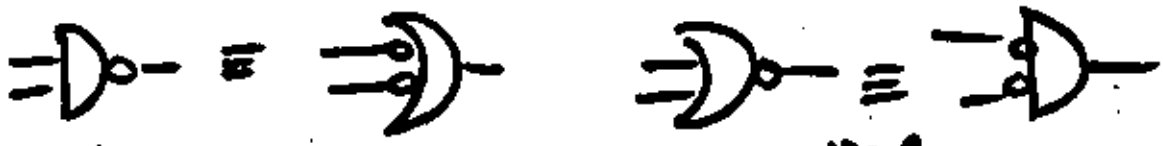


NPN - POLARIZ. POSIT.

PNP - POLARIZ NEGAT

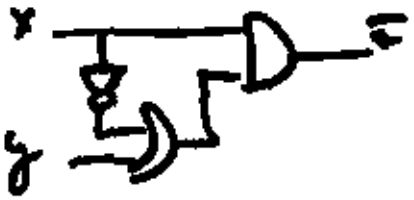
NOMBRE	SIMBOLO	F. ALGEBRAICA	T. DE VERDAD
AND		$X = AB$	A B | X 0 0 | 0 0 1 | 0 1 1 | 1
OR		$X = A + B$	A B | X 0 0 | 0 0 1 | 1 1 0 | 1
INVERSOR (COMPLEM.)		$X = A' = \bar{A}$	A | X 0 | 1 1 | 0
BUFFER		$X = A$, amplifica corriente	A | X 0 | 0 1 | 1
NAND		$X = (AB)'$	A B | X 0 0 | 1 0 1 | 1 1 1 | 0
NOR		$X = (A + B)'$	A B | X 0 0 | 1 0 1 | 0 1 1 | 0
XOR f. non		$X = A \oplus B$ $= A'B + AB'$	A B | X 0 0 | 0 0 1 | 1 1 1 | 0
XNOR f. par		$X = A \odot B$ $= A'B' + AB$	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

DE LAS LEYES DE MORGAN



NAND NOR

EJEMPLO:

1. $f = [(A' + B) \cdot B']'$
2. 

TTL

- $f_{AN\ OUT} \sim 10$
- NOISE MARGIN $\sim 0.4V$
- ENTRADA ABIERTA = ALTO

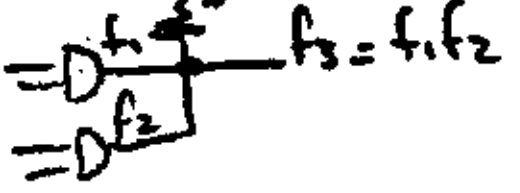
VERSIONES

		ns	mW
ESTANDAR TTL		10	10
LOW POWER	LTTL	33	1
HIGH SPEED	HTTL	6	22
SHOTKY	STTL	3	19
LOW POWER SHOTKY	LSTTL	9.5	2

CONFIGURACIONES A LA SALIDA PARA C/VERSION

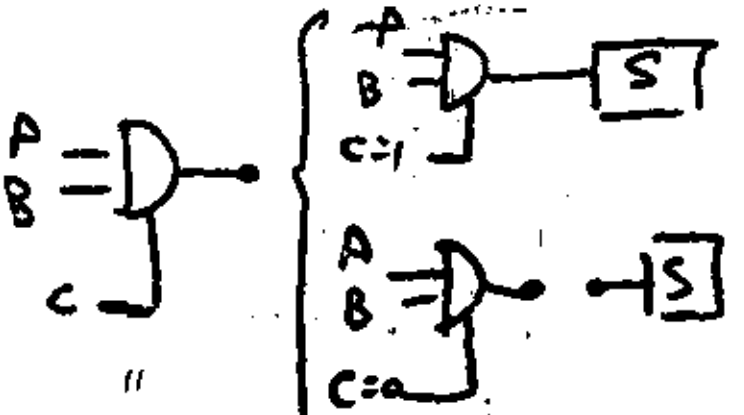
OPEN COLLECTOR: REQUIERE RESISTOR EXTERNO A LA SALIDA CONECTADA A $+V_{cc}$

PERMITE WIRE-AND



TOTEM POLE: SALIDA ESTANDAR POCO RETRASO, MAY FAN OUT

TRISTATE: 3 ESTADOS DE SALIDA
- * 1
- * 0
- * ALTA IMPEDANCIA



MINIMIZACION

FORMAS CANONICAS

- Σ de Π : $f(x_1,\dots,x_n) = \underbrace{(x_1\bar{x}_2\dots x_n)}_{\text{mini:términos}} + (x_1\dots x_n) + \dots$
- Π de Σ : $f(x_1,\dots,x_n) = \underbrace{(x_1+x_2+\dots+\bar{x}_n)}_{\text{max:términos}} \cdot (\bar{x}_1+\dots+x_n) \cdot$
- $f = \Sigma$ de mini:términos para los cuales $f=1$
- $f = \Pi$ de max:términos para los cuales $f=0$

F	x y z	mini:términos		max:términos	
0	0 0 0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0 0 1				
0	0 1 0				
0	0 1 1				
0	1 0 0				
0	1 0 1				
0	1 1 0				
1	1 1 1	xyz	m_7	$x'+y'+z'$	M_7

$$M_i = \bar{m}_i$$

$$f = \Sigma(1,4,7) = m_1 + m_4 + m_7$$

$$f = \Pi(0,2,3,5,6) = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

METODOS DE REDUCCION

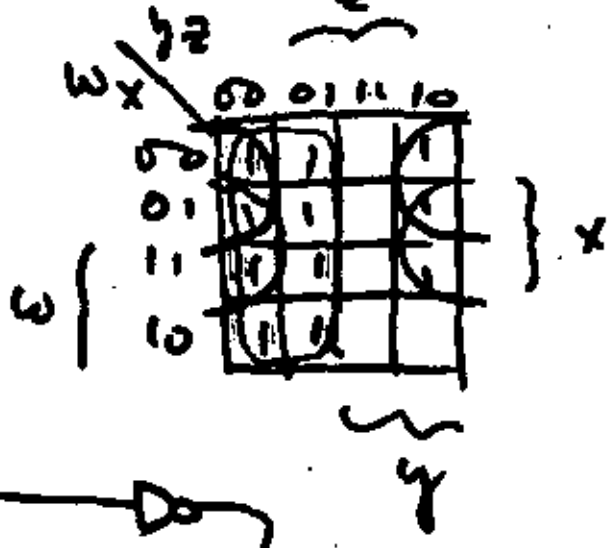
- ALGEBRA BOOLEANA
- MAPAS DE KARNAUGH
- QUINE Mc CLUSKEY.

METODO

- TRAZAR TABLERO CON 2^n CUADROS PARA n VAR. DE ENTRADA
- NUMERAR CUADROS EN FORMA TAL QUE LOS ADYACENTES DIFIERAN EN UN BIT
- LLENAR TABLERO
- AGRUPAR LOS UNOS FORMANDO CONJUNTOS QUE SEAN LO MAS GRANDE POSIBLE. #ELEMENTOS EN EL CONJUNTO = 2^i
- ESCOGER MINIMA CANTIDAD DE CONJUNTOS QUE CUBRA TODOS LOS UNOS (IMPLICANTES PRIMOS)
- ENCONTRAR LA REPRESENTACION ALGEBRAICA DE LOS I.P.
- $f = \Sigma$ DE I.P.

EJEMPLO

$f(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$



$f = \bar{y} + x\bar{z} + \bar{w}\bar{z}$

CONVERSION BINARIA, OCTAL, HEXADEC.

BINARIA ↔ OCTAL ↔ HEXADECIMAL

a) BINARIA → OCTAL $2^3 = 8$

- DIVIDIR # BIN. EN PERIODOS DE 3 ← • →
- OBTENER # DEC. EQUIVALENTE EN CADA PERIODO

101 011 101 011.1101 1

b) OCTAL → BINARIA

- REEMPLAZAR CADA DIGITO OCTAL POR SU EQUIVALENTE BINARIO

$5072.41_{[8]}$

c) BINARIA → HEXADECIMAL $2^4 = 16$

- DIVIDIR # BIN. EN PERIODOS DE 4 ← • →
- OBTENER # DEC. EQUIVALENTE EN CADA PERIODO

$1101\,0111\,0101.1101\,1_{[2]}$

d) HEXADEC. ⇒ BINARIO

- REEMPLAZAR CADA DIGITO HEXADECIMAL POR SU EQUIVALENTE BINARIO

ABC.E9

$$A_F \times M$$

LEER EN ESTA DIRECC. ↓

$$B_{-1} \cdot N_{-1} \times M$$

$$B_{-2} \cdot N_{-2} \times M$$

$$B_{-3} \cdot N_{-3}$$

$$B_{F_{[M]}} = \cdot B_{-1} B_{-2} B_{-3} \cdots$$

$$.6875_{[10]} \rightarrow [2], [7], [16]$$

$$- B_{[M]} = B_E \cdot B_F$$

DIVISION

BASE 10

CO: 5

$$\begin{array}{r|l} 68 & \\ \hline 5 & 340 \quad A \\ -30 & \\ \hline 40 & \\ -40 & \\ \hline 0 & \end{array}$$

B

R: 0

A/B

$CO_i = \begin{cases} 0, & A < B \\ \max(n) \rightarrow n \times B < A \end{cases}$

$R_i = A - n \times B$

NUMERO QUE DEBE SER SUMADO A $n \times B$ PARA OBTENER A.

BASE 2

$$\begin{array}{r} 0 \\ -0 \\ \hline 0 \end{array}$$ > NO TIENE SENTIDO

$\frac{0}{1} = 0$ RESTO

$\frac{1}{1} = 1$ RESTO

EJEMPLO

$$\begin{array}{r|l} & 11 \\ \hline 110 & 10010 \\ & -110 \\ \hline & 110 \\ & -110 \\ \hline & 000 \end{array}$$

REPRESENTACION DE #S SIGNADOS EN 2'COMPLEMENTO CON 'n' BITS

- #POSITIVO
  - REPRESENTAR MAGNITUD EN FORMA BINARIA EMPLEANDO n-1 BITS
  - HACER BIT DEL SIGNO (n) = 0
- #NEGATIVO
  - REPRESENTAR MAGNITUD EN FORMA BINARIA EMPLEANDO n-1 BITS
  - HACER BIT DE SIGNO (n) = 0
  - OBTENER 2'COMPLEMENTO DE LOS 'n' BITS

- PARA CONOCER LA MAGNITUD DE UN #NEGATIVO (BIT DE SIGNO = 1), OBTENER EL 2' COMPLEMENTO DE LOS 'n' BITS Y LEER LA MAGNITUD EN LOS N-1 BITS
- EJEMPLO
  EMPLEAR 4 BITS PARA ±5
- RANGO DE VALORES REPRESENTABLES

$$-2^{n-1} \leq x \leq 2^{n-1} - 1$$

CODIGOS

DEFINICION { FORMA DE REPRESENTAR LA INFORMACION CON SIMBOLOS DIFERENTES

REPRESENTACION DE #S DECIMALES

- CODIGO BINARIO (BCD)

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

* BCD ≠ REPRES. BINARIA
* EJEMPLO

985_{10}

1001 1000 0101

$49_{10} \equiv 110001_2 \equiv 01001001_{BCD}$

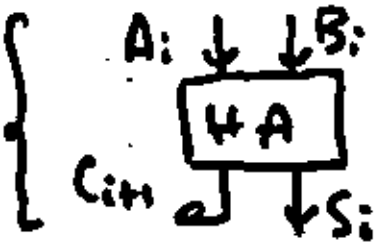
* PARA SUMAR AGREGAR 6_{BCD} A CADA RESULTADO QUE SEA INVALIDO

```
  60   0110 0000
  55   0101 0101
       ---------
       1011 0101
       0110
       ---------
     1 0001 0101   ≡115
```

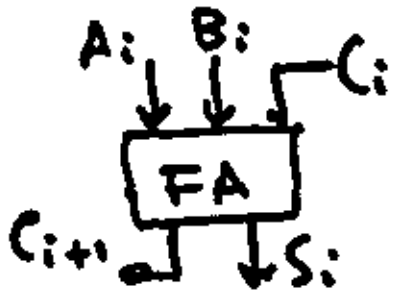
* VENTAJAS: REDUCE CONVERSIONES NECESARIAS E/S
* DESV.: LAS OPERACIONES SON MAS LENTAS

CIRCUITOS DIGITALES BASICOS

MEDIO SUMADOR (HALF ADDER)

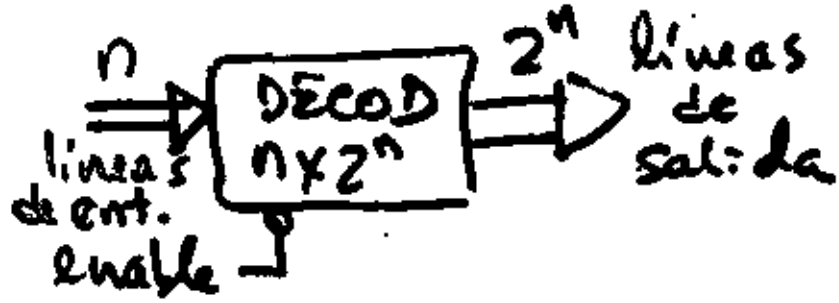


SUMADOR COMPLETO (FULL ADDER)

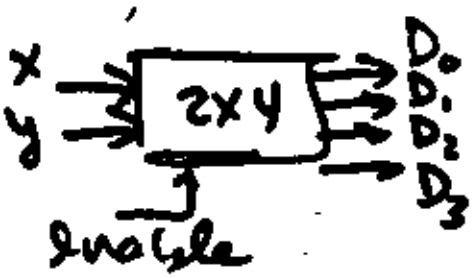


DECODIFICADOR

- CONVIERTE INFORMACION BINARIA DE UN CODIGO A OTRO
- SELECCION DE DISPOSITIVOS

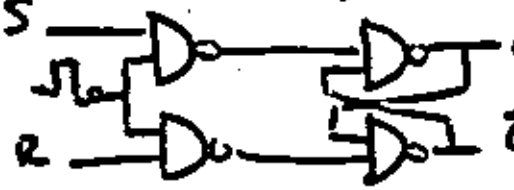


- SOLO UNA LINEA DE SALIDA ADQUIERE VALOR UNITARIO PARA C/COMBINACION DE LAS LINEAS DE ENTRADA
- LOS HAY DE 2x4, 3x8, 4x16, ...
- CON ENABLE SE PUEDEN UNIR DOS PARA GENERAR OTRO DE MAYOR CAPACIDAD



X	Y	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- CIRC. DIGITALES CON MEMORIA
  $Q(t) = f(S_t, R_t, Q_{t-1})$



- UNIDAD BASICA DE MEMORIA (1FF=1BIT)

FLIP-FLOPS

- TIPOS

RS síncrono

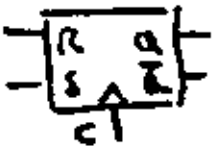


TABLA TRANSICION

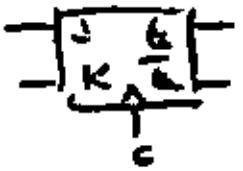
S	R	Q_{t+1}	
0	0	Q_t	
0	1	0	clear
1	0	1	set
1	1	?	

D

D	Q_{t+1}	
0	0	clear
1	1	set

$R = \bar{S}$
$D = S$

JK



J	K	Q_{t+1}	
0	0	Q_t	
0	1	0	clear
1	0	1	set
1	1	$\bar{Q}_t$	

T

T	Q_{t+1}
0	Q_t
1	$\bar{Q}_t$

$J = K$

- TABLAS DE EXCITACION

Q_t	Q_{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

PARAMETROS MAS RELEVANTES

TIEMPO DE ACCESO: Tiempo que transcurre entre la llegada de la señal de lectura y que la información esté disponible a la salida

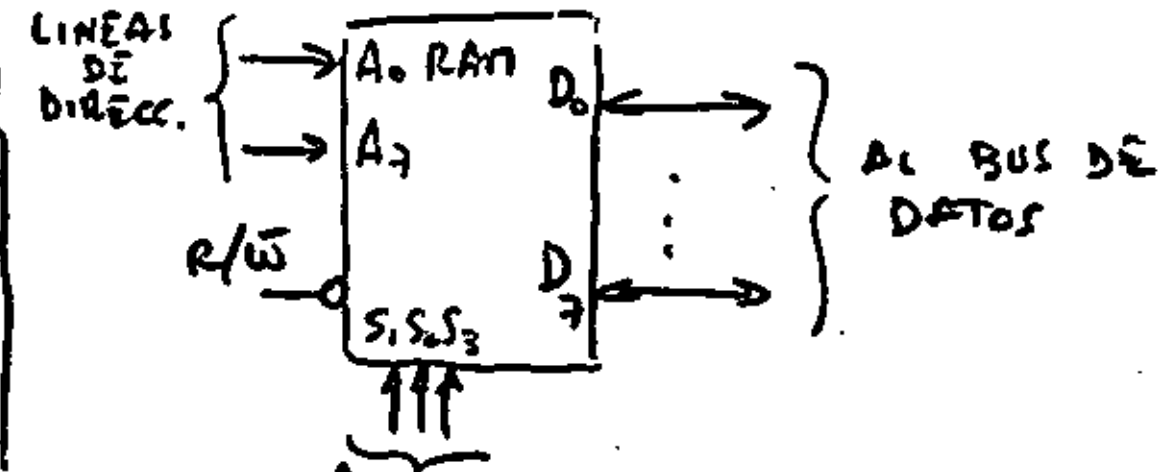
TIEMPO DE CICLO = TIEMPO DE ACCESO + TPO. RESTAURACION PARA CORE.

RAM – 100 ns : 400 ns

CORE – 1 μs

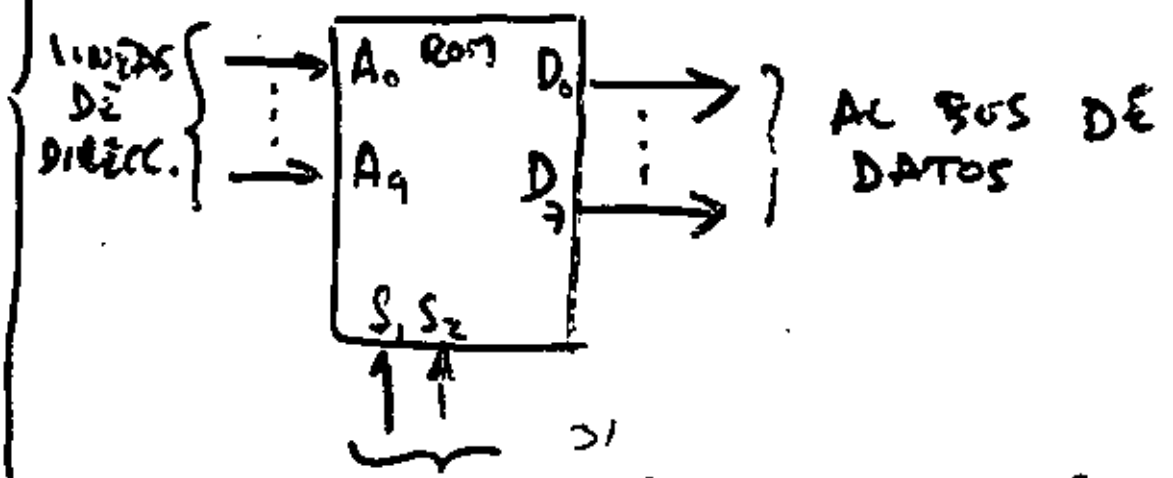
RAM

- TÍPICA : 128 x 8



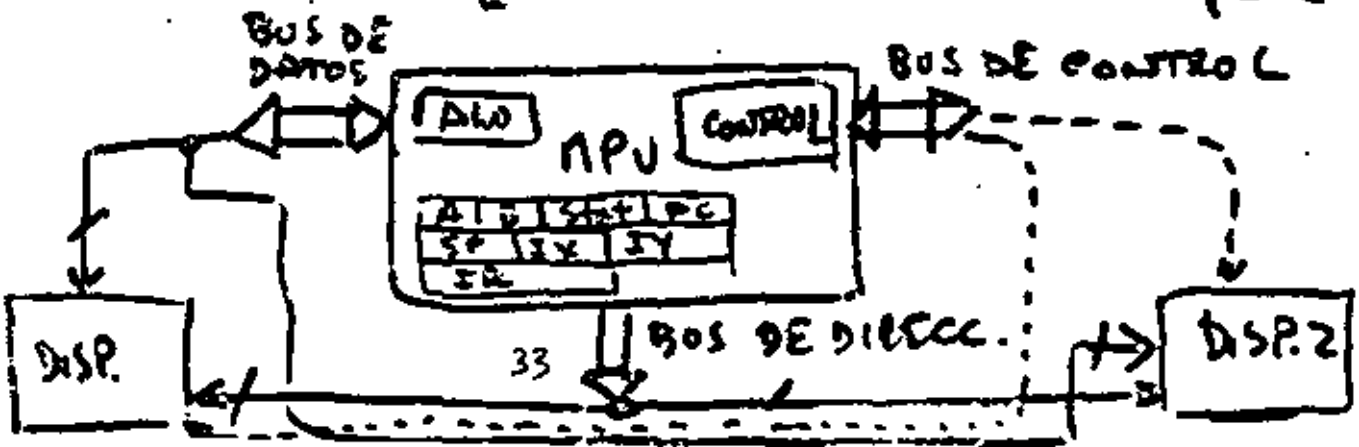
ROM

Típica : 1024 x 8



MICROPROCESADOR (MPU)

- EJECUTA OPERACIONES ← ARITMETICAS, LOGICAS, CONTROL
- GENERA LAS SEÑALES NECESARIAS PARA ACTIVAR PERIFERICOS Y SINCRONIZAR ACTIVIDADES
- REQUIERE DE UN RELOJ
- GENERA LAS DIRECCIONES
- COORDINA EL PROCESAMIENTO DE LA INFORMACION
- #LINEAS DE DIRECC ~ 16 (TIPICO)
- # LINEAS DE DATOS ~ 8 (TIPICO)
- #LINEAS DE CONTROL ~ VARIABLE
  - IOBQ
  - INT
  - NMI
  - R/W
  - ACK
  - IRQ
- ELEMENTOS
  - ALU
  - REGISTROS DE PROP. GRAL { A B C D
  - REGISTRO DE STATUS { E F
  - REGISTROS INDICE { IX IY
  - APUNTADOR DE STACK { SP
  - CONTADOR DE INSTRUCC. { PC
  - REGISTRO DE INSTRUCC. { IR



INTRODUCCION A LOS MICROPROCESADORES (Z-80)

T E R M I N O L O G I A

MARZO, 1981.

GENERAL TERMS and DEFINTIONS

(Read and/or Write)	The time interval between the instant data is inserted into or requested from memory and the instant that data transfer is completed (may be the longest path.)
Accumulator	A temporary storage register(s) that (stores) sums and other arithmetic and logical operations of an arithmetic logic unit (ALU.)
Address	A character or group of bits that identifies a particular location in memory, or other locations of data sources and destinations.
Addressing Modes	The methods of specifying the location(s) of data or program segments in memory or other locations.
Arithmetic and Logic Unit (ALU)	That part of a CPU that performs arithmetic, logical and related operations. Along with memory and control, an essential microprocessor element.
Architecture	The organizational structure of a computing system. Refers mainly to physical makeup of the microcomputer (Section 2) or microprocessor (Section 10) parts in this volume.
Assembler	A computer program used to translate a symbolic-language statement to a machine-language statement on a one-for-one basis.
Assembly Language	A programming language utilizing symbolic representation. The symbolic representation, sometimes called mnemonics, suggests the instruction function and is translatable by the assembler into machine-language.
Asynchronous Operation	A switching network operation with no common timing source. Circuit operation is such that the completion of one event initiates the next.
Baud	A measure of serial data transmission flow in communications applications. Baud rate generally defines signal bits per second transmitted, but may also include character framing bits.
Benchmark Problem	A frequently used (sample) problem employed to compare and evaluate computers (microcomputers.) Permits comparison of the number of instructions, memory words, and operation cycles required to solve the same problem.
Binary Coded Decimal (BCD)	A number coding system in which each decimal digit is represented by a 4-bit binary word. The decimal number 13 becomes the coded-number 0001 0011 in BCD using an 8-4-2-1 binary code.
Bit	The abbreviation of "binary digit" and the single characters in a binary number (1 or 0).

GENERAL TERMS and DEFINITIONS

Data Register	Any register that holds data.
Debug (Program)	A computer program designed to aid in detecting, tracing and eliminating errors in microcomputer (or other computer) programs while they are running. Allows replacing, adding, or revising instructions into the main operating program.
Decrement	A program instruction that decreases the contents of a storage location.
Dedicated Microprocessor	A microprocessor that has been programmed for a specific, single application.
Diagnostic Program	A computer program designed to check the operation of various hardware and software parts of the microcomputer sytem. Typically written for each functional area; e.g., CPU diagnostics for CPU checks, Memory diagnostics for Memory checks, etc. . .
Direct Addressing	An addressing mode in which the address of the instruction or operand is completely specified in the instruction without reference to a base register or index register.
Direct Memory Access (DMA)	A method of inserting input/output data into storage or obtaining input/output data from storage directly without involving the usual flow of data through the processor registers.
Editor (Program)	A computer program designed to allow manipulation of source program text material.
Emulate	To imitate one system with another, the latter being microprogrammable and equipped with a special mode program, so that the imitating system executes the same programs and achieves the same results as the imitated system. (See Simulate.)
Execution Time	The time, normally expressed in clock cycles, required to carry out an instruction.
Fetch	The reading out of an instruction from main memory and the insertion of the instruction into working memory.
Firmware	Programming instructions stored in a read only memory (ROM.)
Flag Bit	An information bit that indicates the occurrence of special conditions such as — overflow, carry, interrupt
Flow Chart	A graphical representation of a problem and the operations to be accomplished to solve the problem.
FORTRAN	A high level programming language used to facilitate the expression of computer programs in arithmetic terms and formulae. Short for "Formula Translator."
Handshaking	A colloquial term that describes the method used by a modem (or other asynchronous devices) to establish a communication link for eventual data transmission.

Jump	An unconditional departure from the normal instruction sequence to a different place in the program.
Loader (Program)	A computer program designed to read (enter) a program from an input device into working storage. (Random Access or Read/Write Memory – RAM.)
Look Ahead	A central processing unit (CPU) feature that allows the computer to mask (ignore) an interrupt request until the following instruction has been executed.
Loop	A sequence of instructions in which the final instruction causes repetition of the sequence a number of times until a termination condition as detected by a branch instruction is reached.
Machine Language	The numeric form of specifying every bit of every instruction in a program. The final binary program code that a computer uses directly.
Machine Cycle	The basic central processing unit (CPU) cycle in which; an address may be sent to memory and a word (data or instruction) may be read or written; or in which a fetched instruction may be executed. One machine cycle is made up of several clock cycles.
Macro Instruction	A symbolic source assembly language statement that combines any group of frequently used commands (instructions). The macro instruction will be expanded by the assembler into several machine language instructions.
Memory Address Register	The register in the central processing unit (CPU) that contains the address of the storage (memory) location being accessed.
Microcomputer	A computer system whose central processing unit (CPU) is a microprocessor. A basic microcomputer includes a microprocessor, memory, and input/output facility.
Microinstruction	An element in a microprogram.
Microprocessor	A single integrated circuit chip, or set of chips, capable of performing the functions of a computer central processor.
Microprogram	The computer instructions stored in a read-only-memory (ROM) that implement the basic instruction set of the computer.
Microprogrammable Computer	A computer in which the microprogram (microinstructions) in the read-only-memory (ROM) can be changed, thus changing the computer's instruction set.

Port (I/O Port)	Terminals that provide electrical access between the (micro)computer and the outside world.
Program Counter	A register in the central processing unit (CPU) whose job is to step the (micro)computer through the (micro)program. This register holds (saves) the address of the instruction under execution in the event of program interrupt or branching activity.
Programmable Logic Array (PLA)	An array of logic elements whose interconnections can be programmed (usually mask programmable, some field programmable) to perform a specific logical function.
Programmable Read-Only Memory (PROM)	A memory array manufactured with a constant logical pattern (all ones or zeros) and that can be written into (programmed) by the user. EAROM's (electrically alterable ROM's) can be electrically erased and reprogrammed. EPROMS can be erased and reprogrammed using ultraviolet light and electrical signals.
Push Down Stack	See "Stack."
Random Access Memory (RAM)	A memory device that has read and write capability and that provides direct access to stored information, regardless of location. Read or write time is independent of data location.
Read	To acquire and/or to interpret data from a memory device.
Register	Small scale memory capable of holding a fixed amount of data, such as a word. Used mainly for temporary storage, and accessible to the central processing unit (CPU.) The number of registers in a microprocessor is directly related to the program efficiency.
Relative Addressing	An addressing mode in which the address is determined by adding an offset address to a base address stored in some register. Permits the computer to relocate blocks of data by changing only one number.
Read-Only-Memory (ROM)	A memory device generally used to store a computer's control programs (firmware), and not alterable by normal operating methods. Exceptions are PROM's, EPROM's and EAROM's, which are ROM's whose contents may be altered in the field by special means.
Scratch Pad Memory	Read/Write (RAM) memory devices or registers that are used to store temporarily intermediate results (data), or memory addresses (pointers.)
Simulate	To imitate one system with another using a program written in assembly or high level language so that the imitating system accepts the same data, executes the same programs, and achieves identical the same results as the system being imitated. (See Emulate.)
Slice	A partition of a microprocessor that enables several identical units to be paralleled to realize and augmented by control logic to realize the CPU.
Software	A collection of computer programs, procedures, rules and documentation used or associated with the operation of the computer.

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

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

DISEÑO Y APLICACIONES DE MICROPROCESADORES

MARZO, 1981

Palacio de Minería Calle de Tacuba 5 primer piso México 1, D. F. Tel: 521-40-20 Apdo. Postal M-2285

CHAPTER 1 OVERVIEW

Although the first microprocessor, the Intel 4004, appeared six years ago, the microprocessor field still seems new and exciting. The rapid advances in microprocessor technology, the creativity of the software designers, and the explosion of microprocessor applications have all contributed to this image. In the popular mind, the computer on a chip has been realized, and the availability of computing power to everyone is at hand. While this is not yet the case, it is true that few of us will long escape the impact of microprocessors. Microprocessors are scheduled for use in most automobiles in the late 70's and early 1980's, and the market for television modules is just now developing. For engineers and scientists, microprocessors will increasingly be employed in instruments, terminals, and a wide variety of military and industrial control and communication systems.

Components - Not Computers (A viewpoint)

There is a wide spectrum of views on microprocessors due in part to the wide variety of available microprocessors. The word "microcomputer" is frequently used to emphasize that the basic operation of these devices and their associated systems is nearly identical to that of conventional computers.* [1] On the other hand digital designers often view microprocessors as convenient replacements for hardwired digital subsystems. One should note that microprocessors are products of the components industry and not of the computer industry.

1. In this text, "microprocessor" will be used when describing the central processing unit of a "microprocessor system" (or "microcomputer").

"Microprocessor Family". Later, Intel developed its systems components for the 8080. The designer can expect a continuing growth in the family size for most microprocessors since the capabilities of the systems can be increased without moving to another processor and it concommitant software investments.

Rather special memory components have also been developed for microprocessors. These chips may have multiple chip selects that simplify the address decoding and may be organized so that a minimum number of chips can be used. The ubiquitous 2102 is organized as a 1024 by 1 bit chip (1024 words, 1 bit long), and a minimum memory for an 8-bit processor would require eight chips. This memory would have 1Kbyte capacity and might be too large for many microprocessor applications. Instead, one could use a Motorola 6810 which, in one chip, has 128 bytes. (In most microprocessor applications, the program is stored in read-only-memory and not in read/write memory.) Ultraviolet erasable read-only-memory (ROM) is also available for proto- type systems.

Finally, read/write and ROM memories have been combined with input/output circuits on the same chip. Obviously, the manufacturers are trying to provide chips that will simplify the design of microprocessor systems by minimizing the number of chips. Further, the load on the address and data buses is reduced and the overall reliability increased.

Computers are Smarter than Programmers

Software remains a costly feature of the design process for a microprocessor system. There are two reasons for this. First, microprocessors have a limited instructions set and limited addressing modes. The programmer simply must work harder and more carefully. Second, in contrast to computer manufacturers,

in exactly the same way). Here the designer must exersize careful judgement is the selection of the processor, memory, programmable input/output, board design, development system, etc. The large number of systems justifies considerable attention to the minimization of costs ,to testing and reliability, and to documentation. Since the same software will be used, the designer can try to optimize the software so that hardware costs can be minimized (e.g. memory size) through various tradeoffs in software. In this latter case, a designer should completely understand the components of the system and be able to cope with the software.

In this tutorial, we will attempt to study some of the fundamental ideas involved in microprocessor system design and to review the basic characteristics of microprocessor system components.

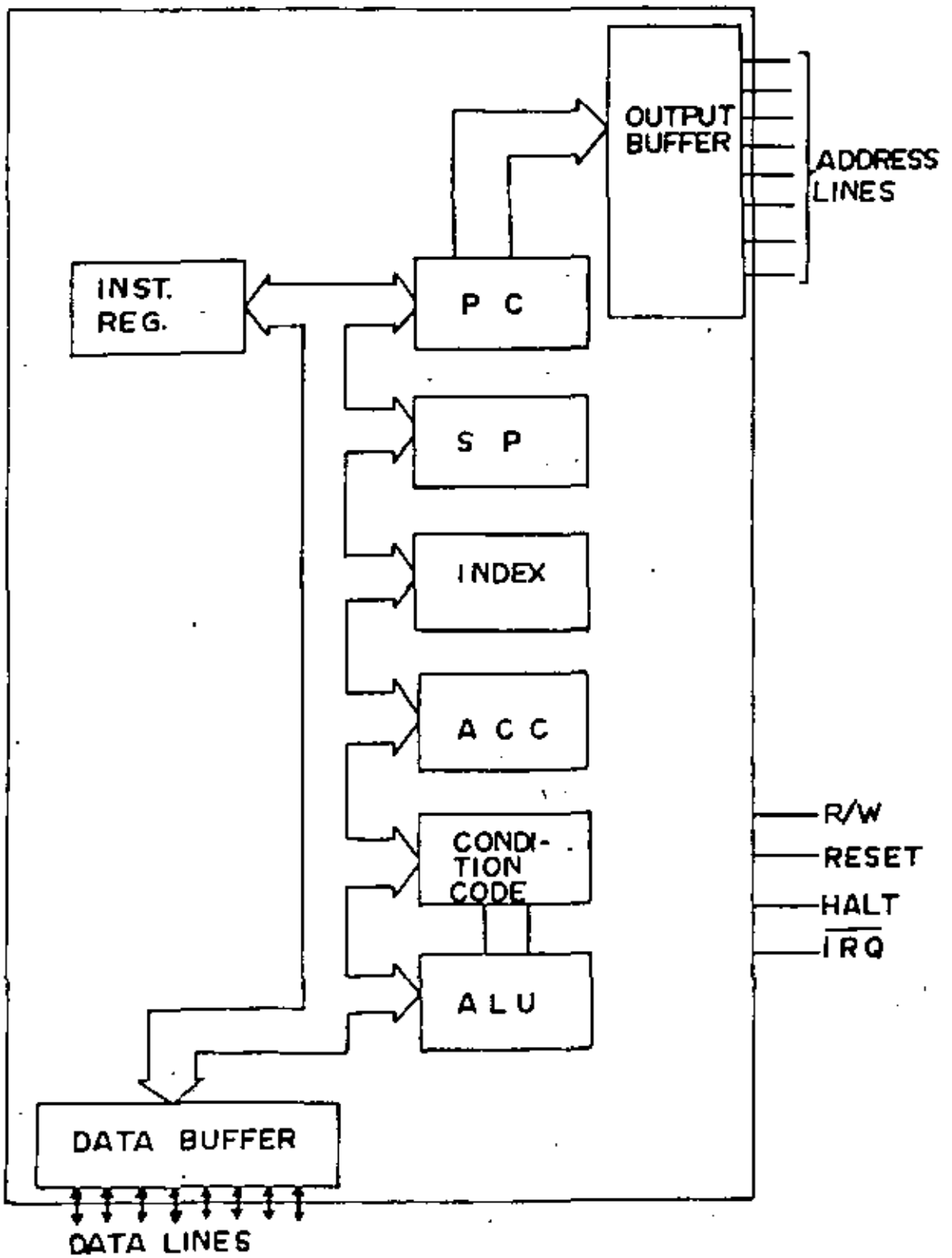

OUTPUT BUFFER
ADDRESS LINES
INST. REG.
P C
S P
INDEX
A C C
CONDI- TION CODE
A L U
R/W
RESET
HALT
IRQ
DATA BUFFER
DATA LINES

Fig. 2-1

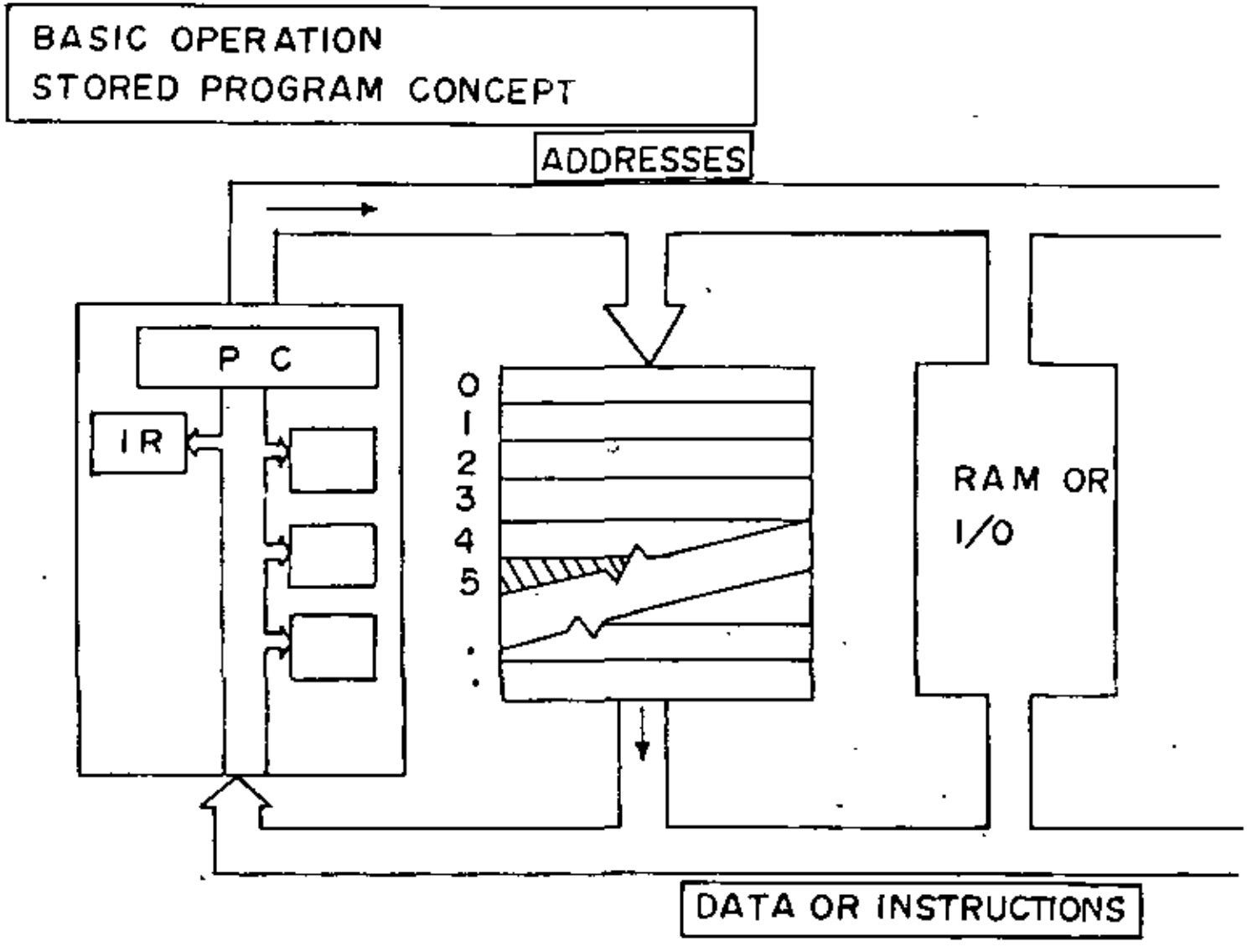
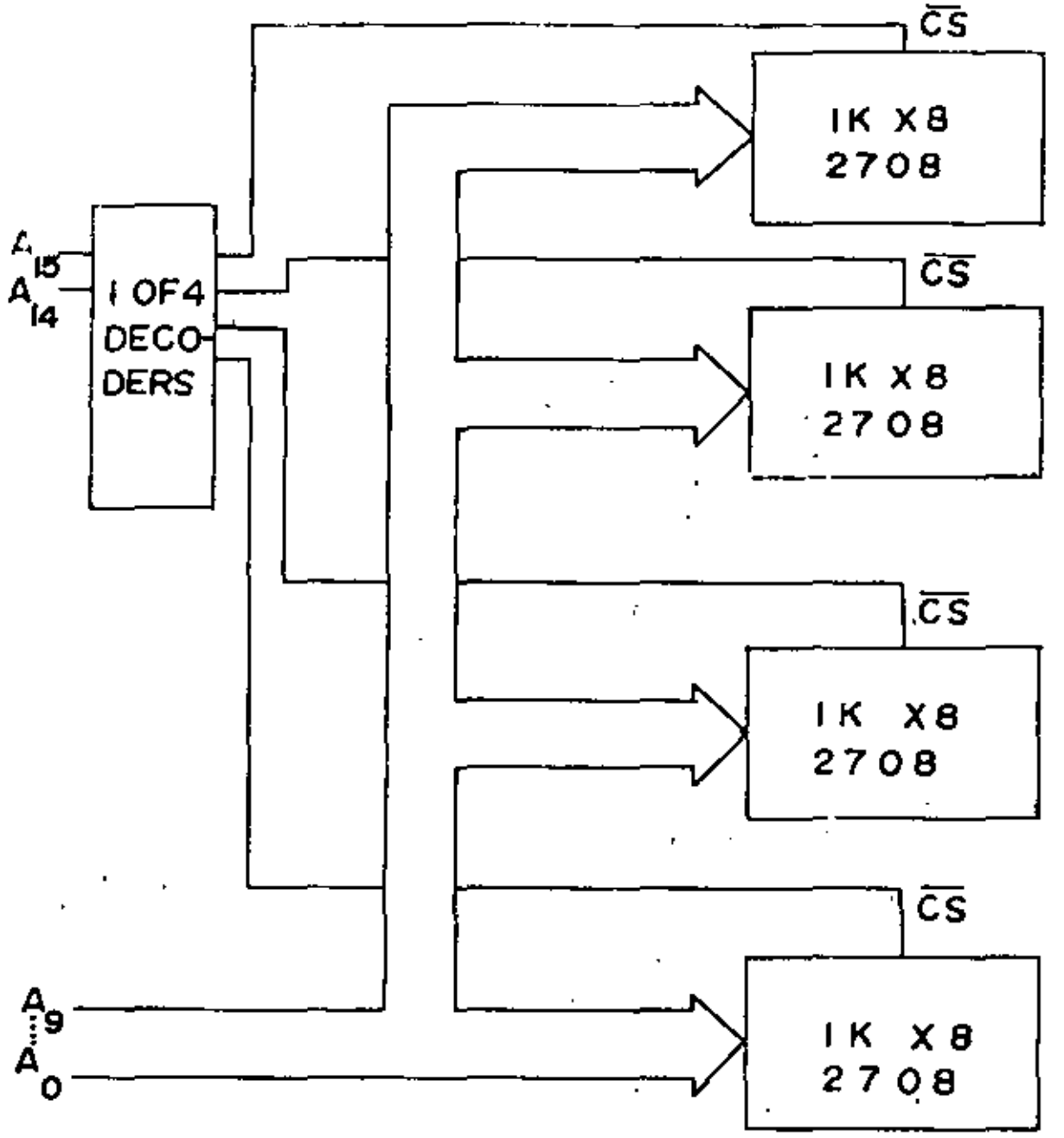

BASIC OPERATION
STORED PROGRAM CONCEPT
ADDRESSES
P C
IR
0
1
2
3
4
5
RAM OR
I/O
DATA OR INSTRUCTIONS

Fig. 2-2a



MEMORY MAP

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	•	•	•	•	X	X	X	X	X	X	X	X	X	X	0000-03FF
0	1	•	•	•	•	X	X							X	X	0400-43FF
1	0	•	•	•	•	X	X									8000-83FF
1	1	•	•	•	•	X	X							X	X	C000-C3FF

• - DON'T CARE

Fig. 2-3b

minicomputers are "fully decoded", hence if the processor references locations where memory does not exist, a "trap" to a special program for such errors usually takes place. Generally, in microprocessor systems the processor may read and write in non-existent memory without impunity or it may address the same physical location through multiple addresses. Thus microprocessor programmers must be more careful because no "hand shaking" occurs between processor and memory to assure the processor that the memory exists or that the location is uniquely defined.

Memory Mapped vs. Isolated Input/Output

In most microprocessor systems, the input and output registers and their associated control and status registers are seen as memory locations by the processor. Thus the input and output registers and the ordinary memory locations occupy the same address space (i.e., they fall within the 65,536 locations addressable by a sixteen bit address bus of a typical processor). This scheme is called memory mapped I/O and is illustrated in Fig. 2-4. The advantages of this scheme is that one can apply any of the memory reference instructions to the input/output. Thus to create a pulse at an output bit, one can simply use the INC (increment) and DEC (decrement) commands which ordinarily increment and decrement memory locations. Some processors have separate I/O instructions; the 8080 is a good example. The 8080 generates input and output signals separate from the memory read and write signals. The input and output signals essentially are read and write signals for the I/O registers selected by eight of the address lines. The advantages of isolated I/O are that less address decoding is necessary and that the full address space can be used for memory. In isolated I/O, special instructions such as IN and OUT are used; these place the contents of the accumulator at the output register or place the contents of the input

register in the accumulator. The 8080 is easily converted to memory mapped as shown in Fig. 2-5 and as discussed in Intel's "8080 Microcomputer Systems User's Manual."

Isolated I/O is used in some of the newer processors such as Intel's 8048 and 8085. The 8085 is a later version of the 8080 chip which helps to minimize the chip count while keeping software compatibility. The 8080 is really a three chip processor because it requires a 8228 data bus chip to capture the status and a 8224 timing chip to generate the clock signals and synchronize control signals. A typical 8085 system is shown in Fig. 2-6, ; the 8085 will be used in addition to the 8080 for illustrations and examples. The 8048, a true microcomputer on a chip, is discussed later in Chp. 5 as one of the best, simpler processors. All of the Intel 80XX processors can be made compatible with simple logic circuits, and even other, quite distinct processors such as the 6800 or 6505 can be interfaced with Intel-like circuits.

A simple microprocessor system based on the MOS Technology 6502 is shown in Fig. 2-7 to illustrate address decoding and memory mapped I/O. A simple 8080 system is included in Appendix A along with other operation details. These simple examples show that the number of chips required to fabricate a working system is quite small. Yet quite sophisticated operations may be performed by these systems. In addition, these systems provide excellent I/O capabilities, typical of most microprocessor systems. In the 8085 system, the 8755 provides 2K bytes of ROM in addition to the I/O ports ad the 8155 provides 256 bytes of RAM. Both the ROM and RAM capacities are suitable for a large number of applications. Note that limited address decoding is provided by the 11th and 12th bits of the address bus. These systems require only a few resistors and capaci-

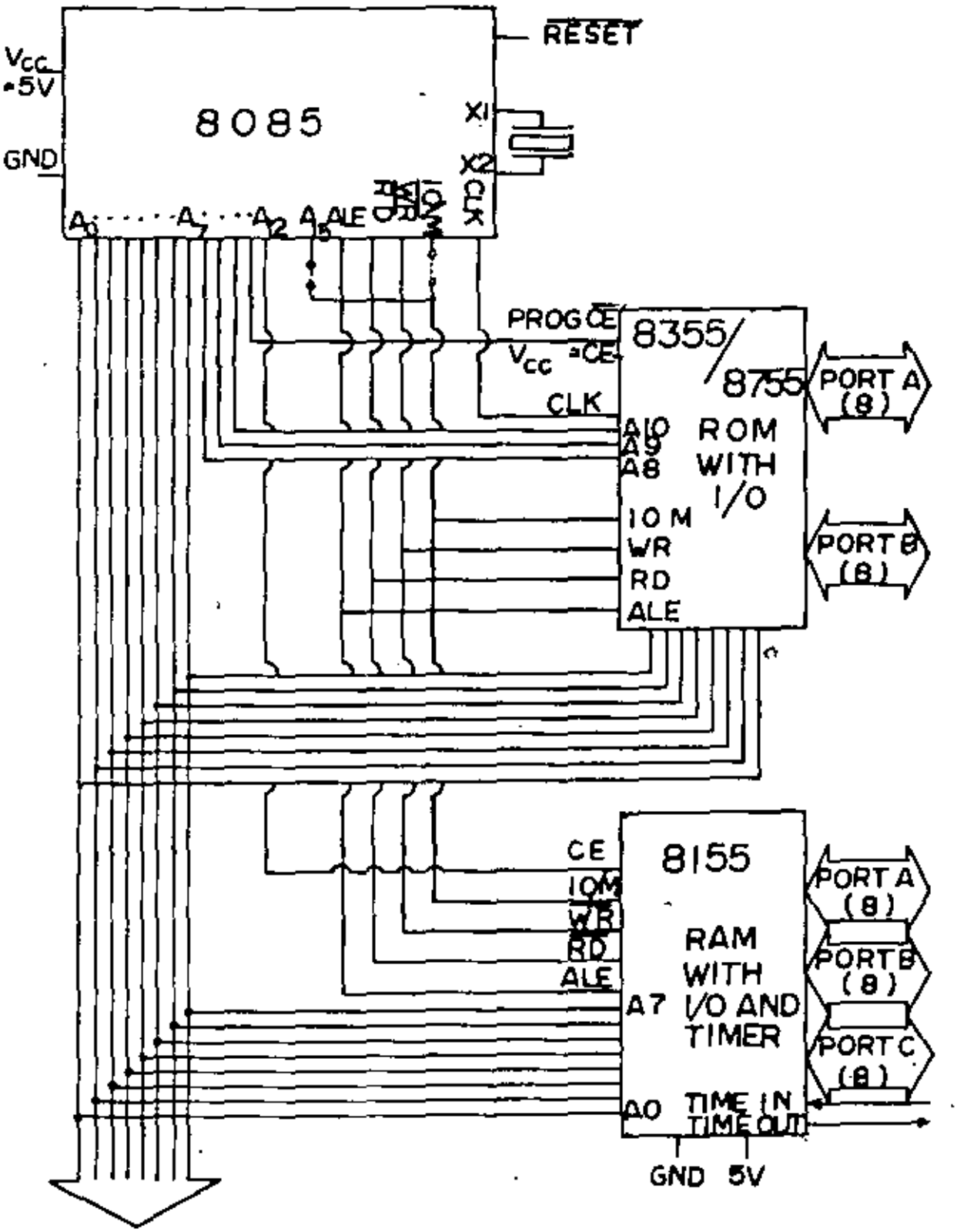

8085
RESET
V_{CC} +5V
GND
X1
X2
CLK
ALE
PROG
V_{CC}
CE
CLK
8355/8755
A10
A9
A8
ROM WITH I/O
IO M
WR
RD
ALE
PORT A (8)
PORT B (8)
8155
CE
IOM
WR
RD
ALE
A7
RAM WITH I/O AND TIMER
A0
TIME IN
TIME OUT
PORT A (8)
PORT B (8)
PORT C (8)
GND
5V

Fig. 2-6

tors in addition to the chips and crystal shown to fabricate a working system. The 6505 systems uses a 1702 ROM with 256 bytes and a Motorola 6810 with 128 bytes. Address decoding is provided by the 7445 decoder chip. Again limited address decoding is used. The 6505 has only 12 address pins, and thus it can address only 4K bytes; but this capacity is very adequate for most systems. The 650X series from MOS Technology resembles the M6800. In some cases, a considerable simplification of the interfacing requirements has been achieved. Note that a very simple clock circuit is used and that the chip select of memory or I/O occurs with phase two of the clock. The system shown is probably the least expensive microprocessor system that can be assembled. It fits on wiring strips such as the Super-Strip and costs less than $30 is single quantities.

Restart Locations

Upon a RESET signal, the program counter of a processor is set to a specific state. In the 8080, the PC is cleared so that it points to location 0. The first instruction of the program must be in that location. Since the program is stored in ROM for nonvolatility (retains data without power), the ROM is usually located in the lower memory locations. In the 6800, the PC is loaded with the contents of location FFFE and location FFFF on RESET, thus the ROM in a 6800 system is usually located at the top of memory. In addition, the 6800 has addressing modes which makes special use of the lowest memory locations, and thus read/write memory (RAM) usually starts a location 0. (Fig. 2-8).

The Data Bus

The data bus of most processors is eight bits wide; a few such as the SP 1600 and the TI 9900 are sixteen bits wide. Eight bits permit the

representation of all of tne alphanumeric characters; a byte is a natural unit for communication purposes. Most 8-bit processors have limited instruction sets because of the width limitation, the Z-80 perhaps is an exception. The 8080 uses 244 of the possible 256 op codes, the 6800 uses 195, and the 8048 uses 239

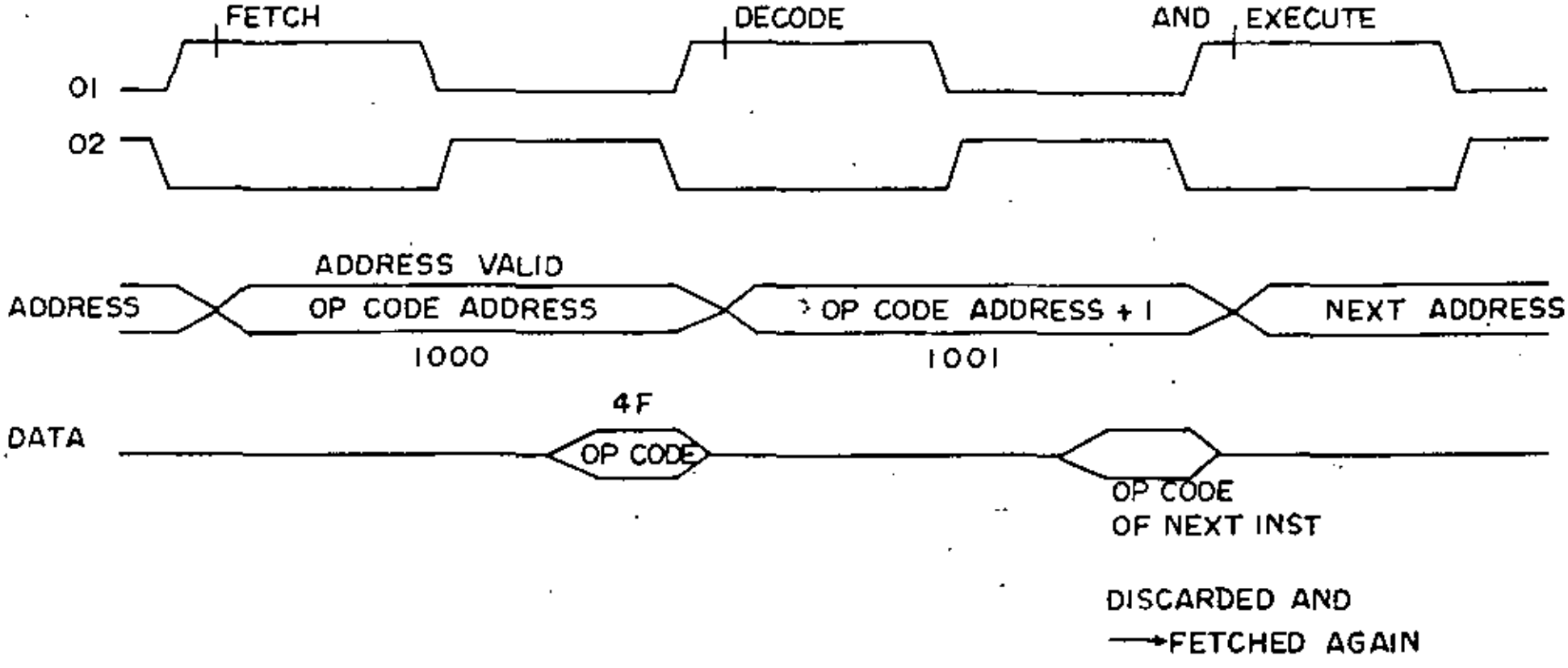
In contrast to the address bus, the data bus is bidirectional, hence data can be read from or written into memory via the same lines.

In a typical system, the outputs of several memory or interface chips are connected to the data bus; however, only one device can place its contents on the bus at any given time. Thus most microprocessor system chips have "tri-state" or high impedance outputs. Unless a chip is selected by the address decoding process, its outputs remain in the off or floating state and do not affect the output. When selected, the outputs are either the 1 or 0 state.

The data and address buses can drive only a limited number of chips. A rule of thumb is that ten family chips may be connected without buffering. Each memory or I/O device places a capacitance on the bus a permits a certain leakage current to flow. The address and data buses must be able to charge the parasitic capacitance of the wires and of the devices and must provide the leakage current in order for the processor to work at its rated speed. Special chips are available to buffer the buses. Intel designed the 8228 chip usually used with the 8080 so that it would buffer the data bus in addition to its other tasks. The data bus is the more difficult because it is bidirectional.

Timing and Clocks

Microprocessor operation is controlled by clock (oscillator) signals. In some cases, the oscillator is on the chip, and external components such a



ONE BYTE INSTRUCTION
(CLEAR ACCUMULATOR).

Fig. 2-9

instruction; the obviously save time and is called "pipelining". All of the decoding of the op-code and execution of an instruction requires operations between internal registers in the processor. The nonoverlapping two phase clock provides four "edges" (rising and falling transitions) that clock or toggle these internal registers in addition to other timing tasks. If only a single phase clock is used, more clock cycles will be required to provide the "edges". An example is shown in Fig. 2-11 for the Intel 8085. Here, the data bus and the lower byte of the address bus share the same pins. The lower address bits are sent out first and must by latched using the ALE as the enabling pulse on D-type latches. Later, the data flows in or out of the processor in the normal manner. While this may seem cumbersome, the complete 8085 system is much simpler than a corresponding 8080 system.

The 6800 and 650X microprocessors have the very simple two phase clocks just discussed. The 8080 also has a two phase clock, but its operations are very much more complicated. A more or less complete discussion is included in Appendix A.

Clock Rates

The clock rate alone is not an indication of the speed of a processor. It is frequently and incorrectly stated that the 8080 is twice as fast as the 6800 because the former's clock rate is twice as fast as the latter's. In fact, the 8080 requires more clock cycles to accomplish the same task, and the two processors have very similar speeds.[3] Of course, for a given processor, a higher clock rate yields faster execution.

---

3. Clock rates are only one of many characteristics that are incorrectly compared even in the most recent literature.

The maximum clock rate is limited by propagation delays, access times, and other factors within the processor. The slowest rate at which the processor will operate depends upon the type of storage in the internal registers; viz. static or dynamic. In dynamic storage, the information is stored on capacitors which will eventually discharge unless the information is refreshed. Thus the minimum clock rate depends upon the rate of discharge. The slowest specified clock rate for the 6800 is 100KHz. At room temperature where the leakage is not particularly large, a 6505 can be operated at 5 KHZ before it ceases operation. The 8080 also has dynamic storage, and its details are included in App. A.

For static storage (the associated clock is sometimes called a "static clock") the clock may be slowed as much as desired. The RCA COSMAC (1802) has a static clock and thus may be single stepped using a debounced switch. Static and dynamic storage in the internal registers is, of course, similar to the static and dynamic memory discussed in Chp. 3.

Finally, some processors provide synchronizing or state signals that may be used for the control of peripherals or for interfacing. The 8080, for instance uses a "synch" signal; the RCA COSMAC state signals, and the 6800 has a VMA (valid memory address) signal. As discussed ealier, some processors do not have sixteen address pins, but time multiplex the address information or share the address bus with the data bus. In this case, the sixteen address bits are sent out in two bytes, and one of the bytes will have to be latched or held by external circuitry. A timing signal for the latch has to be provided such as the ALE on the 8085.

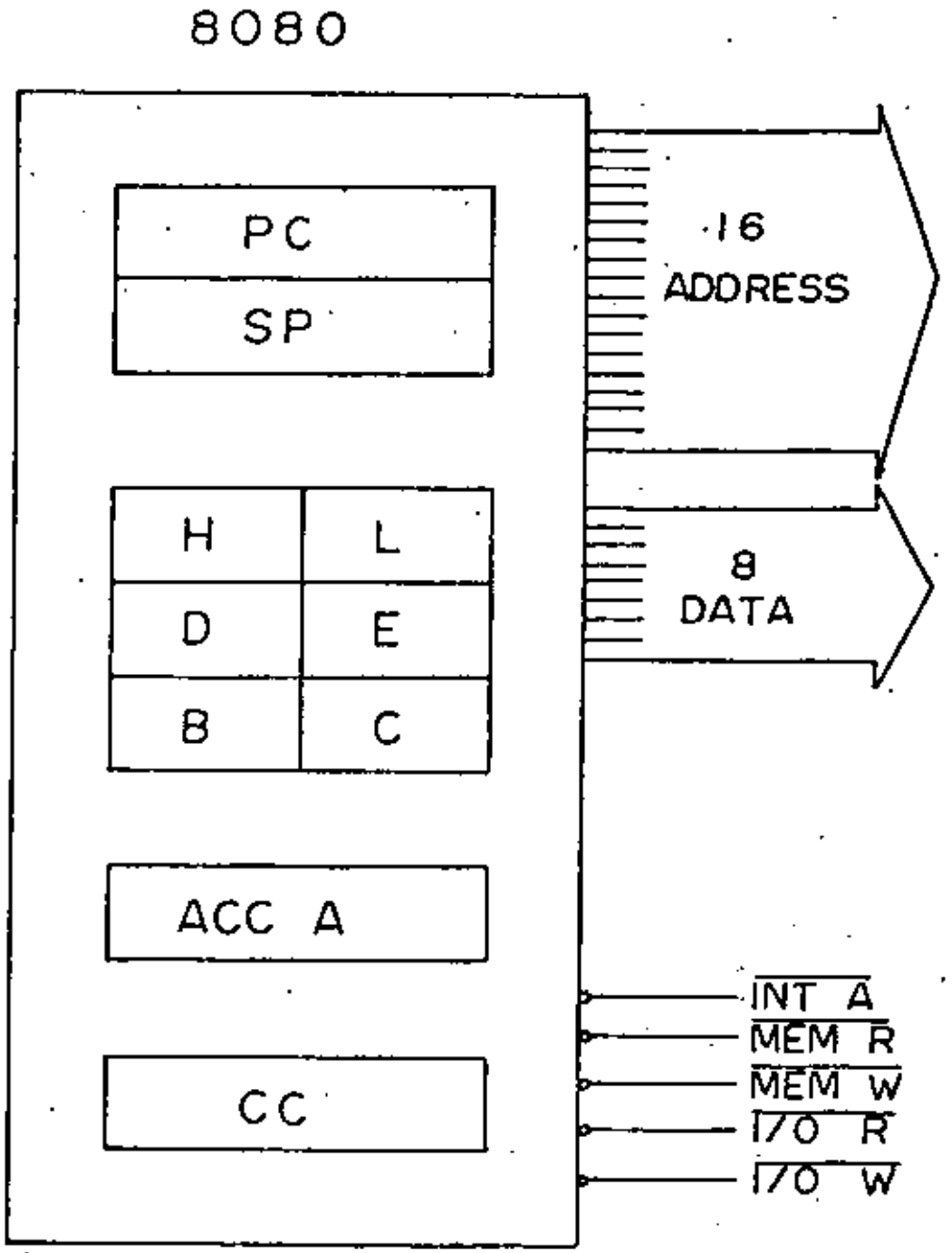

8080
PC
SP
H
L
D
E
B
C
ACC A
CC
16
ADDRESS
8
DATA
INT A
MEM R
MEM W
I/O R
I/O W

Fig. 2-12

CYCLE - BY- CYCLE OPERATION

EXTENDED ADDRESSING:

ADD @#XXXX

ADDRESS BUS	R/W	DATA BUS
Op code address	1	Op code
Op code address+1	1	Address of operand (high byte)
Op code address+2	1	Address of operand (low byte)
Address of operand	1	Operand data

DEC @#XXXX

ADDRESS BUS	R/W	DATA BUS
Op code address	1	Op code
Op code address+1	1	Address of operand (high byte)
Op code address+2	1	Address of operand (low byte)
Address of operand	1	Current Operand data
Address of operand	1	Irrelevant data (note)
Address of operand	0	New Operand data.

note: If VMA is used to enable memory, then VMA will be 0 during this cycle and the data bus will depend on capacitive decay.

Fig. 2-13

(Motorola's names for these modes are not standard names.) Conditional branches use the relative mode. These modes are summarized in the following.

Immediate mode: In the immediate mode, the operand is stored in the location following the op-code. For example consider the instruction which adds the number 5 to accumulator A:

```
ADD #5, ACCA
```

This would be assembled in the following form:

```
ADD #5, ACCA    89    (op-code)
                05    (data)
```

Extended and Direct Modes: The modes are really absolute addressing modes, i.e., the absolute or numerical address of the operand is given in the following byte or bytes. In the extended mode, the specific address of the memory location or I/O register is given in the following two bytes (the full sixteen address bits), the high order byte first. For memory locations with addresses less than 256, the direct addressing mode may also be used. Here, only one byte, specifying one of the first 256 locations is used. This direct mode permits faster executing and, of course, saves memory locations. An example is a load the accumulator with the contents of a memory location:

```
LDA @# 167A
```

being executed.

pointer registers, or the condition code register.

Finally the relative mode is used for the branch commands and consists of adding the second byte of the instruction to the program counter to divert the program execution to another part of the program. The second byte is a 2's complements number, meaning that the contents of the program counter may be increased or decreased. An example is a branch from the current location back to a previous instruction, often in a loop. For instance, one may wish to test the state of a process by testing a done bit. If the done bit is the most significant bit of a register then the register's contents are a negative number when the done bit is set (=1). We suppose that the register is located at 8008 (hex).

```
LOOP:   TST @#8008
        BPL LOOP
```

Thus unless the done bit is set, the loop test the contents of 8008 and branches back for another test. This would be assembled:

```
LOOP:   7D      (test)
        80
        08      (the address)
        2A      (BPL, branch of plus)
        FB      (2's complement of -5)
```

Note the one branches back -5 because the program counter will point to the location following the offset of the branch command while the branch command is

Data Handling Instructions II

Function	Mnemonic	Operation
Clear	CLR,CLRA,CLRB	00->M, A, B
Decrement	DEC,DECA,DECB	M - 1 -> M A - 1 -> A B - 1 -> B
Increment	INC,INCA,INCB	M + 1 -> M A + 1 -> A B + 1 -> B
Negate 2's complement	NEG,NEGA,NEGB	-M,-A,-B
Complement (1's)	COM,COMA,COMB	M, A, B

Data Handling Instruction III

Function	Mnemonic	Operation
Rotate left	ROL,ROLA,ROLB	c ->b0, b7 ->c
Rotate right	ROR,RORA,RORB	b0 ->c, c ->b7
Arithmetic shift left	ASL,ASLA,ASLB	b7 ->c, 0->b0
Arithmetic shift right	ASR,ASRA,ASRB	b7 ->b7, b0->c
Logic shift right	LSR,LSRA,LSRB	0 ->b7, b0->c

A - Accumulator A
B - Accumulator B
M - Memory location
c - carry bit
b0 - bit 0
b1 - bit 7

Conditional Branch and Jump Instructions

Function	Mnemonic	Condition
Branch always	BRA	none
Branch if = zero	BEQ	Z = 1
Branch if ≠ zero	BNE	Z = 0
Branch if plus	BPL	N = 0
Branch if carry set	BCS	C = 1
.		
.		
.		
Branch to subroutine	BSR	
Jump	JMP	
No operation	NOP	

This is a partial list of the branch and jump commands; some depend upon the bits in the condition code register which are summarized below:

Bit no.		Condition Code
0	C	carry - borrow
1	V	overflow
2	Z	zero
3	N	negative
4	I	interrupt mask
5	H	half carry (for dec. arith.)

identical systems, mask programmed ROMs are available from a number of manufacturers which are pin for pin compatible with the erasable ROMs. Even larger capacity mask programmed ROMs are available; e.g. the Intel 8316A with 16K bytes. Other types of programmable ROMs include fusible link ROMs in which a nichrome or silicon link is burned opened to store a 1 or 0 in the corresponding bit location. Fusible link ROMs are usually bipolar hence much faster than the MOS ROMs; but they are less practical since the faster speed is not necessary , they can be programmed only once, and the chip capacities are smaller. Intel, MOS Technology and others have combined ROMs and I/O circuits on the same chips. This helps to minimize the chip count for the system. Some of the ROMs are reprogrammable. The ultimate chip is the 8748, a member of the the MCS-48 family of Intel, which contains the processor, erasable ROM, read/write memory, interval/event timer, and copious I/O.

In addition to program storage, ROM can store tables of data, codes, monitor programs, assemblers, editors, etc. In evaluation kits and development systems, monitor programs assist the user in writing, editing, and executing programs.

Read/Write Memory: RAM

Actually, ROMs are random access memory (RAM), but "RAM" is usually reserved for read/write memory. Developments in RAM have proceeded as least as fast as microprocessor development. Memory chips with 16 K by one bit organizations are available; eight such chips would provide 16k bytes. In microprocessor systems, RAM is usually used for data storage, however the program may be stored in and executed from RAM. RAM is volatile, and thus requires battery backup if a power loss occurs. Many microprocessor systems require only a small

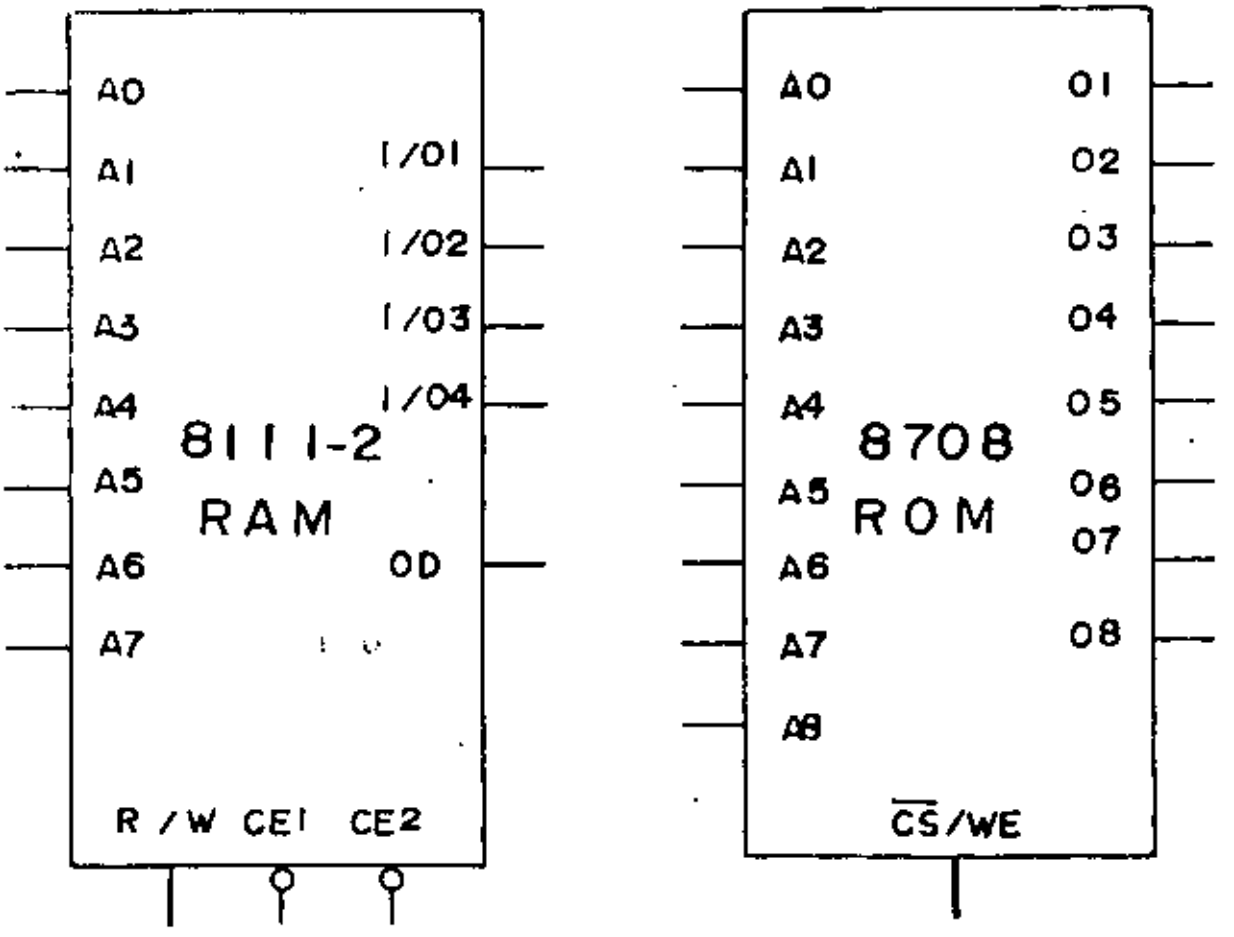

A0
A1
A2
A3
A4
A5
A6
A7
8111-2
RAM
I/01
I/02
I/03
I/04
OD
R/W CE1 CE2
A0
A1
A2
A3
A4
A5
A6
A7
A8
8708
ROM
01
02
03
04
05
06
07
08
CS/WE

Fig. 3-1a

The smaller capacity RAMs are static. Here the bit data is stored on flip-flops and the data will remain so long as power is applied. Power-down options are available which save power when the memory is not accessed by the processor. Static memory is easier to use than dynamic and is the obvious choice for small RAM systems.

Access Time and Cycle Time.

Recall that microprocessor timing is controlled by clock cycles. When the processor accesses memory, it sends out address information for chip selection and for byte location within a chip(s). At the end of phase two in our earlier example in Chp. 2, the processor expects valid data. Thus the memory must be fast enough to yield valid information. The time between the application of valid address and the data valid state is called the "access time" and must at least match that of the the processor. Shown in Fig. 3-2 are the timing data for a 2111. Note also that the chip select may occur after the address is applied. Usually the address information is involved in a chip select, but the phase two clock may also be involved in chip select. Thus a second time, the "chip enable to output" time is also important. Typical times are shown in the figure and in the associated table. Other times are included for completeness. Extremely fast memory, faster than 400 nsec would be a wasted expense for most systems.

Simple Memory System

Shown in Fig. 3-3 is a simple system using one ROM and two 2111 RAMs. The memory is connected to an 8080 bus and some address decoding is accomplished by

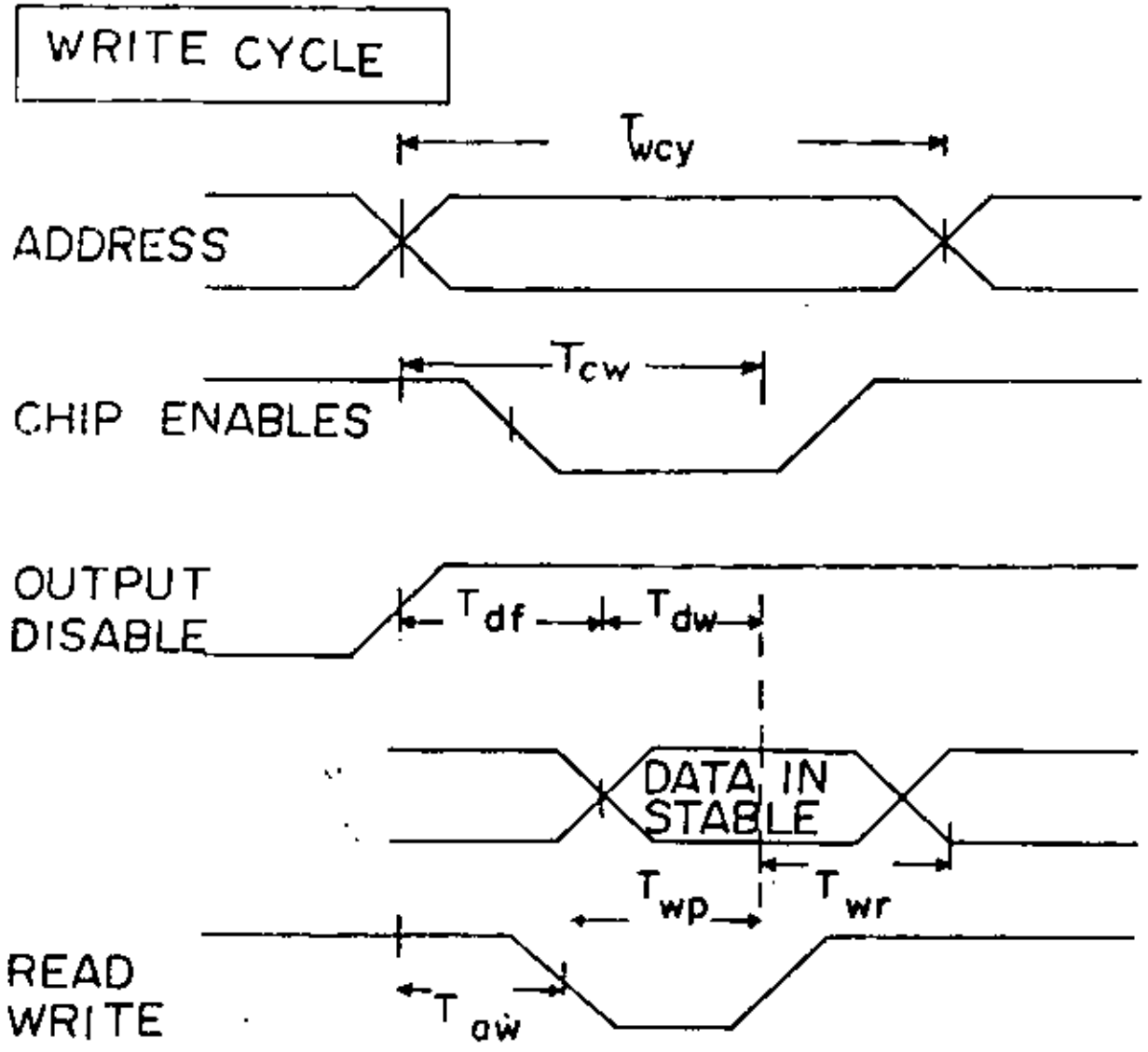

WRITE CYCLE
T_{wcy}
ADDRESS
T_{cw}
CHIP ENABLES
OUTPUT DISABLE
T_{df}
T_{dw}
DATA IN STABLE
T_{wp}
T_{wr}
READ WRITE
T_{aw}

Fig. 3-2b

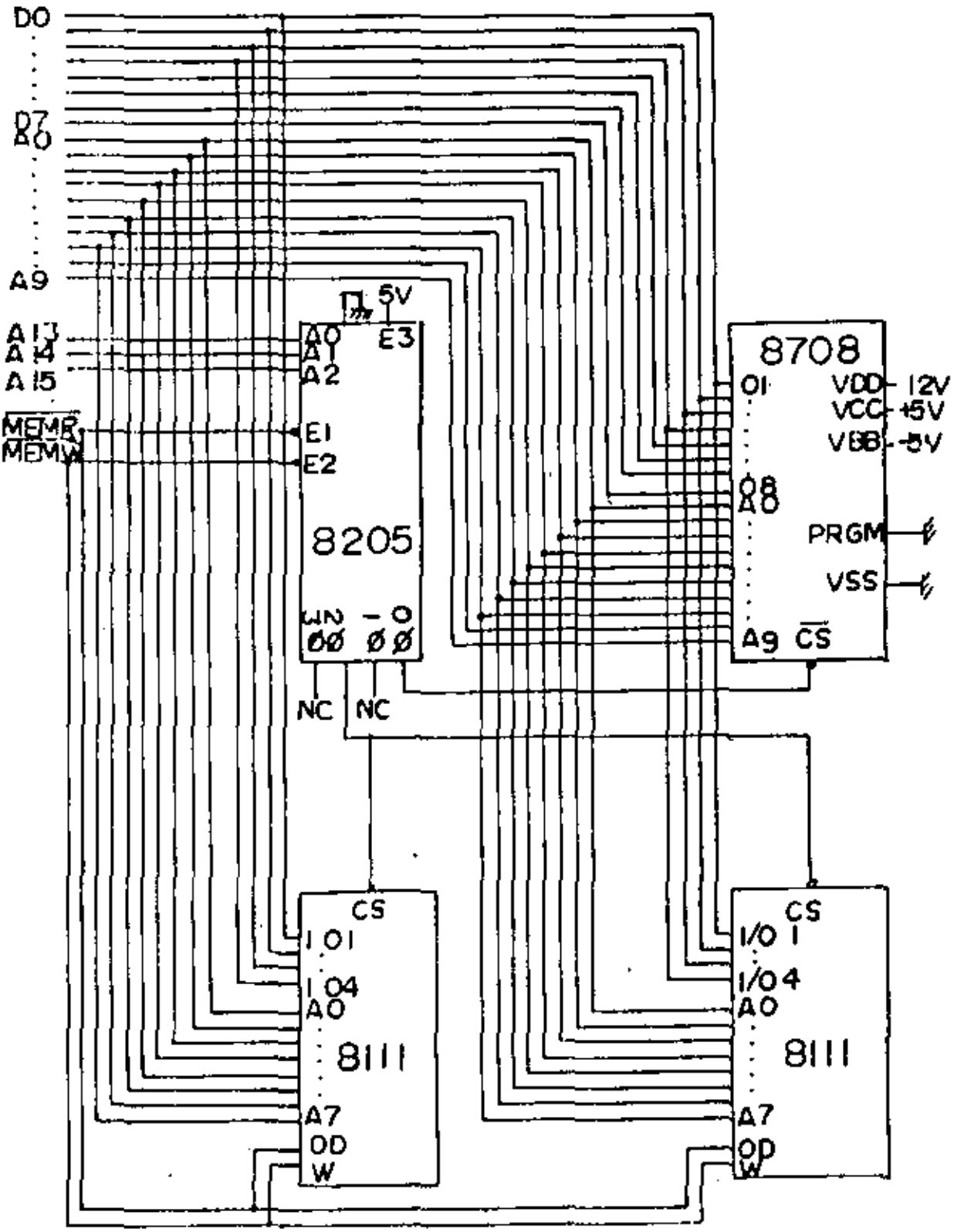

D0
D7
A0
A9
A13
A14
A15
MEMR
MEMW
5V
A0
A1
A2
E3
E1
E2
8205
WN
O
NC
NC
8708
O1
VDD 12V
VCC +5V
VBB -5V
O8
A0
PRGM
VSS
A9
CS
CS
I O1
I O4
A0
8111
A7
OD
W
CS
I/O 1
I/O 4
A0
8111
A7
OD
W

Fig. 3-3

MULTIPLEXED DMA/MPU OPERATION

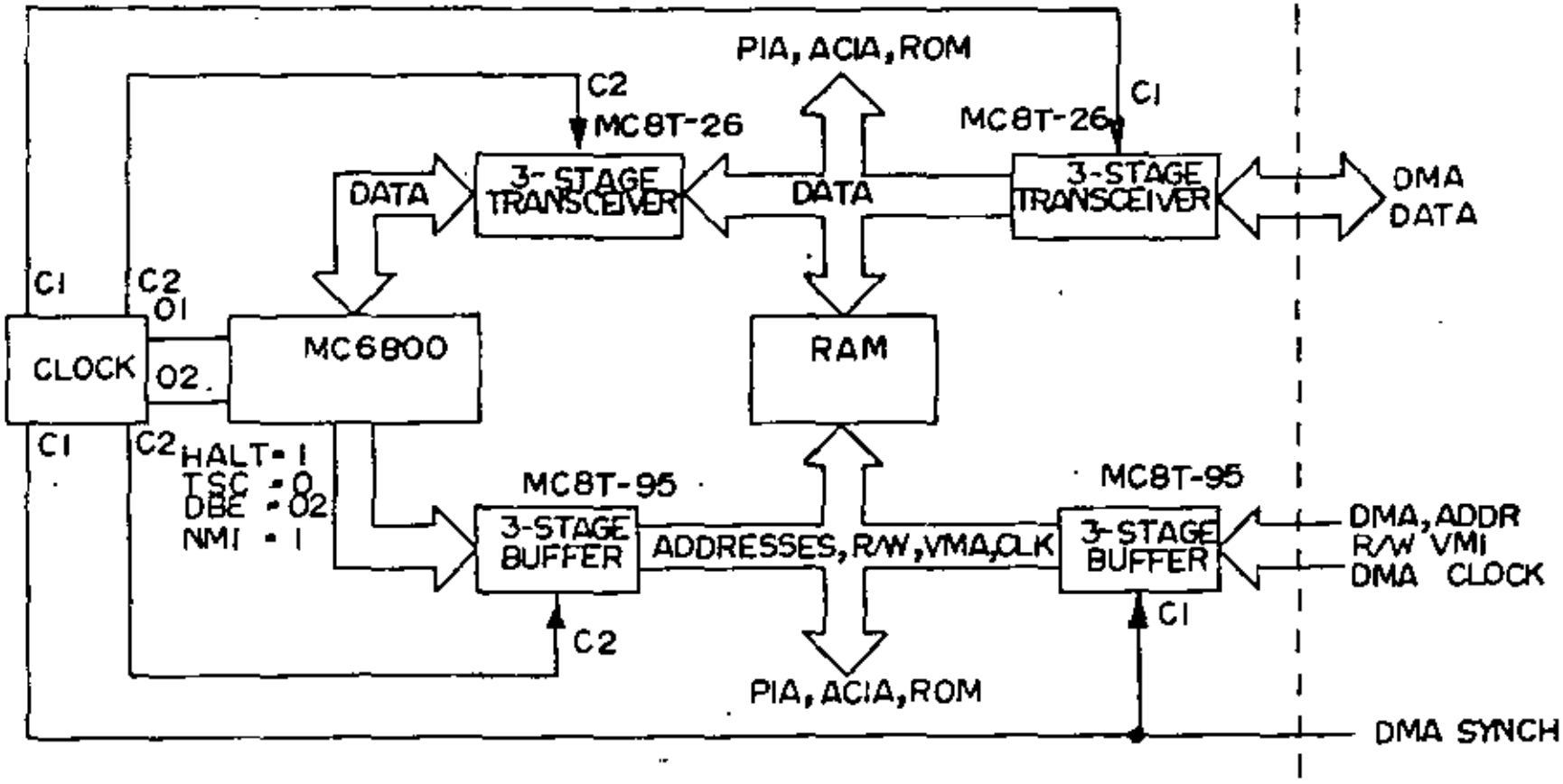


Fig. 3-4

DMA TECHNIQUES

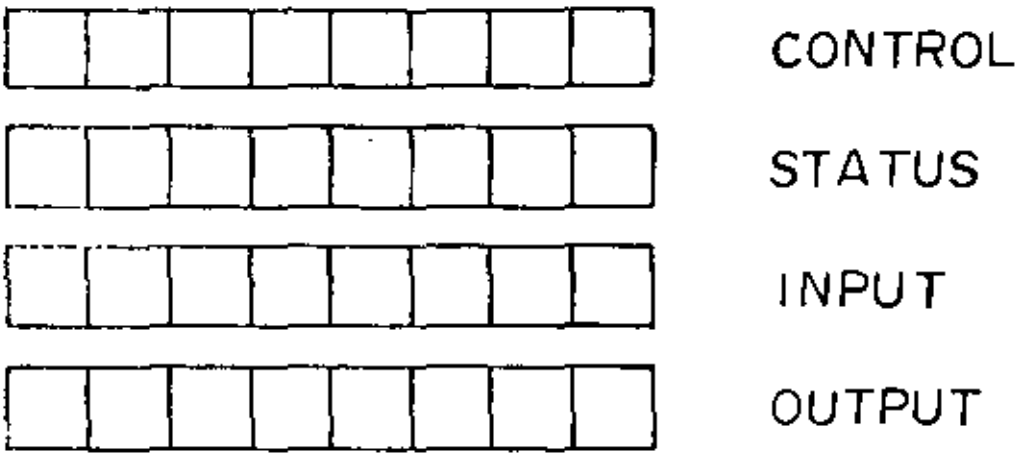
Technique	Maximum Data Rate	MPU program Execution	Hardware Complexity
Halt Processor	1 byte/microsec.	0	Lowest
Cycle Steal	1 byte/2.5 microsec.	1cycle/5 microsec.	Medium
Multiplexed DMA	1 byte/1.2 microsec.	1cycle/1.2 microsec.	Highest
Software (PIA)	1 byte/14 microsec.	Dedicated to Aquisition	Lowest

Notes:

These examples hold for the 6800 processor and are described in Motorola's Applications Manual. The DMA rate for the Halt case is limited by memory speed.

CHAPTER 4
PROGRAMMABLE MICROPROCESSOR SYSTEM CHIPS - INPUT/OUTPUT

When microprocessors first became available, the input and output circuits has to be deisgned using SSI logic. Not only was this task time consuming, but the number of chips in the system increased a great deal. Motorola, and later



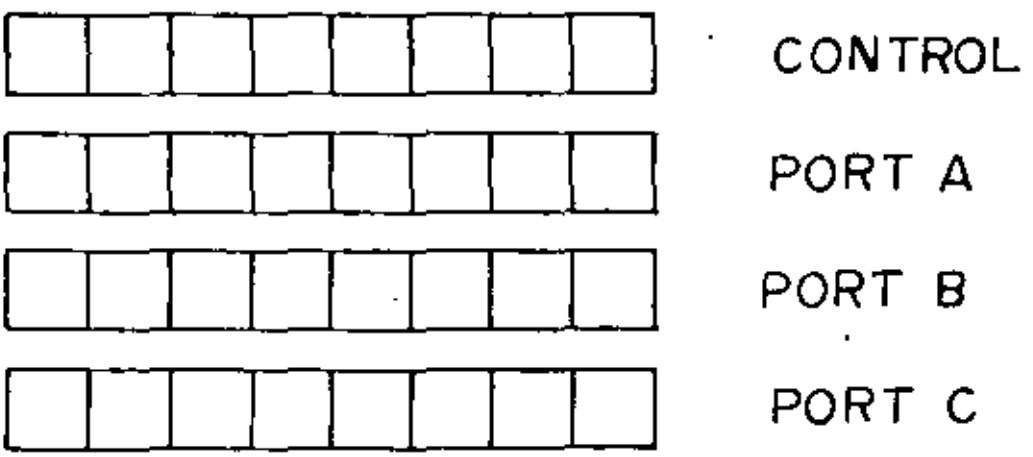


Fig. 4-1

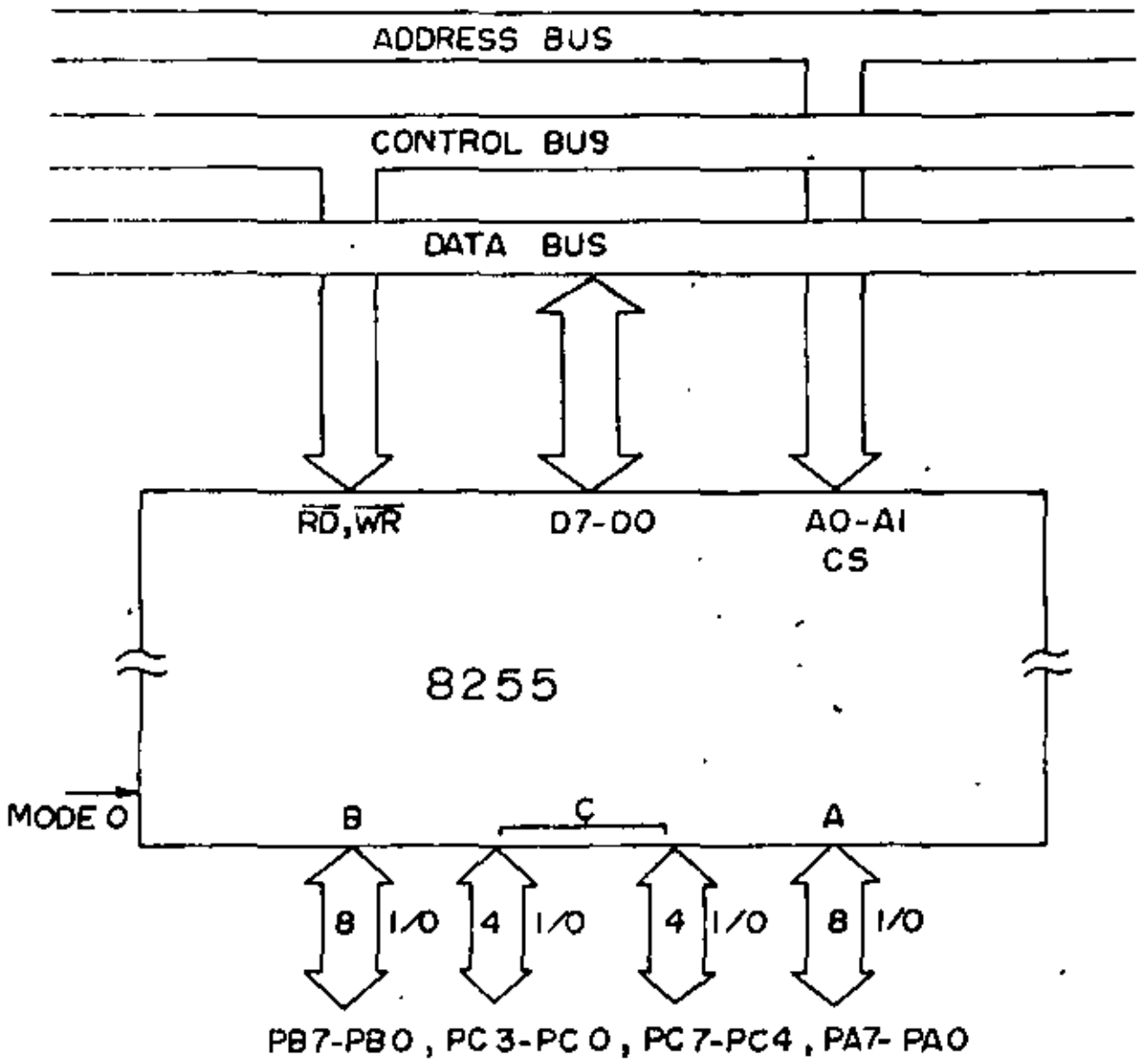

ADDRESS BUS
CONTROL BUS
DATA BUS
RD, WR
D7-D0
A0-A1
CS
8255
MODE 0
B
C
A
8 I/O
4 I/O
4 I/O
8 I/O
PB7-PB0 , PC3-PC0 , PC7-PC4 , PA7- PA0

Fig. 4-2

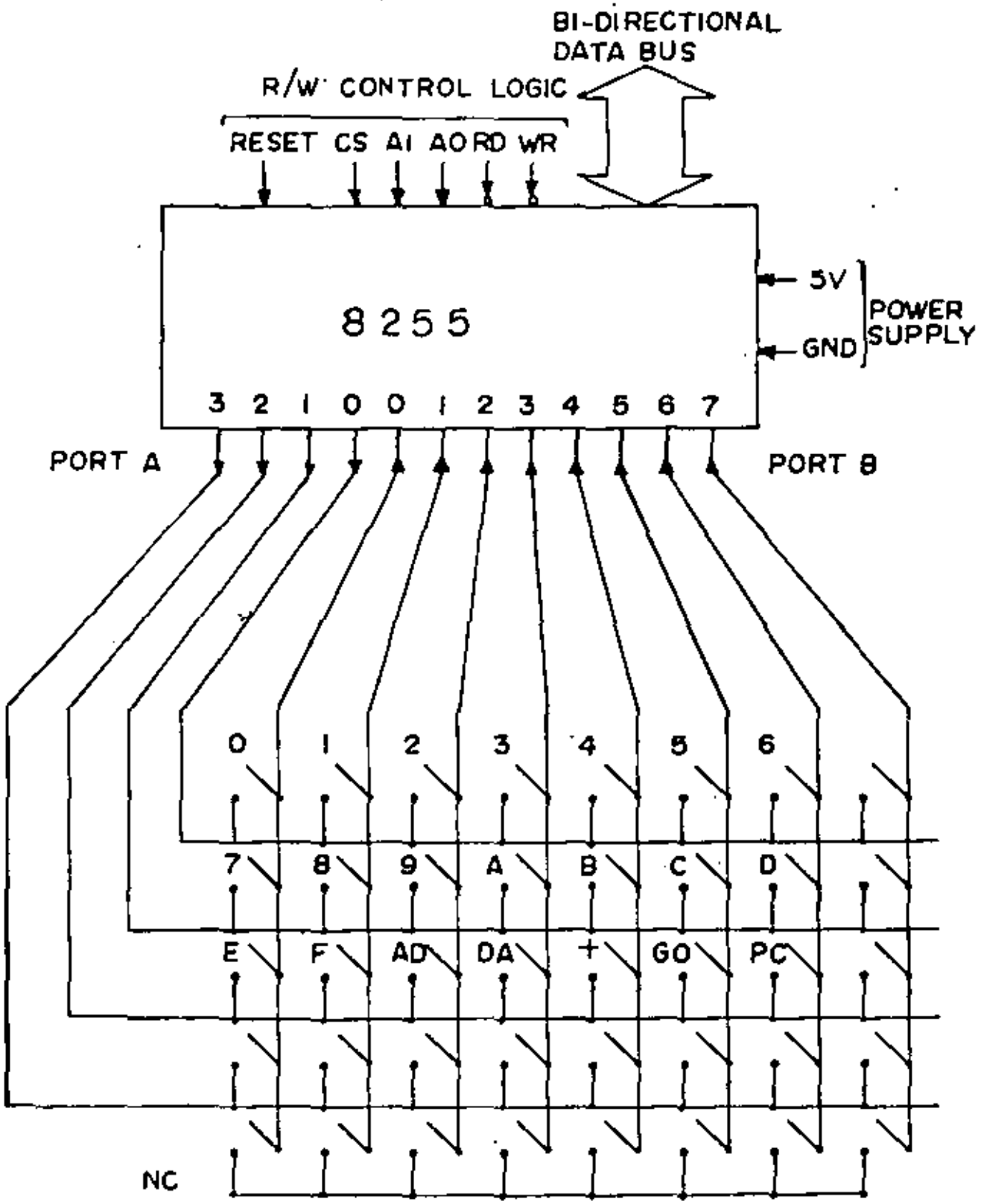

BI-DIRECTIONAL
DATA BUS
R/W CONTROL LOGIC
RESET CS A1 A0 RD WR
8255
5V
GND
POWER
SUPPLY
3 2 1 0 0 1 2 3 4 5 6 7
PORT A
PORT B
0 1 2 3 4 5 6
7 8 9 A B C D
E F AD DA + GO PC
NC

Fig. 4-4

Latched Inputs and Handshaking -Another Mode

In some applications, it is necessary to latch input data since the data may be valid only for a short time. Usually some timing signal or "strobe" is available to indicate when the data are valid. The processor may be occupied with some task which cannot be interrupted or the data valid dstate may be too short for the processor to capture the data. The 8255 may be used in mode 1 as shown in Fig. 4-6 for these cases. (If the processor is not otherwise occupied and if the data valid state exists for at least several microseconds, the processor can check the state of the strobe input and capture the data easily in mode 0 of the previous section.) In the latched input case of Fig. 4-6, the strobe signal loads the data into the input latch. Subsequently the 8255 issue the input buffer full (IBF) signal. If the interrupt enable is set using the bit set of pin 4 of port C, an interrupt request will be generated on pin 3 of port C. The strobe and the IBF constitute the handshaking; the IBF signal is useful for indicating to the inputting device that the data has not been read by the processor. When the contents of port A are read, the IBF signal is reset. Port A and port B may both be operated in this mode. Another use of Mode 1 is strobed output. Here data are placed at the output of A or B and an output buffer full (OBF) signal is issued by the 8255. When the output device connected to the 8255 accepts the data, it should issue an acknowledge signal that can in turn generate an interrupt request. (Interrupts are discussed later in this chapter.)

The 8255 can be operated in still another mode which can provide synchronous input or output. Details can be found in the 8080 Microcomputer Systems User's Manual. In summary, the 8255 provides 24 input or output lines and can

be operated in a number of modes.

There are a number of other chips from Intel and other manufacturers that resemble the 8255. With the new 8085, an 8155 has been made available which has many of the 8255 features: two 8-bit A and B ports, 6-bit C port, basic and strobed input/output. In addition, it has a 14 bit binary down counter, and 256 bytes of RAM, and may be interfaced with the 8080.

Data Direction Registers

On some parallel I/O chips, individual bits may be designated as either an input or an output bit. One example is the Motorola PIA (6820). Here the chip may be divided into two parts, A and B. Each side or part is very similar as shown in Fig. 4-7. (The A output bits are standard TTL compatible, but the B output bits may source up to 1 milliampere, adequate for the base drive of a transistor. The Intel 8255 outputs also provide the 1ma. drive.) Although there are six internal registers, only four are addressable at any time. The data direction registers specify which bits are outputs or inputs. A 1 in a bit of the data direction register defines the corresponding external line as an output; a 0 defines an input. The control registers define and enable the interrupt states and with pins R0 and R1, the register select pins, select the I/O register or the data direction register. Specifically, it the 2nd bit of the control register is a zero, then (R0,R1) = (0,0) selects the data direction register. If the 2nd bit is one, (0,0) selects the I/O register. The control registers are always selectable. The register select is indicated in Fig. 4-8. While cumbersome, this scheme permits a pin that would ordinarily be used as register select, to be used for some other useful task. MOS Technology manufactures an excellent variation of the PIA: the 6530. It contains not only the

INTERNAL ADDRESSING OF THE PIA

RS1	RS0	Control Register Bit		Location Selected
		CRA-2	CRB-2	
0	0	1	x	Peripheral Register A
0	0	0	x	Data Direction Register A
0	1	x	x	Control Register A
1	0	x	1	Peripheral Register B
1	0	x	0	Data Direction Register B
1	1	x	x	Control Register B

RS0 and RS1 are the register select pins normally connected to address pins A0 and A1. CRA-2 and CRB-2 are bit 2 of the control registers.

ANALOG TO DIGITAL CONVERSION

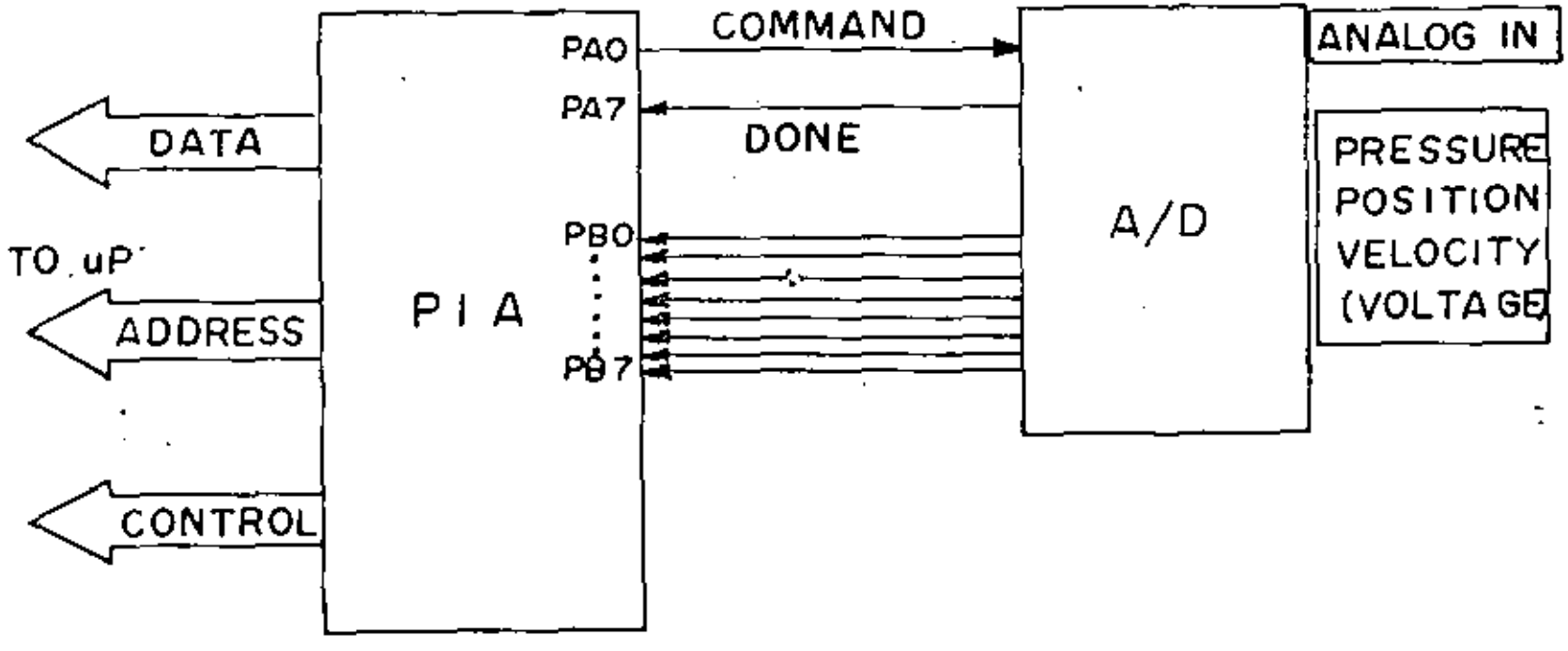


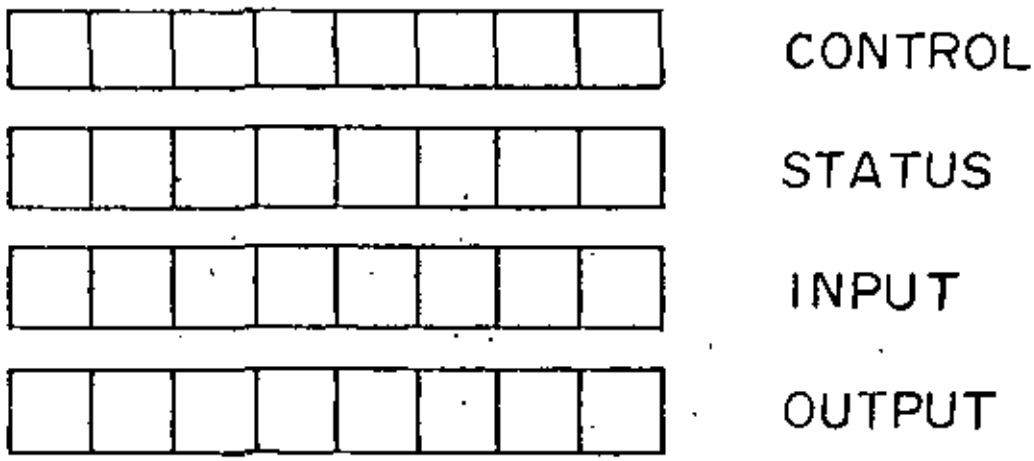
Fig. Fig. 4-8

DATA TRANSFER PROGRAM

This program segment is intended to indicate the maximum rate at which a 6800 may acquire data. Here, we are given a control register (CNTRL) which indicated that status: if bit 7 is high, the data word (DTWR) is ready to be read. The index register has been preloaded with the number of words to be acquired andi is decremented until it is zero in which case the acquisition is done.

```
LOOP:   LDAA    CNTRL     ;load control word        4 cycles
        BPL     LOOP      ;data not ready           4
        LDAA    DTWD      ;get data into accum A.   4
        PSHA              ;store on stack           4
        DEX               ;Index reg. has  count    4
        BNE     LOOP      ;return for next word     4
```

Fig. 4-10



CONTROL REGISTER - WRITE ONLY

STATUS REGISTER - READ ONLY

I/O REGISTERS - DOUBLE BUFFERRED

Fig. 4-11

TRANSMIT DATA REG.
RS · R/W
WRITE ONLY

7	6	5	4	3	2	1	0
DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0

RECEIVE DATA REG.
RS · R/W
READ ONLY

7	6	5	4	3	2	1	0
DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0

CONTROL REG
RS · R/W
WRITE ONLY

7	6 5	4 3	2	1 0
INT. ENABLE	TRANSMIT CONTROL	# OF STOP BIT / BIT PARITY	WORD SELECT	COUNTER DEVIDE (÷1 ÷16 BAUD)

STATUS REG
RS · R/W
READ ONLY

7	6	5	4	3 2	1 0
INT. REQ.	PARITY ERROR	REC. OVERRUN	FRAME ERROR	MODEM STATUS	TRANS-MIT DATA REG

Fig. 4-12

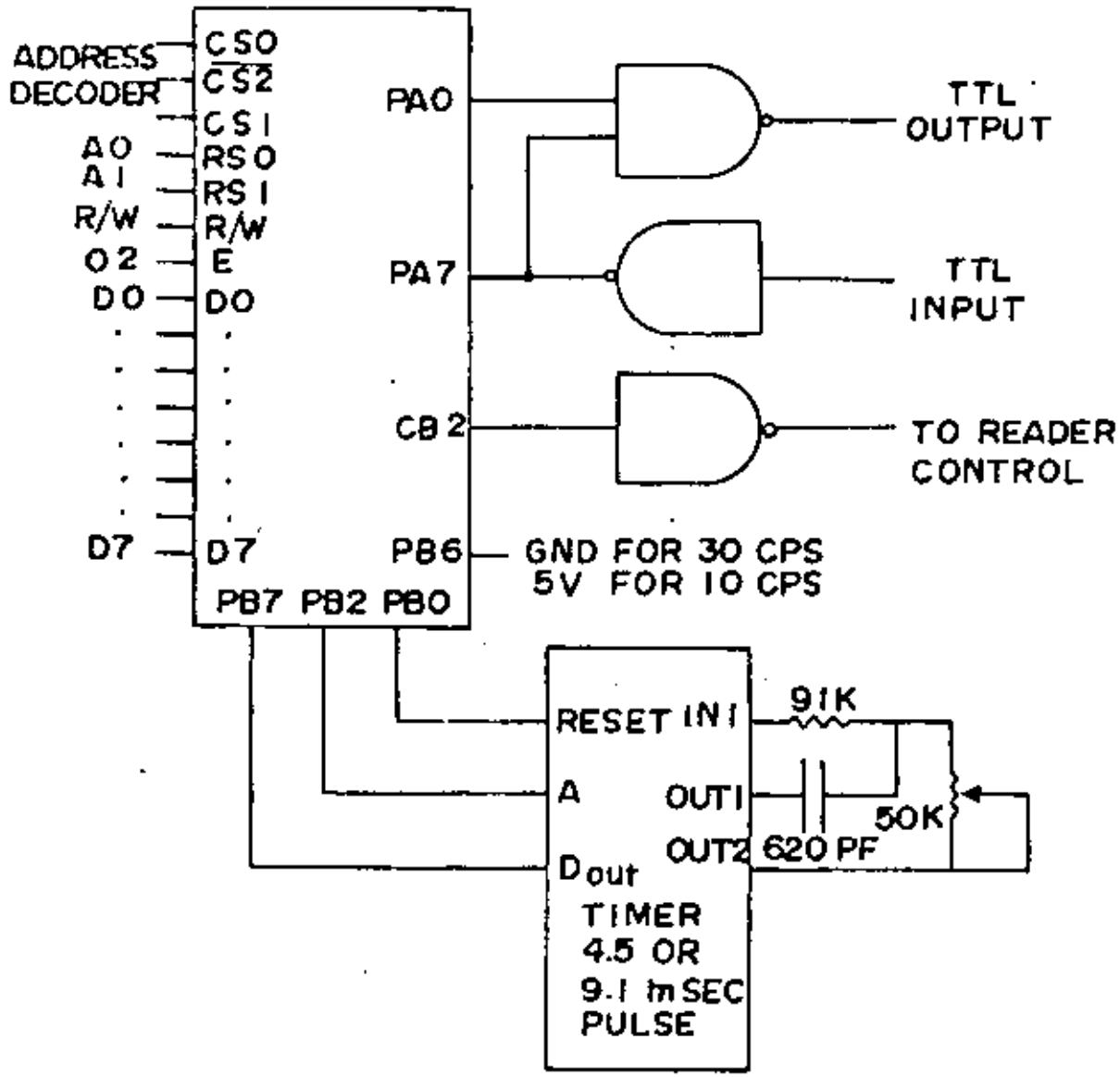

ADDRESS DECODER
CS0
CS2
CS1
AO
A1
RS0
RS1
R/W
R/W
O2
E
DO
DO
D7
D7
PA0
PA7
CB2
PB6
PB7 PB2 PB0
TTL OUTPUT
TTL INPUT
TO READER CONTROL
GND FOR 30 CPS
5V FOR 10 CPS
RESET IN1
A
Dout
OUT1
OUT2
91K
50K
620 PF
TIMER
4.5 OR
9.1 m SEC
PULSE

Fig. 4-14

similar), however the designer should consider the possibility of using several processors rather than coping with complicated multitask programming. Microprocessor are less powerful than minicomputers, the software presents more difficulties, and they cost much less. Of course, if the data or control rates are very slow and simple, a microprocessor can handle many tasks.

Typical Interrupt Procedures

Most interrupts are maskable, that is, the microprocessor's condition code register contains a flag bit which enables or disables (masks) interrupt requests. When the interrupt in not masked, and an interrupt request is made, the processor finishes its current instruction and inters the service routine. The flow chart of a 6800 is shown in Fig. 4-15. Note that there are two interrupt requests: a nonmaskable interrupt (one that cannot be disabled by the processor) and a regular interrupt request. We will discuss only the latter. When the processor finishes an instruction, it checks for a halt signal and then an interrupt request. If the interrupt request is asserted, it then checks the mask. If the mask is set, the processor continues with the next instruction. If the request is not masked, the processor saves all of the internal registers on the stack (excluding of course the contents of the stack pointer) and goes to locations FFF8 and FFF9 for the new program counter. If several devices are capable of interrupting the processor, the service program must POLL the devices to determine which device created the interrupt request. A "priority" of the requests may be established through the order in which the devices are checked. In the 6800 family, the I/O devices have a status register or status/control register which contains a flag bit that signals an interrupt request. Thus when an I/O device is programmed, interrupt enable bits in the control register may be set to permit interrupts by that I/O device.(Fig. 4-16). The Motorola scheme

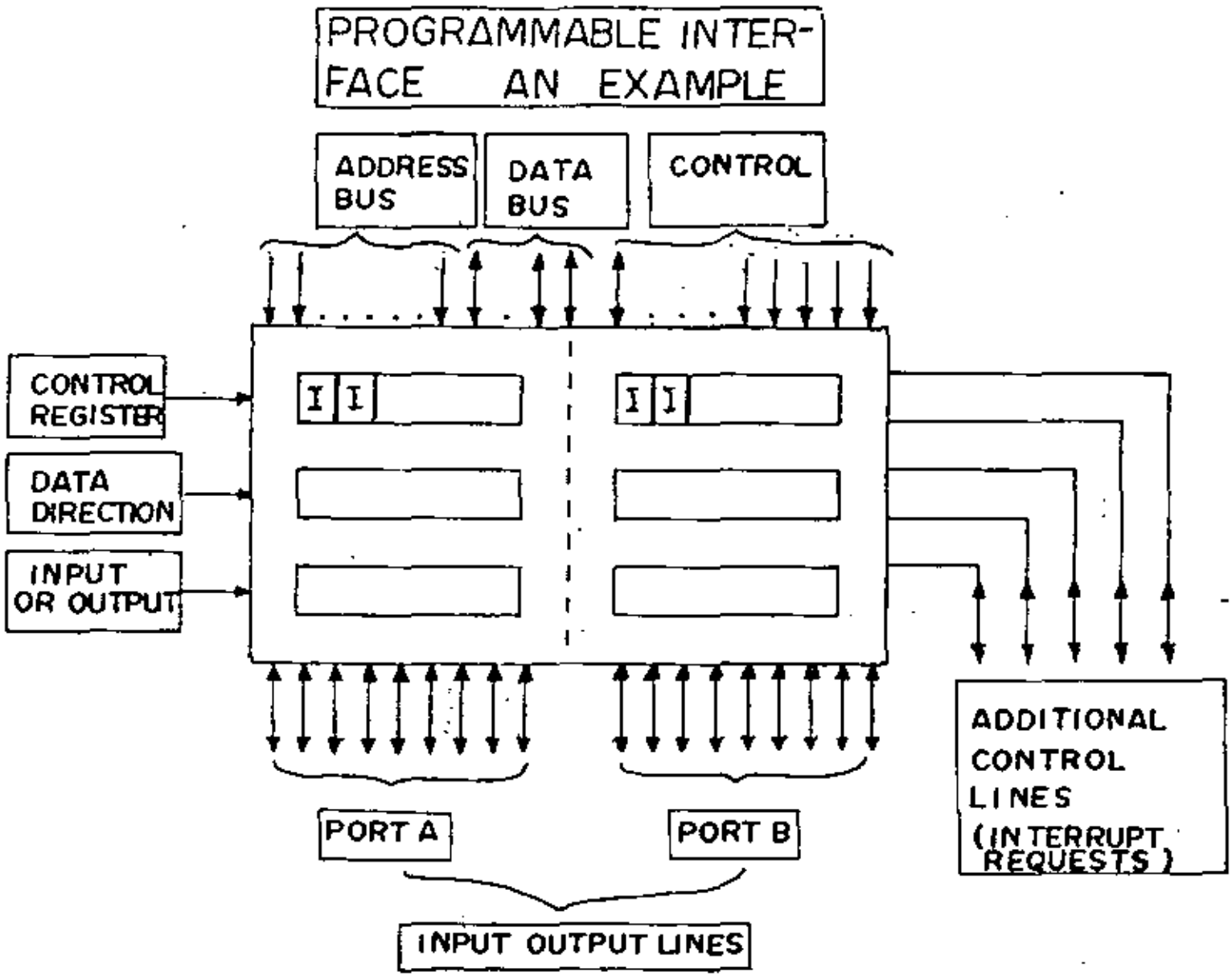

PROGRAMMABLE INTER-
FACE AN EXAMPLE
ADDRESS BUS
DATA BUS
CONTROL
CONTROL REGISTER
DATA DIRECTION
INPUT OR OUTPUT
I I
I I
ADDITIONAL CONTROL LINES (INTERRUPT REQUESTS)
PORT A
PORT B
INPUT OUTPUT LINES

Fig. 4-16

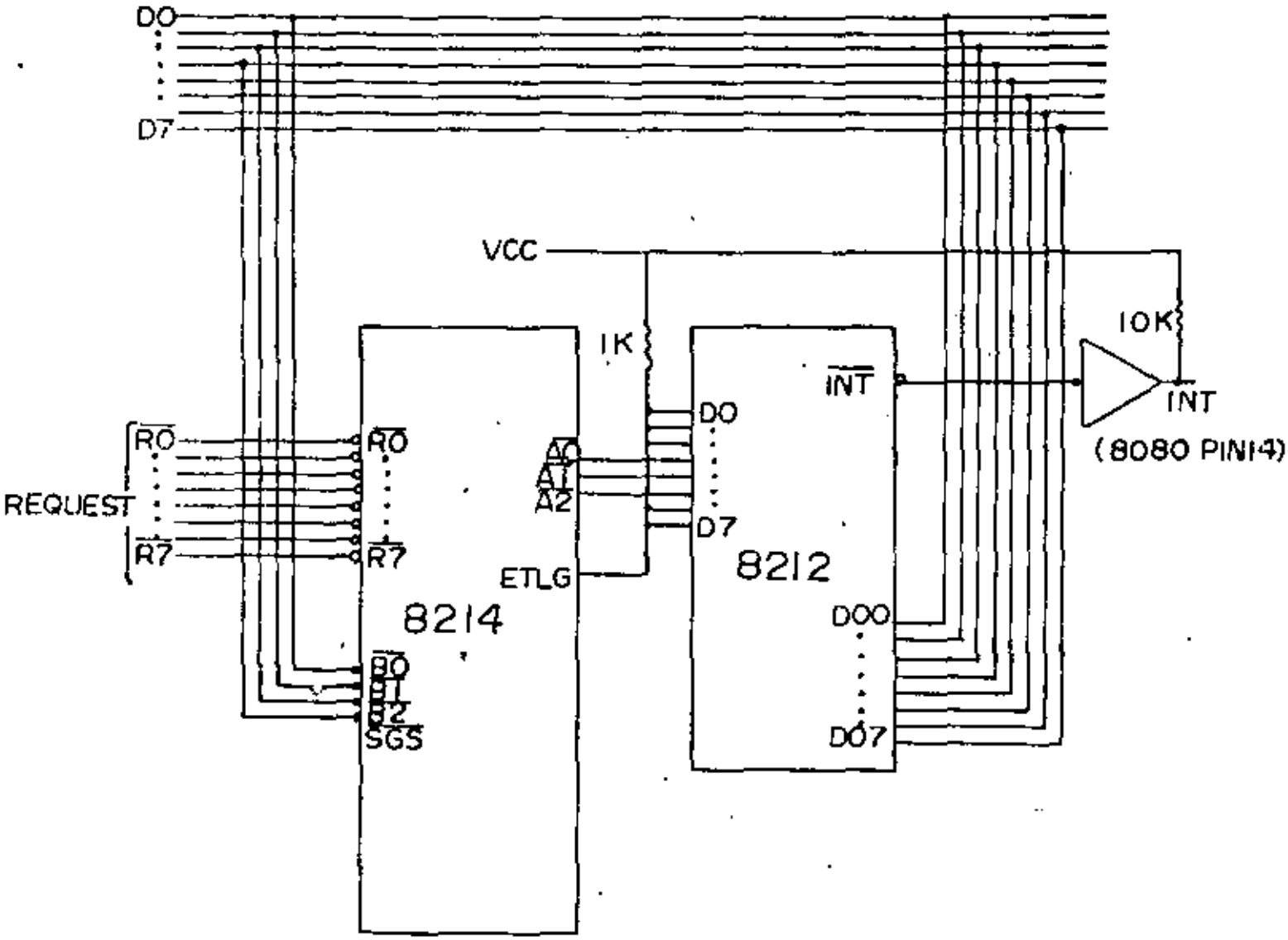

D0
D7
VCC
1K
10K
INT
INT
(8080 PIN14)
REQUEST
R0
R7
8214
8212
ETLG
A0
A1
A2
D0
D7
DO0
DO7
SGS

Fig. 4-16

sequential address locations for the input or output amd uses the processor HOLD feature discussed earlier. Priorities may be set for each of the channels or a rotating priority may be established. Finally, a "terminal count" or the number of bytes that are to be transferred in a single request may be programmed. A disadvantage of using this chip is that a maximum transfer rate of 1 byte/4usec. is possible. For many instrumentation tasks, this rate is much too slow. A hardwired DMA unit as discussed earlier, can handle data as fast as the memory access time will allow. However, in this latter case, that hardware complexity is much greater than the complexity of a single chip.

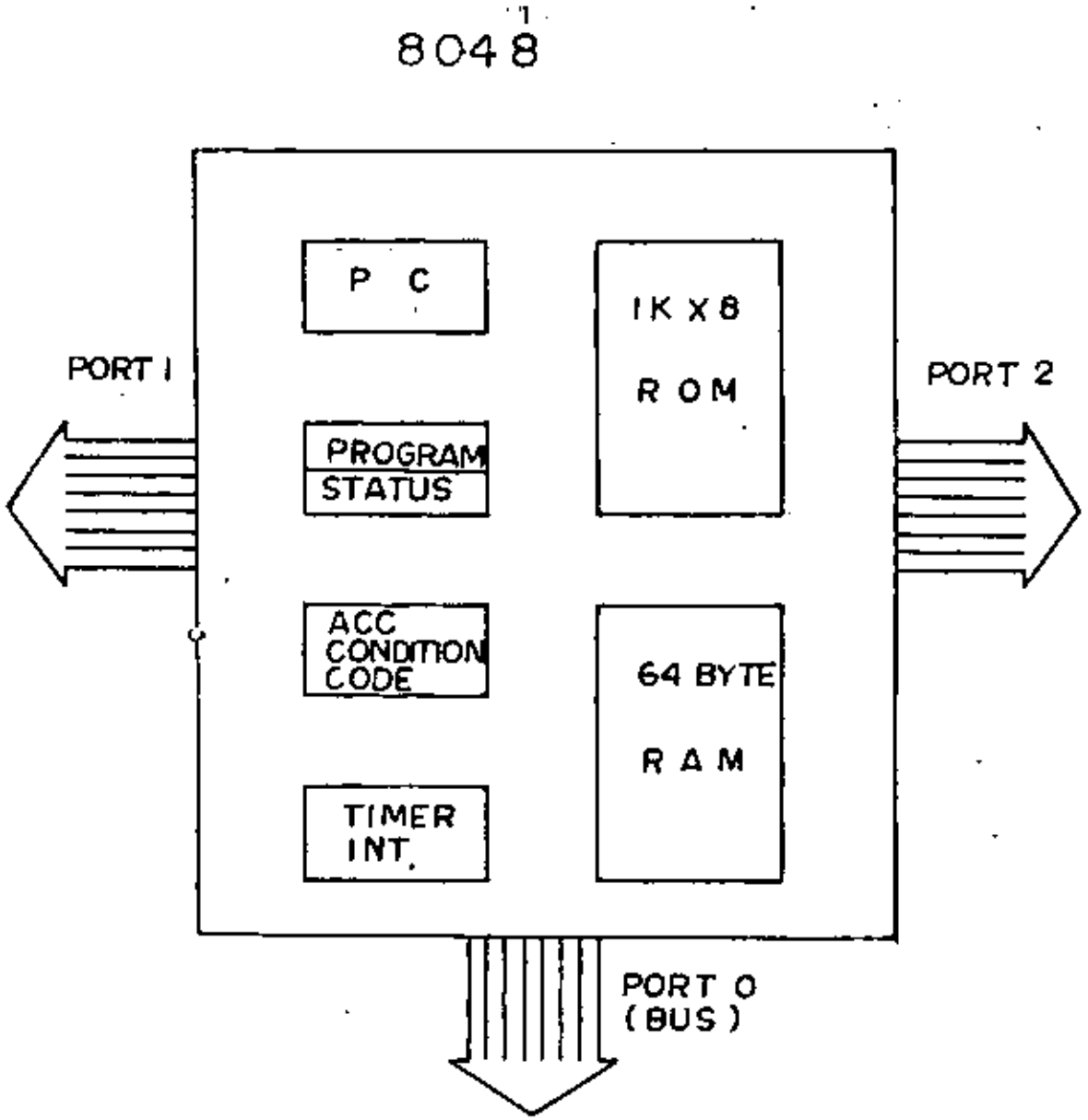

8048
P C
1K X 8
ROM
PORT 1
PORT 2
PROGRAM
STATUS
ACC
CONDITION
CODE
64 BYTE
RAM
TIMER
INT.
PORT 0
(BUS)

Fig. 5-1

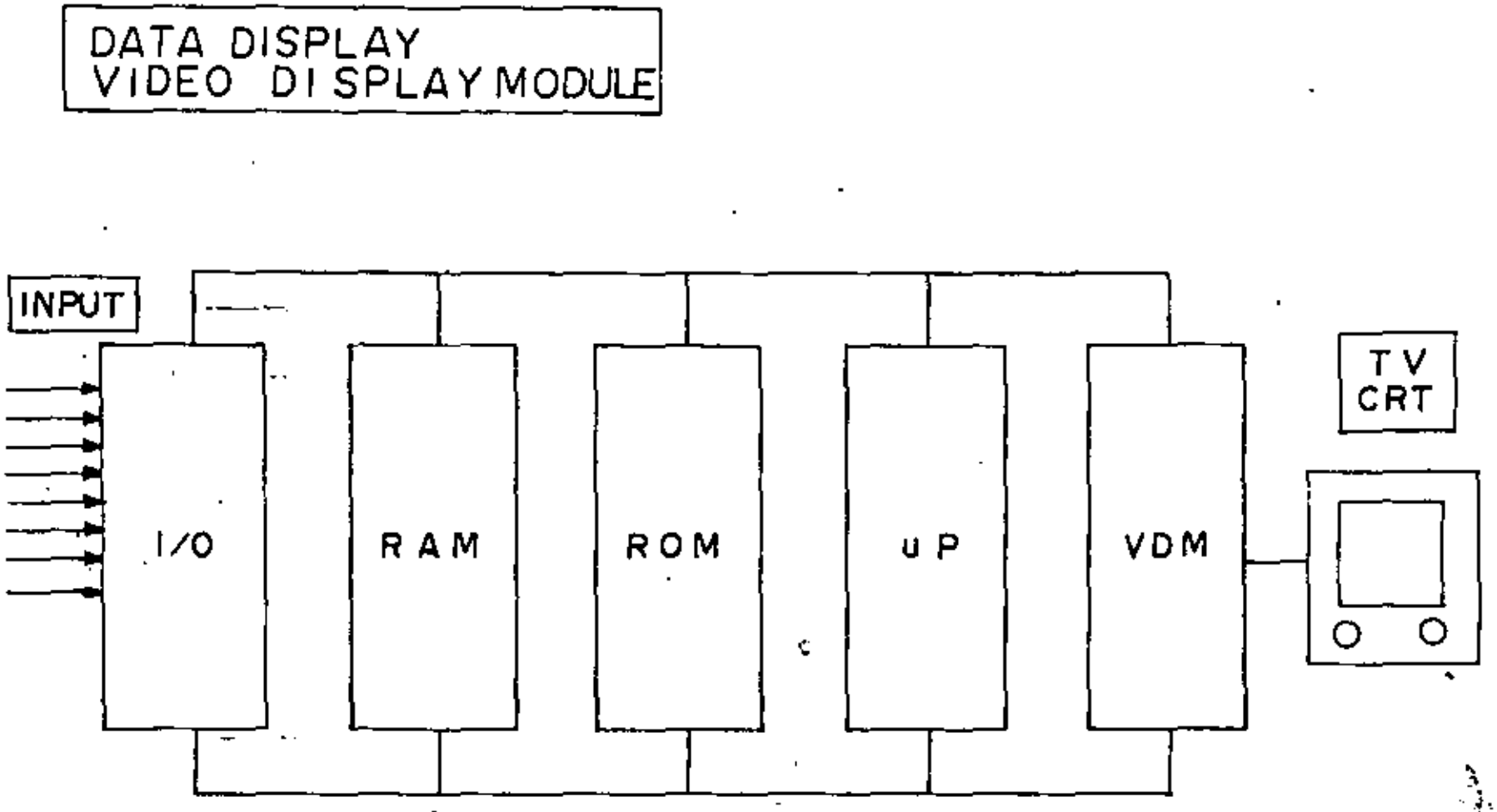

DATA DISPLAY
VIDEO DISPLAY MODULE
INPUT
I/O
RAM
ROM
uP
VDM
TV
CRT

Fig. 5-2

of the RAM. A keyboard may be interfaced to the 8048 using the 8279 keyboard/display or a standard, encoded keyboard may be connected to one of the quasi-bidirectional ports since the VDM already supplies the display. The serial I/O can be provided by using the other quasi-bidirectional port. To generate to proper baud and handle the serial I/O, the processor may use its internal counter for timing.

The 1Kbyte ROM and 64 bytes of RAM are adequate for the necessary program. The processor must spend most if its time monitoring the serial input and output and echoing that response on the VDM. When a key is struck, an interrupt will occur that results in a read of the keyboard encoding chip and transmission of the character via the serial output. Since the internal timer increments every 80 microsec., this scheme is satisfactory up to 600 baud. For higher baud, a serial interface chip such as the 6800 discussed in the previous chapter is required.

Intel's 8085

The 8085 was mentioned in Chp.3 as an extension or simplication of the 8080. In the 8080, status information must be placed on the bus and be captured using an additional chip, the 8228. The 8228 possesses some additional features that make it very useful, but if one is trying to use the minimum number of components, the 8085 system is better. The 8085 has all of the instructions of the 8080 and can be used in similar systems but is not bus compatible. One must use some logic chips to connect the 8085 to an 8080 bus.

The 8085 offers some simplication of system design, especially if the system does not require many priority interrupt request lines. For instance, the

a CMOS (complimentary MOS logic) processor, and thus has some unique features. First, the processor can be operated with a wide range of voltages: 3 - 15 volts. Second, the processor requires extremely low power- microwatts instead of hundreds of milliwatts for more conventional processors. Thus the processor and its associated chips can be operated in applications requiring battery power or very low power input much more successfully than, say, the 8080.

Other special features of the COSMAC include a static clock (no minimum clock frequency); simple input/output either memory mapped, on-chip DMA, or I/O lines on the chip; and 16 16-bit registers, any one of which may be the program counter, index register, or DMA pointer. The processor is thus I/O oriented. Each of these I/O features will be examined in the following.

The processor has four input flags which which may be tested and used for conditional branches and it has 3 output lines which may be used for simplified address decoding or device selection or for additional I/O lines. Finally, a one-bit I/O line is available for serial input/output. Some details are shown in Fig. 5-4.

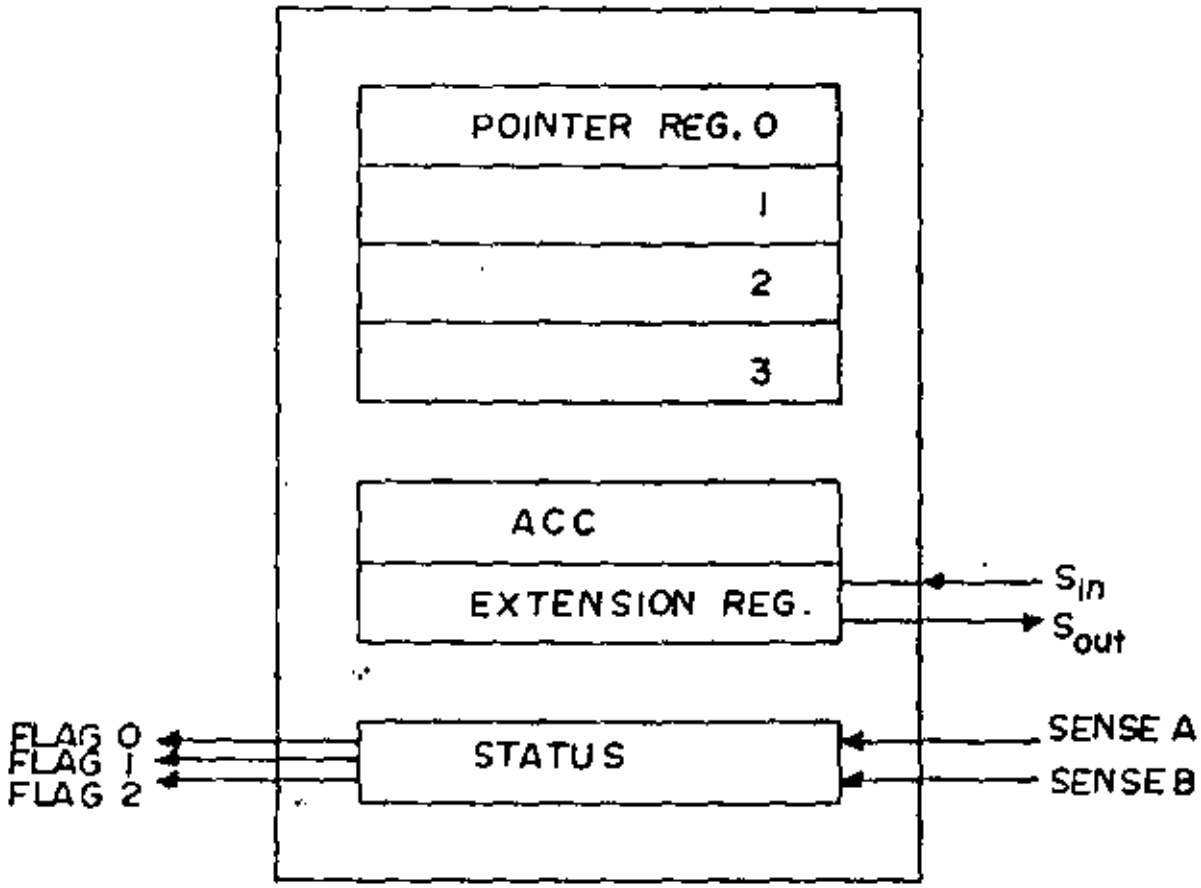
The processor has a DMA request input that essentially provides "cycle stealing" capabilities. If the request is made frequently enough or if it is continuously made, the processor will continue to provide DMA in or DMA out. This feature is very useful for loading memory with the desired program and then executing the program. RCA has made available a "Microtutor" that uses this scheme. The DMA feature is handled by Register 0; i.e. R0 points to the memory locations that are loaded through the DMA requests and is automatically stepped to the next memory location. On RESET, R0 also serves as the program counter, but any register may thereafter be designated the program counter.

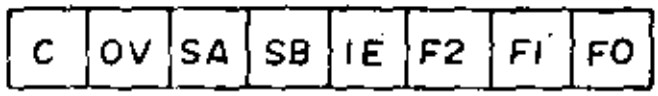
Any of the sixteen 16-bit register may be designated as an index register. Basic input and output is handled through an IN or an OUT instruction which uses an index register as a pointer to input or output a byte. The index register is automatically stepped in this procedure. Further a 3 bit code is specified on the output bits for device selection. The byte is placed on the data bus by the processor in an output operation, or the processor reads the byte on the input operation. With additional external logic, the I/O capabilities may be greatly expanded. RCA has also developed peripheral chips for this processor. The additional I/O lines are possible because only 8 pins are used for the address bits. The 16 address bits are time multiplexed; an additional bit is provided so that the first (most significant) address bits can be latched. This address multiplexing is common on many of the simpler processors.

The SC/MP: National's SC&MP

The original SC/MP was fabricated using a p-MOS technology, and thus it was quite slow: 32 micoroseconds per instruction. A new n-MOS version is now available and is correspondingly faster. The SC/MP has 12 address bits on the chip but can multiplex the additional four for a full 16-bit address using four bits on the data bus. A timing or latching signal is issued by the processor when the additional four bits are valid. The internal register organization is illustrated in Fig. 5-5. The program counter is hardwired as pointer register 0. However, the contents of the PC can be exchanged with the contents of any pointer register using a single instruction. This feature leads to an interesting way to handle subrountines: if the pointer registers are pointing to the beginning of a subrountine, then by exchanging contents, the processor executes the subroutine. The return from subroutine is accomplished by another exchange

SC/MP





STATUS REG.

Fig. 5-5b

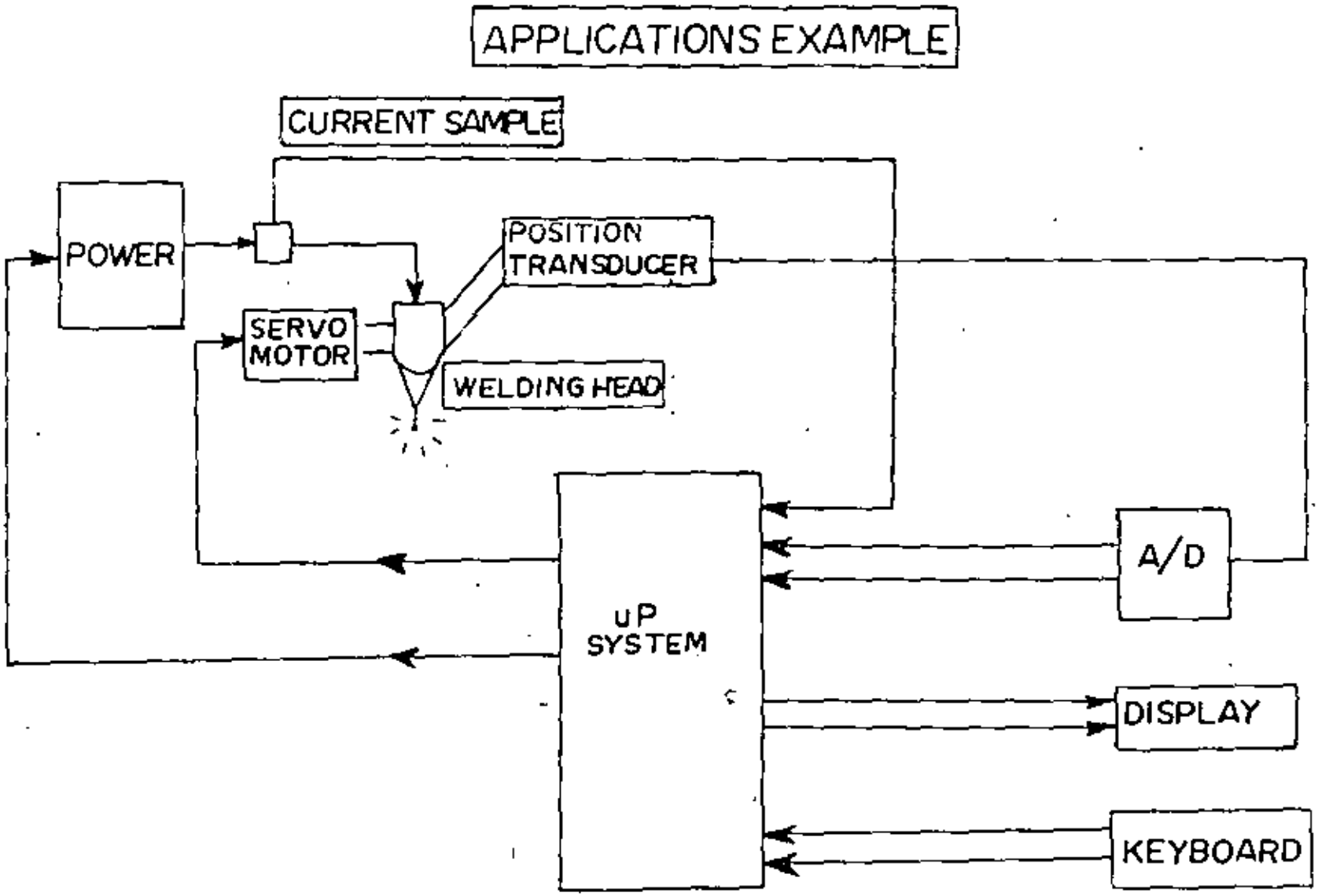

APPLICATIONS EXAMPLE
CURRENT SAMPLE
POWER
POSITION TRANSDUCER
SERVO MOTOR
WELDING HEAD
uP SYSTEM
A/D
DISPLAY
KEYBOARD

Fig. 5-6

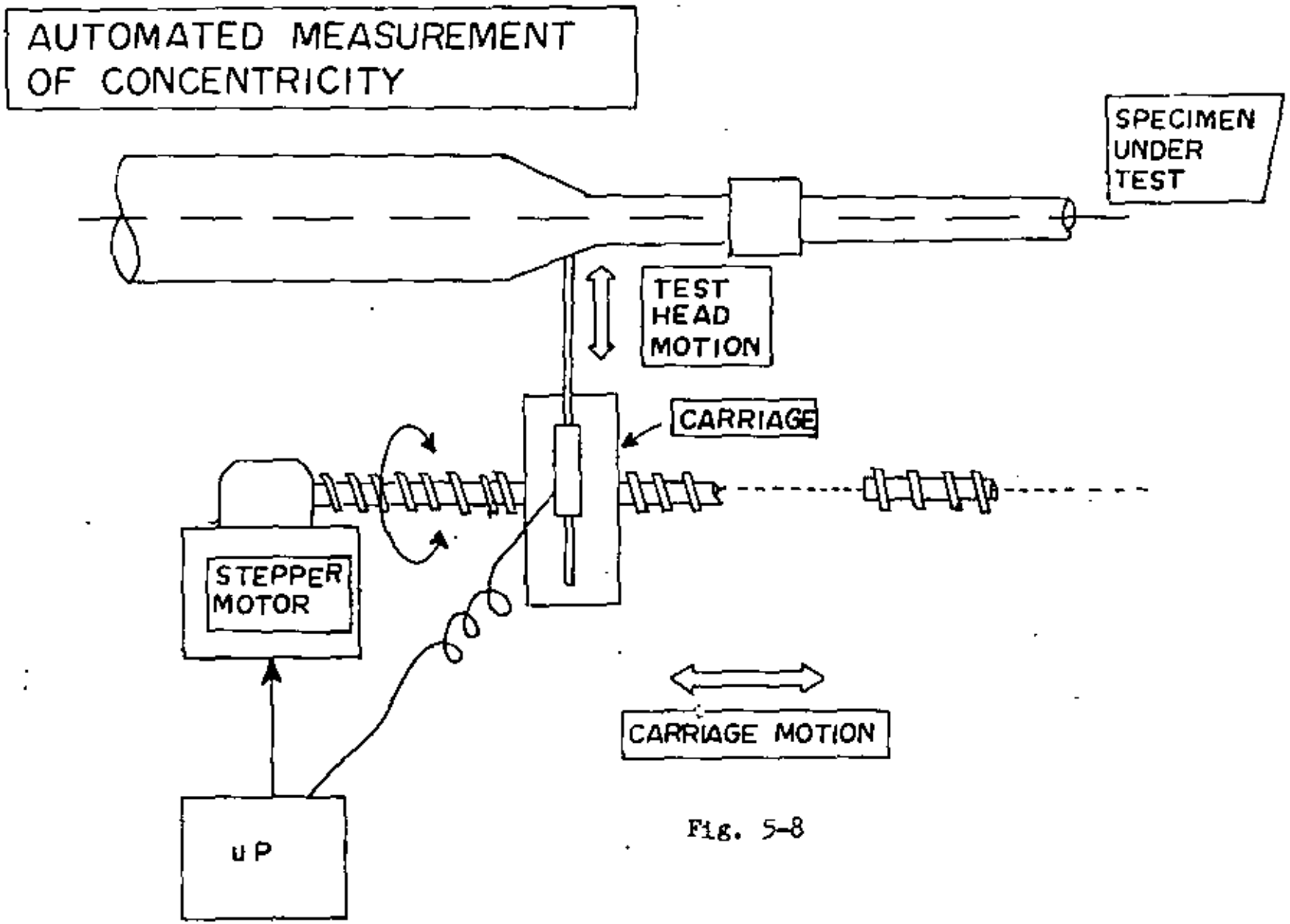

AUTOMATED MEASUREMENT
OF CONCENTRICITY
SPECIMEN
UNDER
TEST
TEST
HEAD
MOTION
CARRIAGE
STEPPER
MOTOR
CARRIAGE MOTION
uP

Fig. 5-8

Evaluation Systems

Microprocessor manufacturers have developed inexpensive evaluation kits or systems for users who wish to explore simple microprocessor operation. The Motorola Evaluation Kit II used in this tutorial is a good example. These systems interface with a terminal or with a keyboard on the system and allow the user to deposit, run, debug, and perhaps store his program on a cassette. The system is controlled by a program, the monitor, stored in a ROM. These systems can be expanded to include more memory and more I/O, however, it is impractical in most cases to add assemblers, disks, etc. These systems cost from $100 to $300, and are often available only as a kit. Other popular evaluation kits include the SDK-80 from Intel, and the KIM-1 from MOS Technology. While I have heard of a firm that used the SDK-80 in a product, I believe that evaulation kits are really suitable for learning about microprocessor and especially in educational programs.

Hobby Systems

This text is not directed toward the design of hobby systems such as the Altair by MITS or IMSAI's 8080 system. They are mentioned here because their costs are modest, and because these systems are occasionally used for inexpensive development of software. In contrast to the evaluation systems, the hobby systems can be easily expanded. Many have assemblers and debugging aids that greatly ease the programming process. Most of the systems can be interfaced to floppy disks where the software can be stored, edited, assembled, etc. High level languages have also been developed for these systems; BASIC in some form is the most popular.

operations as the MDS system. As in the Intel system, it accomodates a number of different systems, and can be expanded to additional memory, ROM programmers, disk interfaces etc. Both the Intel and the Motorola systems deal only with their respective processors. Recently, Tektronix has developed its system, the 8002 system, which can accomodate a number of different processors: 6800, 8080, Z-80,. and will be expanded to include four others. This system will be available at the tutorial and will be discussed below.

The 8002 system consists of a dual-floppy disk, a terminal, and the 8002 mainframe. The first two are standard components and need no elaboration. The mainframe contains the system processor, a Z-80, 16K bytes of system memory, an assembler processor, and an emulator processor. 16K bytes of RAM are available for the users program and the RAm may be expanded to 64K bytes. The assembler processor run the Tektronix assembler while the system processor handles all of the I/O to the disks. The emulator processor is a circuit board which uses the particular commercail processor of interest to the user. It runs the user program while the system processor program detects program errors and runtime errors.

As shown in 6-1, one can start with the software design and produce a source program which in turn may be edited by the 8002 editing program. The source program may be assembled via the assembler processor and tested using the emulator processor. In that testing, the emulator processor executes the user program from the emulator memory and also provides the I/O.

In the hardware design process, one may start with a breadboard or a prototype system. Using a special prototype probe, the hardware system may be checked using the emulator processor. The processor is removed from the

breadboard or prototype system and the prototype probe is inserted in its place. In this mode, the emulator processor runs the hardware while the system processor performs debugging and run time checks. Breakpoints may be inserted, new instruction added or substituted, and timing check in loops or from one part of the user program to another. The user program may be run from the emulator processor, from the prototype system, or both. A special mapping command exists to select which part of the memory and I/O space resides in the emulator memory and which part in the prototype board. A ROM programmer is also available for programming the hardware system's memory.

Finally, a logic analyzer facility is available for system testing. Software debugging is successful only when the hardware is working properly. To check hardware, the analyzer displays the states of the address bus, the data bus, the control bus, and eight other user selected points. The last 100 such states are stored and displayed as a result of a pre-trigger, a post-trigger, or a variable center trigger.

This development system was described in some detail because it offers most of the design aids available in a single system. The design aids are not necessarily unique however; Intel offers an ICE system which is very similar to the emulator processor and the prototype probe. The high initial cost for this system -about $18,000 - is justified if designs using several different processors are necessary. The same system instructions and usage prevails over all of the processors, and is easier than using development systems from different manufacturers.

Time Sharing Systems

costly to optimize in assembly language. Normally, the speed with which a program will execute depends upon a very few portions of the program: a few loops perhaps. If the compiler can be linked to an assembly language program, as most can, these few portions can be optimized and the rest of the program written using assembly language.

Intel offers their compiler as a resident program in their Intellec system (MDS) and claims that this offers considerable savings over timesharing services. Intel also states that the realiability of programs written in PL/M have better software reliability.

Macros and other aids

Programming still seems to be a barrier in the use of computer systems for many people. While high level languages help, they are not always the answer. Macros represent at least a partial solution. A macro is essentially a string of assembly language instructions which represents a single operation or task. An example might be a microprocessor controlled pin ball machine[9] in which the designers (who are not computer experts) wish to program a variety of games. The manufacturer of the processors (if the quantity is very promising) or some other organization might write a series of Macros, each of which performs some task: check the tilt, count the points, check for bonus points, etc. The designer of the game could then simple combine the macros and a few conditional branches to form a specific game. The macros can be named in obvious ways. Essentially, then, macros can be used by non-expert programmers to deal with programmable systems such as microprocessor based systems.

---

9. Mentioned at an IEEE Microprocessor Workshop by a National Semiconductor speaker

APPENDIX A THE 8080 MICROPROCESSOR

The 8080 is best appreciated and understood if it is remembered that it is the first of the "2nd generation" processors. Its predessesors were the 4004 and the 8008, quite limited processors by comparison. The programming model of the 8080 is shown in Fig. A-1. The program counter, the stack pointer, and the accumulator are quite standard registers. The condition code register has the usual bits plus a parity bit that is set when the modulo 2 sum of the bits in the result of the operation is zero. I have never used this feature, and thus I cannot give an example of the application of the parity bit. The register pair, H & L, is frequently used to point to a memory location, but it is not an index register. Registers D & E and B & C, as pairs, also can point to memory and used in some instructions. The registers may also be used for scratch pad purposes.

The 8080 has a number of addressing modes: absolute (exact address), register indirect (location pointed to by register pair), register (op-code specifies one of the internal registers or register pairs), and immediate(operand follows op-code). All conditional branches, called conditional jumps in the 8080, use absolute addressing. The register pair H L is more frequently used for memory-accumulator transfers or ALU operations. The condition codes are not modified by incrementing or decrementing register pairs and there is no operation to compare H L to determine if a transfer a data is complete. This is the principal shortcoming of the instruction set.

The 8080's clock cycles are more complex than most other processors'. Each

The complete processor is shown in Fig. A-3. The 8085 offers considerable ease in design, since it does not need any special clock or status latches. It is almost certainly the processor of choice for most future systems if one wishes to stay in the 8080/8085 software (a considerable investment).

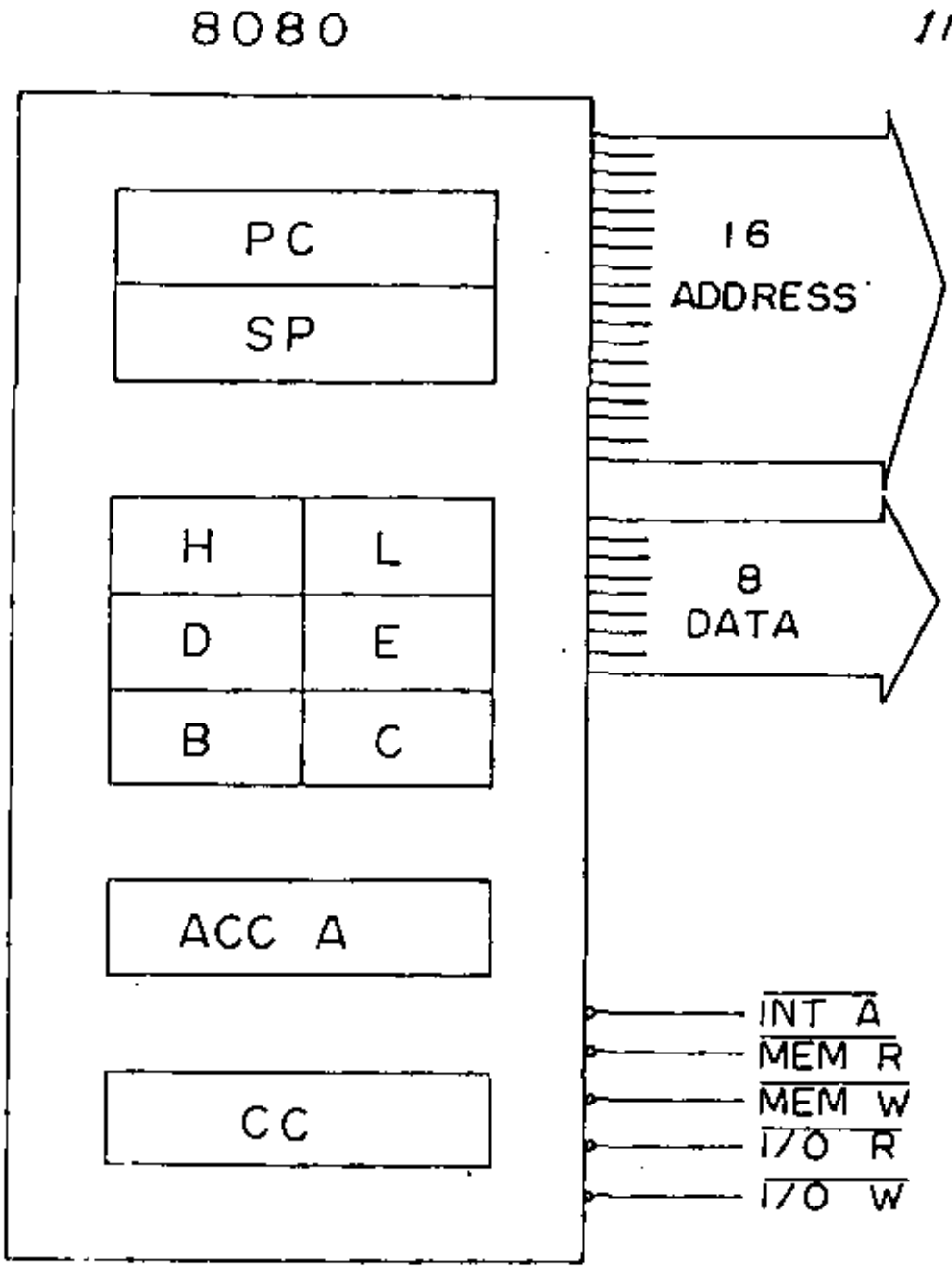


Fig. A-1

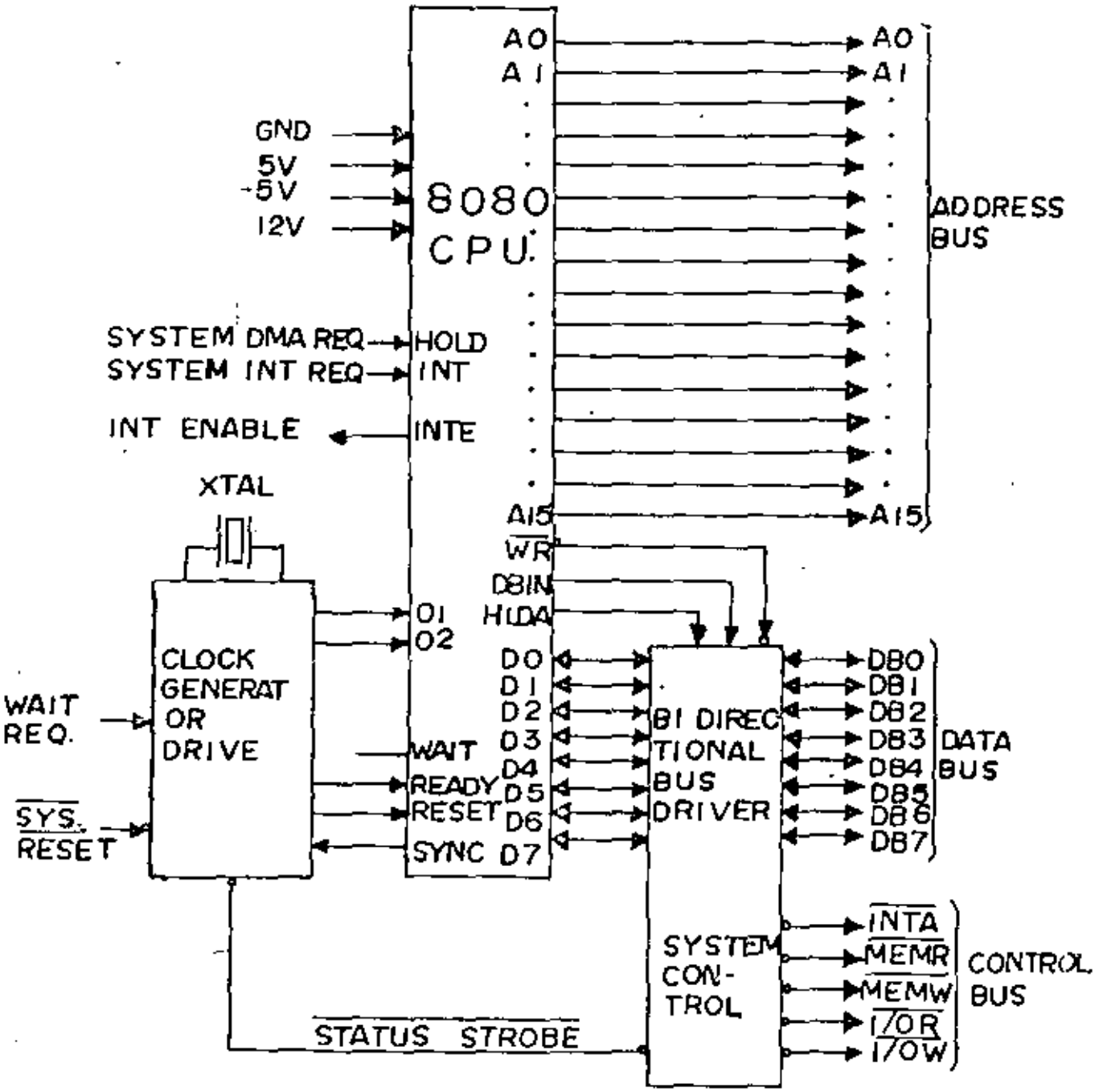

A0
A1
A15
GND
5V
-5V
12V
8080
CPU
ADDRESS
BUS
SYSTEM DMA REQ
HOLD
SYSTEM INT REQ
INT
INT ENABLE
INTE
XTAL
WR
DBIN
HLDA
Ø1
Ø2
CLOCK
GENERAT
OR
DRIVE
WAIT
REQ.
SYS.
RESET
WAIT
READY
RESET
SYNC
D0
D1
D2
D3
D4
D5
D6
D7
BI DIREC
TIONAL
BUS
DRIVER
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DATA
BUS
SYSTEM
CON-
TROL
INTA
MEMR
MEMW
I/OR
I/OW
CONTROL
BUS
STATUS STROBE

Fig. A-3

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

MODOS DE DIRECCIONAMIENTO

MARZO, 1981

IV.- MODOS DE DIRECCIONAMIENTO

1.- ESQUEMAS DE DIRECCIONAMIENTO.

La unidad central de proceso (CPU) en las computadoras debe realizar las siguientes funciones:

- Obtener y traer de memoria primaria al CPU la siguiente instrucción a ejecutar.
- Entender los operandos, esto es, definir la localización de los operandos necesarios para ejecutar la instrucción y traerlos al CPU.
- Ejecutar la instrucción.

Para llevar a cabo las funciones anteriores el CPU debe contar con la siguiente información:

- El código de operación de la instrucción a ejecutar.
- Las direcciones de los operandos y la del resultado.
- La dirección de la siguiente instrucción a ejecutar.

Existen diferentes soluciones que satisfacen los requerimientos anteriores, los cuales determinan la arquitectura de los procesadores que las utilizan.

Se supondrán operaciones aritméticas en las que se tienen dos operandos y un resultado ya que son las que proporcionan el caso más general.

a) Máquinas de "3+1" direcciones

El formato de instrucción en este esquema de direcciona-- miento contiene todos los elementos necesitados por el CPU

donde T1 y T2 representan localidades temporales usadas para guardar resultados aritméticos intermedios.

Las conclusiones más importantes en este esquema son:
Los programas no necesitan estar almacenados en memoria en forma secuencial ya que el campo de dirección de la siguiente instrucción permite conocer donde fueron almacenados.

Debido a que cada instrucción contiene en forma explícita tres direcciones, no es necesario tener en el CPU hardware para guardar los resultados de las operaciones.

b) Máquinas de "3" direcciones

Considerando que los programas se escriben secuencialmente y que por consiguiente es muy lógico almacenarlos en este mismo orden, se llega a un nuevo esquema de direccionamiento en el cual se sustituyen todos los campos de dirección de la siguiente instrucción por un solo registro dentro del procesador que lleva en forma secuencial y automáticamente la dirección de la siguiente instrucción a ejecutar. Un posible formato de instrucción se muestra en la fig. IV.2 .

Dirección de la sig. inst.	Registro en el procesador	Código de operac.	Dirección primer operando	Dirección segundo operando	Dirección resultado	Palabra n de memoria

FIG. IV.2

En este esquema se usará la dirección del segundo operando como la dirección del resultado una vez que la operación se haya efectuado, por lo que el segundo operando será destruído. Así pues la expresión A=(B*C)-(D*E) en FORTRAN, quedaría:

```
MUL   B, C
MUL   D, E
SUB   E, C
ADD   C, A
```

La eliminación del campo de dirección del resultado permite reducir la longitud de la palabra de memoria y los costos de la misma, lo que permite usar este esquema en máquinas medianas y chicas.

d) Máquinas de "1" dirección

Este esquema de direccionamiento permite eliminar de todas las instrucciones el campo de dirección de uno de los operando y sustituirlo por un registro dentro del procesador, el cual contendrá a uno de los operandos. A este registro se le conoce como acumulador.- El formato de instrucción para la máquina de 1 dirección se muestra en la figura IV.4

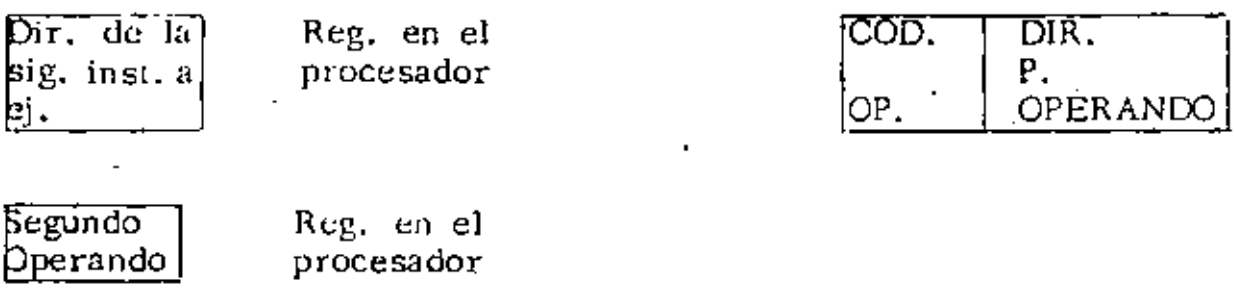


FIG. IV.4

memoria accesadas usando una disciplina UEPS (últimas entradas, primeras salidas). De lo anterior se concluye que en cada momento se tendrá disponible el elemento que se encuentre en el tope del stack.

El formato de instrucción para este esquema de direccionamiento se encuentra en la figura IV.5

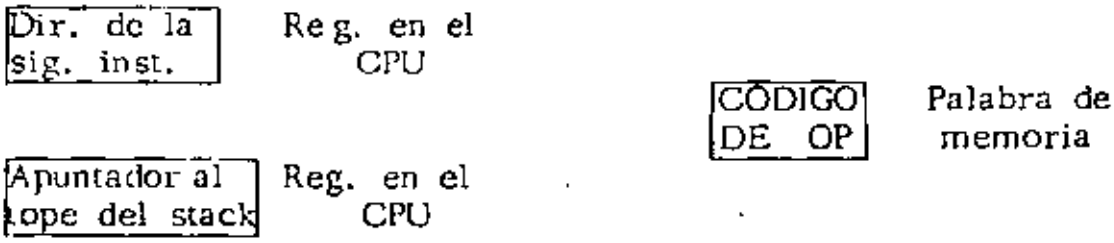
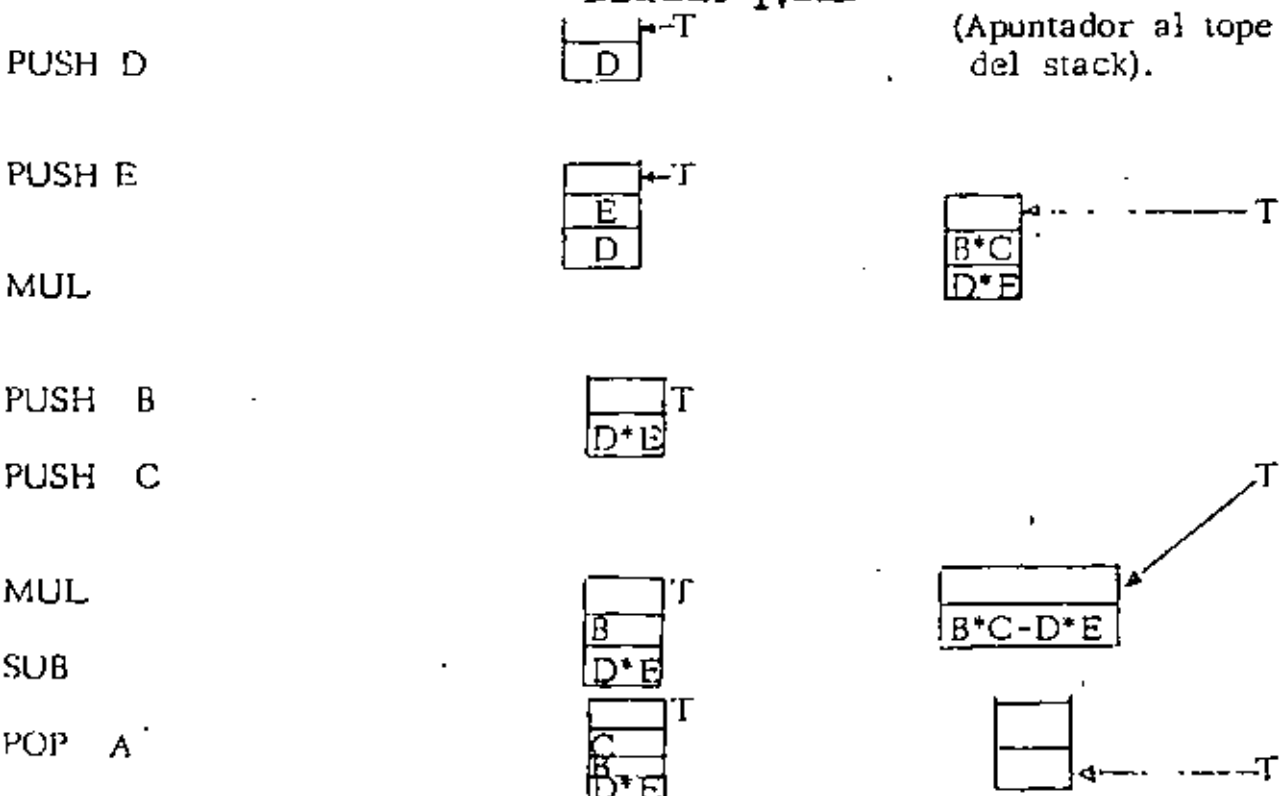


FIG. IV.5

Es necesario contar con instrucciones que permitan meter elementos de memoria al stack (PUSH) y sacar elementos del stack à memoria (POP).

La expresión A=(B*C)-(D*E) en FORTRAN, podría expresarse como:

FIG. IV.6



2.- METODOS DE DIRECCIONAMIENTO

En las máquinas de una sola dirección el formato de las instrucciones que hace referencia a memoria consta de dos campos: el campo de código de operación y el campo de dirección del operando. Si suponemos que el campo de dirección consta de n bits, entonces la máxima capacidad de memoria direccionable será 2^n localidades. Lo anterior puede resultar bastante drástico en el caso de las minicomputadoras ya que por lo general tienen palabras de 12 ó 16 bits y si se asignan cuatro de ellos al campo de código de operación solo se pueden direccionar $2^8 = 256$ localidades de memoria en el caso de palabras de 12 bits ó $2^{12} = 4096$ localidades de memoria en el caso de palabras de 16 bits, lo cual resulta insuficiente para la gran mayoría de las aplicaciones.

Lo anterior ha ocasionado diferentes modos de direccionamiento, en los cuales el campo de dirección sirve para calcular la dirección efectiva del operando, logrando una mayor capacidad de memoria direccionable.

a) Inmediato

En este caso el operando puede estar contenido directamente en el campo de dirección ó en la localidad de memoria siguiente a la instrucción.

Será necesario dedicar un bit de la palabra para saber como se debe interpretar la instrucción.

página actual.

La dirección del operando se determina sumando los bits de orden superior del PC al campo de dirección de la instrucción.

b.3) Relativo al PC

En este modo de direccionamiento el contenido del campo de dirección de la instrucción, interpretado como un entero con signo, se suma al PC para obtener la dirección del operando.

b.4) Relativo a un registro índice

El contenido del campo de dirección de la instrucción, interpretado como un entero con signo, se suma al contenido de un registro índice para obtener la dirección del operando. En caso de existir más de un registro índice es preciso asignar los bits necesarios para su identificación.

c) Indirecto

En el direccionamiento indirecto el campo de dirección de la instrucción contiene un apuntador a la dirección del operando ó este campo combinado con algún registro ó palabra de memoria genera un apuntador a la dirección del operando.

Mediante un bit en la instrucción se puede saber si el direccionamiento usado es directo ó indirecto.

3.- D I R E C C I O N A M I E N T O E N Z - 80

El microprocesador Z-80 es una máquina de una dirección en la que los diferentes modos de direccionamiento son usados por grupos de instrucciones y no se aplican de una forma general a todo el conjunto de instrucciones.

a) Implícito

En este modo de direccionamiento el operando no se define en forma explícita ya que el formato de instrucción es fijo y en los códigos de operación se especifica implícitamente sobre que registros del procesador actúan las instrucciones, por lo que el usuario no puede alterarlo de ninguna manera.
Los grupos de instrucciones, que utilizan este modo de direccionamiento son: carga de 8 bits; carga de 16 bits; intercambio, transferencia de bloques y búsqueda; aritméticas de propósito general y control del CPU.

Ejemplos 1.

b) Inmediato

El operando se encuentra en la localidad de memoria siguiente a la instrucción y se considera que forma parte de la misma. Los valores de los operandos inmediatos en ningun caso podrán exceder la capacidad de representación de un byte. Este modo de direccionamiento se utiliza cuando se desean realizar operaciones con valores constantes.

Los grupos de instrucciones que utilizan este modo de direccionamiento son: carga de 8 bits; aritméticas y lógicas de 8 bits y entrada/salida.

Ejemplos 2.

f) Extendido

La dirección del operando está contenida dentro del campo de operando de la instrucción. El campo de dirección tiene una longitud de 16 bits por lo que la máxima capacidad de memoria direccionable es de 64 K bytes.

Este modo de direccionamiento es utilizado por los grupos de instrucciones de carga de 8 bits; carga de 16 bits; saltos, llamadas y regreso de subrutinas.

Ejemplos 6.

g) Modificado de página cero

En este modo de direccionamiento el campo de dirección del operando se refiere a una localidad de memoria dentro de la página cero. Este campo de dirección consta de 3 bits y para su correcta interpretación se multiplica por 08H, obteniéndose de esta forma la referencia a las localidades deseadas.

Este modo de direccionamiento se utiliza exclusivamente por la instrucción RST.

Ejemplos 7.

h) Relativo

La dirección del operando se determina sumando al contador del programa el contenido del byte siguiente al código de operación de la instrucción.

El desplazamiento anterior se interpretará como un número en complemento a dos, con lo que se logra un rango de direccionamiento de -126 a +129 localidades relativas al contador del programa.

Este modo de direccionamiento es usado por el grupo de instrucciones de salto, llamada y regreso de subrutinas.

Ejemplos 8.

E J E M P L O S

Se asumirá que todos los ejemplos siguientes utilizan el sistema de numeración hexadecimal.

Ejemplos 1.

```
                        ; MODOS DE DIRECCIONAMIENTO DEL MICROPROCESADOR Z-80
                        ; PROGPAMA CARGADO EN CASSETE CON EL NOMBPE DE -.
                        ; "CEC".
                        ;
                        ;
                        ; DIRECCIONAMIENTO IMPLICITO.
                        ;
0000 ED5F                       LD      A,P
                        ;
                        ; CARGA EN EL PEGISTRO A EL CONTENIDO DEL PEGISTRO
                        ; DE REFRESCAMIENTO P.
                        ;
                        ;
0002 2F                         CPL
                        ;
                        ; REALIZA EL COMPLEMENTO LOGICO DEL CONTENIDO DEL
                        ; ACUMULADOR Y LO DEJA EN EL MISMO REGISTRO.
                        ;
                        ;
0003 DD23                       INC     IX
                        ;
                        ; EL CONTENIDO DEL PEGISTRO DE INDICE IX SE IN--
                        ; CREMENTA EN UNO
                        ;
                        ;
```

Ejemplos 2.

```
                        ;
                        ; DIRECCIONAMIENTO INMEDIATO.
                        ;
0005 C634                       ADD     A,34H
                        ;
                        ; SUMA AL CONTENIDO DEL PEGISTRO ACUMULADOR A, EL
                        ; DATO 34H Y DEJA EL RESULTADO EN EL MISMO RE--
                        ; GISTRO.
                        ;
                        ;
0007 E610                       AND     10H
                        ;
                        ; REALIZA LA OPERACION LOGICA AND ENTRE EL CONTE--
                        ; NIDO DEL PEGISTPO A Y EL DATO 10H, DEJANDO EL -.
                        ; RESULTADO EN EL MISMO REGISTRO.
                        ;
                        ;
```

Ejemplos 5.

```
                ;
                ; DIRECCIONAMIENTO DE REGISTRO INDIRECTO
                ;
0014 0A                 LD      A,(BC)
                ;
                ; CARGA EL REGISTRO A CON EL CONTENIDO DE LA LO-
                ; CALIDAD DE MEMORIA APUNTADA POR EL REGISTRO PAR
                ; BC.
                ;
                ;
0015 34                 INC     (HL)
                ;
                ; INCREMENTA EN UNO EL CONTENIDO DE LA LOCALIDAD
                ; DE MEMORIA APUNTADA POR EL REGISTRO PAR HL.
                ;
                ;
0016 12                 LD      (DE),A
                ;
                ; DEPOSITA EL CONTENIDO DEL ACUMULADOR EN LA LOCA-
                ; LIDAD DE MEMORIA APUNTADA POR EL REGISTRO PAR DE.
                ;
                ;
```

Ejemplos 6.

```
                ;
                ; DIRECCIONAMIENTO EXTENDIDO
                ;
0017 3A2010             LD      A,(1020H)
                ;
                ; CARGA EL ACUMULADOR CON EL CONTENIDO DE LA LOCA-
                ; LIDAD DE MEMORIA 1020H.
                ;
                ;
001A FD220400           LD      (0004H),IY
                ;
                ; DEPOSITA EL CONTENIDO DEL REGISTRO DE INDICE EN
                ; LAS LOCALIDADES DE MEMORIA 0004H (BYTE BAJO) Y
                ; 0005H (BYTE ALTO).
                ;
                ;
```

Ejemplos 9.

```
                        ;
                        ; DIRECCIONAMIENTO  INDEXADO
                        ;
0023 FD364313                 LD      (IY+43H),13H
                        ;
                        ; EL  DESPLAZAMIENTO  43H  SE  SUMA  AL CONTENIDO DEL RE-
                        ; GISTRO  IY  PARA DETERMINAR  LA  DIRECCION  EFECTIVA A
                        ; DONDE  SE  DEPOSITARA  EL  DATO, 13H.
                        ;
                        ;
0027 DD8621                   ADD     A,(IX+21H)
                        ;
                        ; EL  DESPLAZAMIENTO  21H  SE  SUMA  AL  CONTENIDO  DEL
                        ; REGISTRO  IX  PARA  DETERMINAR  LA  DIRECCION  DEL  O-
                        ; PERANDO  QUE SERA  SUMADO  AL  REGISTRO  A.  EL  RESUL-
                        ; TADO  QUEDA  EN  EL  REGISTRO  A.
                        ;
                        ;
002A DD3407                   INC     (IX+07H)
                        ;
                        ; EL  DESPLAZAMIENTO  07H  SE  SUMA  AL  CONTENIDO  DEL
                        ; REGISTRO  IX  PARA  DETERMINAR  LA  DIRECCION  DE  LA
                        ; LOCALIDAD  DE  MEMORIA  CUYO  CONTENIDO  SE  INCREMEN-
                        ; TA  EN  UNO.
                        ;
                        ;
```

Ejemplos 10.

```
                        ;
                        ; DIRECCIONAMIENTO  DE  BIT.
                        ;
002D CBC7                     SET     0000H,A
                        ;
                        ; ENCIENDE  EL  BIT  0  DEL  REGISTRO  A.
                        ;
                        ;
002F CBAE                     RES     05H,(HL)
                        ;
                        ; APAGA  EL  BIT  5  DE  LA  LOCALIDAD  DE  MEMORIA  DI-
                        ; RECCIONADA  POR  EL  REGISTRO  HL.
```

ING. LUIS G. CORDERO BORBOA
AGOSTO-79

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

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

PROGRAMAS DE APLICACION

Marzo, 1981

Palacio de Minería Calle de Tacuba 5 primer piso México 1, D. F. Tel: 521-40-20 Apdo. Postal M-2285

Programa para contar frecuencia de una señal externa con CTC

Contar ondas en Canal 1 CLK/TRG por 1/10 seg. usando canal 3 con 4 interrupciones, cada una de $195*256/1.9968x10^6 = .025$ seg. Para probarlo, usar canal 0 salida ZC.

Programa Principal

```
2060   3E 05              LD A,05          Reloj/16, no interrup.
       D3 94              OUT (94),A       Canal 0
       3E 80              LD A,80          128*16 ciclos = 975 Hz
       D3 94              OUT (94),A
2068   ED 5E              IM 2
       3E 20              LD A,20          Byte más significativo
       ED 47              LD I,A           Registro I
       3E 88              LD A,88          Byte menos significativo
2070   D3 94              OUT (94),A       Se escribe en canal 0
       3E 45              LD A,45          Reloj/256, permitir int.
       D3 97              OUT (97),A       Canal 3
       3E C3              LD A,C3          Constante de tiempo=195
       D3 97              OUT (97),A
       3E 55              LD A,55          Modo contador, no int.
       D3 95              OUT (95),A       Canal 1
       AF                 XOR A            Borrar A
       D3 95              OUT (95),A       Constante de tiempo
2081   32 [illegible] 23  LD (DISBUF+5),A  Cero en [illegible]
       D9                 EXX              Cargar registros altern.
       0E 00              LD C,00          Cuenta inicial
       06 04              LD B,04          Cuatro interrupciones
       D9                 EXX
       FB                 EI               Permitir interrupciones
       C3 94 00           JP DISUP         Monitor despliegue
208E   90 20                               Vector de int. canal 3
```

Programa de servicio de interrupciones

```
2090   08                 EX AF,AF'
       D9                 EXX
       05                 DEC B            Contar interrupciones
       20 41              JR NZ RET
       06 04              LD B, 04
       DB 95              IN A,(95)        Cuenta nueva
       5F                 LD E,A           Guardar en E
       57                 LD D,A           Contar con D
       AF                 XOR A            Borrar A y DISBUF
       32 [illegible] 23  LD (DISBUF+2),A
       32 [illegible] 23  LD (DISBUF+3),A
20A2   32 [illegible] 23  LD (DISBUF+4),A
       18 [illegible]     JR CONT
       3A [illegible] 23  NEXT LD A, (DISBUF+4)
       3C                 INC A
       FE 0A              CP A             Diez o mayor?
       38 [illegible]     JR C             No - continuar
       AF                 XOR A            Si - borrar A
20B0   32 [illegible] 23  LD (DISBUF+4),A
```

Programa para Contar Interrupciones

Interrupciones: pulsos hacia arriba en ASTRB. Despliegue numero 1-99 en los LEDs.

```
2200   ED 5E                 IM 2
       3E D8                 LDA D8          Byte menos significativo
       D3 82                 OUT(82),A         del vector de inter.
       3E 4F                 LDA, 4F         Modo entradas
       D3 82                 OUT(82),A       Lado A control
       3E 87                 LDA, 87         Se permiten interrupciones
       D3 82                 OUT(82),A
       3E 07                 LDA, 07         Byte mas significativo
2210   ED 47                 LDI ,A            en registro I
       FB                    EI              CPU acepta interrupciones
       C3 8F 00              JP RESTAR       Retorno al monitor

07D8   D6 23                                 Vector en ROM

23D6   C3 20 22              JP 2220

2220   08                    EX AF,AF'       Intercambiar registros
       D9                    EXX
       3A DC 23              LDA,(DISMEM+5)  LED Buffer No.6
       E6 0F                 AND 0F
       3C                    INC A
       FE 0A                 CP A            Diez o mas?
       38 0B                 JR C, CONT      No
       AF                    XOR A           Si - borrar A
       32 DC 23              LD(DISMEM+5),A
2230   3A DB 23              LDA,(DISMEM+4)  Incrementar 10's
       E6 0F                 AND 0F
       3C                    INC A
       32 DB 23              LD(DISMEM+4),A
       18 03                 JR 03
       32 DC 23              LD(DISMEM+5),A
       21 00 80              LD HL,8000      Esperar desconexion
2241   2D           ESP      DEC L
       20 FD                 JR NZ ESP
       25                    DEC H
       20 FA                 JR NZ ESP
       08                    EX AF,AF'       Reponer registros
       D9                    EXX
       FB                    EI              Permitir interrupciones
       ED 4D                 RETI            Retorno
```

Programa de Reloj Digital

Reloj del sistema = 1.2288 MHz
1.0068 MHz /256 = 2800 Hz: contando 200 ciclos por interrupción y 20 interrupciones por segundo para incrementar segundos.

Programa Principal

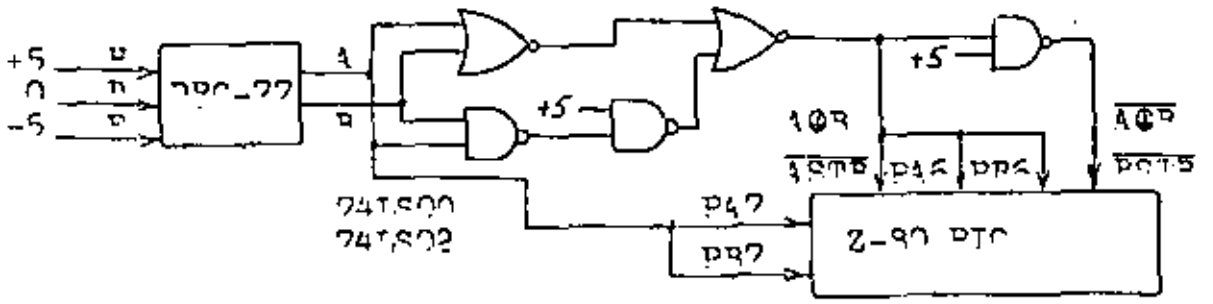
```
2000    MD 5F          TMOD 2          Modo de interrupción 2
        20 20.         THA,20          Byte más significativo
        MD 4D          LDT,A           Registro T
        2D 14          THA, 14         Byte menos significativo
        BZ 84          OUTM (84),A     Canal 0 CMD
        2D A5          THA,15          Permitir int., reloj/256
        BZ 85          OUTM (85),A     Canal 1 CMD
        2D C8          THA,C8          Constante de tiempo = 200
2010    BZ 85          OUTM (85),A
        B9             EXX             Intercambiar registros
        06 27          LD B,20         Número de interrupciones
        B0             EXX
        FB             EI              Se permiten interrupciones
        CZ 9C 00       JP DESPLA       Monitor despliegue
2014    2020                           Vector de interrupción
```

Programa de servicio de interrupciones

```
2010                                   Segundos en 00
2011                                   Minutos en 00
2012                                   Horas en 00

2020    08             EX AF,AF'       Intercambiar registros
        B9             EXX
        05             DEC B           Contar interrupciones
        20 50          JR NZ,SAL       Continuar
        06 27          LD B,27         20 interrupciones por seg.
        3A 10 20       LD A,(2010)     Segundos
        3C             INC A
        27             DAA             Ajuste decimal
        FE 60          CP 60           60 segundos?
        28 26          JR Z            NO
2030    3E             LD A,0          SI - borrar A
        32 10 20       LD (2010),A
        3A 11 20       LD A,(2011)     Minutos
        3C             INC A
        27             DAA
        FE 60          CP 60           60 minutos?
        28 14          JR Z            NO
        3E             LD A,0          SI - borrar A
        32 11 20       LD (2011),A
2041    3A 12 20       LD A,(2012)     Horas
        3C             INC A
        27             DAA
        FE 12          CP 12           12 horas?
        28 03          JR Z            NO
```

Programa para contar angulos usando convertidor incremental



Rutina de servicio de interrupciones

```
2000  08                  EX AF,AF'      Interrupción lado A
      DB 01               IN A,(01)      Leer lado B
      18 03               JR 03
2005  08                  EX AF,AF'      Interrupción lado B
      DB 00               IN A,(00)      Leer lado A
      D5                  PUSH DE        Guardar registros
      E6 C0               AND C0         Examinar bits 6,7
      57                  LD D,A         B = valor previo
      FA 1420             JP M,2014      Negativo?
      20 03               JR Z,2011      Cero?
2011  14                  INC D          Ajustar angulo +
      18 01               JR 01
2014  1D                  DEC E          Ajustar angulo -
      47                  LD B,A         Guardar bits 6,7
      ED 53 0C 20         LD(200C),DE    Guardar angulo
      08                  EX AF,AF'      Intercambiar registros
      D5                  PUSH DE
      FB                  EI             Permitir interrupciones
      ED 4D               RETI           Retorno
```

Programa Principal

```
2020  ED 5E               IM 2
      3E 00               LD A,00        Byte menos significativo
      D3 02               OUT(02),A      Vector lado A
      3E 04               LD A,04
      D3 03               OUT(03),A      Vector lado B
      3E 4F               LD A,4F        Modo entradas
      D3 02               OUT(02),A      Lado A
      D3 03               OUT(03),A      Lado B
2030  3E 97               LD A,97        Permitir interrupciones
      D3 02               OUT(02),A      Lado A
      D3 03               OUT(03),A      Lado B
      3E 20               LD A,20        Byte más significativo
      ED 47               LD I,A         Registro I
      5E                  IM 2           Permitir interrupciones
      ED 21 10 20         LD IX,2010     IX= buffer
      3A 0C 20            LD A,(200C)    Byte más significativo
2043  E6 C0               AND C0         6 bits menos signif.
      DD 96 01            SUB (IX+1),A
      32 0C 20            LD (200C),A
```

Programa para Salida Audio -- PIO PA0

440 Hz (=la) =1.13636 mseg. por 1/2 ciclo
Reloj del Starter Kit = 1.9968 MHz
2269.1 ciclos = 1.13636 mseg.

```
2260    3E 0F           LDA 0F      Modo de salidas
        D3 82           OUT(82),A   Canal A control
        AF              XOR A       Registro A = 0
        D3 80     REP   OUT(80),A   Canal A datos
        06 8C           LD B,8C     140 veces por el bucle
        05              DEC B
        20 FD           JR NZ
        3C              INC A       Cambiar bit 0
        18 F6           JR  REP
```

Cuenta: 16*139 + 11 + 11 + 7 + 4 + 12 = 2269 ciclos.

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

DATOS TECNICOS

Marzo, 1981

Z80™-PIO
Z80™A-PIO

Product Specification

The Zilog Z-80 product line is a complete set of microcomputer components, development systems and support software. The Z-80 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 Z-80 Parallel I/O (PIO) Interface Controller is a programmable, two port device which provides TTL compatible interfacing between peripheral devices and the Z80-CPU. The Z80-CPU configures the Z80-PIO to interface with standard peripheral devices such as tape punches, printers, keyboards, etc.

Structure

- N-Channel Silicon Gate Depletion Load technology
- 40 Pin DIP
- Single 5 volt supply
- Single phase 5 volt clock
- Two independent 8-bit bidirectional peripheral interface ports with "handshake" data transfer control

Features

- Interrupt driven "handshake" for fast response
- Any one of the following modes of operation may be selected for either port:
  - Byte output
  - Byte input
  - Byte bidirectional bus (available on Port A only)
  - Bit Mode
- Programmable interrupts on peripheral status conditions.
- 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.

PIO Architecture

A block diagram of the Z80-PIO is shown in figure 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. A typical application might use Port A as the data transfer channel and Port B for the status and control monitoring.

The Port I/O logic is composed of 6 registers with "handshake" control logic as shown in figure 2. The registers include: an 8-bit input register, an 8-bit 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. The last three registers are used only when the port has been programmed to operate in the bit mode.

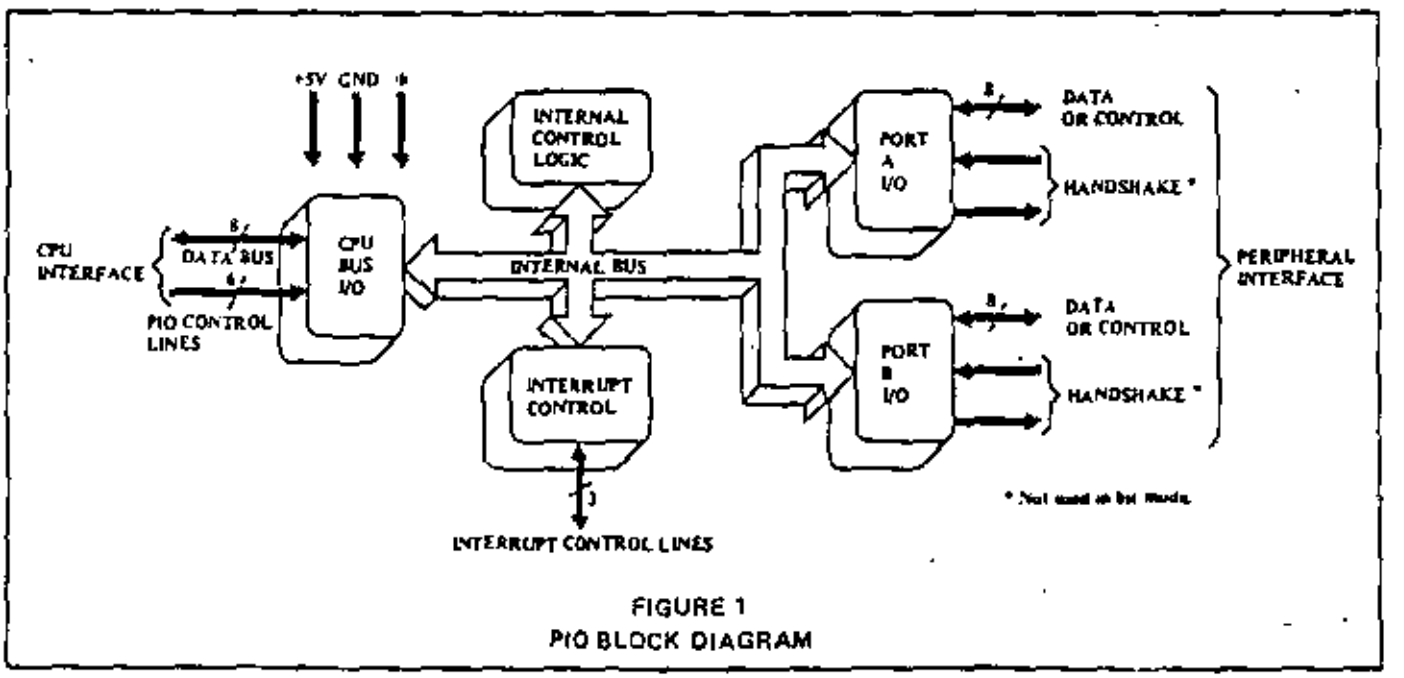
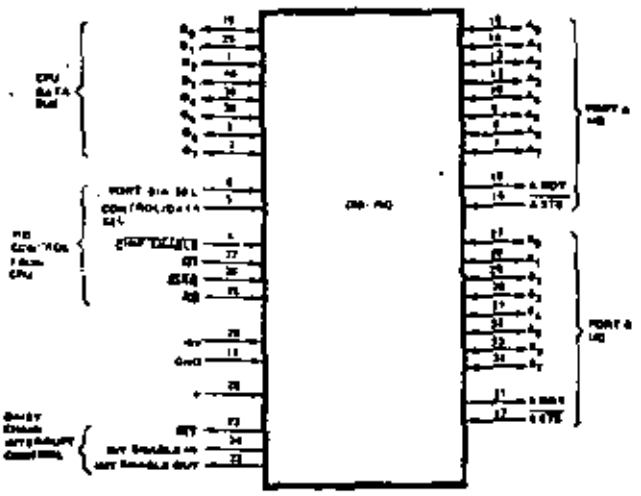


FIGURE 1
PIO BLOCK DIAGRAM

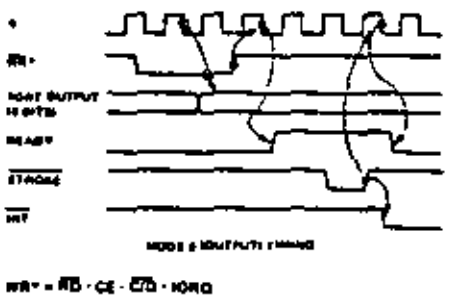


D_7-D_0	Z80-CPU Data Bus (bidirectional, tristate)
B/A Sel	Port B or A Select (input, active high)
C/D Sel	Control or Data Select (input, active high)
$\overline{CE}$	Chip Enable (input, active low)
Φ	System Clock (input)
$\overline{M1}$	Machine Cycle One Signal from CPU (input, active low)
$\overline{IORQ}$	Input/Output Request from Z80-CPU (input, active low)
$\overline{RD}$	Read Cycle Status from the Z80-CPU (input, active low)
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.
$\overline{INT}$	Interrupt Request (output, open drain, active low)
A_0-A_7	Port A Bus (bidirectional, tristate)
$\overline{A\ STB}$	Port A Strobe Pulse from Peripheral Device (input, active low)
A RDY	Register A Ready (output, active high)
B_0-B_7	Port B Bus (bidirectional, tristate)
$\overline{B\ STB}$	Port B Strobe Pulse from Peripheral Device (input, active low)
B RDY	Register B Ready (output, active high)

Timing Waveforms

OUTPUT MODE

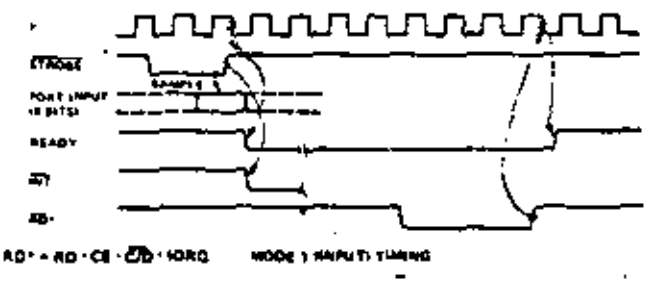
An output cycle is always started by the execution of an output instruction by the CPU. The $\overline{WR}$ pulse from the CPU latches the data from the CPU data bus into the selected port's output register. The write pulse sets the ready flag after a low going edge of Φ, indicating data is available. Ready stays active until the positive edge of the strobe line is received indicating that data was taken by the peripheral. The positive edge of the strobe pulse generates an $\overline{INT}$ if the interrupt enable flip flop has been set and if this device has the highest priority.



$\overline{WR} = \overline{M1} \cdot CE \cdot \overline{C/D} \cdot \overline{IORQ}$

INPUT MODE

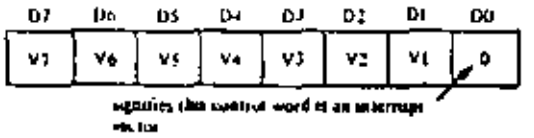
When $\overline{STROBE}$ goes low data is loaded into the selected port input register. The next rising edge of strobe activates $\overline{INT}$ if interrupt enable is set and this is the highest priority requesting device. The following falling edge of Φ resets Ready to an inactive state, indicating that the input register is full and cannot accept any more data until the CPU completes a read. When a read is complete the positive edge of $\overline{RD}$ will set Ready at the next low going transition of Φ. At this time new data can be loaded into the PIO.



$\overline{RD} = \overline{RD} \cdot CE \cdot \overline{C/D} \cdot \overline{IORQ}$

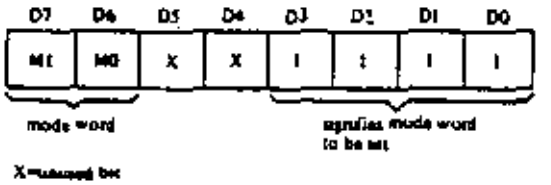
LOAD INTERRUPT VECTOR

The Z80-CPU requires an 8-bit interrupt vector be supplied by the interrupting device. The CPU forms the address for the interrupt service routine of the port using this vector. During an interrupt acknowledge cycle the vector is placed on the Z-80 data bus by the highest priority device requesting service at that time. 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.



SELECTING AN OPERATING MODE

When selecting an operating mode, the 2-bit mode control register is set to one of four values. These two bits are the most significant bits of the register, bits 7 and 6; bits 5 and 4 are not used while bits 3 through 0 are all set to 1111 to indicate "set mode."



Mode	M_1	M_0
Output	0	0
Input	0	1
Bidirectional	1	0
Bit	1	1

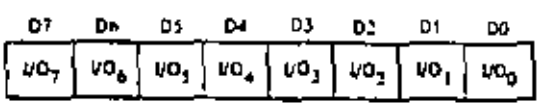
MODE 0 active indicates that data is to be written from the CPU to the peripheral.

MODE 1 active indicates that data is to be read from the peripheral to the CPU.

MODE 2 allows data to be written to or read from the peripheral device.

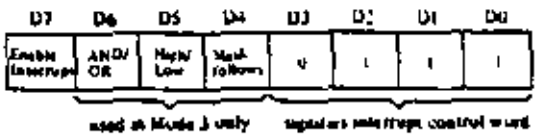
MODE 3 is intended for status and control applications. When selected, the next control word must set the I/O Register to indicate which lines are to be input and which lines are to be output.

I/O = 1 sets bit to input.
I/O = 0 sets bit to output.

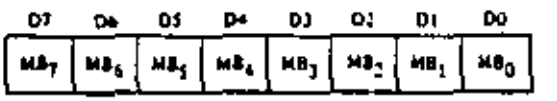


INTERRUPT CONTROL

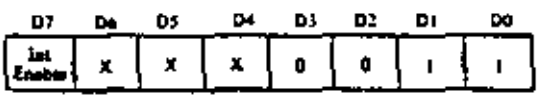
Bit 7 = 1	interrupt enable is set—allowing interrupt to be generated.
Bit 7 = 0	indicates the enable flag is reset and interrupts may not be generated.
Bits 6,5,4	are used in the bit mode interrupt operations; otherwise they are disregarded.
Bits 3,2,1,0	signify that this command word is an interrupt control word.



If the "mask follows" bit is high (D4 = 1), the next control word written to the port must be the mask.



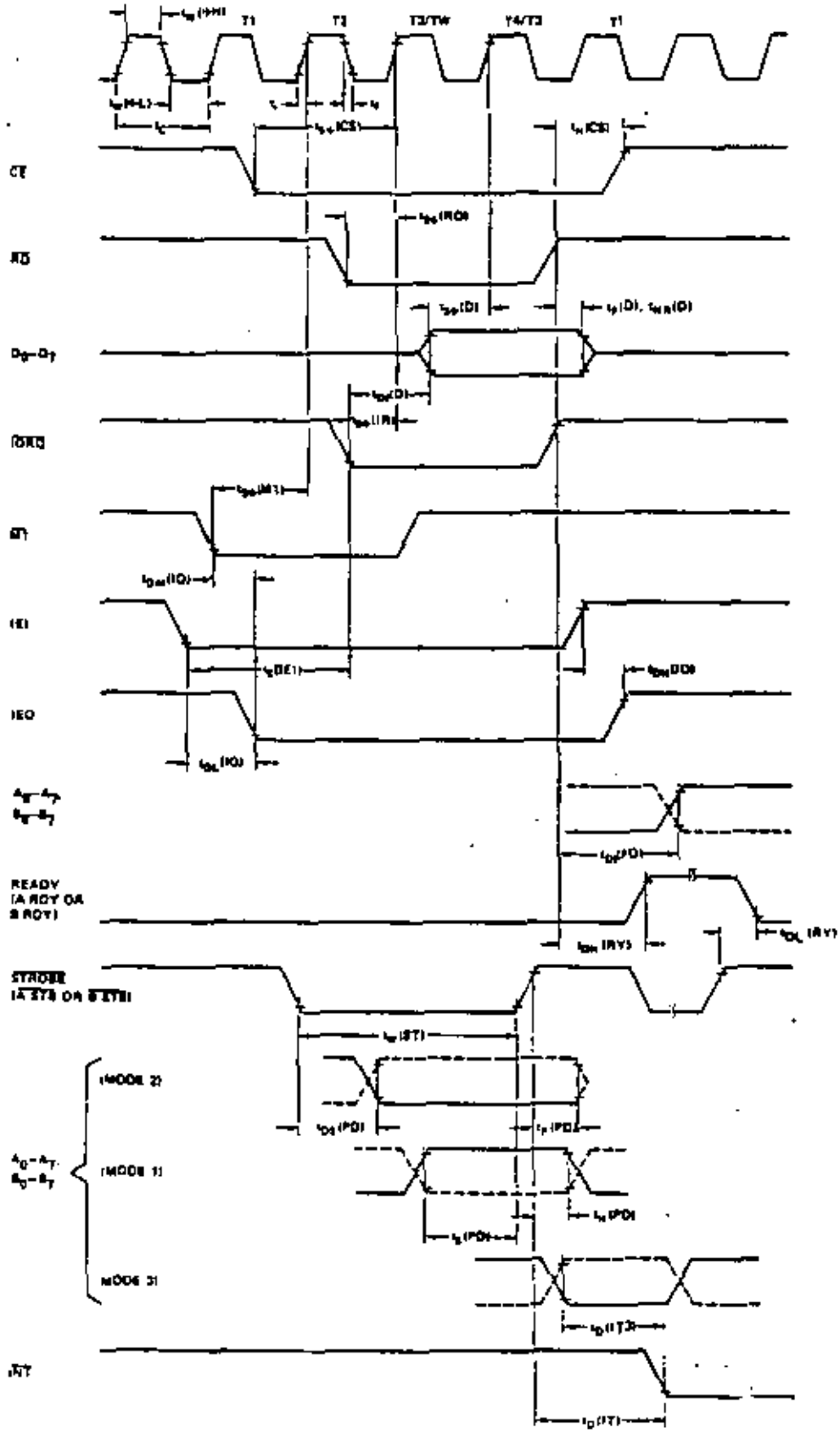
The interrupt enable flip-flop of a port may be set or reset without modifying the rest of the interrupt control word by the following command.



A.C. 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



TA = 0° C to 70° C, Vcc = +5 V ± 5%, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_c $t_{w(\Phi H)}$ $t_{w(\Phi L)}$ t_r, t_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	
CS, $\overline{CS}$ ETC.	$t_{su(CS)}$	Control Signal Set-Up Time to Rising Edge of Φ During Read or Write Cycle	145		nsec	
D_0-D_7	$t_{DR(D)}$ $t_{su(D)}$ $t_{D(INTA)}$ $t_{f(D)}$	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 INTA Cycle Delay to Floating Bus (Output Buffer Disable Time)	50	320 290 110	nsec nsec nsec nsec	[2] C_L = 50 pf [3]
IEI	$t_{s(IEI)}$	IEI Set-Up Time to Falling edge of $\overline{IORQ}$ During 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.		190 130 190	nsec nsec nsec	[5] [5] C_L = 50 pf [5]
$\overline{IORQ}$	$t_{su(IR)}$	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle.	115		nsec	
$\overline{M1}$	$t_{su(M1)}$	$\overline{M1}$ Set-Up Time to Rising Edge of Φ During INTA or $\overline{M1}$ Cycle. See Note B	80		nsec	
$\overline{RD}$	$t_{su(RD)}$	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	110		nsec	
A_0-A_7, B_0-B_7	$t_{s(PD)}$ $t_{DS(PD)}$ $t_{f(PDS)}$ $t_{DH(PDR)}$	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 WR Cycle (Mode 0)	230	 270 150 150	nsec nsec nsec nsec	 [5] C_L = 50 pf [5]
$\overline{ASTB}$, $\overline{BSTB}$	$t_{w(ST)}$	Pulse Width, $\overline{STROBE}$	150 [4]		nsec nsec	
$\overline{INT}$	$t_{D(IT)}$ $t_{D(ITB)}$	$\overline{INT}$ Delay time from Rising Edge of $\overline{STROBE}$ $\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		440 380	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 + 380	nsec nsec	[5] C_L = 50 pf [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 3 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 50 pf increase in loading up to 300 pf max.

[3] Increase $t_{D(D)}$ by 10 nsec for each 50 pf increase in loading up to 200 pf max.

[4] For Mode 3: $t_{w(ST)} > t_{s(PD)}$

[5] Increase these values by 2 nsec for each 10 pf increase in loading up to 100 pf max.

Z80™-DMA
Z80A-DMA

Product Specification

OCTOBER 1977

PRELIMINARY

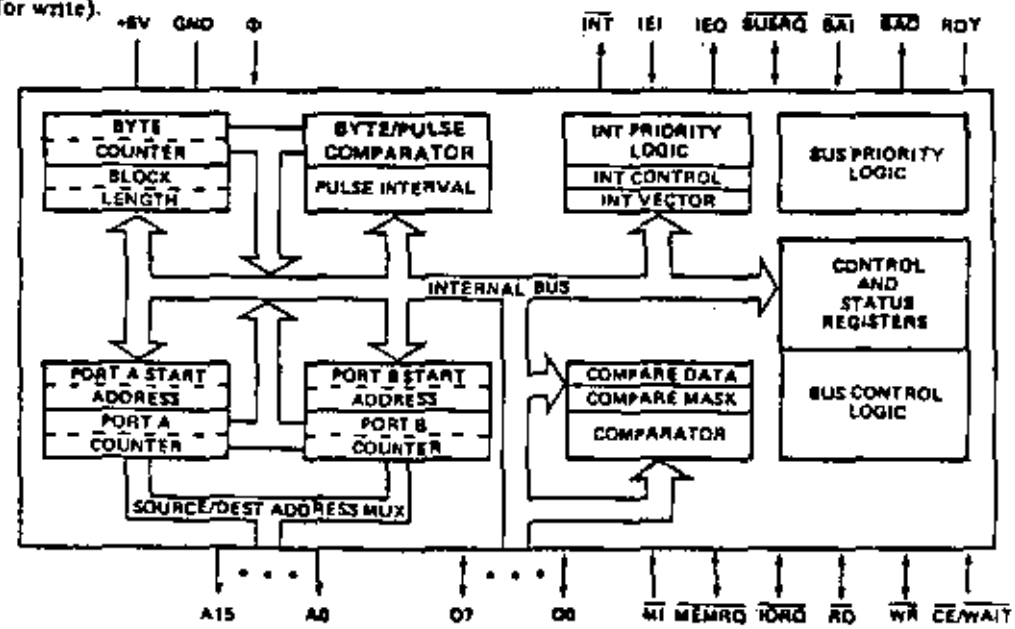
Zilog's Z80 microcomputer product line includes a third generation LSI component set, development systems and support software. The component set includes all the logic circuits necessary for the user to build high performance microcomputer systems with virtually no external logic and a minimal number of standard low-cost memory components. The Z80-DMA (Direct Memory Access) circuit is a programmable single-channel device which provides all address, timing and control signals to effect the transfer of blocks of data between two ports within a Z80-CPU based system. These ports may be either system main memory or any system peripheral I/O device. The DMA can also search a block of data for a particular byte (bit maskable), with or without a simultaneous transfer.

Structure

- N-channel Silicon Gate Depletion Load Technology
- 40 Pin DIP
- Single 5 volt supply
- Single phase 5 volt clock
- Single channel, two port

Features

- Three classes of operation:
  - –Transfer Only
  - –Search Only
  - –Search-Transfer
- Address and Block Length Registers fully buffered. Values for next operation may be loaded without disturbing current values.
- Dual addresses generated during a transfer (one for read port and one for write).
- Programmable data transfers and searches, automatically incrementing or decrementing the port addresses from programmed starting addresses (they can also remain fixed).
- Four modes of operation:
  - –Byte-at-a-time: One byte transferred per request
  - –Burst: Continues as long as ports are ready
  - –Continuous: Locks out CPU until operation complete
  - –Transparent: Steals refresh cycles
- Timing may be programmed to match the speed of any port.
- Interrupts on Match Found, End of Block, or Ready, may be programmed.
- An entire previous operation may be repeated automatically or on command. (Auto restart or Load)
- The DMA can signal when a specified number of bytes has been transferred, without halting transfer.
- Multiple DMA's easily configured for rotating priority.
- The channel may be enabled, disabled or reset under software control.
- Complete channel status upon program (CPU) request.
- Up to 1.25 megabyte Search or Transfer Rate.
- Daisy-chain priority interrupt and bus acknowledge included to provide automatic interrupt vectoring and bus request control, without need for additional external logic.
- TTL compatible inputs and outputs
- The CPU can read current Port counters, Byte counter, or Status Register. A mask byte can be set which defines which registers can be accessed during read operations.



DMA Internal Block Diagram

Fig. 1

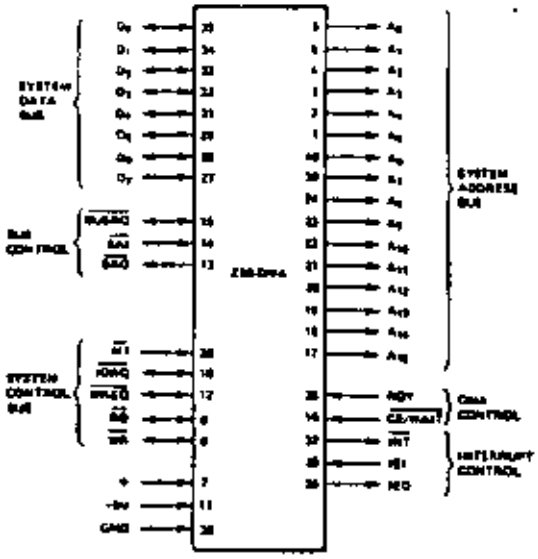
Addressing

The DMA's addressing of ports is either fixed or sequential, incrementing or decrementing from a starting address. The length of the operation (number of bytes) is specified by the programmed contents of a block length register. The DMA can address block lengths of up to 64K bytes. During a transfer two separate port addresses are generated, one during the Read cycle and one during the Write cycle.

Operating Sequence

Once the DMA has been programmed it may be "Enabled" (command byte 2d or 2a). In the enabled condition when Ready goes active the DMA will request the bus by bringing $\overline{BUSRQ}$ low. The CPU will acknowledge this with a $\overline{BUSACK}$ which will normally be attached to $\overline{BAI}$. When the DMA receives $\overline{BAI}$ it will start its programmed operation releasing $\overline{BUSRQ}$ to a "high" state when it is through.

Z80-DMA Pin Description



A_0–A_{15} System Address Bus. All sixteen of these pins are used by the DMA to address system main memory or an I/O port (output)

D_0–D_7 System Data Bus. Commands from the CPU, DMA status and data from memory or peripherals are transferred on these tristate pins (input/output)

+5V Power

GND Ground

Φ System clock (input)

$\overline{MI}$ Machine cycle One signal from CPU (input)

$\overline{IORQ}$ Input/Output Request to and from the System Bus (input/output)

$\overline{MREQ}$ Memory REQuest to the System Bus (input/output)

$\overline{RD}$ ReaD to and from the System Bus (input/output)

$\overline{WR}$ WRite to and from the System Bus (input/output)

$\overline{CE/WAIT}$ Chip Enable; may also be programmed to be $\overline{WAIT}$ during time when $\overline{BAI}$ is low (input)

$\overline{BUSRQ}$ BUS ReQuest. Requests control of the CPU Address Bus, Data Bus and Status/Control Bus (input/output, open drain)

$\overline{BAI}$ Bus Acknowledge In. Signals that the system buses have been released for DMA control (input)

$\overline{BAO}$ Bus Acknowledge Out. $\overline{BAI}$ and $\overline{BAO}$ form a daisy-chain connection for system-wide priority bus control (output)

$\overline{INT}$ INTerrupt request (output, open drain)

IEI Interrupt Enable In (input)

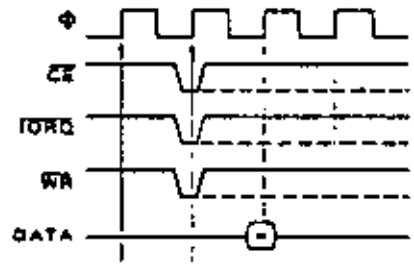
IEO Interrupt Enable Out. IEI and IEO form a daisy-chain connection for system-wide priority interrupt control (output)

RDY ReaDY is monitored by the DMA to determine when a peripheral device associated with a DMA port is ready for a read or write operation (input, programmable as active high or low)

DMA Timing Waveforms

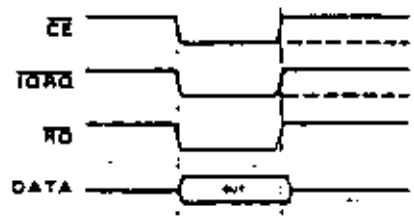
DMA Command Write Cycle

Illustrated here is the timing associated with a command byte or control byte being written to the DMA which is to be loaded into internal registers. Z80 Output instructions satisfy this timing.



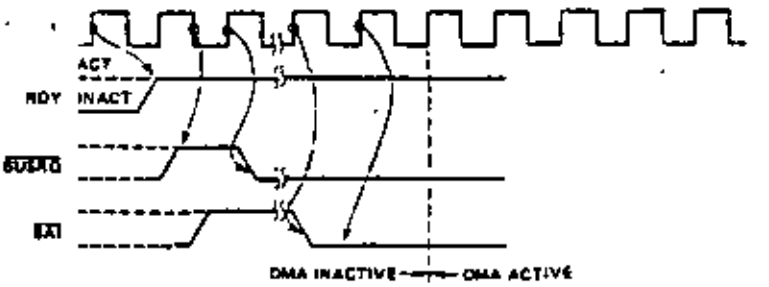
DMA Register Read Cycle

This timing is used when a read operation is performed on the DMA to access the contents of the Status Register, Address Counter or other readable registers. Z80 Input instructions satisfy this timing.



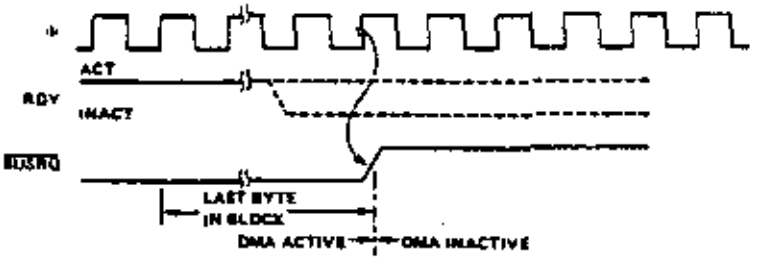
DMA Bus Request and Acceptance for Byte-at-a-Time, Burst, and Continuous Mode

*Ready is sampled on every rising edge of Φ. When it is found to be active, the following rising edge of Φ generates $\overline{BUSRQ}$. After receiving $\overline{BUSRQ}$ the CPU will grant a $\overline{BUSAK}$ which will be connected to $\overline{BAI}$ either directly or through the Bus Acknowledge Daisy Chain. When a low is detected on $\overline{BAI}$ (sampled on every rising edge of Φ), the next rising edge of Φ will start an active DMA cycle.



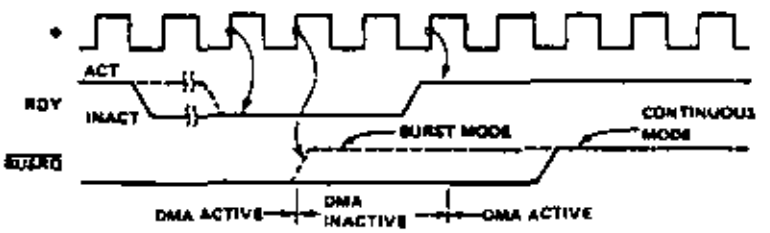
DMA Bus Release at End of Block for Burst or Continuous Mode

Timing for End of Block and DMA not programmed for Auto-restart.



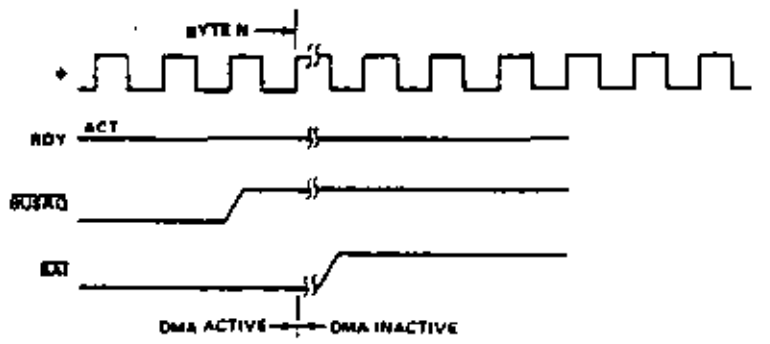
DMA Bus Release with 'Ready' for Burst and Continuous Mode

The DMA will relinquish the bus after RDY has gone inactive (Burst mode) or after an End of Block or a Match is found (Continuous mode). With RDY inactive, the DMA in Continuous mode is inactive but maintains control of the bus ($\overline{BUSRQ}$ low) until the cycle is resumed when RDY goes active.



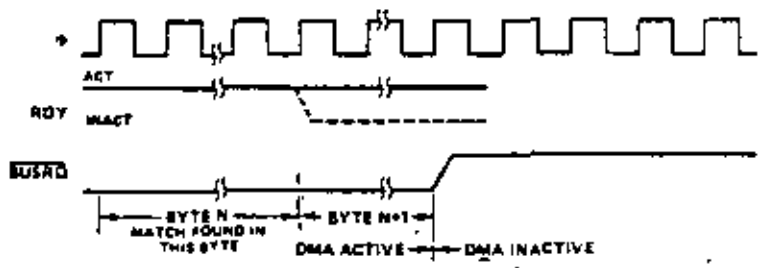
DMA Bus Release for Byte-at-a-Time Mode

In the Byte mode the DMA will release $\overline{BUSRQ}$ on the rising edge of Φ prior to the end of each Read cycle in Search Only or each Write cycle in a Transfer, regardless of the state of RDY. The next bus request will come after both $\overline{BUSRQ}$ and $\overline{BAI}$ have returned high.



DMA Bus Release with Match for Burst or Continuous Modes

When a Match is found and the DMA is programmed to stop on Compare, the DMA performs an operation on the next byte and then releases bus.

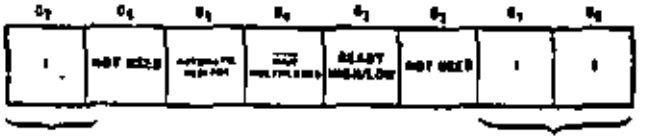


Command Byte 2B

Specifies Group 2 — Specifies Byte 2B

M_1	M_0	Mode
0	0	Byte
0	1	Continuous
1	0	Burst
1	1	Transparent

Command Byte 2C



Specifies Group 2 — Specifies Byte 2C

$D_5 = 1$ Automatically repeats entire operation when end of block is reached.

$D_5 = 0$ No affect.

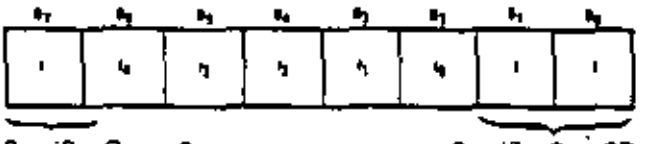
$D_4 = 1$ $\overline{CE}$ and $\overline{WAIT}$ multiplexed on same pin.

$D_4 = 0$ $\overline{CE}$ only.

$D_3 = 1$ Ready active high.

$D_3 = 0$ Ready active low.

Command Byte 2D



Specifies Group 2 — Specifies Byte 2D

Hex	f_4	f_3	f_2	f_1	f_0	
C3	1	0	0	0	0	Reset
C7	1	0	0	0	1	Reset Port A Timing
CB	1	0	0	1	0	Reset Port B Timing
CF	1	0	0	1	1	Load
D3	1	0	1	0	0	Continue
AB	0	1	0	1	0	Enable Int
AF	0	1	0	1	1	Disable Int
A3	0	1	0	0	0	Reset Int
87	0	0	0	0	1	Enable DMA
83	0	0	0	0	0	Disable DMA
BB	0	1	1	1	0	Read Byte Follows
A7	0	1	0	0	1	Reset RD
BF	0	1	1	1	1	RD Status
B3	0	1	1	0	0	Force Ready
B7	0	1	1	0	1	Enable After RETI
8B	0	0	0	1	0	Reset Status

Command Byte 2D Summary

Reset: Resets all interrupt circuitry, disables interrupts and bus req. logic.

Reset Timing A or B: Resets timing for Port A or B to standard Z80-CPU timing.

Load: Zeros Byte Counter and loads Starting Address for both Ports.

Continue: Resets byte counter only. Addresses continue from present location.

Enable Interrupt: Permits interrupt to occur.

Disable Interrupt: Inhibits interrupt from occurring.

Reset Interrupt: Resets and disables all interrupt circuits (similar to RETI).

Enable DMA, Disable DMA: Overall enable or disable for all operations except interrupts: does not reset any functions.

Read Byte Follows: Next write to DMA will contain a mask to program which readable registers are to be read.

Reset RD: Next read will be from 1st register set as readable by response mask.

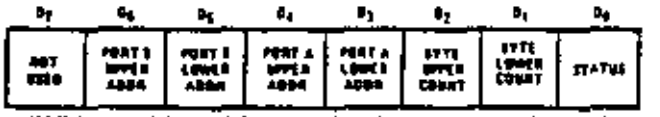
RD Status: Next read will be from status register.

Force Ready: Ready will be considered active regardless of the state of external RDY pin. Used for Mem-Mem operations where no RDY signal is needed.

Enable after RETI: DMA will not request bus until after it has received a RETI.

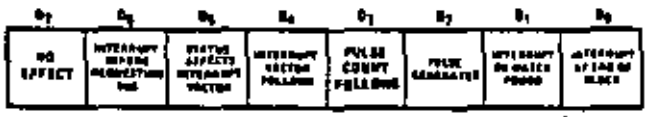
RST Status: Resets Match and End of Block status bits.

Read Byte



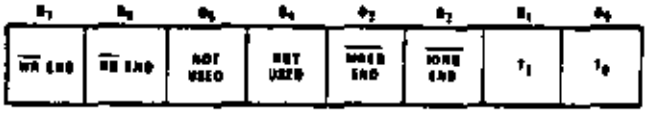
A "1" in any bit position enables that register to be read.

Interrupt Control Byte



A "1" in a bit position selects the option.

Timing Control Byte



T_1	T_0	Cycle Length
0	0	4
0	1	3
1	0	2
1	1	1

A "0" in D_2, D_3, D_6, or D_7 will cause the corresponding control signal to end ½ clock time before the end of the cycle. Note: the total operation (Read and Write in Transfer or Read in Search) must be at least 2 cycles long.

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	1.5W

Note: All AC and DC characteristics remain the same for the military grade parts except I_{CC}.

I_{CC} = 200 mA.

*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.

Z80-DMA 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}−0.6		V_{CC}+0.3	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} = 1.8 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −250 µA
I_{CC}	Power Supply Current			150	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 ≤ V_{IN} ≤ V_{CC}

Z80A-DMA D.C. Characteristics

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

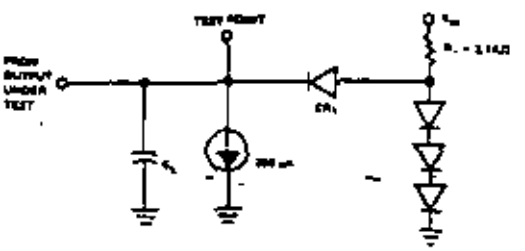
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC}−0.6		V_{CC}+0.3	V	
V_{IL}	Input Low Voltage	−0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −250 µA
I_{CC}	Power Supply Current		90	200	mA	t_C = 250nsec
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 ≤ V_{IN} ≤ V_{CC}

Capacitance

T_A = 25°C, f = 1 MHz

Symbol	Parameter	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	

Load Circuit for Output



Z80-DMA as a Peripheral Device (Inactive State).
T_A = 0°C to 70°C, Vcc = +5V±5%, Unless Otherwise Noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_{cy} $t_{w(\phi H)}$ $t_{w(\phi L)}$ $t_{r,f}$	Clock Period Clock Pulse Width, Clock High Clock Pulse Width, Clock Low Clock Rise and Fall Times	400 170 170	(1) 2000 2000 30	nsec nsec nsec nsec	
	t_{sh}	Any Hold Time for Specified Setup Time	0		nsec	
CE, WR IORQ	$t_{s\phi(CS)}$	Control Signal Setup Time to Rising Edge of Φ During Write Cycle (IORQ, WR, CE)	280		nsec	
D_{0-7}	$t_{DR(D)}$ $t_{s\phi(D)}$ $t_{DF(D)}$ $t_{F(D)}$	Data Output Delay from Falling Edge of RD Data Setup Time to Rising Edge of Φ During Write or M1 Cycle Data Output Delay from Falling Edge of IORQ During INTA Cycle Delay to Floating Bus (Output Buffer Disable Time)	60	430 340 180	nsec nsec nsec nsec	(2) C_L = 50pF (3)
IEI	$t_{s(IEI)}$	IEI Setup Time to Falling Edge of IORQ During 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 M1 (Interrupt Occurring Just Prior to M1) See Note A.		210 150 300	nsec nsec nsec	C_L = 50pF
M1	$t_{s\phi(M1)}$	M1 Setup Time to Rising Edge of Φ During INTA or RT Cycle. See Note B.	210		nsec	
RD	$t_{s\phi(RD)}$	RD Setup Time to Rising Edge of Φ During RT Cycle	240		nsec	
INT	$t_{D(IT)}$	INT Delay Time from Conditions Causing INT, INT generated only when DMA is Inactive.		500	nsec	
BAO	$t_{DH(BO)}$ $t_{DL(BO)}$	BAO Delay from Rising Edge of BAI BAO Delay from Falling Edge of BAI	150 150	200 200	nsec nsec	

(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 $t_{DF(D)}$ by 10 nsec for each 50pF increase in loading up to 200pF max.

A. 2.5 t_c > (N−2) $t_{DL(BO)}$ + $t_{DM(IO)}$ + $t_{s\phi(IEI)}$ + TTL Buffer Delay, if any

Z80-DMA as a Bus Controller (Active State)
T_A = 0°C to 70°C, Vcc = +5V±5%, Unless Otherwise Noted.

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_c	Clock Period	4	[12]	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	180	2000	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	t_r, t_f	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{d(AD)}$	Address Output Delay		145	nsec	
	$t_{f(AD)}$	Delay to Float		110	nsec	C_L = 50pF
	t_{ASM}	Address Stable Prior to MREQ (Memory Cycle)	[1]		nsec	D
	t_{ASI}	Address Stable Prior to IORQ, RD or WR (I/O Cycle)	[2]		nsec	D
	t_{CA}	Address Stable From RD or WR	[3]		nsec	D
	t_{caf}	Address Stable From RD or WR During Float	[4]		nsec	D
D_{0-7}	$t_{d(D)}$	Data Output Delay		290	nsec	
	$t_{f(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{ds(D)}$	Data Setup Time to Rising Edge of Clock During Read When Rising Edge Ends RD	50		nsec	C_L = 300pF
	$t_{ds(\Phi)D}$	Data Setup Time to Falling Edge of Clock During Read When Falling Edge Ends RD	60		nsec	
	t_{dsm}	Data Stable Prior to WR (Memory Cycle)	[5]		nsec	D
	t_{dsi}	Data Stable Prior to WR (I/O Cycle)	[6]		nsec	D
	t_{dsf}	Data Stable From WR	[7]		nsec	D
	t_{dh}	Any Hold Time for Setup Time	0		nsec	
MREQ	$t_{d(\Phi L)MR}$	MREQ Delay from Falling Edge of Clock, MREQ Low		100	nsec	
	$t_{d(\Phi H)MR}$	MREQ Delay from Rising Edge of Clock, MREQ High		100	nsec	
	$t_{d(\Phi H)MR}$	MREQ Delay from Falling Edge of Clock, MREQ High		100	nsec	
	$t_{d(\Phi L)MR}$	MREQ Delay from Falling Edge of Clock, MREQ Low		100	nsec	C_L = 50pF
	$t_{w(MRL)}$	Pulse Width, MREQ Low	[8]		nsec	D
	$t_{w(MRH)}$	Pulse Width, MREQ High	[9]		nsec	D
IORQ	$t_{d(\Phi H)IR}$	IORQ Delay from Rising Edge of Clock, IORQ Low		80	nsec	
	$t_{d(\Phi L)IR}$	IORQ Delay from Falling Edge of Clock, IORQ Low		130	nsec	
	$t_{d(\Phi H)IR}$	IORQ Delay from Rising Edge of Clock, IORQ High		100	nsec	C_L = 50pF
	$t_{d(\Phi H)IR}$	IORQ Delay from Falling Edge of Clock, IORQ High		110	nsec	
RD	$t_{d(\Phi H)RD}$	RD Delay from Rising Edge of Clock, RD Low		100	nsec	
	$t_{d(\Phi H)RD}$	RD Delay from Falling Edge of Clock, RD Low		130	nsec	
	$t_{d(\Phi H)RD}$	RD Delay from Rising Edge of Clock, RD High		100	nsec	C_L = 50pF
	$t_{d(\Phi H)RD}$	RD Delay from Falling Edge of Clock, RD High		110	nsec	
WR	$t_{d(\Phi H)WR}$	WR Delay from Rising Edge of Clock, WR Low		80	nsec	
	$t_{d(\Phi H)WR}$	WR Delay from Falling Edge of Clock, WR Low		80	nsec	
	$t_{d(\Phi H)WR}$	WR Delay from Falling Edge of Clock, WR High		100	nsec	C_L = 50pF
	$t_{d(\Phi H)WR}$	WR Delay from Rising Edge of Clock, WR High		100	nsec	
	$t_{w(WRL)}$	Pulse Width, WR Low	[10]		nsec	
WAIT	$t_{s(WT)}$	WAIT Setup Time to Falling Edge of Clock	70		nsec	
BUSRQ	$t_{d(BQ)}$	BUSRQ Delay Time from Rising Edge of Clock	100		nsec	
	t_{FD}	Delay to Float (MREQ, IORQ, RD and WR)		100	nsec	

[12] t_c = $t_{w(\Phi H)}$+$t_{w(\Phi L)}$+t_r+t_f

[1] t_{ASM} = $t_{w(\Phi H)}$+t_f−75
[2] t_{ASI} = t_c−80
[3] t_{CA} = $t_{w(\Phi L)}$+t_r−40
[4] t_{caf} = $t_{w(\Phi L)}$+t_r−40

[5] t_{dsm} = t_c−190
[6] t_{dsi} = $t_{w(\Phi L)}$+t_r+190
[7] t_{dsf} = $t_{w(\Phi L)}$+t_r−60

[8] $t_{w(MRL)}$ = t_c−40
[9] $t_{w(MRH)}$ = t_c−40 Min. CPU Timing
$t_{w(MRH)}$ = $t_{w(WRH)}$+t_r−30 Variable 1 Cycle.

[10] $t_{w(WRL)}$ = t_c−40 Min. CPU Timing
$t_{w(WRL)}$ = $t_{w(\Phi H)}$+t_r−30 Variable 1 Cycle.

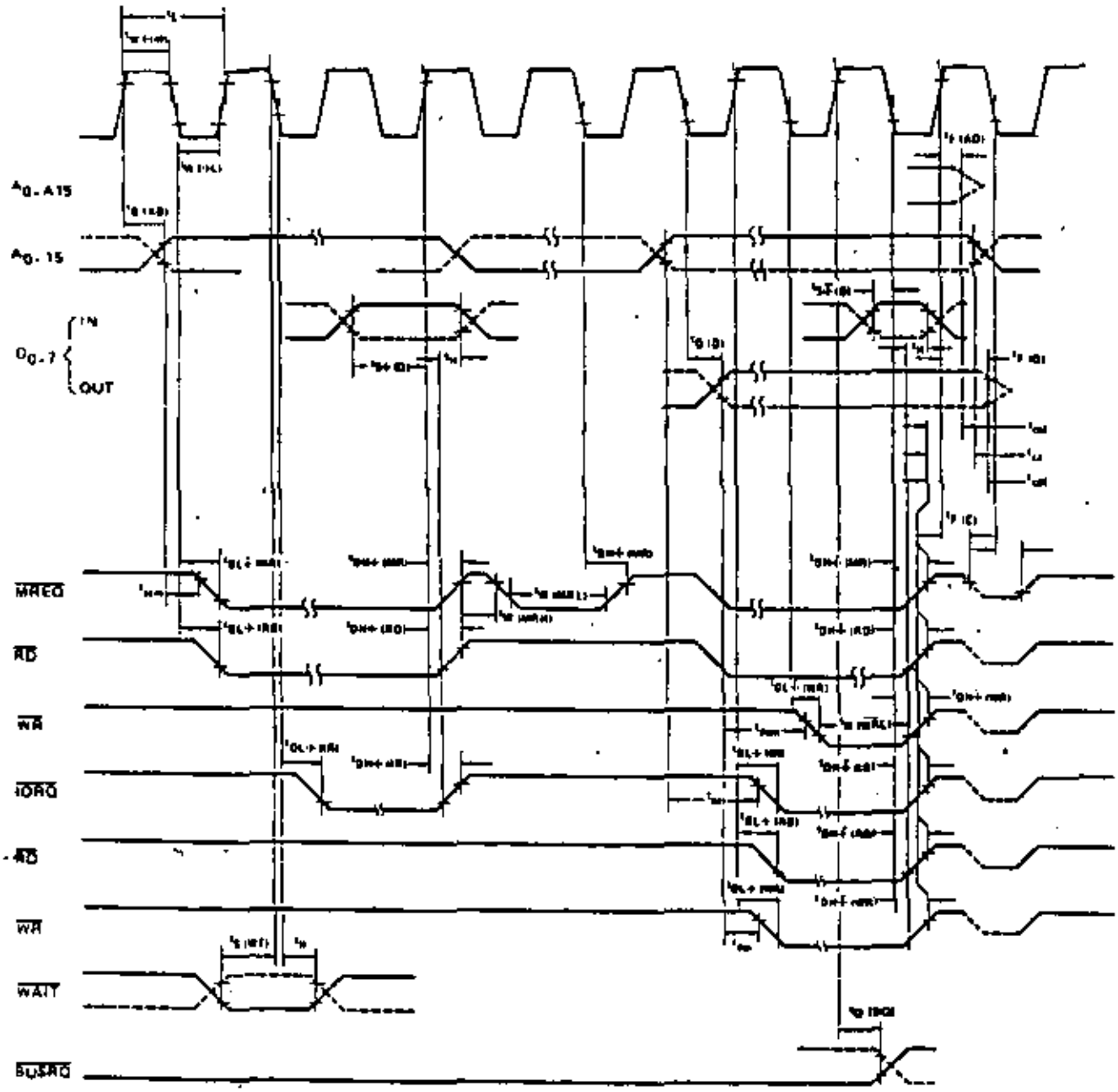
NOTES:
A. Data should be sampled only on the DMA data bus when RD is active.
B. All control inputs are internally synchronized, so they may be totally asynchronous with respect to the clock.
C. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V±5%
(1) ΔC_L = +100pF (Ag—A15 and Control Signals), add 30 nsec to timing shown.
D. During Standard CPU Timing

A.C. Timing Diagrams

Z80 and Z80A as a Bus Controller (Active State)

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

	"1"	"0"
CLOCK	4.2V	0.6V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	ΔV =	+0.5V



Z80®-CTC
Z80®A-CTC

Product Specification

APRIL 1978

The Zilog 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-Counter Timer Circuit (CTC) is a programmable, four channel device that provides counting and timing functions for the Z80-CPU. The Z80-CPU configures the Z80-CTC's four independent channels to operate under various modes and conditions as required.

Structure

- N-Channel Silicon Gate Depletion Load Technology
- 28 Pin DIP
- Single 5 volt supply
- Single phase 5 volt clock
- Four independent programmable 8-bit counter/16-bit timer channels

Features

- Each channel may be selected to operate in either a counter mode or timer mode.
- Programmable interrupts on counter or timer states.
- A time constant register automatically reloads the down counter at zero and the cycle is repeated.
- Readable down counter indicates number of counts-to-go until zero.
- Selectable 16 or 256 clock prescaler for each timer channel.
- Selectable positive or negative trigger may initiate timer operation.
- Three channels have zero count/timeout outputs capable of driving Darlington transistors.
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.
- All inputs and outputs fully TTL compatible.
- Outputs directly compatible with Z80-SIO.

CTC Architecture

A block diagram of the Z80-CTC is shown in figure 1. The internal structure of the Z80-CTC consists of a Z80-CPU bus interface, internal control logic, four counter channels, and interrupt control logic. Each channel has an interrupt vector for automatic interrupt vectoring, and interrupt priority is determined by channel number with channel 0 having the highest priority.

The channel logic is composed of 2 registers, 2 counters and control logic as shown in figure 2. The registers include an 8-bit time constant register and an 8-bit channel control register. The counters include an 8-bit readable down counter and an 8-bit prescaler. The prescaler may be programmed to divide the system clock by either 16 or 256.

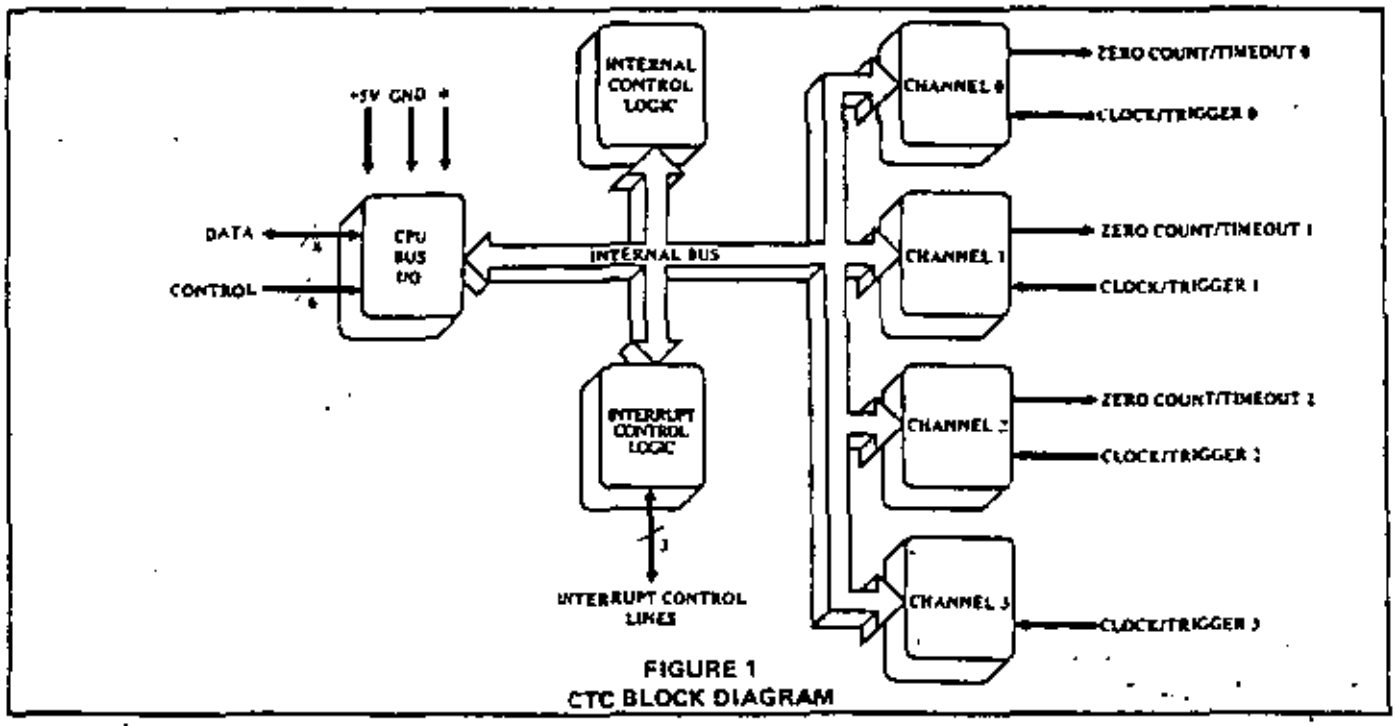


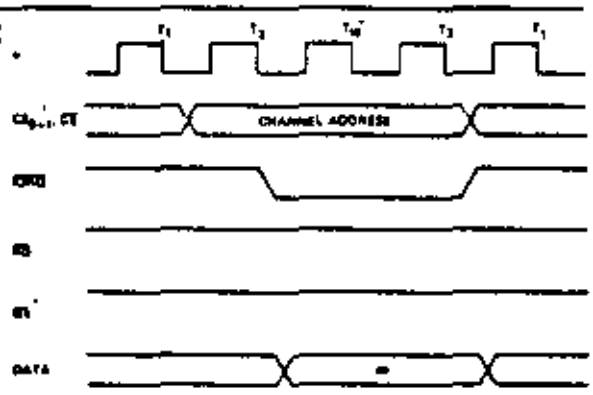
FIGURE 1
CTC BLOCK DIAGRAM

ZC/TO_1	Channel 1 Zero Count or Timeout (output, active high)
ZC/TO_2	Channel 2 Zero Count or Timeout (output, active high)
$CS_1 - CS_0$	Channel Select (input, active high). These form a 2-bit binary address of the channel to be accessed.
$D7 - D0$	Z80-CPU Data Bus (bidirectional, tristate)
$\overline{CE}$	Chip Enable (input, active low)
ϕ	System Clock (input)
$\overline{M1}$	Machine Cycle One Signal from Z80-CPU (input, active low)
$\overline{IORQ}$	Input/Output Request from Z80-CPU (input, active low)
$\overline{RD}$	Read Cycle Status from the Z80-CPU (input, active low)
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
$\overline{INT}$	Interrupt Request (output, open drain, active low)
$\overline{RESET}$	RESET stops all channels from counting and resets channel interrupt enable bits in all control registers. During reset time ZC/TO_{0-2} and $\overline{INT}$ go to the inactive states. IEO reflects the state of IEI, and the data bus output drivers go to the high impedance state (input, active low)

Timing Waveforms

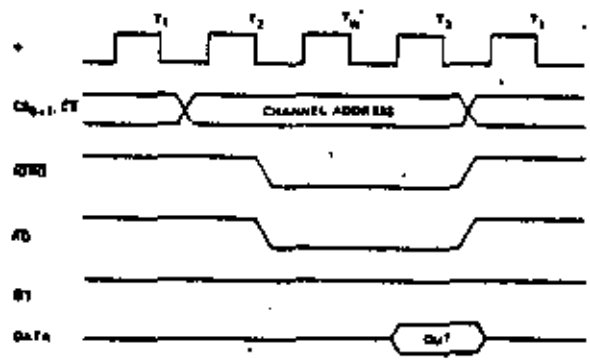
CTC WRITE CYCLE

Illustrated here is the timing for loading a channel control word, time constant and interrupt vector. No wait states are allowed for writing to the CTC other than the automatically inserted (T_W*). Since the CTC does not receive a specific write signal, it internally generates its own from the lack of an $\overline{RD}$ signal.



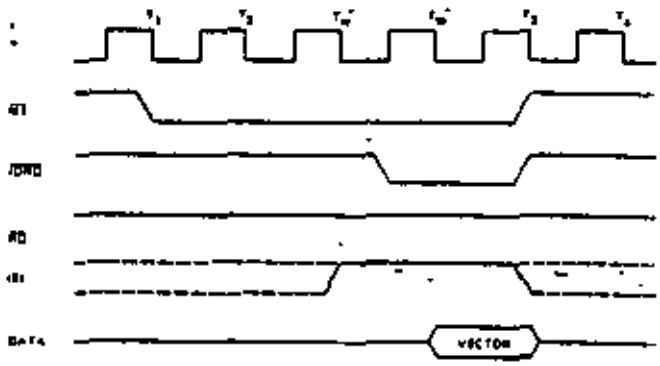
CTC READ CYCLE

Illustrated here is the timing for reading a channel's Down Counter when in Counter Mode. The value read onto the data bus reflects the number of external clock's rising edges prior to the rising edge of cycle (T_2). No wait states are allowed for reading the CTC other than the automatically inserted (T_W*).



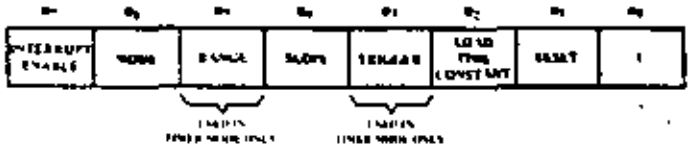
INTERRUPT ACKNOWLEDGE CYCLE

Some time after an interrupt is requested by the CTC, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). During this time the interrupt logic of the CTC will determine the highest priority channel 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. If the CTC Interrupt Enable Input (IEI) is active, then the highest priority interrupting channel places the contents of its interrupt vector register onto the Data Bus when $\overline{IORQ}$ goes active. Additional wait cycles are allowed.



SELECTING AN OPERATING MODE

When selecting a channel's operating mode, bit Ø is set to 1 to indicate this word is to be stored in the channel control register.



Bit 7 = Ø — Channel interrupts disabled.

Bit 7 = 1 — Channel interrupts enabled to occur every time Down Counter reaches a count of zero. Setting Bit 7 does not let a preceding count of zero cause an interrupt.

Bit 6 = Ø — Timer Mode – Down counter is clocked by the prescaler. The period of the counter is:
t_C • P • TC
t_C = system clock period
P = prescale of 16 or 256
TC = 8 bit binary programmable time constant (256 max)

Bit 6 = 1 — Counter Mode – Down Counter is clocked by external clock. The prescaler is not used.

Bit 5 = Ø — Timer Mode Only–System clock Φ is divided by 16 in prescaler.

Bit 5 = 1 — Timer Mode Only–System clock Φ is divided by 256 in prescaler.

Bit 4 = Ø — Timer Mode – negative edge trigger starts timer operation.
Counter Mode – negative edge decrements the down counter.

Bit 4 = 1 — Timer Mode – positive edge trigger starts timer operation.
Counter Mode – positive edge decrements the down counter.

Bit 3 = Ø — Timer Mode Only – Timer begins operation on the rising edge of T_2 of the machine cycle following the one that loads the time constant.

Bit 3 = 1 — Timer Mode Only – External trigger is valid for starting timer operation after rising edge of T_2 of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles.

Bit 2 = Ø — No time constant will follow the channel control word. One time constant must be written to the channel to initiate operation.

Bit 2 = 1 — The time constant for the Down Counter will be the next word written to the selected channel. If a time constant is loaded while a channel is counting, the present count will be completed before the new time constant is loaded into the Down Counter.

Bit 1 = Ø — Channel continues counting.

Bit 1 = 1 — Stop operation. If Bit 2 = 1 channel will resume operation after loading a time constant, otherwise a new control word must be loaded.

LOADING A TIME CONSTANT

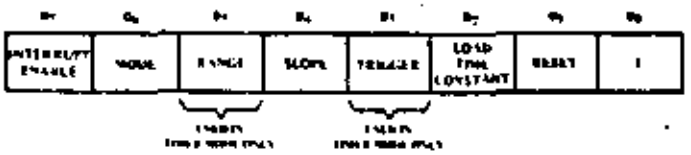
An 8-bit time constant is loaded into the Time Constant register following a channel control word with bit 2 set. All zeros indicate a time constant of 256.

LOADING AN INTERRUPT VECTOR

The Z80-CPU requires that an 8-bit interrupt vector be supplied by the interrupting channel. The CPU forms the address for the interrupt service routine of the channel using this vector. During an interrupt acknowledge cycle the vector is placed on the Z80 Data Bus by the highest priority channel requesting service at that time. The desired interrupt vector is loaded into the CTC by writing into channel Ø with a zero in DØ. D_7-D_3 contain the stored interrupt vector, D_2 and D_1 are not used in loading the vector. When the CTC responds to an interrupt acknowledge, these two bits contain the binary code of the highest priority channel which requested the interrupt and D_0 contains a zero since the address of the interrupt service routine starts at an even byte. Channel Ø is the highest priority channel.

SELECTING AN OPERATING MODE

When selecting a channel's operating mode, bit Ø is set to 1 to indicate this word is to be stored in the channel control register.



Bit 7 = Ø — Channel interrupts disabled.

Bit 7 = 1 — Channel interrupts enabled to occur every time Down Counter reaches a count of zero. Setting Bit 7 does not let a preceding count of zero cause an interrupt.

Bit 6 = Ø — Timer Mode — Down counter is clocked by the prescaler. The period of the counter is:
t_c * P * TC
t_c = system clock period
P = prescale of 16 or 256
TC = 8 bit binary programmable time constant (256 max)

Bit 6 = 1 — Counter Mode — Down Counter is clocked by external clock. The prescaler is not used.

Bit 5 = Ø — Timer Mode Only–System clock Φ is divided by 16 in prescaler.

Bit 5 = 1 — Timer Mode Only–System clock Φ is divided by 256 in prescaler.

Bit 4 = Ø — Timer Mode — negative edge trigger starts timer operation.
Counter Mode — negative edge decrements the down counter.

Bit 4 = 1 — Timer Mode — positive edge trigger starts timer operation.
Counter Mode — positive edge decrements the down counter.

Bit 3 = Ø — Timer Mode Only — Timer begins operation on the rising edge of T_2 of the machine cycle following the one that loads the time constant.

Bit 3 = 1 — Timer Mode Only — External trigger is valid for starting timer operation after rising edge of T_2 of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles.

Bit 2 = Ø — No time constant will follow the channel control word. One time constant must be written to the channel to initiate operation.

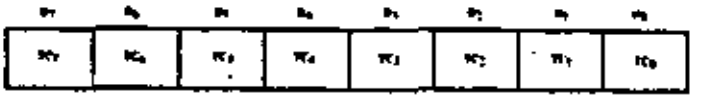
Bit 2 = 1 — The time constant for the Down Counter will be the next word written to the selected channel. If a time constant is loaded while a channel is counting, the present count will be completed before the new time constant is loaded into the Down Counter.

Bit 1 = Ø — Channel continues counting.

Bit 1 = 1 — Stop operation. If Bit 2 = 1 channel will resume operation after loading a time constant, otherwise a new control word must be loaded.

LOADING A TIME CONSTANT

An 8-bit time constant is loaded into the Time Constant register following a channel control word with bit 2 set. All zeros indicate a time constant of 256.



LOADING AN INTERRUPT VECTOR

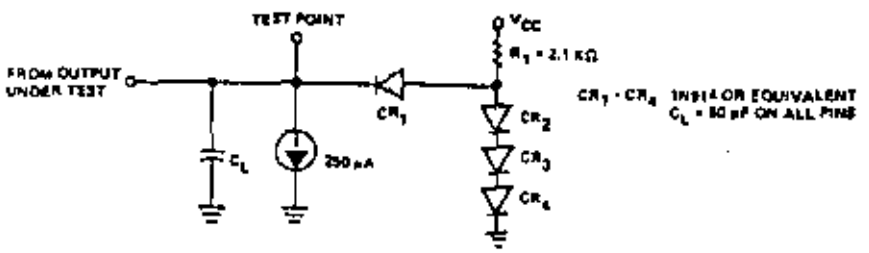
The Z80-CPU requires that an 8-bit interrupt vector be supplied by the interrupting channel. The CPU forms the address for the interrupt service routine of the channel using this vector. During an interrupt acknowledge cycle the vector is placed on the Z80 Data Bus by the highest priority channel requesting service at that time. The desired interrupt vector is loaded into the CTC by writing into channel Ø with a zero in DØ. D_7-D_3 contain the stored interrupt vector, D_2 and D_1 are not used in loading the vector. When the CTC responds to an interrupt acknowledge, these two bits contain the binary code of the highest priority channel which requested the interrupt and DØ contains a zero since the address of the interrupt service routine starts at an even byte. Channel Ø is the highest priority channel.

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

Signal	Symbol	Parameter	Min	Max	Unit	Comments
Φ	t_C	Clock Period	250	[1]	ns	
	$t_W(\Phi H)$	Clock Pulse Width, Clock High	105	2000	ns	
	$t_W(\Phi L)$	Clock Pulse Width, Clock Low	105	2000	ns	
	t_r, t_f	Clock Rise and Fall Times		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		ns	
CS, $\overline{CE}$, etc	$t_{S\Phi}(CS)$	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	60		ns	
D_0–D_7	$t_{DR}(D)$	Data Output Delay from Falling Edge of $\overline{RD}$ During Read Cycle		380	ns	[2]
	$t_{S\Phi}(D)$	Data Setup Time to Rising Edge of Φ During Write or M1 Cycle	50		ns	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of IORQ During INTA Cycle		180	ns	[2]
	$t_F(D)$	Delay to Floating Bus (Output Buffer Disable Time)		110	ns	
IEI	$t_S(IEI)$	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	140		ns	
IEO	$t_{DH}(IO)$	IEO Delay Time from Rising Edge of IEI		180	ns	[3]
	$t_{DL}(IO)$	IEO Delay Time from Falling Edge of IEI		130	ns	[3]
	$t_{DM}(IO)$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		190	ns	[3]
$\overline{IORQ}$	$t_{S\Phi}(IR)$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	115		ns	
$\overline{M1}$	$t_{S\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or M1 Cycle	90		ns	
$\overline{RD}$	$t_{S\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or M1 Cycle	115		ns	
$\overline{INT}$	$t_{DCK}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(\Phi) + 140$		Counter Mode
	$t_{D\Phi}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(\Phi) + 140$		Timer Mode
CLK/TRG_{0-3}	$t_C(CK)$	Clock Period	$2t_C(\Phi)$			Counter Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		50		
	$t_S(CK)$	Clock Setup Time to Rising Edge of Φ for Immediate Count	210			Counter Mode
	$t_S(TR)$	Trigger Setup Time to Rising Edge of Φ for enabling of Prescaler on Following Rising Edge of Φ	210			Timer Mode
	$t_W(CTH)$	Clock and Trigger High Pulse Width	200			Counter and Timer Modes
	$t_W(CTL)$	Clock and Trigger Low Pulse Width	200			Counter and Timer Modes
ZC/TO_{0-2}	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		190		Counter and Timer Modes
	$t_{DL}(ZC)$	ZC/TO Delay Time from Falling Edge of Φ, ZC/TO Low		190		Counter and Timer Modes

Notes: [1] $t_C = t_W(\Phi H) + t_W(\Phi L) + t_r + t_f$.
[2] Increase delay by 10 nsec for each 50 pF increase in loading, 200 pF maximum for data lines and 100 pF for control lines.
[3] Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum.
[4] $\overline{RESET}$ must be active for a minimum of 3 clock cycles.

OUTPUT LOAD CIRCUIT



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	−0.3 V to +7 V
Power Dissipation	0.8W

***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 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

TA = 0° C to 70° C, Vcc = 5 V ± 5% unless otherwise specified

Z80-CTC

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	.45	V	
V_{IHC}	Clock Input High Voltage [1]	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 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −250 μA
I_{CC}	Power Supply Current		120	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_{OHD}	Darlington Drive Current	−1.5		mA	V_{EXT} = 1.5V R_{EXT} = 390Ω

Z80A-CTC

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	.45	V	
V_{IHC}	Clock Input High Voltage [1]	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 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −250 μA
I_{CC}	Power Supply Current		120	mA	T_C = 250 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_{OHD}	Darlington Drive Current	−1.5		mA	V_{EXT} = 1.5V R_{EXT} = 390Ω

Capacitance

TA = 25° C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_ϕ	Clock Capacitance	20	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

Z80®-SIO
Z80A-SIO

Product Specification

AUGUST 1978

The Z80-SIO (Serial Input/Output) is a dual-channel multi-function peripheral component designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic function is a serial-to-parallel, parallel-to-serial converter/controller, but—within that role—it is configurable by systems software so its "personality" can be optimized for a given serial data communications application.

The Z80-SIO is capable of handling asynchronous formats, synchronous byte-oriented protocols such as IBM Bisync, and synchronous bit-oriented protocols such as HDLC and SDLC. This versatile device can also be used to support virtually any other serial protocol for applications other than data communications (cassette or floppy disk interfaces, for example).

The Z80-SIO can generate and check CRC codes in any synchronous mode and can be programmed to check data integrity in various modes. The device also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

N-channel silicon-gate depletion-load technology

40-pin DIP

Single 5 V power supply

Single-phase 5 V clock

All inputs and outputs TTL compatible

Two independent full-duplex channels

Data rates in synchronous or isosynchronous modes:

- 0–550K bits/second with 2.5 MHz system clock rate
- 0–880K bits/second with 4.0 MHz system clock rate

Receiver data registers quadruply buffered; transmitter doubly buffered.

Asynchronous features:

- 5, 6, 7 or 8 bits/character

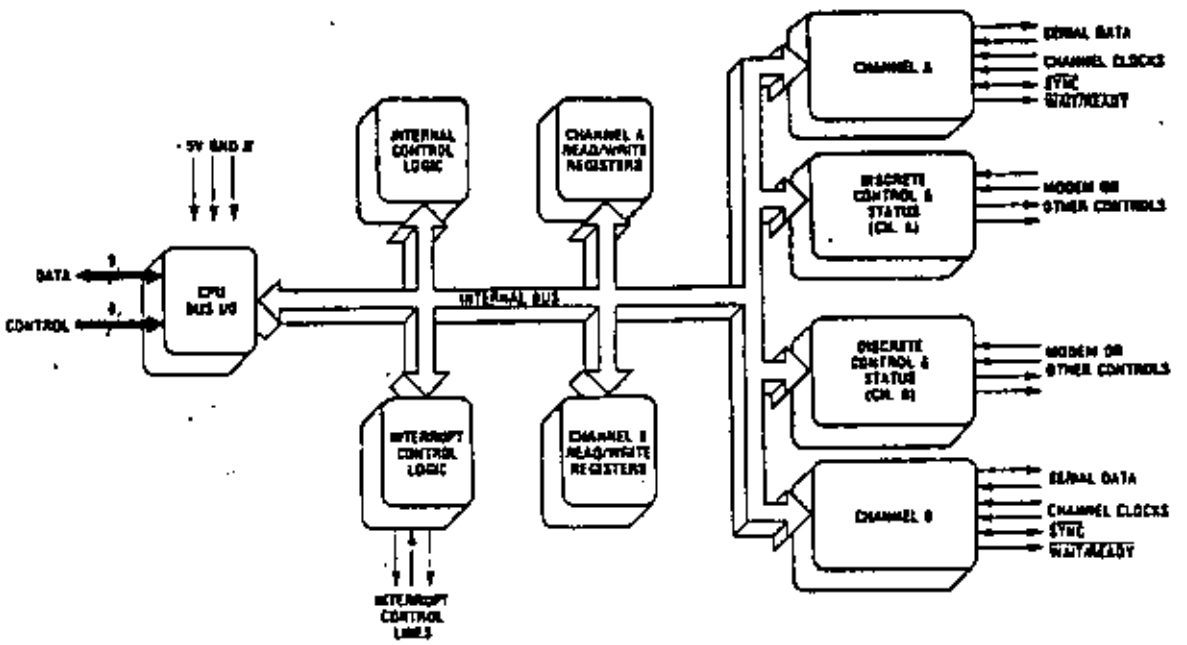


Figure 1. Z80-SIO Block Diagram

and $\overline{IORQ}$ as an interrupt acknowledge if the Z80-SIO is the highest priority device that has interrupted the Z80-CPU.

$\overline{IORQ}$. *Input/Output Request* (input from CPU, active Low). $\overline{IORQ}$ is used in conjunction with B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ and $\overline{RD}$ to transfer commands and data between the CPU and the Z80-SIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are all active, the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active, but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information as specified by C/$\overline{D}$. As mentioned previously, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the Z80-SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{RD}$. *Read Cycle Status.* (input from CPU, active Low). If $\overline{RD}$ is active, a memory or I/O read operation is in progress. $\overline{RD}$ is used with B/$\overline{A}$, $\overline{CE}$ and $\overline{IORQ}$ to transfer data from the Z80-SIO to the CPU.

$\overline{RESET}$. *Reset* (input, active Low). A Low $\overline{RESET}$ disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the Z80-SIO is reset and before data is transmitted or received.

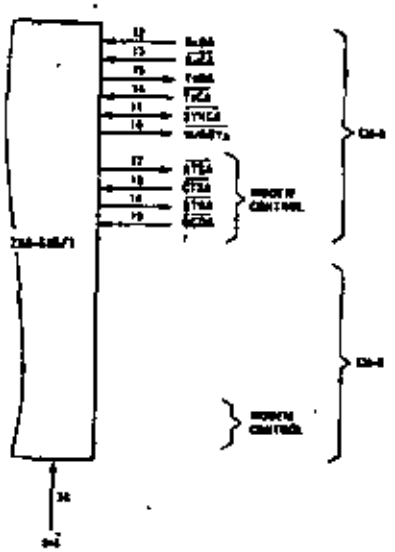
IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80-SIO. 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 the Z80-SIO is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{W/RDYA}$, $\overline{W/RDYB}$. *Wait/Ready A, Wait/Ready B* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80-SIO data rate. The reset state is open drain.

$\overline{CTSA}$, $\overline{CTSB}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime signals. The Z80-SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger buffering does not guarantee a specified noise-level margin.



Z80-SIO/1 Pin Configuration

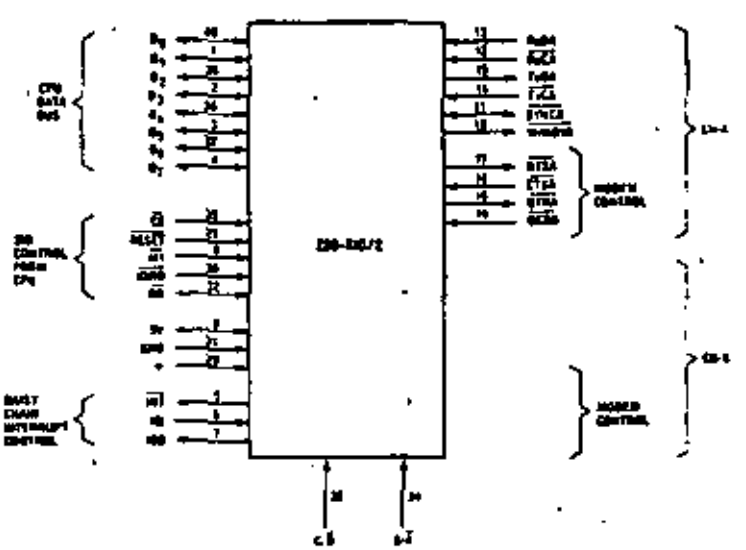


Figure 4. Z80-SIO/2 Pin Configuration

WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and control
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

Write Register Functions

Table 1. Functional Assignments of Read and Write Registers

The logic for both channels provides formats, synchronization and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send ($\overline{CTS}$) and Data Carrier Detect ($\overline{DCD}$) are monitored by the discrete control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/Status interrupts are prioritized in that order within each channel.

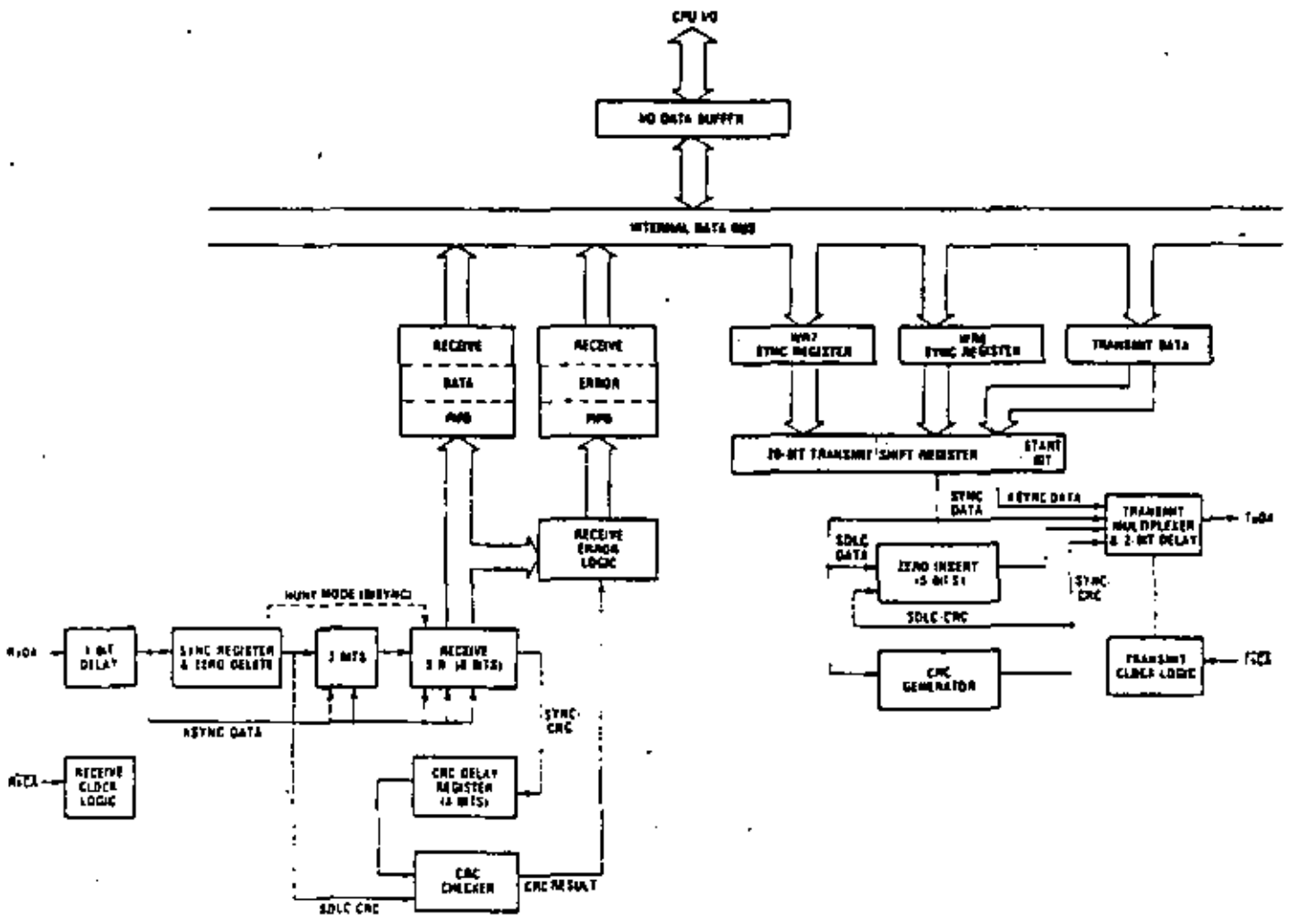


Figure 5. Transmit and Receive Data Path

the accurate timing of the Break/Abort condition in external logic.

CPU/DMA Block Transfer. The Z80-SIO provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers (Z80-DMA or other designs). The Block Transfer mode uses the $\overline{\text{WAIT/READY}}$ output in conjunction with the Wait/Ready bits of Write Register 1. The $\overline{\text{WAIT/READY}}$ output can be defined under software control as a $\overline{\text{WAIT}}$ line in the CPU Block Transfer mode or as a $\overline{\text{READY}}$ line in the DMA Block Transfer mode.

To a DMA controller, the Z80-SIO READY output indicates that the Z80-SIO is ready to transfer data to or from memory. To the CPU, the WAIT output indicates that the Z80-SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle. The programming of bits 5, 6 and 7 of Write Register 1 and the logic states of the $\overline{\text{WAIT/READY}}$ line are defined in the Write Register 1 description (Z80-SIO Programming section).

In addition to the I/O capabilities previously discussed, the Z80-SIO provides two independent full-duplex channels that can be programmed for use in Asynchronous, Synchronous and SDLC (HDLC) modes. These different modes are provided to facilitate the implementation of commonly used data communications protocols. The following is a short description of the data communications protocols supported by the Z80-SIO. A more detailed explanation of these modes can be found in the *Z80-SIO Technical Manual*.

Asynchronous Modes. The Z80-SIO offers transmission and reception of five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one and a half or two stop bits per character and can provide a break output at any time. The receiver break detection logic interrupts the CPU only at the start and end of a received break. Reception is protected from spikes by a transient spike rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the Receive Data input. If the Low does not persist—as in the case of a transient—the character assembly process is not started.

Framing errors and overrun errors are detected and buffered together with the partial character on which they occurred. Vectored interrupts allow fast servicing of error conditions using dedicated routines. Furthermore, a built-in checking process avoids interpreting a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit is begun.

The Z80-SIO does not require symmetric Transmit and Receive Clock signals—a feature that allows it to be used with a Z80-CTC or any other clock source. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32 or 1/64 of the clock rate supplied to the Receive and Transmit Clock inputs.

In Asynchronous modes, the $\overline{\text{SYNC}}$ pin may be programmed for an input that can be used for functions such as monitoring a ring indicator.

Synchronous Modes. The Z80-SIO supports both byte-oriented and bit-oriented synchronous communication. Synchronous byte-oriented protocols can be handled in several modes that allow character synchronization with an 8-bit sync character (Monosync), any 16-bit sync pattern (Bisync), or with an external sync signal. Leading sync characters can be removed without interrupting the CPU. CRC checking for synchronous byte-oriented modes is delayed by one character time so the CPU may disable CRC checking on specific characters. This permits implementation of protocols such as IBM Bisync.

Both CRC-16 ($X^{16}+X^{15}+X^2+1$) and CCITT ($X^{16}+X^{12}+X^5+1$) error checking polynomials are supported. In all non-SDLC modes, the CRC generator is initialized to 0's; in SDLC modes, it is initialized to 1's. (This means that the Z80-SIO cannot generate or check CRC for IBM-compatible soft-sectored disks.) The Z80-SIO also provides a feature that automatically transmits CRC data when no other data is available for transmission. This allows very high-speed transmissions under DMA control with no need for CPU intervention at the end of a message. When there is no data or CRC to send in Synchronous modes, the transmitter inserts 8- or 16-bit sync characters regardless of the programmed character length. Since the CPU can read status information from the Z80-SIO, it can determine the type of transmission (data, CRC or sync characters) that is taking place at any time.

The Z80-SIO supports synchronous bit-oriented protocols such as SDLC and HDLC by performing automatic flag sending, zero insertion and CRC generation. A special command can be used to abort a frame in transmission. The Z80-SIO automatically transmits the CRC and trailing flag when the transmit buffer becomes empty. An interrupt warns the CPU of this status change so an abort may be issued if a transmitter underrun has occurred. One to eight bits per character can be sent, which allows transmission of a message exactly as received with no prior information about the character structure in the information field of a frame.

The receiver automatically synchronizes on the leading flag of a frame and provides a synchronization signal that can be programmed to interrupt. In addition, an interrupt on the first received character or on every character can be selected. The receiver automatically deletes all zeroes inserted by the transmitter during character assembly. It also calculates and automatically checks the CRC to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers. The receiver can be programmed to search for frames addressed to only a specified user-selectable address or to a global broadcast address. In this mode, frames that do not match the user-

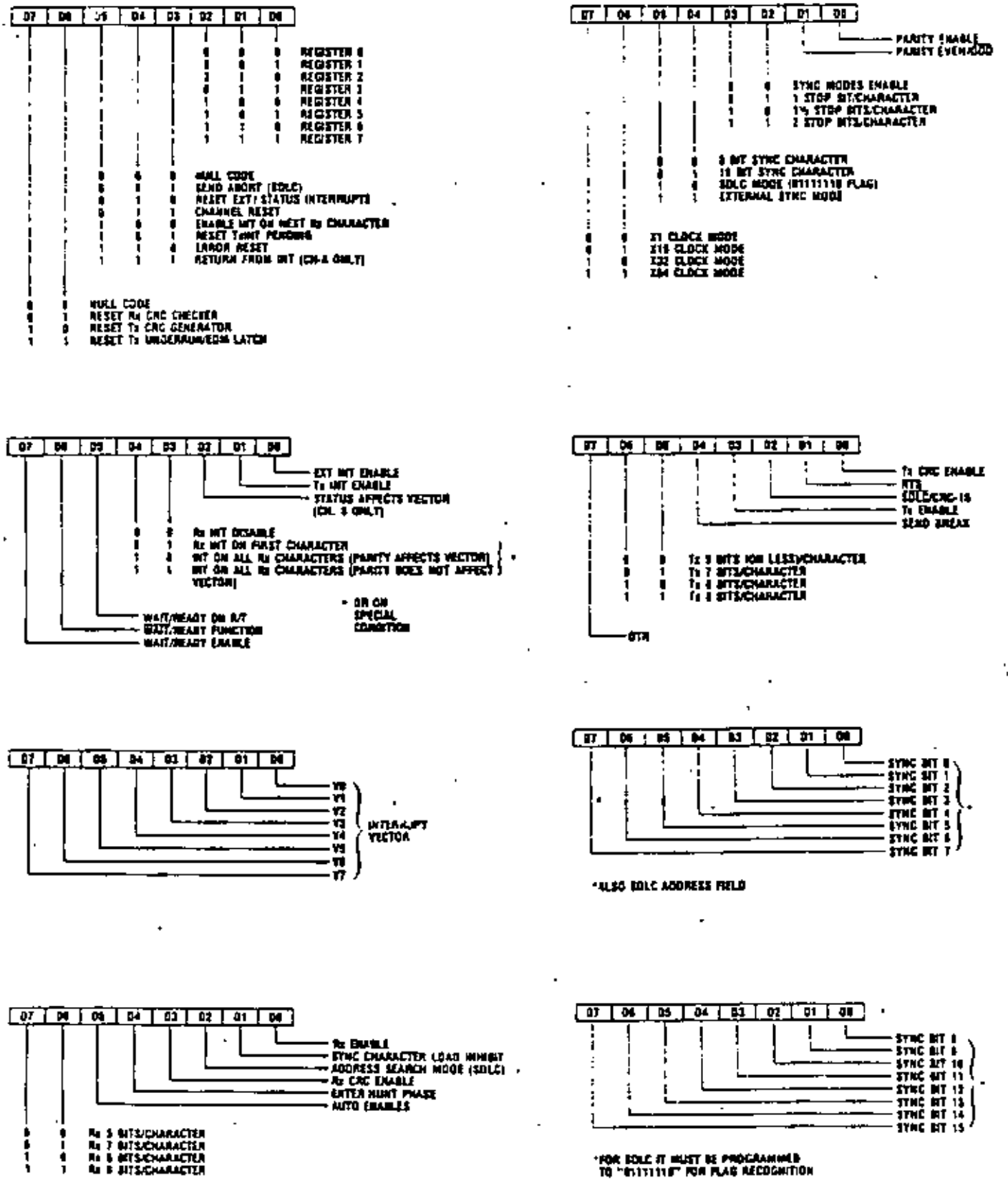


Figure 7. Write Register Bit Functions

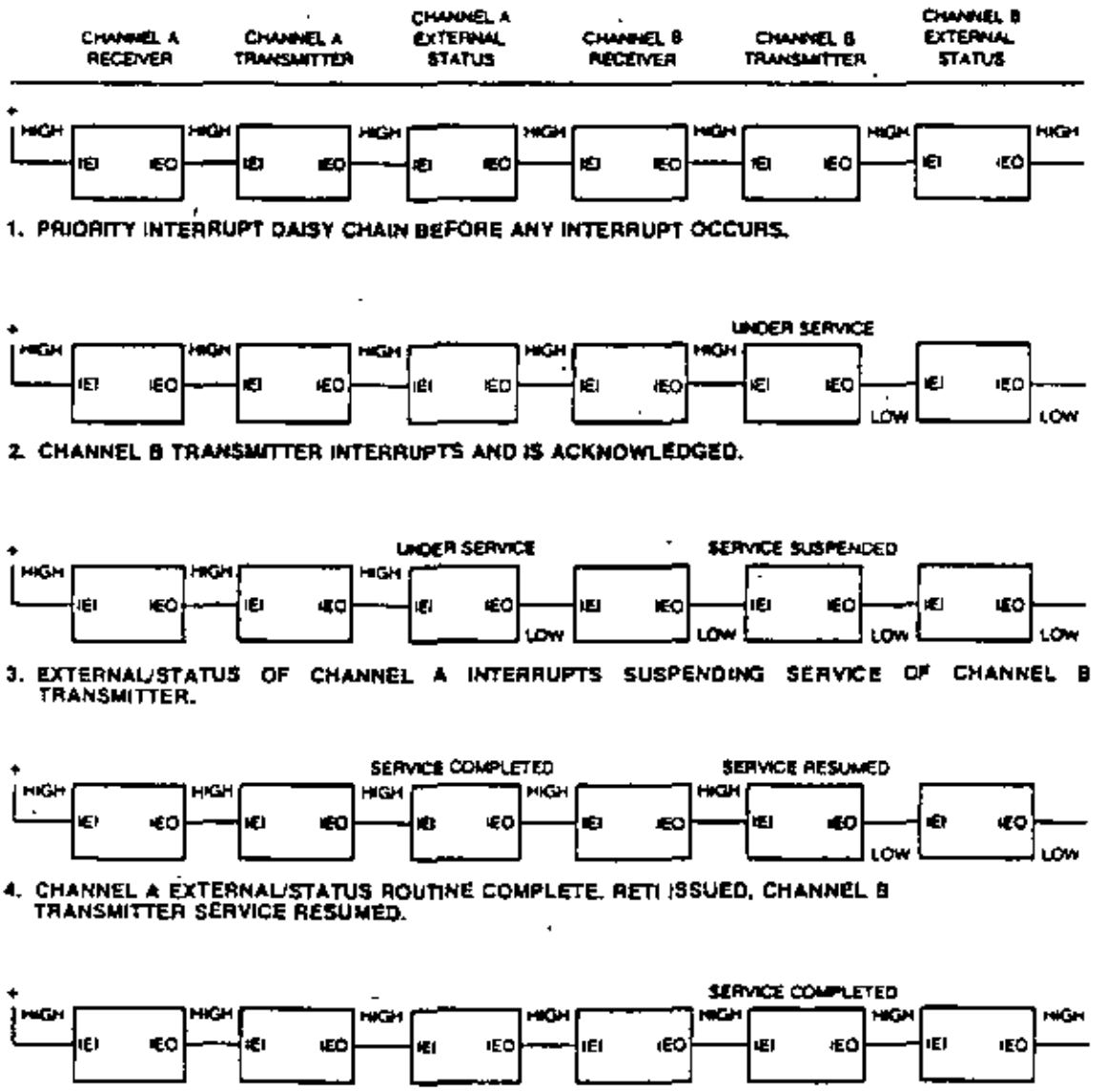
High and an IEO Low. If the next opcode byte is "4D," the interrupt-under-service latch is reset.

The ripple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-look-ahead, or by extending the interrupt acknowledge cycle. For further information about techniques for increasing the number of daisy-chained devices, refer to Zilog Application Note 03-0041-01 (*The Z80 Family Program Interrupt Structure*).

Figure 9 illustrates the daisy chain configuration of interrupt circuits and their behavior with nested interrupts (an interrupt that is interrupted by another with a higher priority).

Each box in the illustration could be a separate external Z80 peripheral circuit with a user-defined order of interrupt priorities. However, a similar daisy chain structure also exists inside the Z80-SIO, which has six interrupt levels with a fixed order of priorities.

The case illustrated occurs when the transmitter of Channel B interrupts and is granted service. While this interrupt is being serviced, it is interrupted by a higher priority interrupt from Channel A. The second interrupt is serviced and—upon completion—a RETI instruction is executed or a RETI command is written into the Z80-SIO, resetting the interrupt-under-service latch of the Channel A interrupt. At this time, the service rotine for Channel B is resumed. When it is completed, another RETI instruction is executed to complete the interrupt service.



1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.

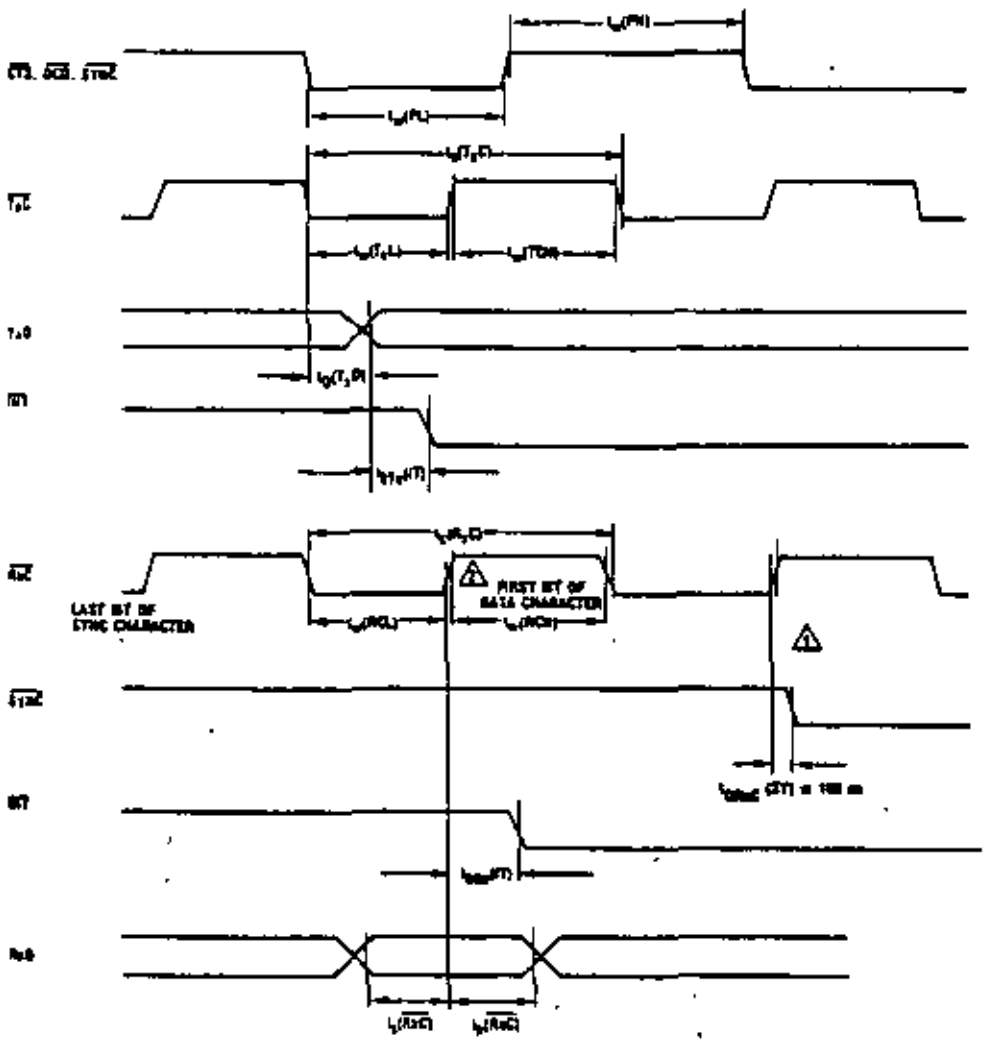
2. CHANNEL B TRANSMITTER INTERRUPTS AND IS ACKNOWLEDGED.

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 SERVICE ROUTINE COMPLETE, SECOND RETI ISSUED.

Figure 9. Typical Interrupt Sequence

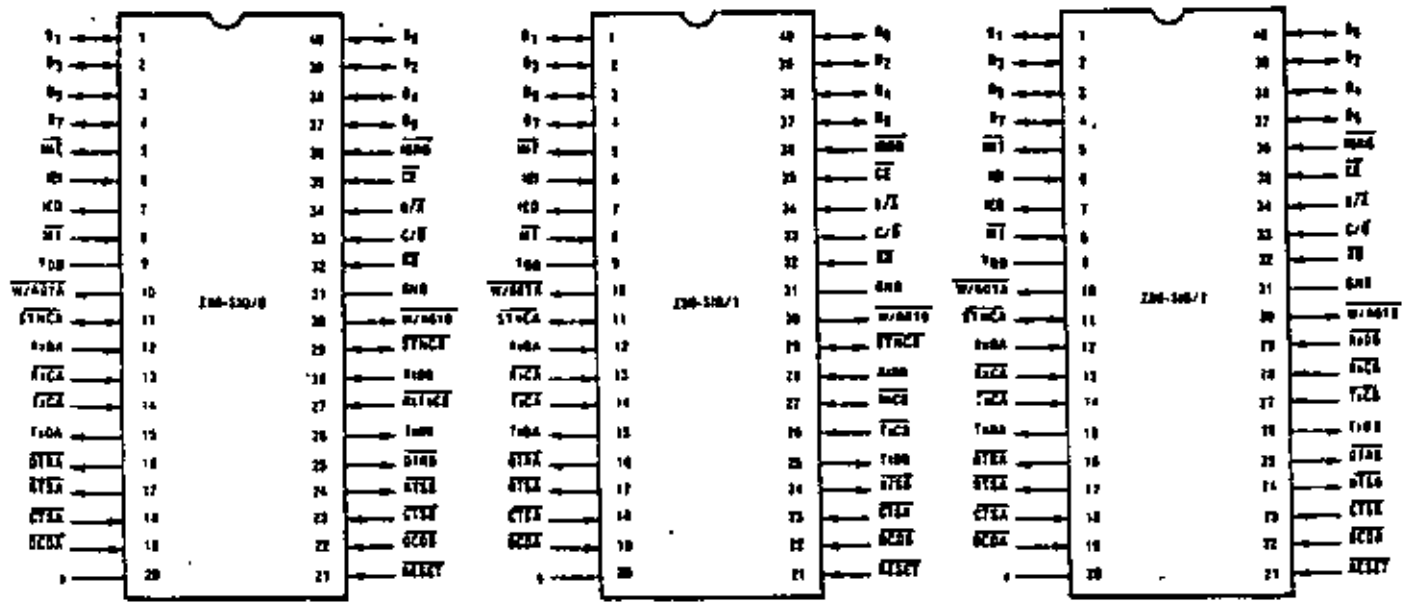


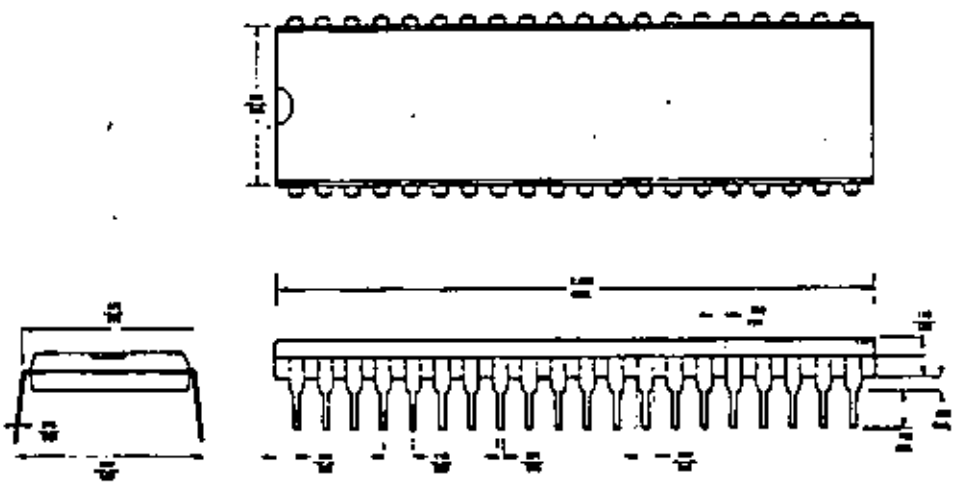
NOTES:

1. The $\overline{SYNC}$ inputs must be driven Low on the rising edge of $\overline{RxC}$ delayed two complete clock cycles from the last bit of the sync character.
2. Data character assembly begins on the next Receive Clock cycle after the last bit of the sync character is received.

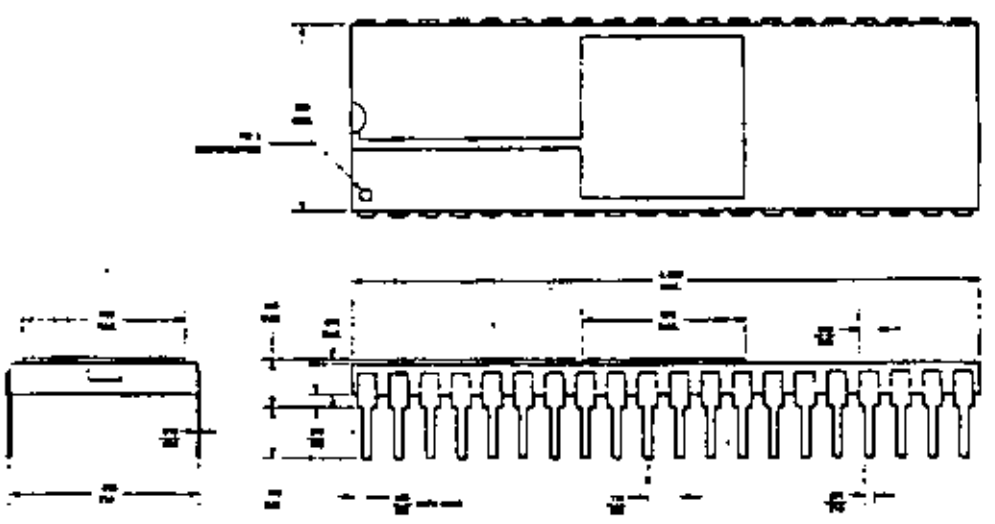
Signal	Symbol	Parameter	Z80-SIO Min	Z80-SIO Max	Z80A-SIO Min	Z80A-SIO Max	Unit
$\overline{INT}$	$t_{cx}(IT)$	INT Delay Time from rising edge of RxC	10	13	10	13	ϕ periods
	$t_{rx}(IT)$	INT Delay Time from transition of Xmit Data Bit	1	8	5	9	ϕ periods
$\overline{CTSA}$, $\overline{CTSB}$, $\overline{DCDA}$, $\overline{DCDB}$, $\overline{SYNCA}$, $\overline{SYNCB}$	$t_w(PH)$	Minimum HIGH Pulse Width for latching state into register and generating interrupt	200		200		ns
	$t_w(PL)$	Minimum LOW Pulse Width for latching state into register and generating interrupt	200		200		ns
$\overline{SYNCA}$, $\overline{SYNCB}$	$t_{cx}(SY)$	Sync Pulse Delay Time from rising edge of $\overline{RxC}$ Output Modes	4	7	4	7	ϕ periods
	$t_{wx}(SY)$	Sync Pulse Delay Time from rising edge of $\overline{RxC}$ External Sync Mode		100		100	ns
$\overline{TxCA}$, $\overline{TxCB}$	$t_c(TxC)$	Transmit Clock Period	400	*	400	*	ns
	$t_w(TCH)$	Transmit Clock Pulse Width, clock HIGH	180	*	180	*	ns
	$t_w(TCL)$	Transmit Clock Pulse Width, clock LOW	180	*	180	*	ns
TxDA, TxDB†	$t_d(TxD)$	TxD Output Delay from falling Edge of $\overline{TxC}$ (x1 Clock Mode)		400		200	ns
$\overline{RxCA}$, $\overline{RxCB}$	$t_c(RxC)$	Receive Clock Period	400	*	400	*	ns
	$t_w(RCH)$	Receive Clock Pulse Width, clock HIGH	180	*	180	*	ns
	$t_w(RCL)$	Receive Clock Pulse Width, clock LOW	180	*	180	*	ns
RxDA, RxDB†	$t_s(RxC)$	Setup Time to rising edge of $\overline{RxC}$, x1 mode	0		0		ns
	$t_h(RxC)$	Hold Time from rising edge of $\overline{RxC}$, x1 mode	140		140		ns

†In all speeds, the system clock (ϕ) rate must be at least 4.5 times the maximum data rate. $\overline{RESET}$ must be active a minimum of one complete ϕ cycle.





40-Pin Plastic



40-Pin Ceramic

Z80®-CPU
Z80A-CPU

Product Specification

MARCH 1978

The Zilog 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.

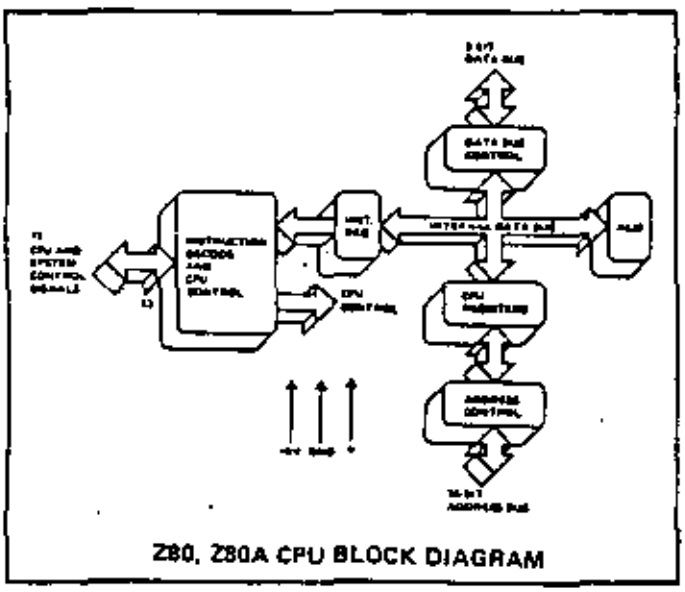
The Z80 and Z80A CPU's are third generation single chip microprocessors with unrivaled computational power. This increased computational power results in higher system through-put and more efficient memory utilization when compared to second generation microprocessors. In addition, the Z80 and Z80A CPU's are very easy to implement into a system because of their single voltage requirement plus all output signals are fully decoded and timed to control standard memory or peripheral circuits. The circuit is implemented using an N-channel, ion implanted, silicon gate MOS process.

Figure 1 is a block diagram of the CPU, Figure 2 details the internal register configuration which contains 208 bits of Read/Write memory that are accessible to the programmer. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or as 16-bit register pairs. There are also two sets of accumulator and flag registers. The programmer has access to either set of main or alternate registers through a group of exchange instructions. This alternate set allows foreground/background mode of operation or may be reserved for very fast Interrupt response. Each CPU also contains a 16-bit stack pointer which permits ample implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data handling.

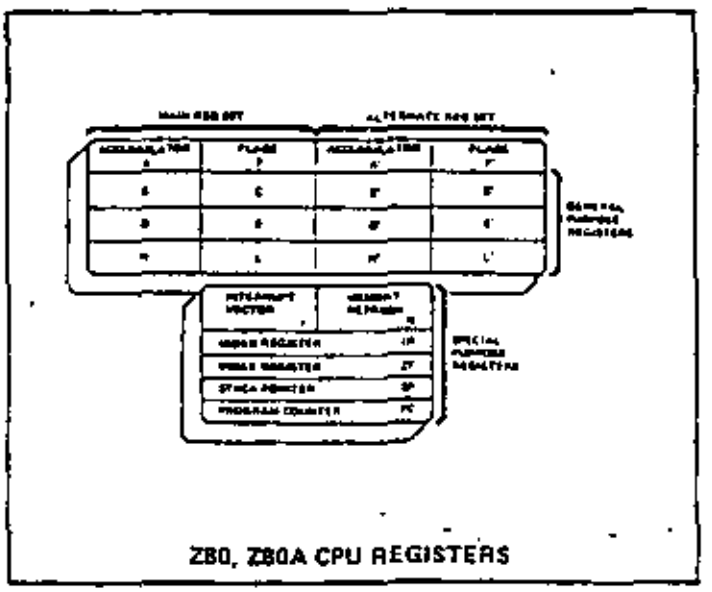
The two 16-bit index registers allow tabular data manipulation and easy implementation of relocatable code. The Refresh register provides for automatic, totally transparent refresh of external dynamic memories. The I register is used in a powerful interrupt response mode to form the upper 8 bits of a pointer to a interrupt service address table, while the interrupting device supplies the lower 8 bits of the pointer. An indirect call is then made to this service address.

FEATURES

- Single chip, N-channel Silicon Gate CPU.
- 158 instructions—includes all 78 of the 8080A instructions with total software compatibility. New instructions include 4-, 8- and 16-bit operations with more useful addressing modes such as indexed, bit and relative.
- 17 internal registers.
- Three modes of fast interrupt response plus a non-maskable interrupt.
- Directly interfaces standard speed static or dynamic memories with virtually no external logic.
- 1.0 μs instruction execution speed.
- Single 5 VDC supply and single-phase 5 volt Clock.
- Out-performs any other single chip microcomputer in 4-, 8-, or 16-bit applications.
- All pins TTL Compatible
- Built-in dynamic RAM refresh circuitry.



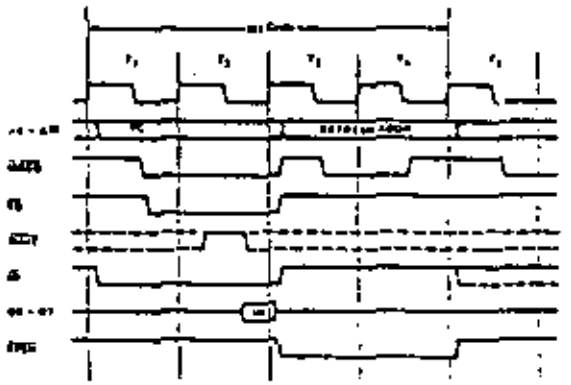
Z80, Z80A CPU BLOCK DIAGRAM



Z80, Z80A CPU REGISTERS

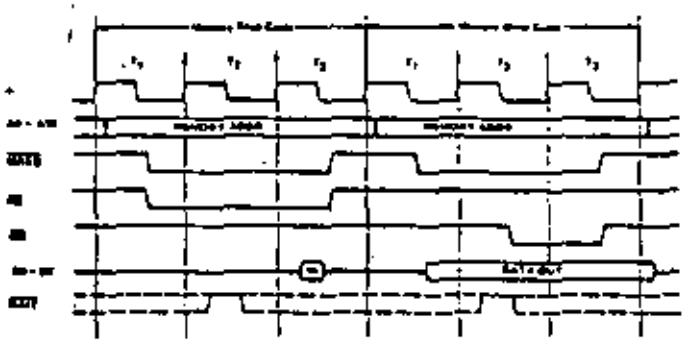
INSTRUCTION OP CODE FETCH

The program counter content (PC) is placed on the address bus immediately at the start of the cycle. One half clock time later $\overline{MREQ}$ goes active. The falling edge of $\overline{MREQ}$ can be used directly as a chip enable to dynamic memories. $\overline{RD}$ when active indicates that the memory data should be enabled onto the CPU data bus. The CPU samples data with the rising edge of the clock state T_3. Clock states T_3 and T_4 of a fetch cycle are used to refresh dynamic memories while the CPU is internally decoding and executing the instruction. The refresh control signal $\overline{RFSH}$ indicates that a refresh read of all dynamic memories should be accomplished.



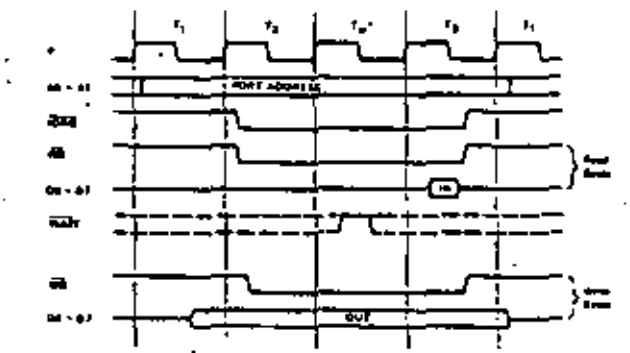
MEMORY READ OR WRITE CYCLES

Illustrated here is the timing of memory read or write cycles other than an OP code fetch (M_1 cycle). The $\overline{MREQ}$ and $\overline{RD}$ signals are used exactly 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.



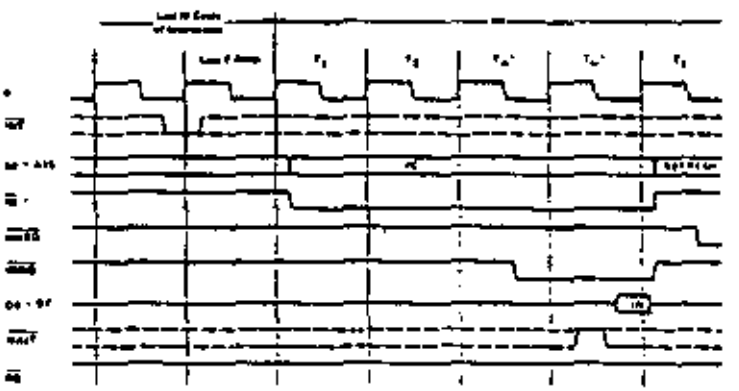
INPUT OR OUTPUT CYCLES

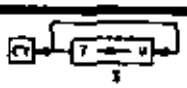
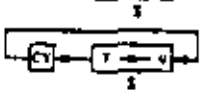
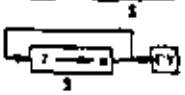
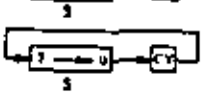
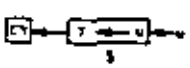
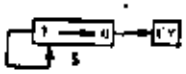
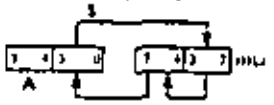
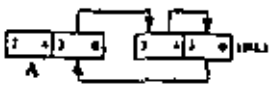
Illustrated here is the timing for an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted (Tw*). The reason for this is that during I/O operations this extra state allows sufficient time for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required.



INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

The interrupt signal is sampled by the CPU with the rising edge of the last clock at the end of any instruction. When an interrupt is accepted, a special M_1 cycle is generated. During this M_1 cycle, the $\overline{IORQ}$ signal becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Two wait states (Tw*) are automatically added to this cycle so that a ripple priority interrupt scheme, such as the one used in the Z80 peripheral controllers, can be easily implemented.

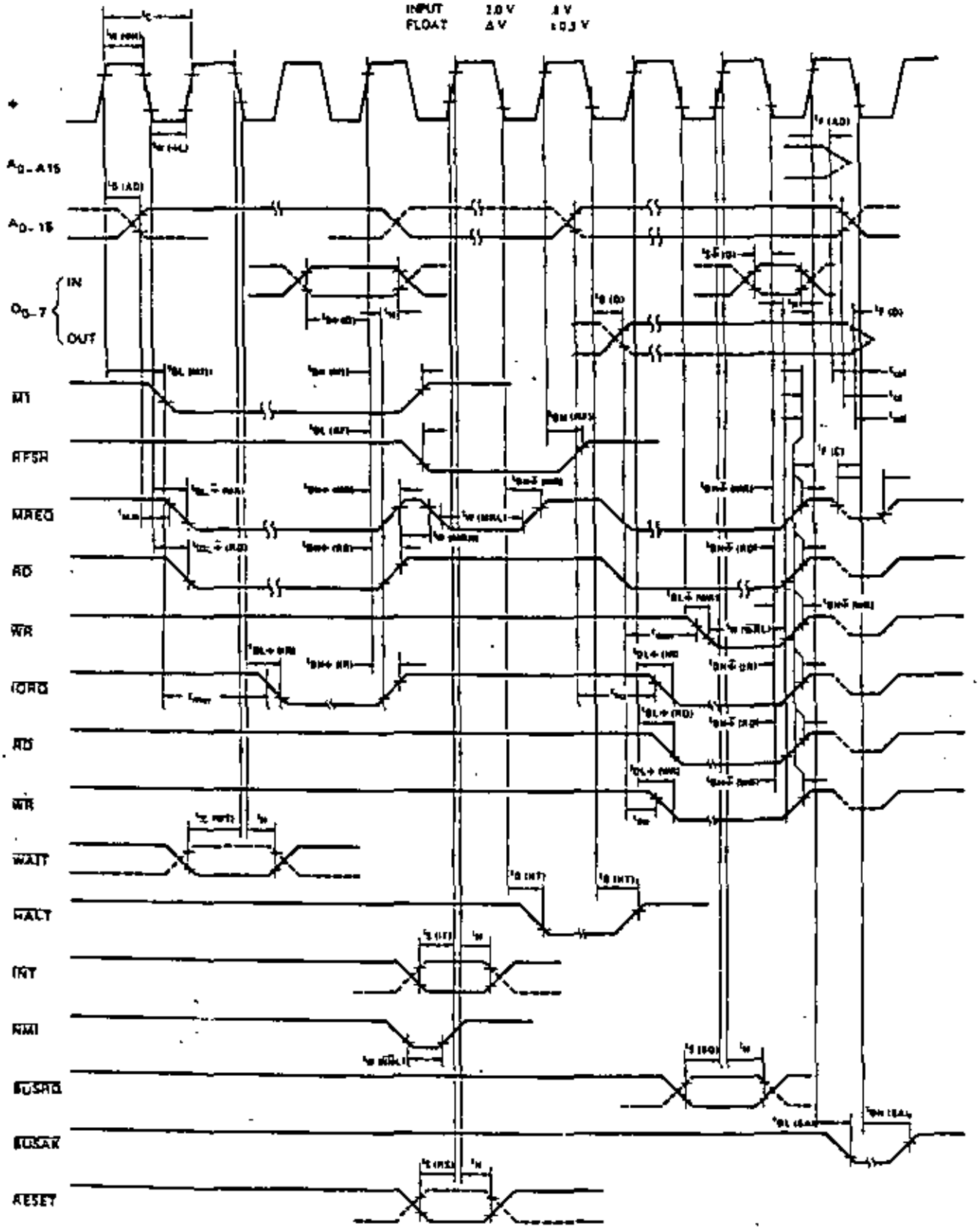


	Mnemonic	Symbolic Operation	Comments
8-BIT ALU	CP s	A - s	s = r, n (HL) (IX+e), (IY+e)
	INC d	d ← d + 1	d = r, (HL) (IX+e), (IY+e)
	DEC d	d ← d - 1	
16-BIT ARITHMETIC	ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE HL, SP
	ADC HL, ss	HL ← HL + ss + CY	
	SBC HL, ss	HL ← HL - ss - CY	
	ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP
	ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
	INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
	DEC dd	dd ← dd - 1	dd ≡ BC, DE, HL, SP, IX, IY
GP ACC & FLAG	DAA	Converts A contents into packed BCD following add or subtract	Operands must be in packed BCD format
	CPL	$A \leftarrow \overline{A}$	
	NEG	A ← 00 - A	
	CCF	$CY \leftarrow \overline{CY}$	
	SCF	CY ← 1	
MISCELLANEOUS	NOP	No operation	
	HALT	Halt CPU	
	DI	Disable Interrupts	
	EI	Enable Interrupts	
	IM 0	Set interrupt mode 0	8080A mode
	IM 1	Set interrupt mode 1	Call to 0038_H
	IM 2	Set interrupt mode 2	Indirect Call
ROTATES AND SHIFTS	RLC s		s ≡ r, (HL) (IX+e), (IY+e)
	RL s		
	RRC s		
	RR s		
	SLA s		
	SRA s		
	SRL s		
	RLD		
	RRD		

	Mnemonic	Symbolic Operation	Comments
BIT S, R, & T	BIT b, s	$Z \leftarrow \overline{s_b}$	Z is zero flag
	SET b, s	$s_b \leftarrow 1$	s ≡ r, (HL) (IX+e), (IY+e)
	RES b, s	$s_b \leftarrow 0$	
INPUT AND OUTPUT	IN A, (n)	A ← (n)	
	IN r, (C)	r ← (C)	Set flags
	INI	(HL) ← (C), HL ← HL + 1 B ← B - 1	
	INIR	(HL) ← (C), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	IND	(HL) ← (C), HL ← HL - 1 B ← B - 1	
	INDR	(HL) ← (C), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
	OUT(n), A	(n) ← A	
	OUT(C), r	(C) ← r	
	OUTI	(C) ← (HL), HL ← HL + 1 B ← B - 1	
	OTIR	(C) ← (HL), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	OUTD	(C) ← (HL), HL ← HL - 1 B ← B - 1	
	OTDR	(C) ← (HL), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
JUMPS	JP nn	PC ← nn	
	JP cc, nn	If condition cc is true PC ← nn, else continue	cc ≡ NZ PO, Z PE, NC P, C M
	JR e	PC ← PC + e	
	JR kk, e	If condition kk is true PC ← PC + e, else continue	kk ≡ NZ NC, Z C
	JP (ss)	PC ← ss	ss = HL, IX, IY
	DJNZ e	B ← B - 1, if B = 0 continue, else PC ← PC + e	
CALLS	CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, PC ← nn	
	CALL cc, nn	If condition cc is false continue, else same as CALL nn	cc ≡ NZ PO, Z PE, NC P, C M
RESTARTS	RST L	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow 0$ $PC_L \leftarrow L$	
RETURNS	RET	$PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$	
	RET cc	If condition cc is false continue, else same as RET	cc ≡ NZ PO, Z PE, NC P, C M
	RETI	Return from interrupt, same as RET	
	RETN	Return from non-maskable interrupt	

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

	"1"	"0"
CLOCK	V_{CC} -.6V	.45V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	Δ V	±0.5 V



T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	250	[12]	nsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	110	[13]	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	110	2000	nsec	
	t_r, t_f	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{d(A)}$	Address Output Delay		110	nsec	C_L = 50pF
	$t_{f(A)}$	Delay to Float		90	nsec	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[3]		nsec	
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{d(D)}$	Data Output Delay		150	nsec	C_L = 50pF
	$t_{f(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{su(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	[5]		nsec	
	$t_{su(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	50		nsec	
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[6]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[7]		nsec	
	t_{daf}	Data Stable From $\overline{WR}$	[7]			
	t_H	Any Hold Time for Setup Time		0	nsec	
$\overline{MREQ}$	$t_{dL(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		85	nsec	C_L = 50pF
	$t_{dH(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		85	nsec	
	$t_{dH(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		85	nsec	
	$t_w(\overline{MRL})$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	$t_w(\overline{MRH})$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{dL(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		75	nsec	C_L = 50pF
	$t_{dL(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		85	nsec	
	$t_{dH(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		85	nsec	
	$t_{dH(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		85	nsec	
$\overline{RD}$	$t_{dL(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		85	nsec	C_L = 50pF
	$t_{dL(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		95	nsec	
	$t_{dH(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		85	nsec	
	$t_{dH(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		85	nsec	
$\overline{WR}$	$t_{dL(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		65	nsec	C_L = 50pF
	$t_{dL(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		80	nsec	
	$t_{dH(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		80	nsec	
	$t_w(\overline{WRL})$	Pulse Width, $\overline{WR}$ Low	[10]		nsec	
$\overline{M1}$	$t_{dL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ Low		100	nsec	C_L = 50pF
	$t_{dH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		100	nsec	
$\overline{RFSH}$	$t_{dL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		130	nsec	C_L = 50pF
	$t_{dH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		120	nsec	
$\overline{WAIT}$	$t_s(WT)$	$\overline{WAIT}$ 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	
$\overline{NMI}$	$t_w(\overline{NMIL})$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_s(BQ)$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	50		nsec	
$\overline{BUSAK}$	$t_{dL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		100	nsec	C_L = 50pF
	$t_{dH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		120	nsec	
$\overline{RESET}$	$t_s(RS)$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	60		nsec	
	$t_{f(C)}$	Delay to Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		80	nsec	
	t_{mr}	M1 Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	[11]		nsec	

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 65$

[2] $t_{aci} = t_c - 70$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 50$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 45$

[5] $t_{dcm} = t_c - 170$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 170$

[7] $t_{daf} = t_{w(\Phi L)} + t_r - 70$

[8] $t_w(\overline{MRL}) = t_c - 30$

[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_r - 20$

[10] $t_{w(\overline{WRL})} = t_c - 30$

[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f + 65$

[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$

NOTES:

A. 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.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

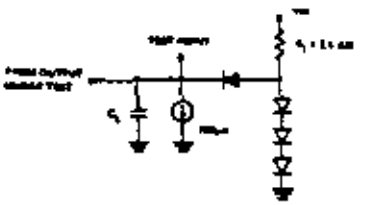
C. The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance

TA = 70°C Vcc = +5V ±5%

Add 10nsec delay for each 50pf increase in load up to maximum of 200pf for data bus and 100pf for address & control lines.

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 usec maximum



Load circuit for Output

APPENDIX C

16-BIT MICROPROCESSOR FAMILY

INTEGRATED COMPUTER SYSTEMS™

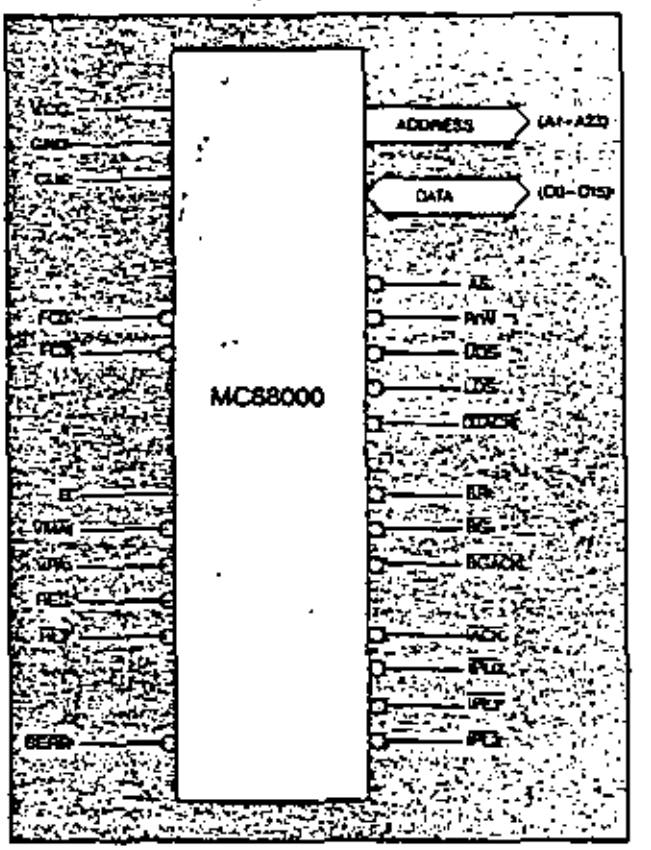
16-BIT MICROPROCESSOR FAMILY EXAMPLES

- MC - 68000 FAMILY
- Z - 8000 CPU
- Z - 8010 MMU
- Z - 8034 UPC
- Z - 8036 CIO
- Z - 8030 SCC

INTRODUCING THE MC68000 ...

MOTOROLA'S ADVANCED COMPUTER SYSTEM ON SILICON

The MC68,000 microprocessor is housed in a 64-pin package that allows the use of separate (non-multiplexed) address and data buses. This large package provides optimum flexibility while at the same time maximizing bus through-put.



PIN IDENTIFICATION & DEFINITIONS

A1-A23	Address Leads	23-bit address bus; capable of addressing 16,777,216 bytes in conjunction with UDS and LDS.
D0-D15	Data Leads	16-bit data bus; transfers 8 or 16 bits of information.
AS	Address Strobe	Indicates valid address & provides a bus lock for indivisible operations.
R/W	Read/Write	Defines bus operation as Read or Write and controls external bus buffers.
UDS, LDS	Data Strobes	Identifies the byte(s) to be operated on according to R/W and AS.
DTACK	Data Transfer Acknowledge	Allows the bus cycle to synchronize with slow devices or memories.
BR	Bus Request	Input to the Processor from a device requesting the bus.
BG	Bus Grant	Output from the processor granting bus arbitration.
BGACK	Bus Grant Acknowledge	Confirmation signal from BG indicating a valid selection from the arbitration process.
IACK	Interrupt Acknowledge	Identifies that the bus is performing an interrupt service cycle.
IPL0, IPL1, IPL2	Interrupt Priority Level	Provides the priority level of the interrupting function to the processor.
FC0, FC1	Function Code	Provides external devices with information about the current bus cycle.
CLK	Clock	Master TTL input clock to the processor.
RES	Reset	Provides reset (initialization) signal to the processor and peripheral devices.
HLT	Halt	Stops the processor and allows single stepping.
BERR	Bus Error	Provides termination of a bus cycle if no response or an invalid response is received.
E	Enable	Enable clock for M6800 systems.
VPA	Valid Peripheral Address	Identifies addressed area as a 6800 compatible area.
VMA	Valid Memory Address	Indicates to 6800 family devices that a valid address is on the bus.
V_{CC}	+5 Volts	—
GND	Ground (2 pins)	—

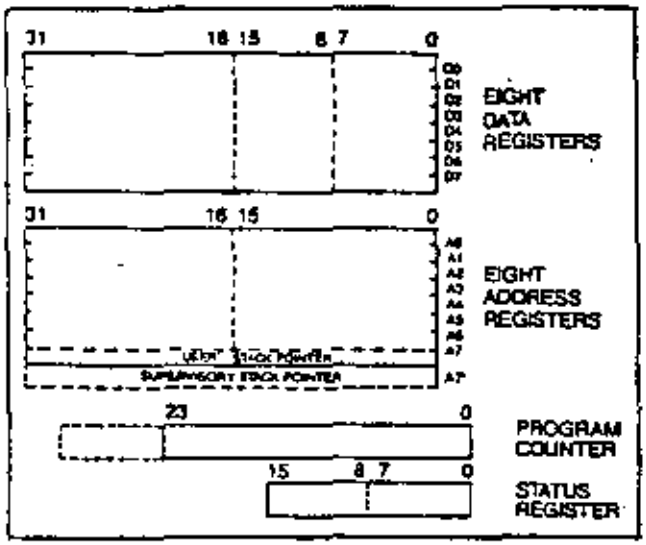


Figure 2: MC68000 Programming Model

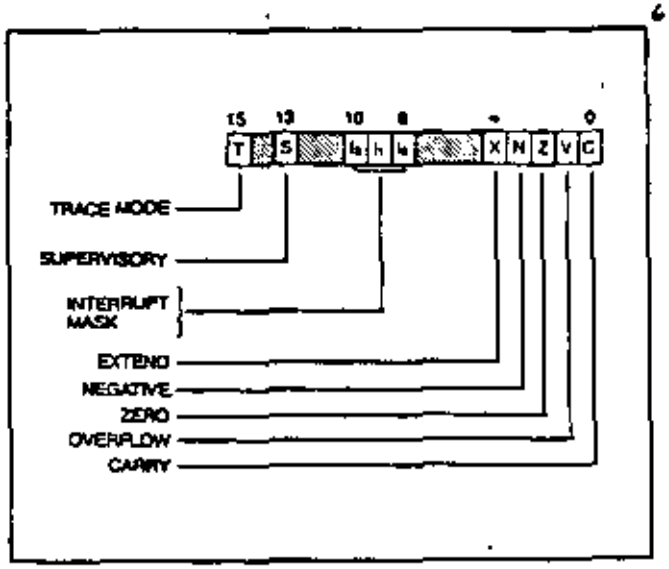


Figure 3: MC68000 Status Register

THE MC68000 CPU

Advanced architecture processors must not only offer efficient solutions to large complex problems but must be able to handle the small, simple problems with proportional efficiency. The CPU has been designed to offer the maximum in performance and versatility to solve simple and complex problems efficiently.

The MC68000 offers sixteen 32-bit registers in addition to the 24-bit program counter and 16-bit status register (Figure 2). The first eight registers (D0–D7) are used as data registers for byte (8-bit), word (16-bit) and long word (32-bit) operations. The second set of eight registers (A0–A7) may be used as software Stack Pointers and Base Address Registers. In addition, the second set of eight registers may be used for word and long word data operations. All of the sixteen registers may be used as Index Registers.

The 24-bit Program Counter provides a memory addressing range of more than 16 mega-bytes (actually 16,777,216 bytes). This large range of addressing capability, coupled with a Memory Management Unit, allows large, modular programs to be developed and operated without resorting to cumbersome and time consuming software bookkeeping and paging techniques.

The Status Register (Figure 3) contains the Interrupt Level Mask (8 levels available) as well as the Condition Code: Overflow (V), Zero (Z), Negative (N), Carry (C), and Extend (X). Additional status bits indicate that the processor is in a TRACE (T) mode or in a SUPERVISORY (S) state. Ample space remains in the Status Register for future extensions of the M68000 family.

Six basic data types are supported. These data types are:

- Bits
- BCD digits
- ASCII characters
- Bytes (8-bits)
- Words (16-bits)
- Long words (32-bits)

In addition operations on other data types such as memory addresses, status word data, etc. are provided for in the instruction set.

DEFINITIONS

EA = Effective Address
Ax = Address Register
Dx = Data Register
Rx = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
Dn = Eight-Bit Offset
Dn = Sixteen-Bit Offset
N = 1 for Byte, 2 for Word and 4 for Long Word
( ) = Contents of
← = Replaces

TABLE I: MC68000 DATA ADDRESSING MODES

REGISTER DIRECT ADDRESSING	
Data Register Direct	EA = Dx
Address Register Direct	EA = Ax
Status Register Direct	EA = SR
ABSOLUTE DATA ADDRESSING	
A. Absolute Short	EA = (Next Word)
B. Absolute Long	EA = (Next two Words)
PROGRAM COUNTER RELATIVE ADDRESSING	
Relative with Offset	EA = (PC) + Dn
Relative with Index & Offset	EA = (PC) + (Rx) + Dn
REGISTER INDIRECT ADDRESSING	
Register Indirect	EA = (Ax)
Post-increment Register Indirect	EA = (Ax), Ax ← Ax + N
Pre-decrement Register Indirect	Ax ← Ax – N, EA = (Ax)
Register Indirect with Offset	EA = (Ax) + Dn
Indexed Register Indirect with Offset	EA = (Ax) + (Rx) + Dn
IMMEDIATE DATA ADDRESSING	
Immediate	DATA = Next Word(s)
Quick Immediate	INHERENT DATA

The addressing modes have been kept simple without sacrificing efficiency. All fourteen addressing modes operate consistently and are independent of the instruction operation itself. Additionally, all address registers may be used for the Direct, Register Indirect and Indexed addressing modes. (Immediate, Program Counter Relative and Absolute addressing by definition do not use address registers). For increased flexibility, any data register — as well as any address register — may be used as an Index Register. Address register consistency is maintained for stacking operations since any of the eight address registers may be utilized as User Program Stack pointers with the Register Indirect Post-increment/Pre-decrement addressing modes. Register A7, however, is a special register that, in addition to its normal addressing capability, functions as the System Stack Pointer when stacking the Program Counter and Status Register for subroutine calls, traps and interrupts; while in the supervisory mode.

Structured Modular Programming — The art of programming microprocessors has evolved rapidly in the past few years. Numerous advanced techniques have been developed to allow easier, more consistent and reliable generation of software. In general, these techniques require that the programmer be more disciplined in observing a defined programming structure such as modular programming. Modular programming allows a required function or process to be broken down in short modules or subroutines that are concisely defined and easily programmed and tested. Such a technique is greatly simplified by the availability of advanced macro assemblers and block structured High Level Languages such as PASCAL. Such concepts are virtually useless, however, unless parameters are easily transferred between and within software modules that operate on a reentrant and recursive basis. (To be reentrant a routine must be usable by interrupt and non-interrupt driven programs without the loss of data. A recursive routine is one that may call or use itself). The MC68000 microprocessor provides the necessary architectural features to allow efficient reentrant modular programming. The "LINK" and "UNLINK" instructions reduce subroutine call overhead in two complementary instructions by allowing the manipulation of linked lists of data areas on the stack. The "STM" (Store Multiple Registers) and "LDM" (Load Multiple Registers) instructions also reduce subroutine call programming overhead. These allow the loading or storing, via an effective address, multiple registers that are specified by the programmer. Sixteen software trap vectors are provided with the "TRAP" instruction and are useful in operating system call routines or user generated "macro routines." Other instructions that support modern structured programming techniques are PEA (Push Effective Address), LEA (Load Effective Address), RTR (Return to Restore) as well as the normal JSR, BSR and RTS.

Of course, the powerful vectored priority interrupt structure of the microprocessor allows straightforward generation of reentrant modular Input/Output routines. Eight maskable levels of priority with 192 vector locations provide maximum flexibility for I/O control. (A total of 256 vector locations are available for interrupts, hardware traps, and software traps.)

Improved Software Testability — One of the major tasks the system programmer encounters when writing software for microcomputers is the detection and correction of errors, or "debugging." The time taken to "debug" software nearly always exceeds the time it takes to write the software. In practice, the old 20/80 rule often applies: "The last 20% of the job requires 80% of the effort." The microprocessor incorporates several features that reduce the chance for errors. These features, such as Orthogonality and the Structured Modular Programming capability, have already been discussed.

Of major importance to the systems programmer are features that have been incorporated specifically to detect the occurrence of programming errors or "bugs." Several hardware traps, provided to indicate abnormal internal conditions of the MC68000 processor, detect the following error conditions:

- Word access with an odd address
- Illegal instructions
- Unimplemented instructions
- Illegal addressing mode
- Illegal Memory access (bus error)
- Overflow on divide (divide by zero)
- Overflow condition code (separate instruction TRAPV)

Additionally, the sixteen software TRAP instructions may be utilized by the programmer to provide applications oriented error detection or correction routines.

An additional error detection tool is the CHK (Check Register Against Bounds) instruction used for array bound checking by verifying that $0 \leq (REG) < LIMIT$. A trap occurs if the register contents are negative or greater than the limit.

Finally, the MC68000 includes a facility that allows instruction-by-instruction tracing of a program being debugged. This TRACE MODE results in a trap being made to a tracing routine after each instruction execution. The TRACE MODE is available to the programmer when the microprocessor is in the SUPERVISORY state as well as the USER state, but may only be entered while in the supervisory state. The SUPERVISORY/USER states provide an additional degree of error protection for the microprocessor by providing memory protection of selected areas of memory when an external memory management device is used.

by permitting the programmer to generate as many as eight concurrent stacks or queues. Another feature that allows the programmer to manage the use of memory is the CHK (Check Register Against Bounds) instruction. This instruction permits the software implementation of a basic memory protection/management structure.

Still another significant feature provided in the MC68000 microprocessor is the distinction between a USER and a SUPERVISOR mode. The SUPERVISOR mode permits certain protected operations within the processor system. Of particular interest is that an external Memory Management Controller may be used when the processor is in the USER mode to manage the large address space for the programmer. The controller's memory management operations are transparent to the programmer when in the USER mode and can be changed or updated only in the SUPERVISOR mode. The Memory Management Controller provides both management of a variable number of variable size segments (Memory Segmentation) and dynamic management of multi-task memory relocation and protection. The Memory Management Controller regulates access to storage segments that are dedicated to read only data, read/write data, program code and protected data/code.

REDUCED CODE DENSITY AND IMPROVED SPEED

With the advent of low cost, very high density VLSI RAMS and ROMS, it might incorrectly be assumed that the number of bytes of code needed to execute a given program is no longer important. Code density, however, is very critical, since microprocessor speed is highly dependent upon the number of executed instruction words. During the early development of Motorola's MC68000 microprocessor, extensive studies were made of the use of instructions and sequences of instructions in many microprocessor applications. These studies identified not only statically frequent instructions but also dynamically frequent instructions. (The dynamic frequency of instructions is a measure of how often an instruction is executed while static frequency is a measure of how often it occurs in a program listing or is encountered by an assembler). The major contributer to the increased efficiency, as a result of the studies, is the highly regular or orthogonal structure of the architecture. The consistency of the architecture, instruction set, and addressing modes significantly reduces the number of instructions needed to accomplish a given task. Additionally, many instructions have been included to specifically improve code density and speed. For example, single word Add and Subtract instructions using Quick Immediate addressing allow fast, small value arithmetic operations on data registers and memory. A Load Quick Immediate (LDQ) provides the ability to load a small (8-bit) signed word into any register in a single word operation. In order to improve the speed of loop operations, a single word instruction for Decrement Count by One and Branch if non-zero (DCNT) is included. Of course, the TRAP, Store Multiple Registers (STM), Load Multiple Registers (LDM), Link Stack (LINK), Unlink Stack (UNLK) and Check Limit (CHK) instructions significantly reduce code requirements for subroutines, operating system calls and stacking operations.

Other instructions that help reduce coding requirements and improve performance of arithmetic operations are Signed and Unsigned Multiply (MULS and MULU), Signed and Unsigned Divide (DIVS and DIVU), BCD Arithmetic (ABCD, SBCD, PACK and UNPK) as well as the standard binary integer operations. In order to improve the efficiency of moving or transferring data, a powerful MOVE data instruction has been incorporated that allows the transfer of bytes, words and long words and operates in all data addressing modes. Thus; register-to-register, register-to-memory, memory-to-register and memory-to-memory transfers are permitted.

In addition to the powerful instructions that provide a substantial improvement in processor through-put, numerous architectural features significantly reduce the execution times for all instructions. The separate (non-multiplexed) address and data buses, instruction pre-fetch pipeline and 32-bit internal registers are major contributors to the processor's unequaled performance. As an example of the performance capability of the MC68000 Table III and the accompanying graphs in figures 5 and 6 summarize the execution times for a number of common instructions. For comparison purposes, similar information is provided for Zilog's Z-8000 microprocessor. It is interesting to note that the MC68000 has significantly faster execution times.

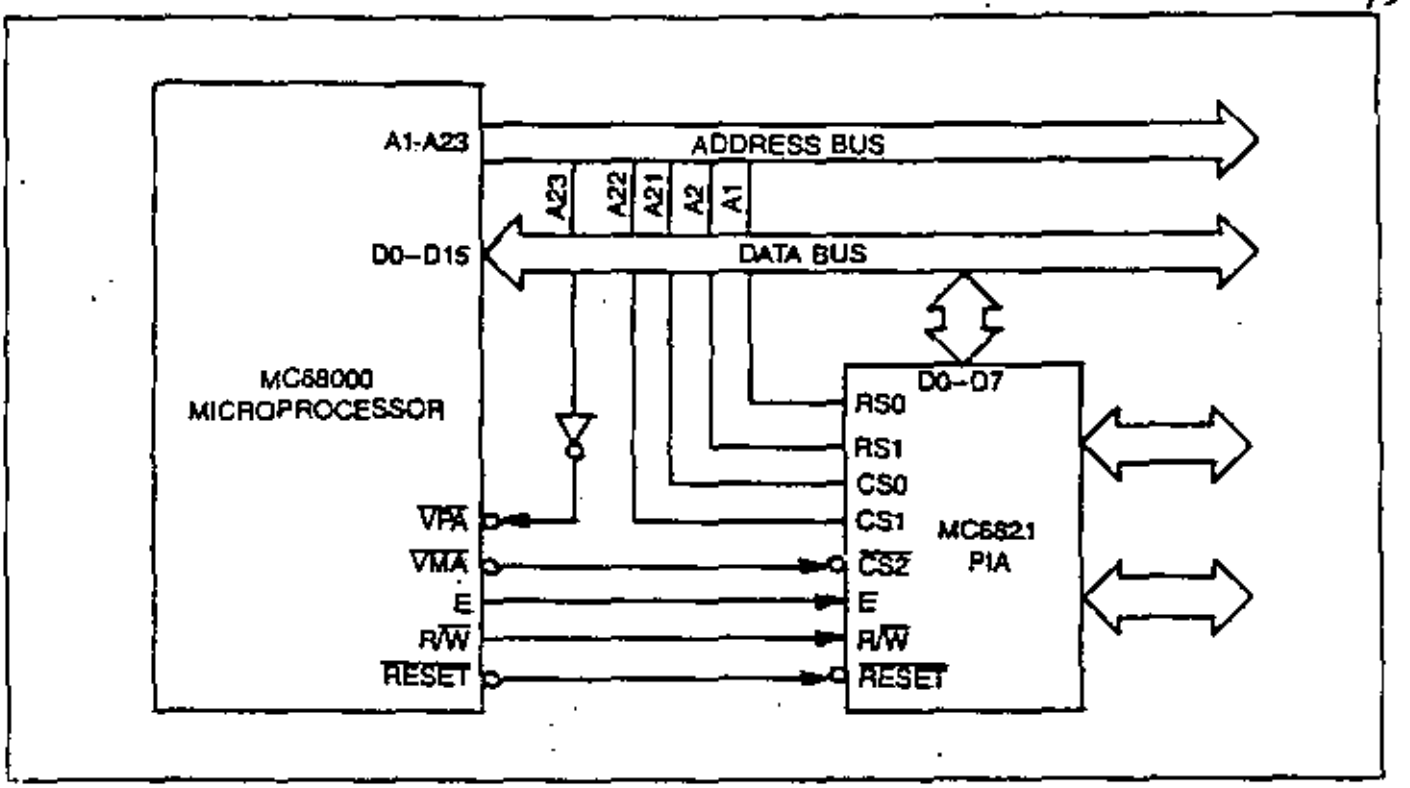


Figure 7: Example of MC68000 Interface Connections for MC6821 Peripheral Interface Adapter

SOFTWARE SUPPORT AND MC6800 COMPATIBILITY

The system designers and programmers using the MC68000 in an application have available a complete, compatible system of hardware and software. The microprocessor is supported by a full range of software development tools including disc operating systems, debug aids, assemblers, and high level languages. In addition, a translator will allow the present M6800 Family user to convert existing programs to run on the MC68000 with a minimum of programmer intervention.

The careful planning of this new microprocessor provides a superset of the MC6800 instruction set enhanced by the addition of more and larger registers, powerful orthogonal structure and many flexible addressing modes. This allows efficient translation of existing MC6800 programs, which can then be further optimized by taking full advantage of the versatile and powerful features of the MC68000.

This careful planning of similarities between the MC68000 and the MC6800 does not stop at software compatibility (by translation) but also extends to peripheral controller interfacing. Motorola's extensive line of intelligent M6800 family peripherals (including the MC6854 Advanced Data Link Controller and the MC68488 General Purpose Interface Adapter) can be directly and easily interfaced to the MC68000. Three signal lines: Enable (E), Valid Memory Address (VMA), and Valid Peripheral Address (VPA) are provided to simplify the interface to Motorola's standard MC6800 peripherals as shown in Figure 7. Interface to the new MC6801E (Single Chip Programmable Controller) is also possible, allowing user implementation of specialized input/output functions. In addition, the MC68000 is supported by unique peripheral controllers expected of an advanced architecture microprocessor, including a DMA Controller and a Memory Management Unit.

The MC68000 is not just a component. By a unique blend of VLSI design, software engineering and careful planning, the MC68000 *is* Motorola's Advanced Computer System on Silicon.

Z8001/Z8002 CPU Central Processing Unit

77

Zilog

Product Brief

August 1979

Features

- Regular, easy-to-use architecture.
- Instruction set more powerful than many minicomputers.
- Directly addresses 8M bytes.
- Eight user-selectable addressing modes.
- Seven data types that range from bits to 32-bit long words and word strings.
- System and normal operating modes; separate code, data and stack spaces.
- Sophisticated interrupt structure.
- Resource-sharing capabilities for multiprocessing systems.
- Multi-programming and compiler support.
- Memory management and protection provided by Z8010 Memory Management Unit.
- 32-bit operations, including signed multiply and divide.
- Z-Bus compatible.

Description

The Z8000 is an advanced high-end 16-bit microprocessor that spans a wide variety of applications ranging from simple stand-alone computers to complex parallel-processing systems. Essentially a monolithic minicomputer central processing unit, the Z8000 CPU is characterized by an instruction set more powerful than many minicomputers; resources abundant in registers, data types, addressing modes and addressing range; and a regular architecture that enhances throughput by avoiding critical bottlenecks such as implied or dedicated registers.

CPU resources include sixteen 16-bit general-purpose registers, seven data types that range from bits to 32-bit long words and word strings, and eight user-selectable addressing modes. The 110 distinct instruction types can be combined with the various data types and addressing modes to form a powerful set of 414 instructions. Moreover, the instruction set exhibits a high degree of regularity: most instructions can use any of the five main addressing modes and can operate on byte, word and long-word data types.

The CPU can operate in either system or normal modes. The distinction between these two modes permits privileged operations, thereby improving operating system organization and implementation. Multiprogramming is supported by the "atomic" Test and Set instruction; multiprocessing by a combination of instruction and hardware features; and compilers by multiple stacks, special instructions and addressing modes.

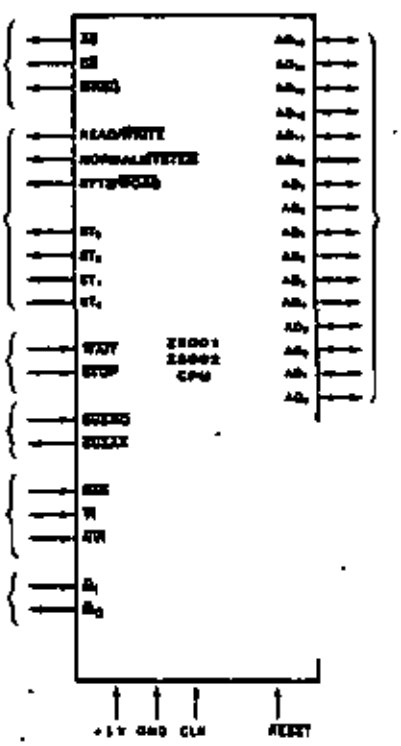


Figure 1. Pin Functions

MSE C-13

Z8010 MMU Memory Management Unit

Zilog

Product Brief

Preliminary

August 1979

Features

Dynamic segment relocation makes software addresses independent of physical memory addresses.

Sophisticated access validation protects memory areas from unauthorized or unintentional access.

MMU architecture supports multiprogramming systems.

Sixty-four variable-sized segments from 256 to 64K bytes can be managed within a total physical address space of 16M bytes; all 64 segments are randomly accessible.

Multiple MMUs can support several translation tables for each of the six Z8001 address spaces.

Description

Declining memory costs coupled with the increasing power of microprocessors has accelerated the use of high-level languages, sophisticated operating systems, complex programs and large data bases in microcomputer systems. The Z8001 microprocessor CPU supports these trends with an eight megabyte direct address space as well as a rich and powerful instruction set. The Z8010 Memory Management Unit (MMU) provides flexible and efficient support for this large address space by offering dynamic segment relocation as well as numerous memory-protection features.

The primary memory of a computer is one of its major resources. As such, the management of this resource becomes a major concern as demands on it increase. These demands arise from multiple users (or multiple tasks within a dedicated application), the need to increase system integrity by limiting access

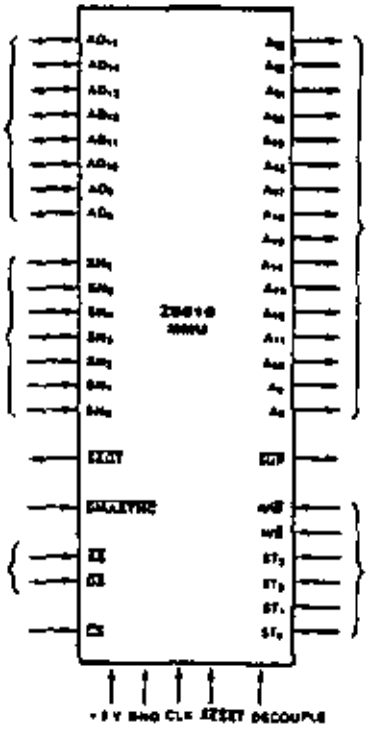


Figure 1. Pin Functions

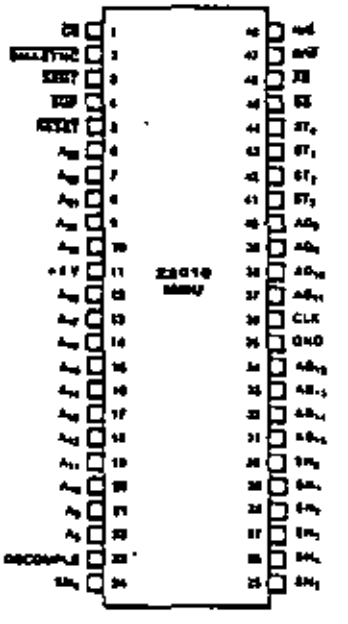


Figure 2. Pin Assignments

MSE C-15

Z8034 UPC Universal Peripheral Controller

61

Zilog

Product Brief

Preliminary

August 1979

Features

- Complete slave microcomputer, for distributed-processing Z-bus use.
- Unmatched power of Z8 architecture, instruction set.
- Three programmable I/O ports, two with 2-wire handshake, or any combination of data and control lines.
- Six levels of priority interrupts to Z-UPC.
- Two programmable 8-bit counter/timers with 6-bit prescalers.
- 256 byte register file, accessible by both master CPU and Z-UPC, as allocated by Z-UPC program.
- 2K bytes of on-chip program ROM for efficiency, versatility.

Description

The Z-UPC Universal Peripheral Controller is a distributed microcomputer that performs the three basic interfacing functions needed to interface a CPU with peripherals: device control by ROM-resident internal software, data manipulation, such as reformatting or arithmetic, and data buffering in internal registers.

The Z-UPC is similar to the Z8 microcomputer and uses the Z8 instruction set. Under program control, its three 8-line I/O ports can be tailored to the needs of its user. Permanently configured as a single-chip controller with 2K bytes of internal ROM, the Z-UPC executes instructions in 2.2 μs average using a 4-MHz clock source. Its register file contains 256 bytes, of which 234 are general-purpose registers, 19 are status and control registers, and three are port registers.

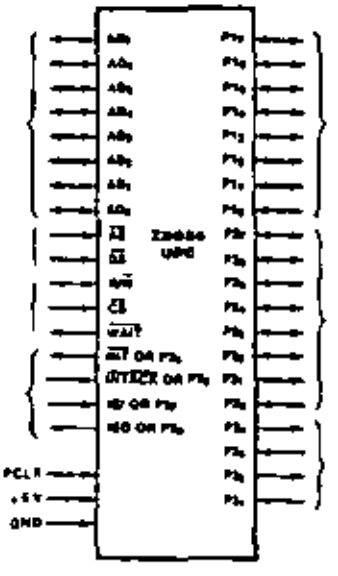


Figure 1. Pin Functions

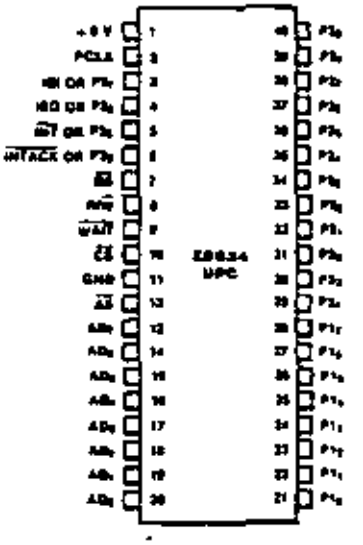


Figure 2. Pin Assignments

Z8036 CIO Counter/Timer and Parallel I/O Unit

Zilog

Product Brief

Preliminary

August 1979

Features

- Two independent 8-bit double-buffered bidirectional I/O ports plus a special-purpose 4-bit I/O port.
- Four handshake modes including IEEE-488.
- Wait/Request line for high speed data transfer.
- Three independent 16-bit counters.
- All registers read/write and directly addressable.
- Flexible pattern recognition logic, programmable as 16-input interrupt controller.

Description

The Z8036 CIO Counter/timer and Parallel I/O element is a general purpose peripheral circuit that satisfies most counter/timer and parallel I/O needs encountered in system designs. This versatile device contains three I/O ports and three counter/timers. Many programmable options tailor its configuration to specific applications. The use of the device is simplified by making all internal registers (command, status, and data) readable and (except for status bits) writable. Also, each register is given its own unique address so it can be accessed directly—no special sequential operations are required. The Z-CIO is directly Z-bus compatible.

Either 8-bit I/O port can be a handshake byte port or a bit port. In the bit mode, data direction is programmable bit by bit. In the handshake mode, the ports can be input, output, or bidirectional, and they may be linked to form a 16-bit port. The four handshake modes include IEEE-488, interlocked (for interfacing to a Z-UPC, Z-FIO or another Z-CIO), strobed and pulsed. The pulsed mode connects one counter/timer with the handshake logic for interfacing a mechanical device such as a printer. The 4-bit port provides handshake controls, special controls (Wait/Request) or general-purpose I/O.

The counter/timer section contains three 16-bit counters, two of which can be software-configured as a 32-bit counter/timer. Up to

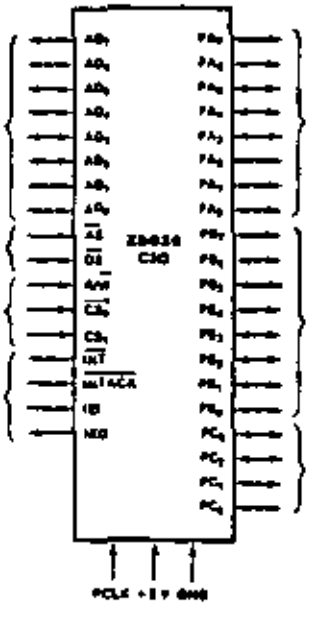


Figure 1. Pin Functions

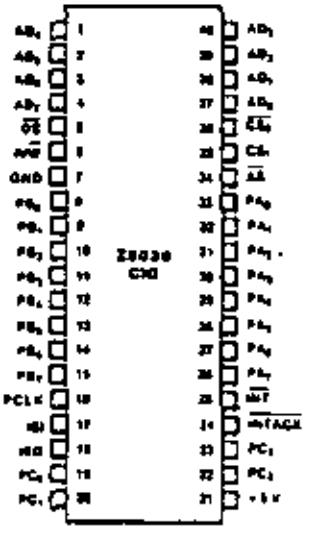


Figure 2. Pin Assignments

MSE C-19

Z8030 SCC Serial Communications Controller

65

Zilog

Product Brief

Preliminary

August 1979

Features

Two independent, 0 to 1 Megabit-per-second, full-duplex channels, each with its own quartz oscillator, baud-rate generator, and digital phase-locked loop for clock recovery.

Multi-protocol operation under program control.

Asynchronous mode with 5 to 8 bits and 1, 1½, or 2 stop bits per character; programmable clock factor; break detection and generation; parity, overrun, and framing error detection.

Local loopback and auto-echo modes.

Bisynchronous mode with internal or external character synchronization on one or two sync characters and CRC generation and checking with CRC-16 or CRC-CCITT preset to either 1s or 0s.

SDLC/HDLC mode with comprehensive frame-level control, automatic zero insertion and deletion, I-field residue handling, abort generation and detection, CRC generation and checking, and loop mode operation.

Programmable for NRZ, NRZI, or FM coding.

Description

The Z-SCC Serial Communication Controller is a dual-channel, multi-protocol data communication peripheral for Z-bus use. It is software-configured to satisfy a wide variety of serial communication applications. Its basic function is serial-to-parallel and parallel-to-serial conversion. However, the Z-SCC also contains a repertoire of new, sophisticated internal functions that minimize the need for external random logic on the circuit card.

The Z-SCC handles asynchronous formats, synchronous byte-oriented protocols such as IBM Bisync, and synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device also supports virtually any other serial data transfer application (cassette or diskette interface, for example).

The device can generate and check CRC

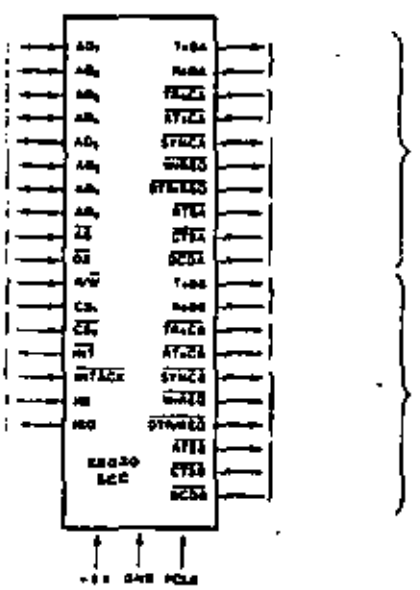


Figure 1. Pin Functions

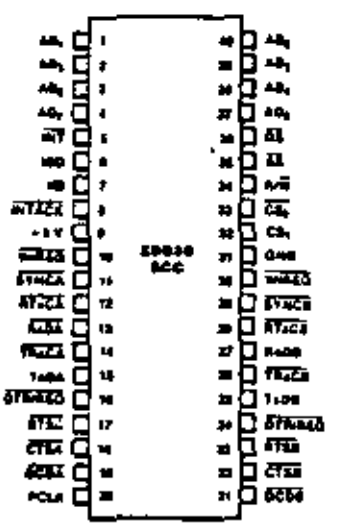


Figure 2. Pin Assignments

MSE C-21

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

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

MANUAL DEL PROCESADOR Z-80

MARZO, 1981

Palacio de Minería Calle de Tacuba 5 primer piso México 1, D. F. Tel: 521-40-20 Apdo. Postal M-2285

TABLE OF CONTENTS

1.0 INTRODUCTION

The term "microcomputer" has been used to describe virtually every type of small computing device designed within the last few years. This term has been applied to everything from simple "microprogrammed" controllers constructed out of TTL MSI up to low end minicomputers with a portion of the CPU constructed out of TTL LSI "bit slices." However, the major impact of the LSI technology within the last few years has been with MOS LSI. With this technology, it is possible to fabricate complete and very powerful computer systems with only a few MOS LSI components.

The Zilog Z-80 family of components is a significant advancement in the state-of-the art of microcomputers. These components can be configured with any type of standard semiconductor memory to generate computer systems with an extremely wide range of capabilities. For example, as few as two LSI circuits and three standard TTL MSI packages can be combined to form a simple controller. With additional memory and I/O devices a computer can be constructed with capabilities that only a minicomputer could previously deliver. This wide range of computational power allows standard modules to be constructed by a user that can satisfy the requirements of an extremely wide range of applications.

The major reason for MOS LSI domination of the microcomputer market is the low cost of these few LSI components. For example, MOS LSI microcomputers have already replaced TTL logic in such applications as terminal controllers, peripheral device controllers, traffic signal controllers, point of sale terminals, intelligent terminals and test systems. In fact the MOS LSI microcomputer is finding its way into almost every product that now uses electronics and it is even replacing many mechanical systems such as weight scales and automobile controls.

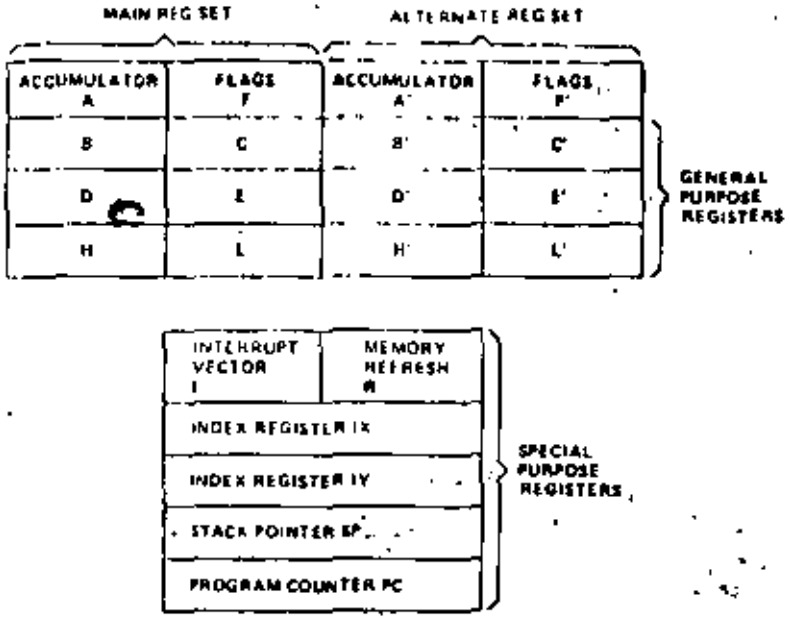
The MOS LSI microcomputer market is already well established and new products using them are being developed at an extraordinary rate. The Zilog Z-80 component set has been designed to fit into this market through the following factors:

1. The Z-80 is fully software compatible with the popular 8080A CPU offered from several sources. Existing designs can be easily converted to include the Z-80 as a superior alternative.
2. The Z-80 component set is superior in both software and hardware capabilities to any other microcomputer system on the market. These capabilities provide the user with significantly lower hardware and software development costs while also allowing him to offer additional features in his system.
3. For increased throughput the Z80A operating at a 4 MHZ clock rate offers the user significant speed advantages over competitive products.
4. A complete product line including full software support with strong emphasis on high level languages and a disk-based development system with advanced real-time debug capabilities is offered to enable the user to easily develop new products.

Microcomputer systems are extremely simple to construct using Z-80 components. Any such system consists of three parts:

1. CPU (Central Processing Unit)
2. Memory
3. Interface Circuits to peripheral devices

The CPU is the heart of the system. Its function is to obtain instructions from the memory and perform the desired operations. The memory is used to contain instructions and in most cases data that is to be processed. For example, a typical instruction sequence may be to read data from a specific peripheral device, store it in a location in memory, check the parity and write it out to another peripheral device. Note that the Zilog component set includes the CPU and various general purpose I/O device controllers, while a wide range of memory devices may be used from any source. Thus, all required components can be connected together in a very simple manner with virtually no other external logic. The user's effort then becomes primarily one of software development. That is, the user can concentrate on describing his problem and translating it into a series of instructions that can be loaded into the microcomputer memory. Zilog is dedicated to making this step of software generation as simple as possible. A good example of this is our



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.

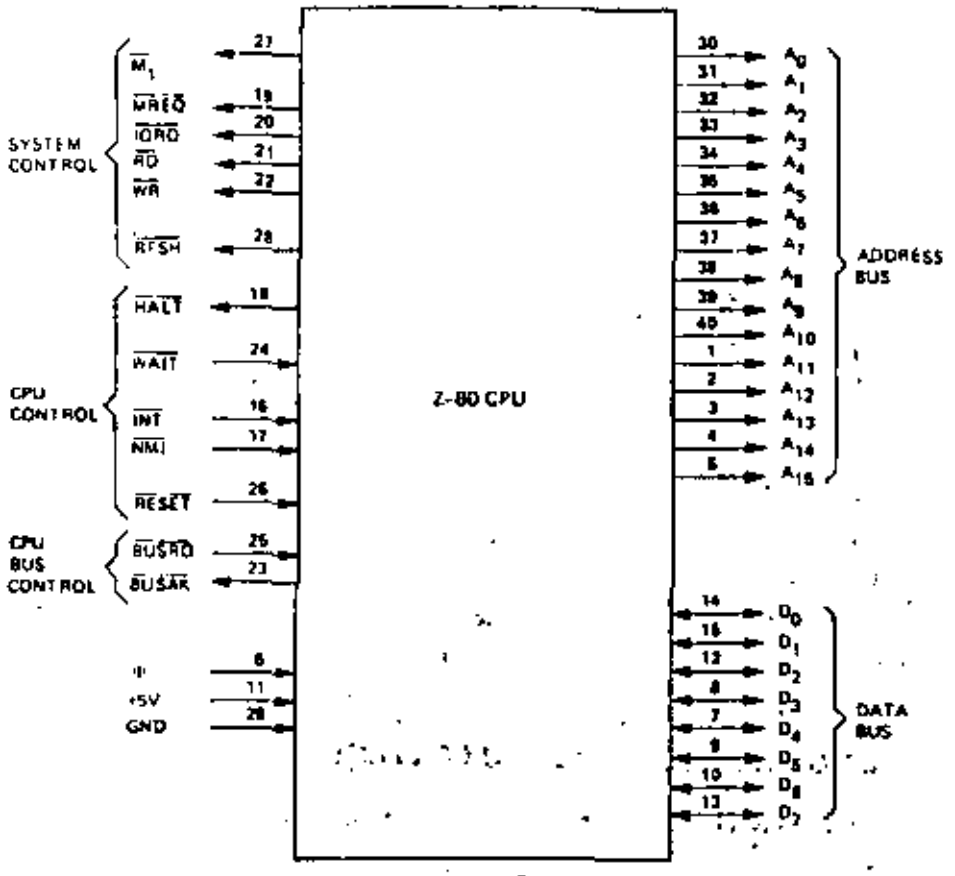
5. **Memory Refresh Register (R).** The Z-80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. Seven bits of this 8 bit register are automatically incremented after each instruction fetch. The eighth bit will remain as programmed as the result of an LD R, A instruction. 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. During refresh, the contents of the I register are placed on the upper 8 bits of the address bus.

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.

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.

$\overline{RESET}$

Input, active low. $\overline{RESET}$ 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.

$\overline{BUSRQ}$
(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 $\overline{BUSRQ}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.

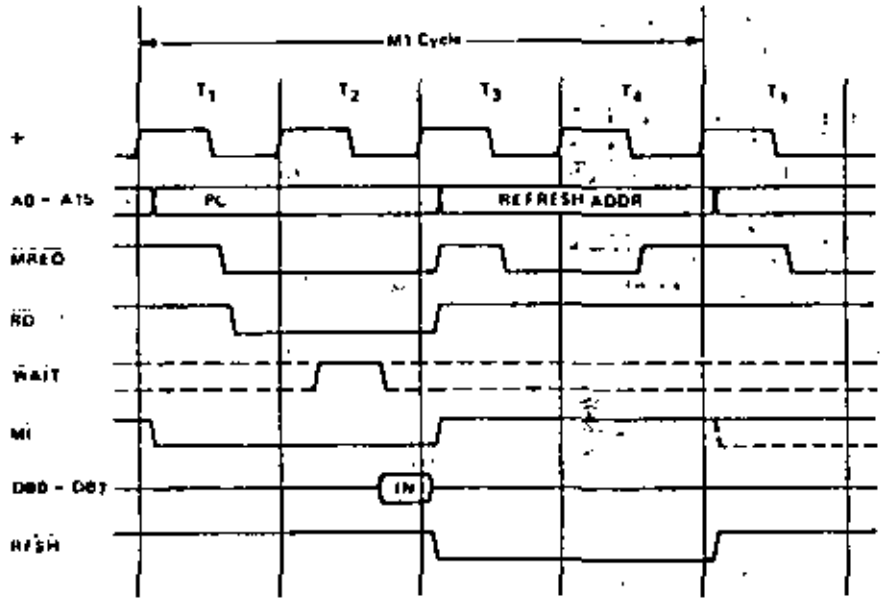
$\overline{BUSAK}$
(Bus Acknowledge)

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.

Single phase TTL level clock which requires only a 330 ohm pull-up resistor to +5 volts to meet all clock requirements.

INSTRUCTION FETCH

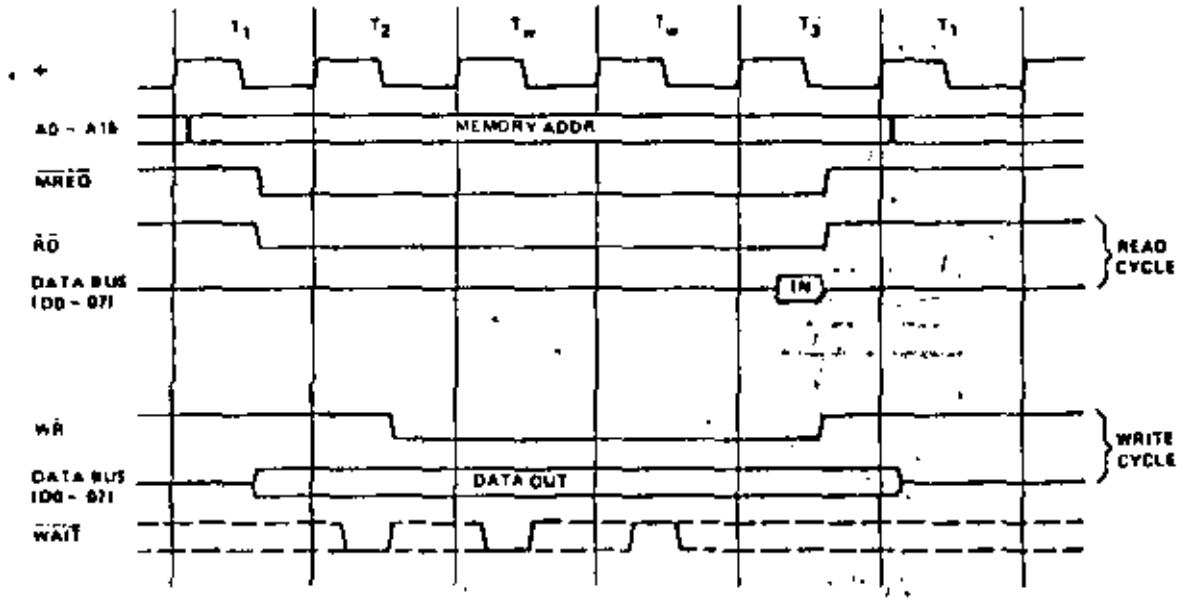
Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling 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.

Figure 4.0-2A illustrates how a $\overline{WAIT}$ 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

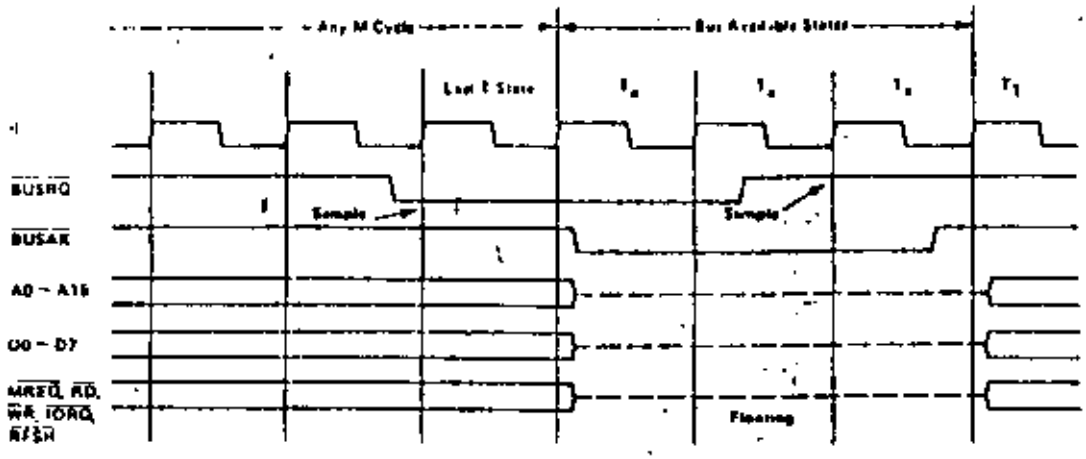
INPUT OR OUTPUT CYCLES

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 $\overline{IORQ}$ signal goes active until the CPU must sample the $\overline{WAIT}$ 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 $\overline{WAIT}$ 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 $\overline{WAIT}$ request signal is sampled. During a read I/O operation, the $\overline{RD}$ 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 $\overline{WR}$ 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 $\overline{WAIT}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

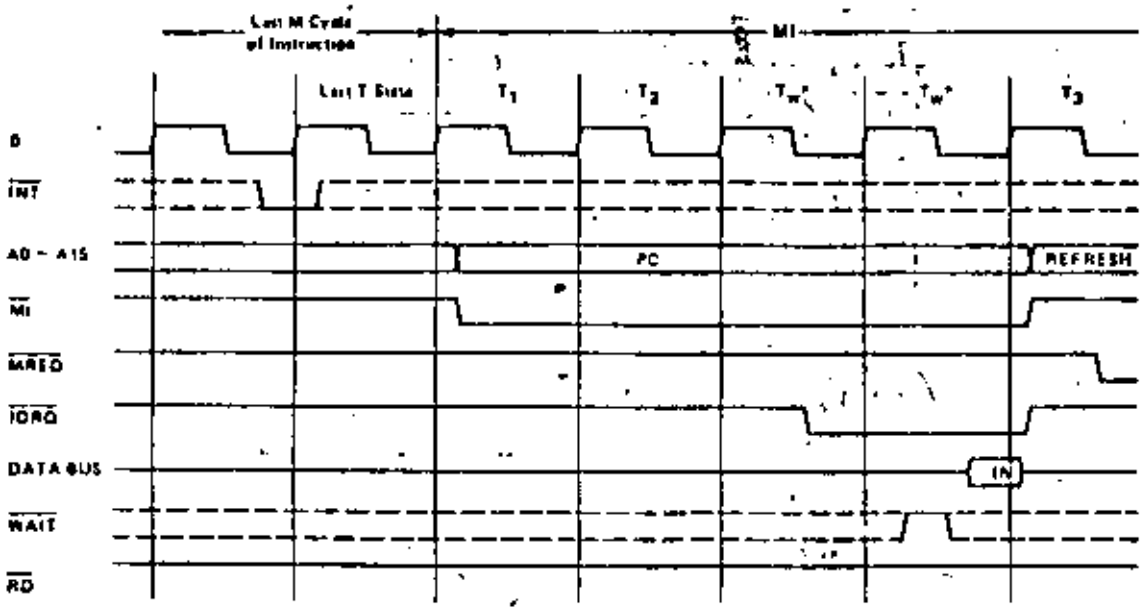
Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{BUSRQ}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{BUSRQ}$ signal is active, the CPU will set its address, data and tri-state control signals 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 $\overline{NMI}$ or an $\overline{INT}$ signal.



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.



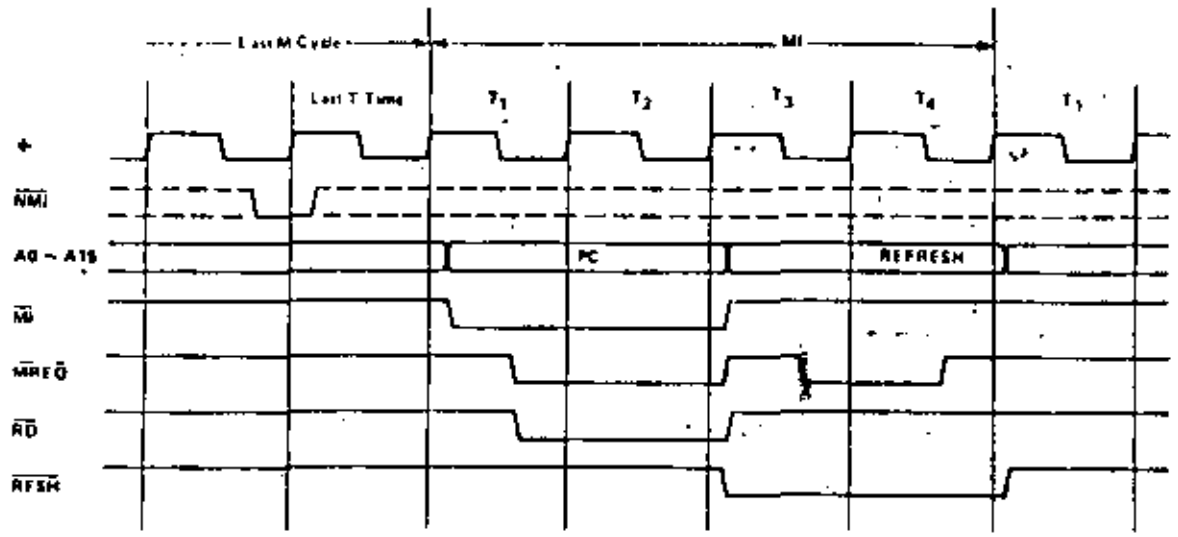
INTERRUPT REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4.0-5

NON MASKABLE INTERRUPT RESPONSE

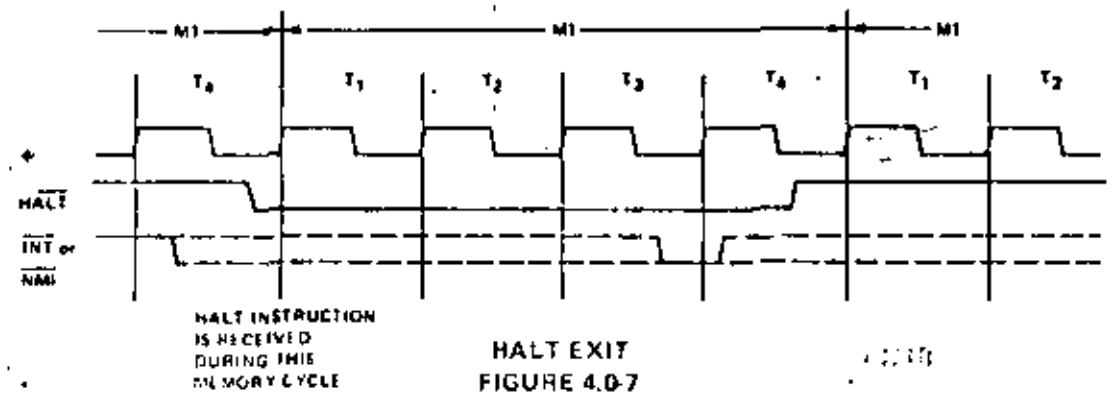
Figure 4.0-6 illustrates the request/acknowledge cycle for the non maskable interrupt. This signal is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response 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

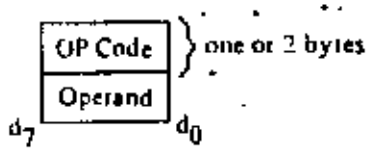
The input/output group of instructions in the Z-80 allow for a wide range of transfers between external memory locations or the general purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower 8 bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z-80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O ports to share common software driver routines. This is not possible when the address is part of the OP code if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z-80 CPU includes single instructions that can move blocks of data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z-80 CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, data and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip flop or setting the mode of interrupt response.

5.2 ADDRESSING MODES

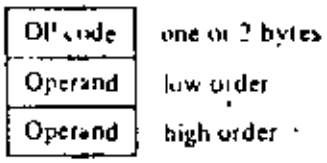
Most of the Z-80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z-80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing the byte following the OP code in memory contains the actual operand.



Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the OP code.

Immediate Extended. This mode is merely an extension of immediate addressing in that the two bytes following the OP codes are the operand.



Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z-80 are referred to as IX and IY. To indicate indexed addressing the notation:

(IX+d) or (IY+d)

is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z-80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.

An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z-80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

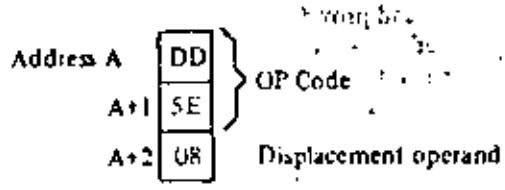
specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing. The Z-80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the OP code specify which of the eight bits is to be manipulated.

ADDRESSING MODE COMBINATIONS

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to specify the destination.

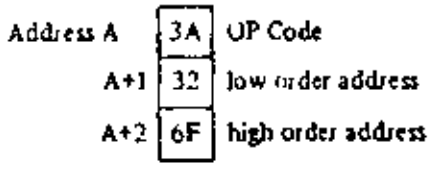
The instruction sequence for this in memory would be:



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, (6F32H)

and its instruction sequence would be:

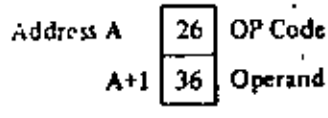


Notice that the low order portion of the address is always the first operand.

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

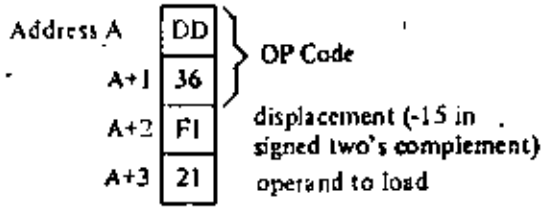
and its sequence would be:



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:



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:

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

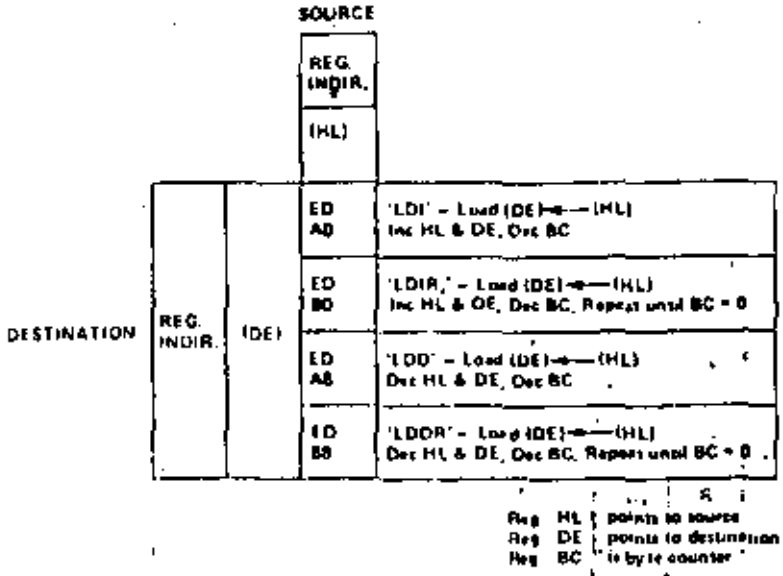
Table 5.3-4 lists the extremely powerful block transfer instructions. All of these instructions operate with three registers.

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.



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

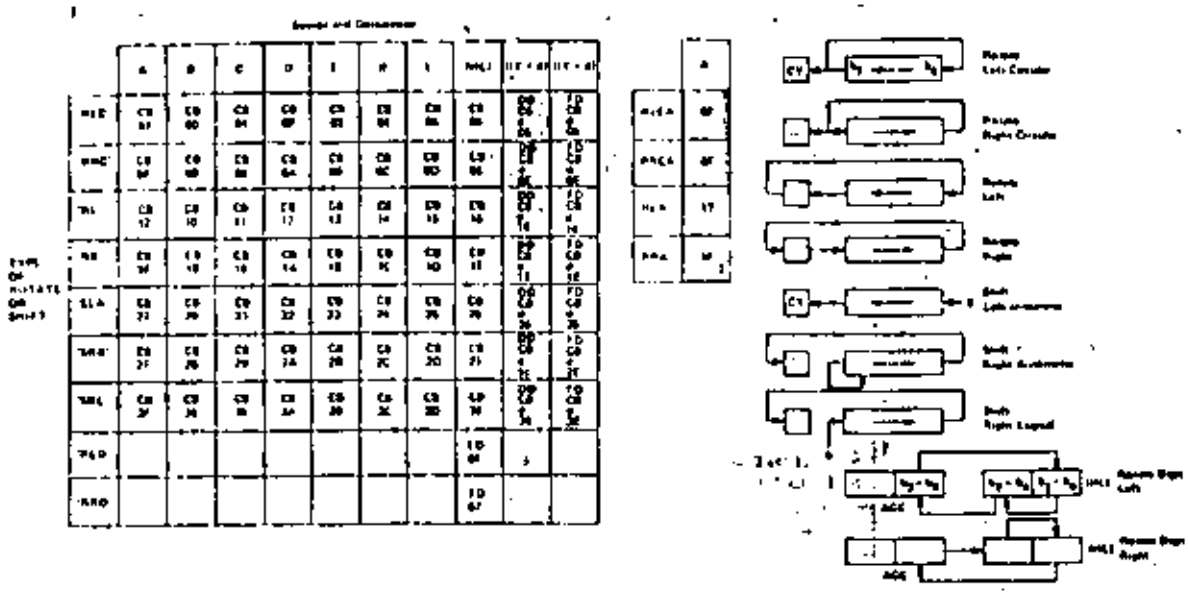
SOURCE

	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



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 from 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:

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 one byte EI instruction and the two byte RETI instruction in the Z80. This is important since interrupt service time often must be minimized.

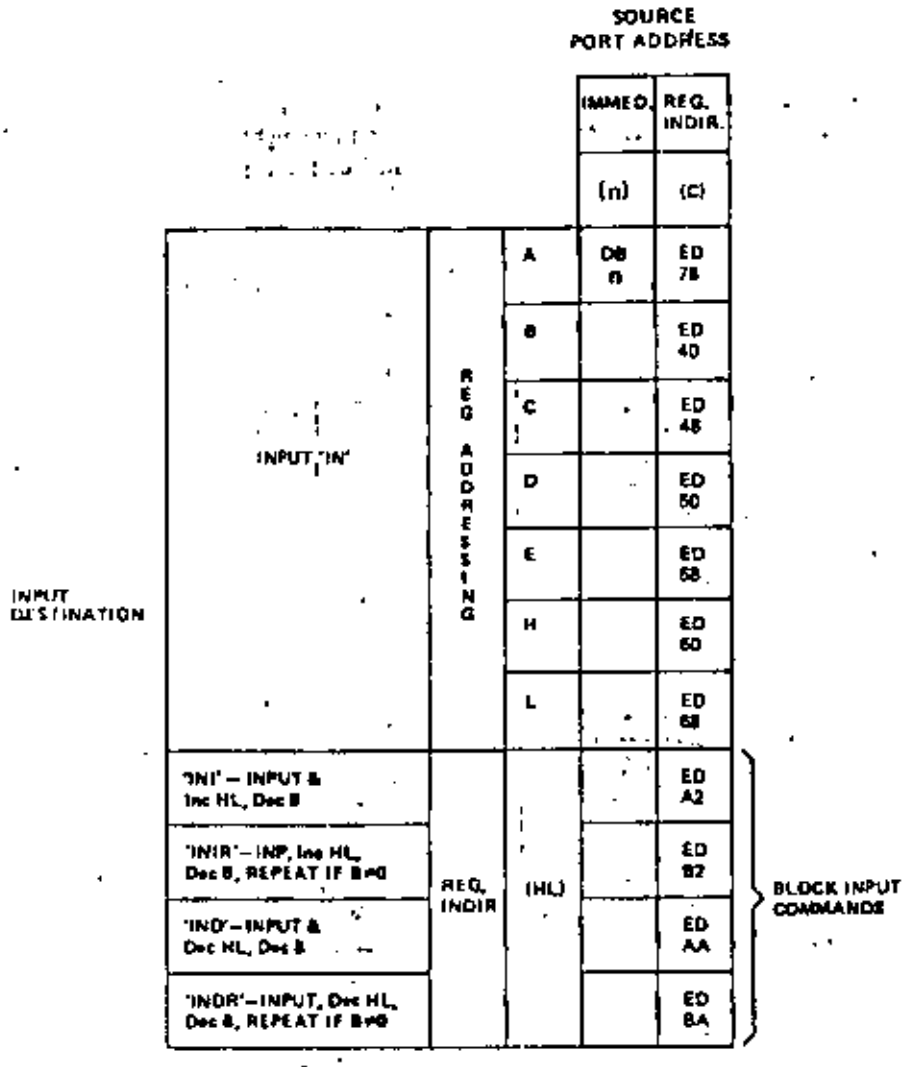
To facilitate program loop control the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. 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	D9	C8	C0	E8	E9	F8	F9	
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 4.0 FOR DETAILS

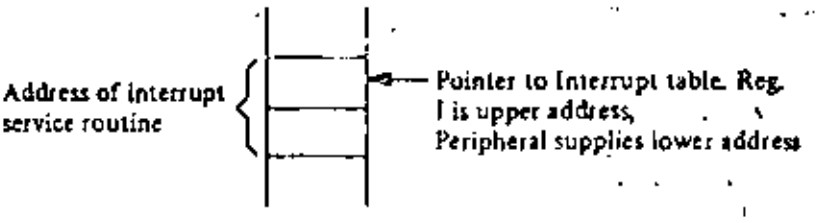
JUMP, CALL and RETURN GROUP
TABLE 5.3–11



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.



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 it 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.

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 location m
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	•	↕	↕	↕	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.
g	Any one of the two index registers IX or IY.
R	Refresh register.
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 6.0-1

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 Cycles	Comments	
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

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
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix}\begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	•	•	•	•	•	•	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 ①	•	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 ①	•	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	•	‡	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	•	‡	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	•	‡	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	•	‡	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

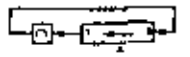
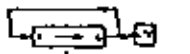
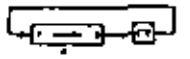
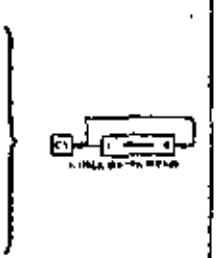
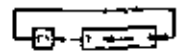
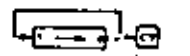
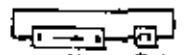
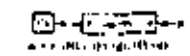
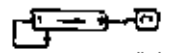
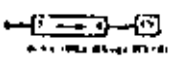
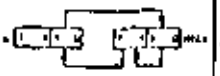
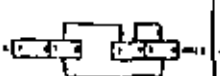
TABLE 7.0-3

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
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	$A \leftarrow \bar{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	$A \leftarrow 0 - A$	‡	‡	V	‡	1	‡	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	‡	•	•	•	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 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 7.0-7

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
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: 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

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
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	11	n to $A_0 - A_7$ Acc to $A_8 - A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	•	①	P	‡	0	‡	11 101 101 01 r 000	2	3	12	C to $A_0 - A_7$ B to $A_8 - A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	①	X	X	1	X	11 101 101 10 100 010	2	4	16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	①	X	X	1	X	11 101 101 10 110 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	①	X	X	1	X	11 101 101 10 101 010	2	4	16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	①	X	X	1	X	11 101 101 10 111 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 ← n →	2	3	11	n to $A_0 - A_7$ Acc to $A_8 - A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	2	3	12	C to $A_0 - A_7$ B to $A_8 - A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	①	X	X	1	X	11 101 101 10 100 011	2	4	16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
OTIR*	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	①	X	X	1	X	11 101 101 10 110 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	①	X	X	1	X	11 101 101 10 101 011	2	4	16	C to $A_0 - A_7$ B to $A_8 - A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	①	X	X	1	X	11 101 101 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 - A_7$ B to $A_8 - A_{15}$

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

Figure 8.0-1 is a summary of the effect of different instructions on the two enable flip flops.

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	$IFF_2 \rightarrow$ Parity flag
LD A, R	•	•	$IFF_2 \rightarrow$ Parity flag
Accept NMI	0	•	
RETN	IFF_2	•	$IFF_2 \rightarrow IFF_1$

"•" indicates no change

FIGURE 8.0-1
INTERRUPT ENABLE/DISABLE FLIP FLOPS

CPU RESPONSE

Non Maskable

A nonmaskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction but, it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 of memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of the memory. Often this will be a restart instruction since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction such as a 3 byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Section 5.0 illustrates the detailed timing for an interrupt response. After the application of RESET the CPU will automatically enter interrupt Mode 0.

Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non maskable interrupt except that the call location is 0038H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

9.0 HARDWARE IMPLEMENTATION EXAMPLES

This chapter is intended to serve as a basic introduction to implementing systems with the Z80-CPU.

MINIMUM SYSTEM

Figure 9.0-1 is a diagram of a very simple Z-80 system. Any Z-80 system must include the following five elements:

1) Five volt power supply
2) Oscillator
3) Memory devices
4) I/O circuits
5) CPU

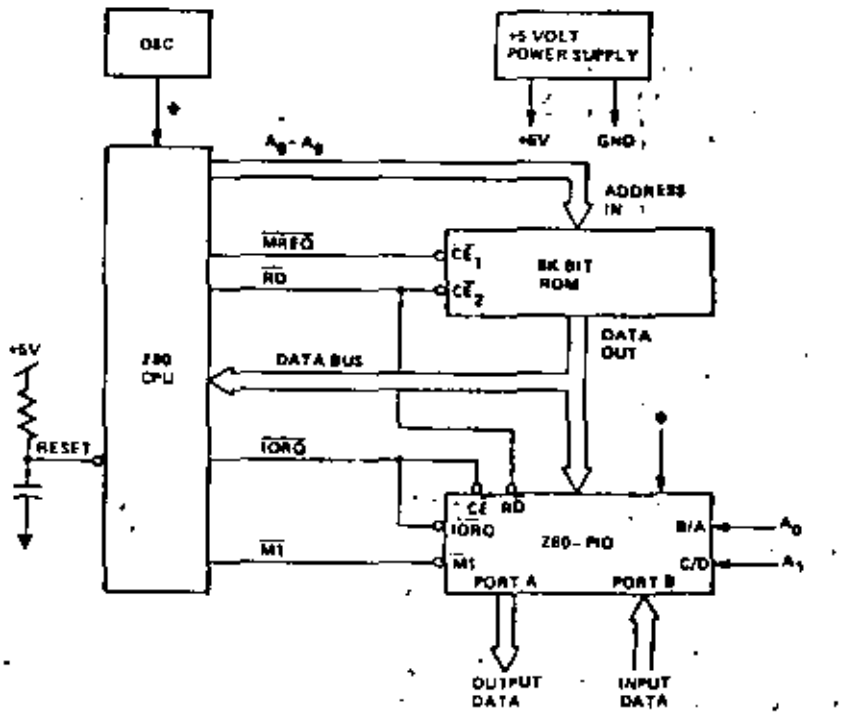


FIGURE 9.0-1
MINIMUM Z80 COMPUTER SYSTEM

Since the Z80-CPU only requires a single 5 volt supply, most small systems can be implemented using only this single supply.

The oscillator can be very simple since the only requirement is that it be a 5 volt square wave. For systems not running at full speed, a simple RC oscillator can be used. When the CPU is operated near the highest possible frequency, a crystal oscillator is generally required because the system timing will not tolerate the drift or jitter that an RC network will generate. A crystal oscillator can be made from inverters and a few discrete components or monolithic circuits are widely available.

The external memory can be any mixture of standard RAM, ROM, or PROM. In this simple example we have shown a single 8K bit ROM (1K bytes) being utilized as the entire memory system. For this example we have assumed that the Z-80 internal register configuration contains sufficient Read/Write storage so that external RAM memory is not required.

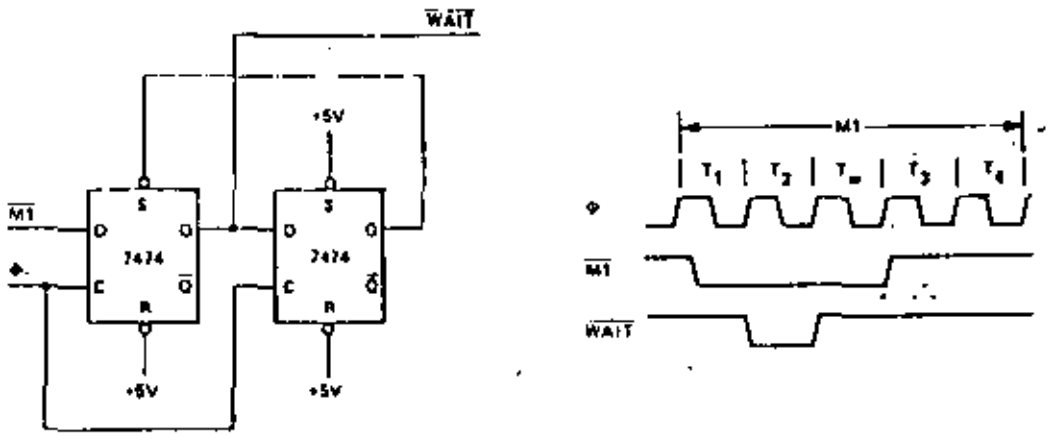


FIGURE 9.0-3
ADDING ONE WAIT STATE TO AN M1 CYCLE

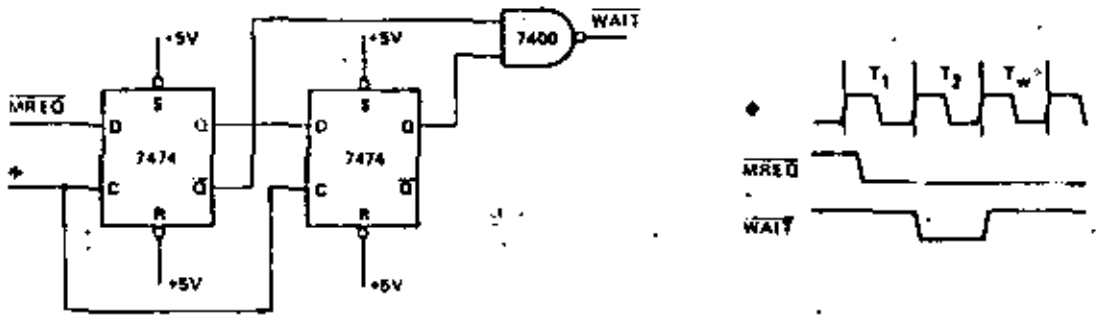


FIGURE 9.0-4
ADDING ONE WAIT STATE TO ANY MEMORY CYCLE

INTERFACING DYNAMIC MEMORIES

This section is intended only to serve as a brief introduction to interfacing dynamic memories. Each individual dynamic RAM has varying specifications that will require minor modifications to the description given here and no attempt will be made in this document to give details for any particular RAM. Separate application notes showing how the Z80-CPU can be interfaced to most popular dynamic RAM's are available from Zilog.

Figure 9.0-5 illustrates the logic necessary to interface 8K bytes of dynamic RAM using 18 pin 4K dynamic memories. This figure assumes that the RAM's are the only memory in the system so that A_{12} is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the proper refresh address on lines A_0 through A_6. To add additional memory to the system it is necessary to only replace the two gates that operate on A_{12} with a decoder that operates on all required address bits. For larger systems, buffering for the address and data bus is also generally required.

10.0 SOFTWARE IMPLEMENTATION EXAMPLES

10.1 METHODS OF SOFTWARE IMPLEMENTATION

Several different approaches are possible in developing software for the Z-80 (Figure 10.1). First of all, Assembly Language or PL/Z may be used as the source language. These languages may then be translated into machine language on a commercial time sharing facility using a cross-assembler or cross-compiler or, in the case of assembly language, the translation can be accomplished on a Z-80 Development System using a resident assembler. Finally, the resulting machine code can be debugged either on a time-sharing facility using a Z-80 simulator or on a Z-80 Development System which uses a Z80-CPU directly.

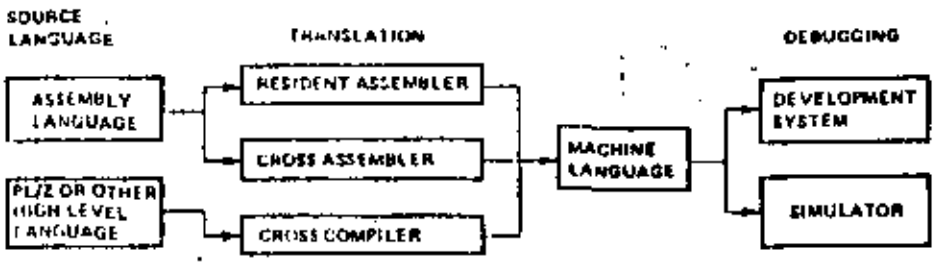


FIGURE 10.1

In selecting a source language, the primary factors to be considered are clarity and ease of programming vs. code efficiency. A high level language such as PL/Z with its machine independent constructs is typically better for formulating and maintaining algorithms, but the resulting machine code is usually somewhat less efficient than what can be written directly in assembly language. These tradeoffs can often be balanced by combining PL/Z and assembly language routines, identifying those portions of a task which must be optimized and writing them as assembly language subroutines.

Deciding whether to use a resident or cross assembler is a matter of availability and short-term vs. long-term expense. While the initial expenditure for a development system is higher than that for a time-sharing terminal, the cost of an individual assembly using a resident assembler is negligible while the same operation on a time-sharing system is relatively expensive and in a short time this cost can equal the total cost of a development system.

Debugging on a development system vs. a simulator is also a matter of availability and expense combined with operational fidelity and flexibility. As with the assembly process, debugging is less expensive on a development system than on a simulator available through time-sharing. In addition, the fidelity of the operating environment is preserved through real-time execution on a Z80-CPU and by connecting the I/O and memory components which will actually be used in the production system. The only advantage to the use of a simulator is the range of criteria which may be selected for such debugging procedures as tracing and setting breakpoints. This flexibility exists because a software simulation can achieve any degree of complexity in its interpretation of machine instructions while development system procedures have hardware limitations such as the capacity of the real-time storage module, the number of breakpoint registers and the pin configuration of the CPU. Despite such hardware limitations, debugging on a development system is typically more productive than on a simulator because of the direct interaction that is possible between the programmer and the authentic execution of his program.

B. Let's assume that a string in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" until an ASCII $ character (used as string delimiter) is found. Let's also assume that the maximum string length is 132 characters. The operation can be performed as follows:

```
        LD      HL , DATA       ;STARTING ADDRESS OF DATA STRING
        LD      DE , BUFFER     ;STARTING ADDRESS OF TARGET BUFFER
        LD      BC , 132        ;MAXIMUM STRING LENGTH
        LD      A ,'$'          ;STRING DELIMITER CODE
LOOP:CP         (HL)            ;COMPARE MEMORY CONTENTS WITH DELIMITER
        JR      Z , END - $     ;GO TO END IF CHARACTERS EQUAL
        LDI                     ;MOVE CHARACTER (HL) to (DE)
                                ;INCREMENT HL AND DE, DECREMENT BC
        JP      PE , LOOP       ;GO TO "LOOP" IF MORE CHARACTERS
END:                            ;OTHERWISE, FALL THROUGH
                                ;NOTE: P/V FLAG IS USED
                                ;TO INDICATE THAT REGISTER BC WAS
                                ;DECREMENTED TO ZERO.
```

19 bytes are required for this operation.

C. Let us assume that a 16-digit decimal number represented in packed BCD format (two BCD digits/byte) has to be shifted as shown in the Figure 10.2 in order to mechanize BCD multiplication or division. The operation can be accomplished as follows:

```
        LD      HL , DATA       ;ADDRESS OF FIRST BYTE
        LD      B , COUNT       ;SHIFT COUNT
        XOR     A               ;CLEAR ACCUMULATOR
ROTAT:  RLD                     ;ROTATE LEFT LOW ORDER DIGIT IN ACC
                                ;WITH DIGITS IN (HL)
        INC     HL              ;ADVANCE MEMORY POINTER
        DJNZ    ROTAT - $       ;DECREMENT B AND GO TO ROTAT IF
                                ;B IS NOT ZERO, OTHERWISE FALL THROUGH
```

11 bytes are required for this operation.

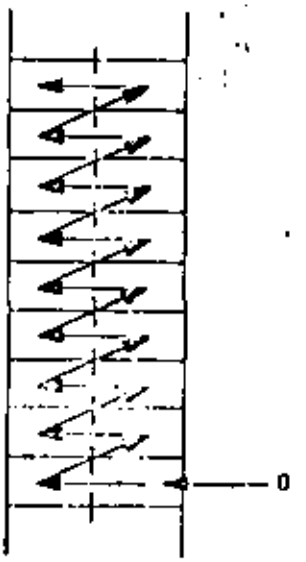


FIGURE 10.2

```
01/22/76     11:14:37                BUBBLE LISTING                    PAGE 1
LOC    OBJ CODE    STMT  SOURCE STATEMENT

                     1   ; *** STANDARD EXCHANGE (BUBBLE) SORT ROUTINE ***
                     2   ;
                     3   ; AT ENTRY:  HL CONTAINS ADDRESS OF DATA
                     4                C CONTAINS NUMBER OF ELEMENTS TO BE SORTED
                     5                (1<C<256)
                     6   ;
                     7   ; AT EXIT: DATA SORTED IN ASCENDING ORDER
                     8   ;
                     9   ; USE OF REGISTERS
                    10   ;
                    11   ; REGISTER   CONTENTS
                    12   ;
                    13   ; A          TEMPORARY STORAGE FOR CALCULATIONS
                    14   ; B          COUNTER FOR DATA ARRAY
                    15   ; C          LENGTH OF DATA ARRAY .
                    16   ; D          FIRST ELEMENT IN COMPARISON
                    17   ; E          SECOND ELEMENT IN COMPARISON
                    18   ; H          FLAG TO INDICATE EXCHANGE
                    19   ; L          UNUSED
                    20   ; IX         POINTER INTO DATA ARRAY
                    21   ; IY         UNUSED
                    22   ;
0000   222600       23   SORT:  LD     (DATA), HL     ;SAVE DATA ADDRESS
0003   CB84         24   LOOP:  RES    FLAG, H        ;INITIALIZE EXCHANGE FLAG
0005   41           25          LD     B, C           ;INITIALIZE LENGTH COUNTER
0006   05           26          DEC    B              ;ADJUST FOR TESTING
0007   DD2A2600     27          LD     IX, (DATA)     ;INITIALIZE ARRAY POINTER
000B   DD7E00       28   NEXT:  LD     A, (IX)        ;FIRST ELEMENT IN COMPARISON
000E   57           29          LD     D, A           ;TEMPORARY STORAGE FOR ELEMENT
000F   DD5E01       30          LD     E, (IX+1)      ;SECOND ELEMENT IN COMPARISON
0012   93           31          SUB    E              ;COMPARISON FIRST TO SECOND
0013   3008         32          JR     NC, NOEX-$     ;IF FIRST > SECOND, NO JUMP
0015   DD7300       33          LD     (IX), E        ;EXCHANGE ARRAY ELEMENTS
0018   DD7201       34          LD     (IX+1), D
001B   CBC4         35          SET    FLAG, H        ;RECORD EXCHANGE OCCURRED
001D   DD23         36   NOEX:  INC    IX             ;POINT TO NEXT DATA ELEMENT
001F   10EA         37          DJNZ   NEXT-$         ;COUNT NUMBER OF COMPARISONS
                    38                                ;REPEAT IF MORE DATA PAIRS
0021   CB44         39          BIT    FLAG, H        ;DETERMINE IF EXCHANGE OCCURRED
0023   20DE         40          JR     NZ, LOOP-$     ;CONTINUE IF DATA UNSORTED
0025   C9           41          RET                   ;OTHERWISE, EXIT
                    42   ;
0026                43   FLAG:  EQU    0              ;DESIGNATION OF FLAG BIT
0026                44   DATA:  DEFS   2              ;STORAGE FOR DATA ADDRESS
                    45          END
```

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	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 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.

Note: For Z80-CPU all AC and DC characteristics are the same for the military grade parts except I_{CC}

I_{CC} = 200 mA

Z80-CPU 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} -.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} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current			150	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}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{CC}$

Z80A-CPU 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} - 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}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current		90	200	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}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{CC}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C
E – Extended 5V ±5% -40° to 85°C
M – Military 5V ±10% -55° to 125°C

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

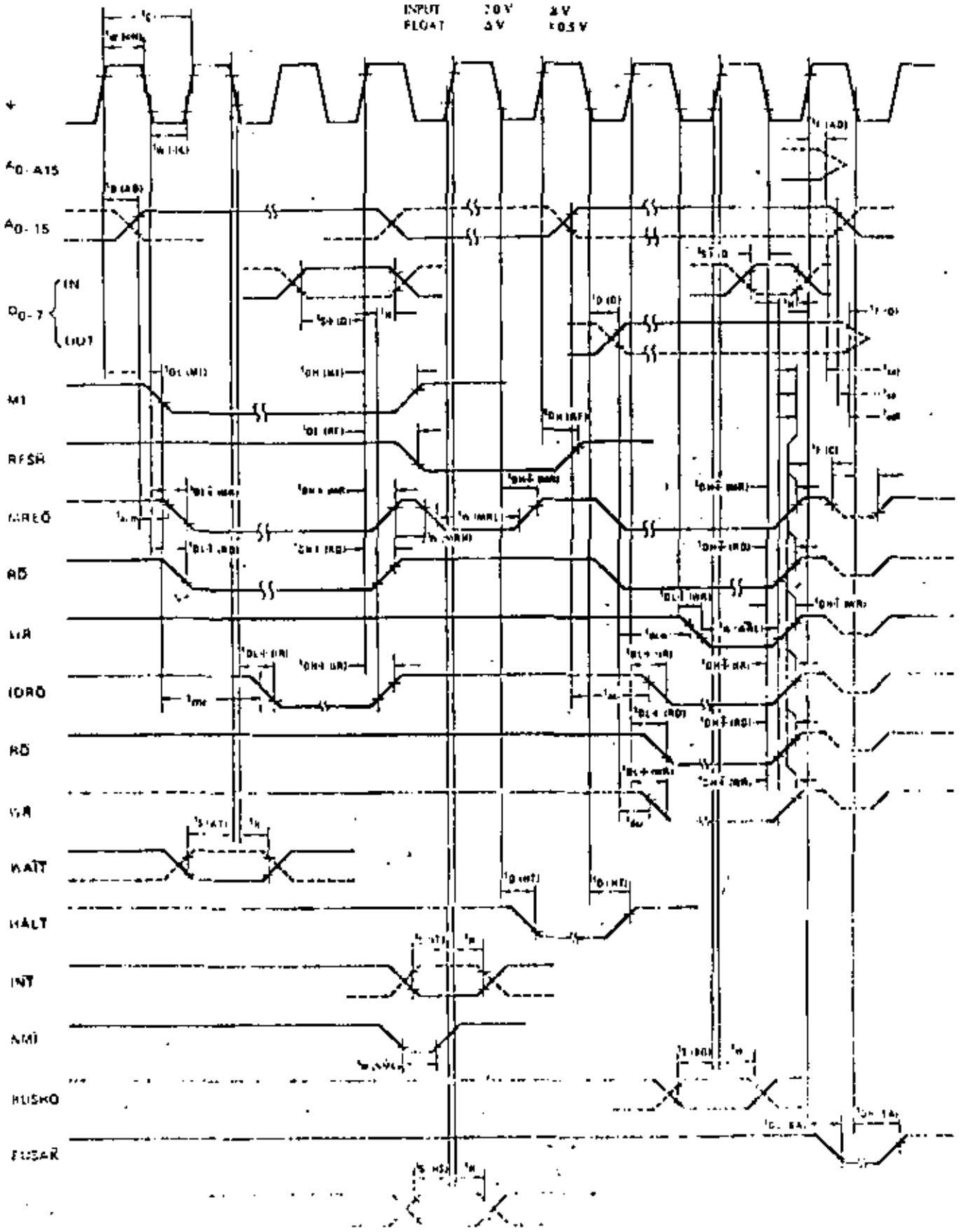
Symbol	Parameter	Max	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80A-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C

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

	"1"	"0"
CLOCK	V_{CC} + 6V	.45V
OUTPUT	2.0V	.8V
INPUT	2.0V	.8V
FLOAT	.5V	±0.5V



Zilog Z80-CPU INSTRUCTION SET

ADC HL, ss	Add with Carry Reg. pair ss to HL
ADC A, s	Add with carry operand s to Acc.
ADD A, n	Add value n to Acc.
ADD A, r	Add Reg. r to Acc.
ADD A, (HL)	Add location (HL) to Acc.
ADD A, (IX+d)	Add location (IX+d) to Acc.
ADD A, (IY+d)	Add location (IY+d) to Acc.
ADD HL, ss	Add Reg. pair ss to HL
ADD IX, pp	Add Reg. pair pp to IX
ADD IY, rr	Add Reg. pair rr to IY
AND s	Logical 'AND' of operand s and Acc.
BIT b, (HL)	Test BIT b of location (HL)
BIT b, (IX+d)	Test BIT b of location (IX+d)
BIT b, (IY+d)	Test BIT b of location (IY+d)
BIT b, r	Test BIT b of Reg. r
CALL cc, nn	Call subroutine at location nn if condition cc if true
CALL nn	Unconditional call subroutine at location nn
CCF	Complement carry flag
CP s	Compare operand s with Acc.
CPD	Compare location (HL) and Acc. decrement HL and BC
CPDR	Compare location (HL) and Acc. decrement HL and BC, repeat until BC=0
CPI	Compare location (HL) and Acc. increment HL and decrement BC
CPIR	Compare location (HL) and Acc. increment HL, decrement BC repeat until BC=0
CPL	Complement Acc. (1's comp)
DAA	Decimal adjust Acc.
DEC m	Decrement operand m
DEC IX	Decrement IX
DEC IY	Decrement IY
DEC ss	Decrement Reg. pair ss
DI	Disable interrupts
DJNZ e	Decrement B and Jump relative if B≠0
EI	Enable interrupts
EX (SP), HL	Exchange the location (SP) and HL
EX (SP), IX	Exchange the location (SP) and IX
EX (SP), IY	Exchange the location (SP) and IY
EX AF, AF'	Exchange the contents of AF and AF'
EX DE, HL	Exchange the contents of DE and HL
EXX	Exchange the contents of BC, DE, HL with contents of BC', DE', HL' respectively
HALT	HALT (wait for interrupt or reset)
IM 0	Set interrupt mode 0
IM 1	Set interrupt mode 1
IM 2	Set interrupt mode 2
IN A, (n)	Load the Acc. with input from device n
IN r, (C)	Load the Reg. r with input from device (C)
INC (HL)	Increment location (HL)
INC IX	Increment IX
INC (IX+d)	Increment location (IX+d)
INC IY	Increment IY
INC (IY+d)	Increment location (IY+d)
INC r	Increment Reg. r
INC ss	Increment Reg. pair ss
IND	Load location (HL) with input from port (C), decrement HL and B
INDR	Load location (HL) with input from port (C), decrement HL and decrement B, repeat until B=0
INI	Load location (HL) with input

[illegible] IX	[illegible] IX with top of stack
POP IY	Load IY with top of stack
POP qq	Load Reg. pair qq with top of stack
PUSH IX	Load IX onto stack
PUSH IY	Load IY onto stack
PUSH qq	Load Reg. pair qq onto stack
RES b, m	Reset Bit b of operand m
RET	Return from subroutine
RET cc	Return from subroutine if condition cc is true
RETI	Return from interrupt
RETN	Return from non maskable interrupt
RL m	Rotate left through carry operand m
RLA	Rotate left Acc. through carry
RLC (HL)	Rotate location (HL) left circular
RLC (IX+d)	Rotate location (IX+d) left circular
RLC (IY+d)	Rotate location (IY+d) left circular
RLC r	Rotate Reg. r left circular
RLCA	Rotate left circular Acc.
RLD	Rotate digit left and right between Acc. and location (HL)
RR m	Rotate right through carry operand m
RRA	Rotate right Acc. through carry
RRC m	Rotate operand m right circular
RRCA	Rotate right circular Acc.
RRD	Rotate digit right and left between Acc. and location (HL)
RST p	Restart to location p
SBC A, s	Subtract operand s from Acc. with carry
SBC HL, ss	Subtract Reg. pair ss from HL with carry
SCF	Set carry flag (C=1)
SET b, (HL)	Set Bit b of location (HL)
SET b, (IX+d)	Set Bit b of location (IX+d)
SET b, (IY+d)	Set Bit b of location (IY+d)
SET b, r	Set Bit b of Reg. r
SLA m	Shift operand m left arithmetic
SRA m	Shift operand m right arithmetic
SRL m	Shift operand m right logical
SUB s	Subtract operand s from Acc.
XOR s	Exclusive 'OR' operand s and Acc.

Z80™-PIO
Z80A™-PIO

Technical Manual

1.0 INTRODUCTION

The Z-80 Parallel I/O (PIO) 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 are 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.

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.

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 $\overline{INT}$ is active the Z80-PIO is requesting an interrupt from the Z80-CPU.

$A_0 \cdot A_7$ — Port A Bus (bidirectional, tristate)
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 the 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.
4) Control mode: This signal is disabled and forced to a low state.

$B_0 \cdot B_7$ — 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.

$\overline{B\ STB}$ — Port B Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal is similar to that of $\overline{A\ STB}$ 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.

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:

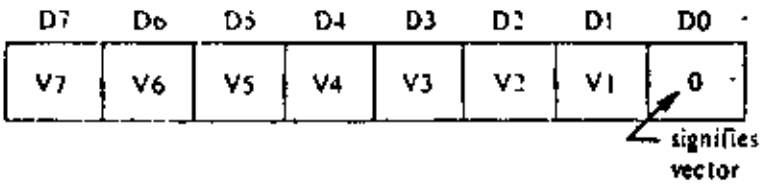
1) Both port mask registers are reset to inhibit all port data bits.
2) Port data bus lines are set to a high impedance state and the Ready "handshake" signals are inactive (low). Mode 1 is automatically selected.
3) The vector address registers are *not* reset.
4) Both port interrupt enable flip flops 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.

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 Z-80 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:



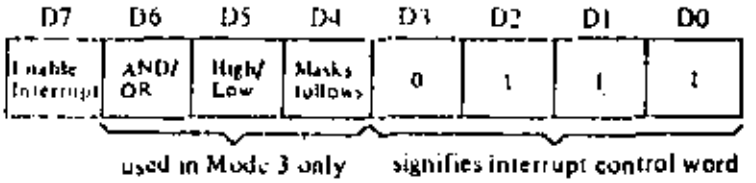
D0 is used in this case as a flag bit which when low causes V7 thru V1 to be loaded into the vector register. At interrupt acknowledge time, the vector of the interrupting port will appear on the Z-80 data bus exactly as shown in the format above.

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. 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

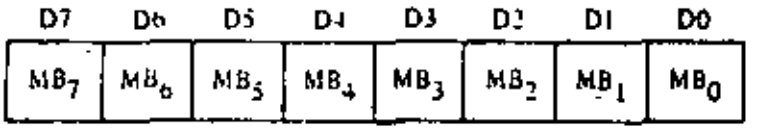
The interrupt control word for each port has the following format:



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 is pending when the enable flag is set, it will then be enabled onto the CPU interrupt request line. Bits D6, D5, and D4 are used only with Mode 3 operation. However, setting bit D4 of the interrupt control word during any mode of operation will cause any 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, an 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:



Only those port lines whose mask bit is zero will be monitored for generating an interrupt.

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 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.

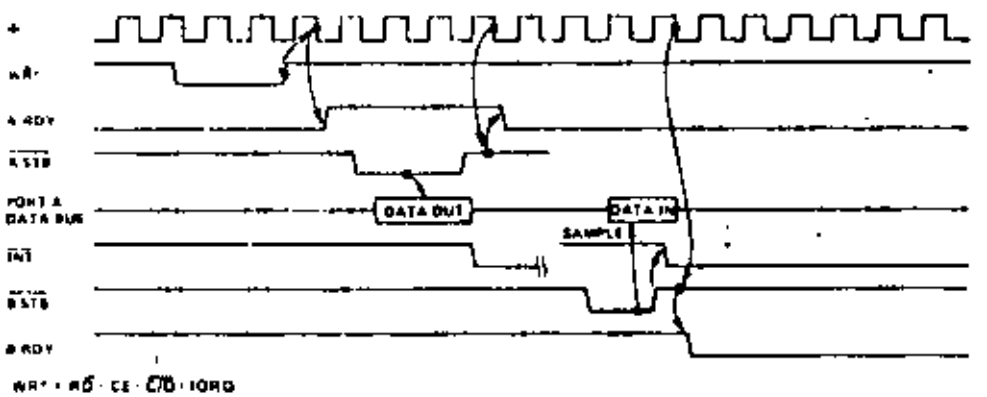


FIGURE 5.0-3
PORT A, MODE 2 (BIDIRECTIONAL) TIMING

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 latch 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.

5.4 CONTROL MODE (MODE 3)

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 $\overline{\text{RD}}$. See Figure 5.0-4.

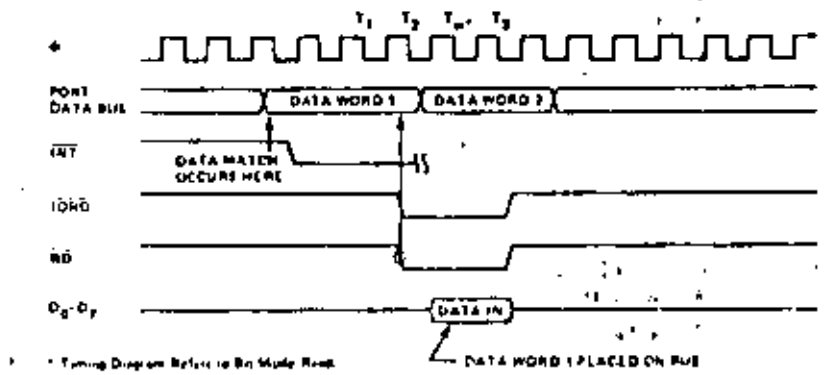


FIGURE 5.0-4

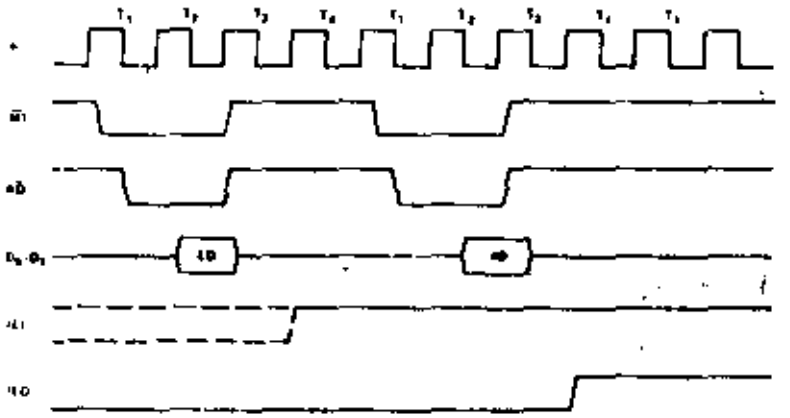


FIGURE 6.0-2
RETURN FROM INTERRUPT CYCLE

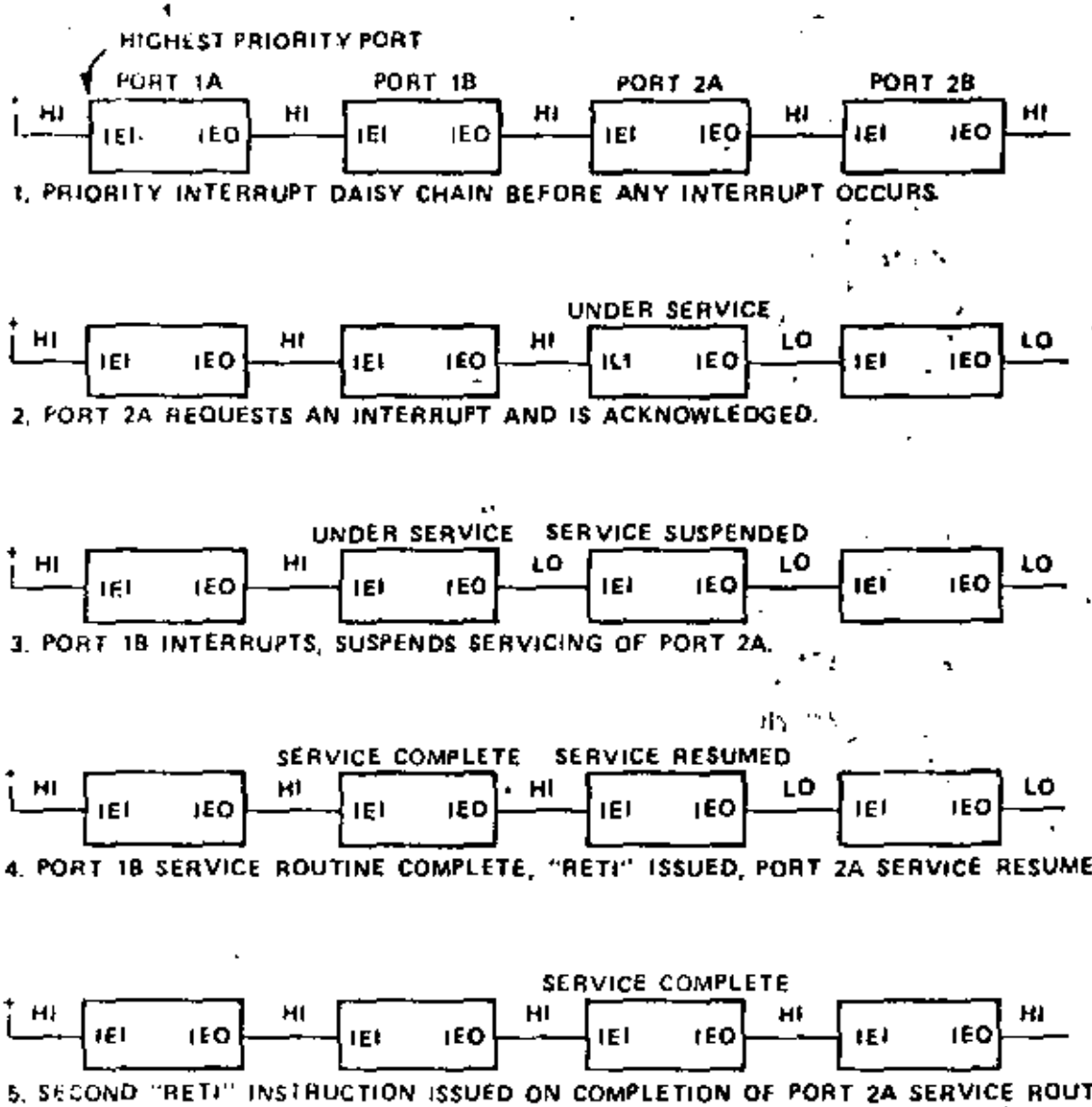


FIGURE 6.0-3
DAISY CHAIN INTERRUPT SERVICING

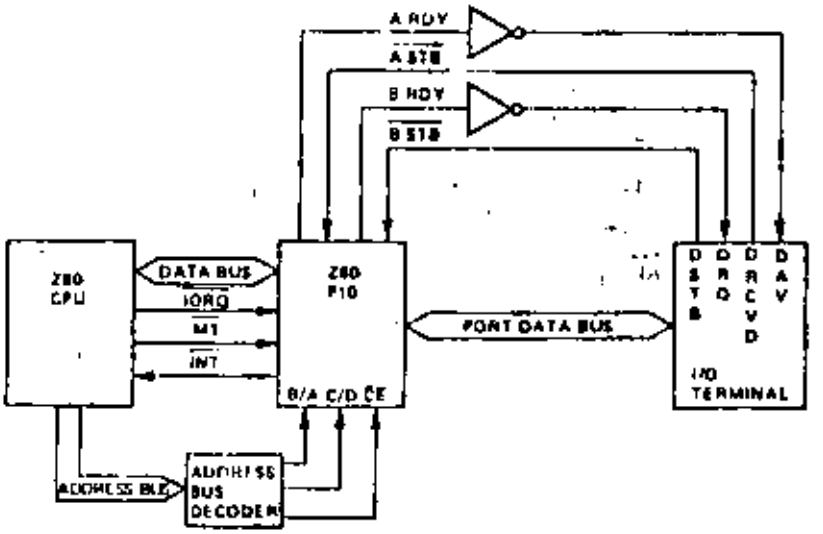


FIGURE 7.0-2

EXAMPLE I/O INTERFACE

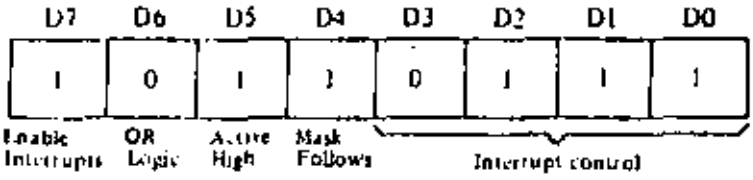
Next, the proper interrupt vector is loaded (refer to CPU Manual for details on the operation of the interrupt).

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

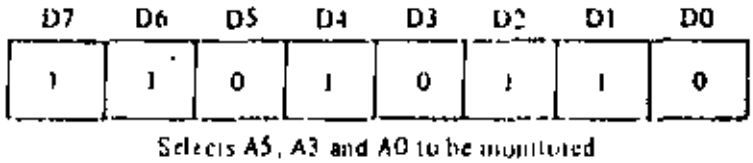
Interrupts are then enabled by the rising edge of the first $\overline{M1}$ after the interrupt mode word is set unless that $\overline{M1}$ defines an interrupt acknowledge cycle. If a mask follows the interrupt mode word, interrupts are enabled by the rising edge of the first $\overline{M1}$ following 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.

An interrupt control word is next sent to the port:

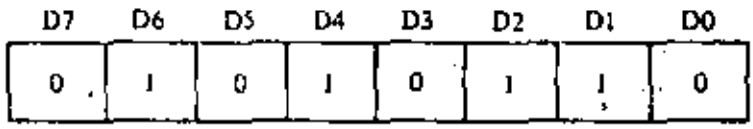


The mask word following the interrupt mode word is:



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:



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 (1110 00). 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.

Absolute Maximum Ratings

Temperature Under Bias	Specified operating range
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect To Ground	−0.3 V to +7 V
Power Dissipation	1 W

**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 specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note All AC and DC characteristics remain the same for the military grade parts except I_{CC}.

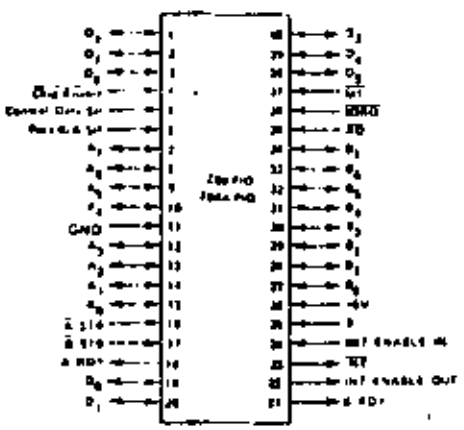
I_{CC} = 130 mA

Z80-PIO and Z80A-PIO D.C. Characteristics

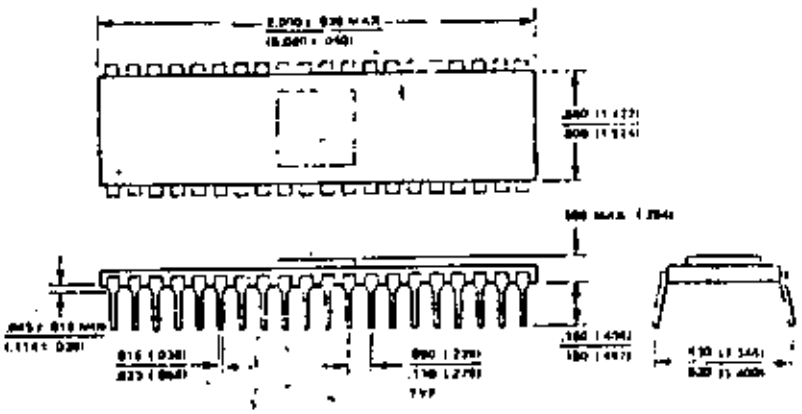
T_A = 0°C to 70°C, V_{CC} = 5 V ± 5%, unless otherwise specified

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}−0.6	V_{CC}+0.3	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} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −250 μA
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	$0 \leq V_{IN} \leq V_{CC}$
I_{OHD}	Darlington Drive Current	−1.5	−3.0	mA	V_{OH} = 1.5 V R_{EXT} = 390 Ω Port B Only

Package Configuration



Package Outline

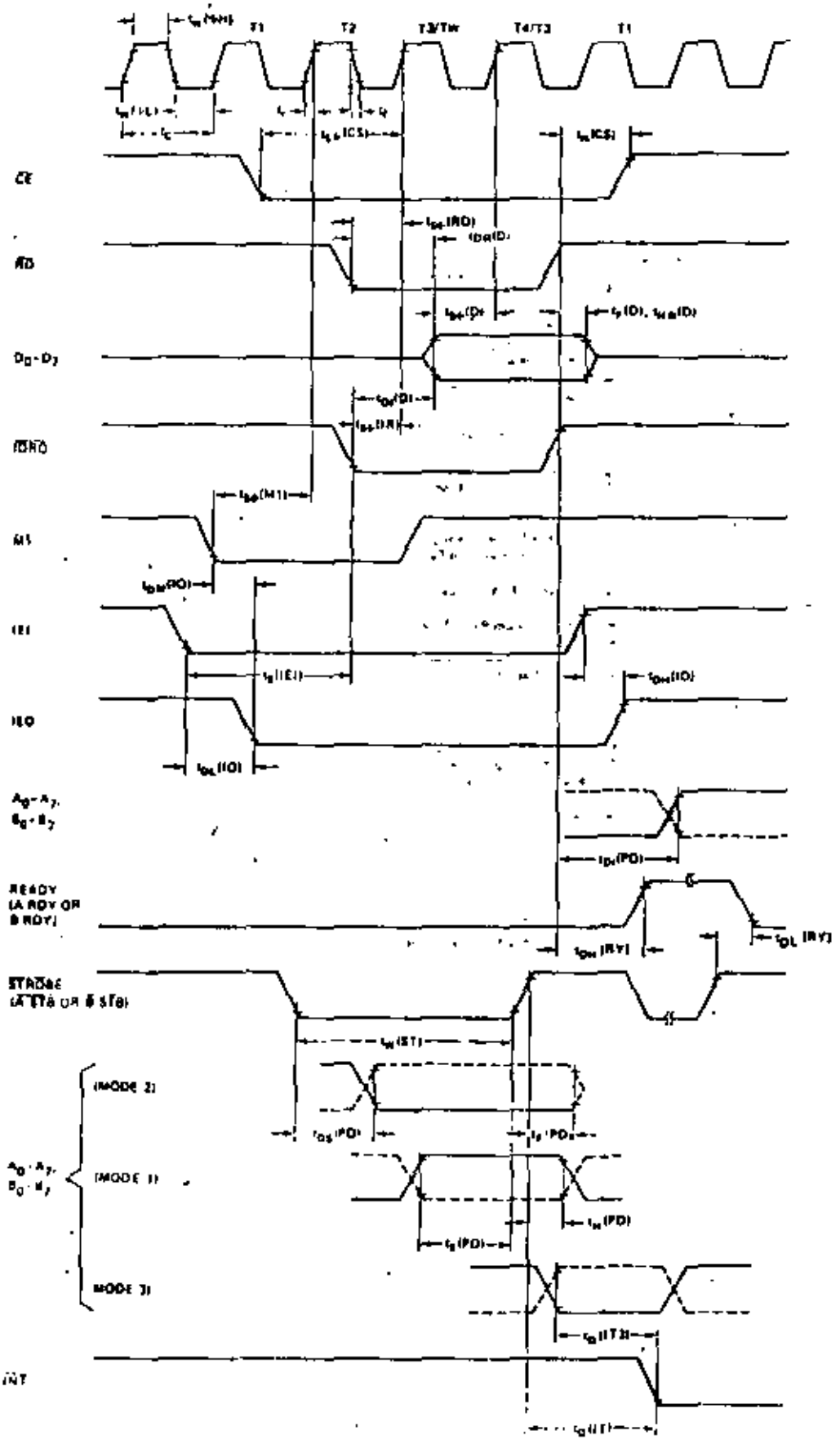


NOTE: Dimensions in parentheses are for metric system (cm).

A.C. Timing Diagram

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

	"1"	"0"
CLOCK	$V_{CC}-.4$	.45V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	ΔV	+0.5V



Z-80® SIO
TECHNICAL MANUAL

The Z80-SIO (Serial Input/Output) is a dual-channel multi-function peripheral component designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic function is a serial-to-parallel, parallel-to-serial converter/controller, but—within that role—it is configurable by systems software so its "personality" can be optimized for a given serial data communications application.

The Z80-SIO is capable of handling asynchronous and synchronous byte-oriented protocols such as IBM Bisync, and synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device can also be used to support virtually any other serial protocol for applications other than data communications (cassette or floppy disk interfaces, for example).

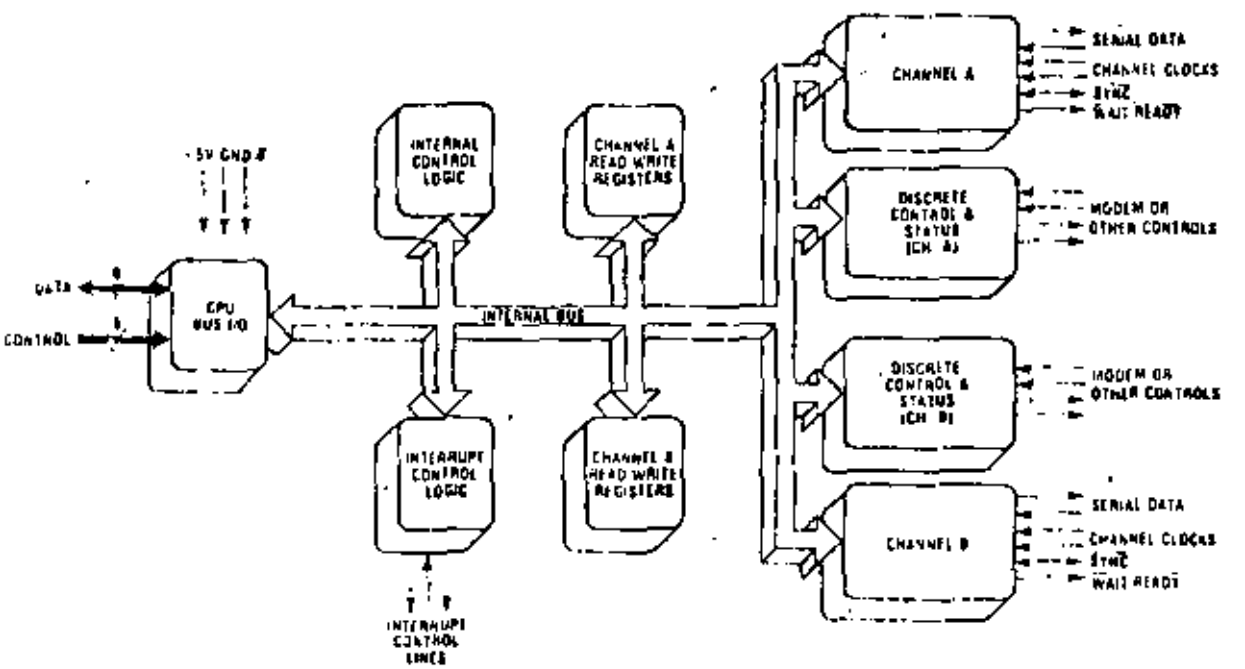
The Z80-SIO can generate and check CRC codes in any synchronous mode and can be programmed to check data integrity in various modes. The device also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

STRUCTURE

- N-channel silicon-gate depletion-load technology
- 40-pin DIP
- Single 5 V power supply
- Single-phase 5 V clock
- All inputs and outputs TTL compatible

FEATURES

- Two independent full-duplex channels
- Data rates in synchronous or isosynchronous modes:
  - 0-550K bits/second with 2.5 MHz system clock rate
  - 0-880K bits/second with 4.0 MHz system clock rate
- Receiver data registers quadruply buffered; transmitter doubly buffered.
- Asynchronous features:
  - 5, 6, 7 or 8 bits/character
  - 1, 1½ or 2 stop bits
  - Even, odd or no parity
  - ×1, ×16, ×32 and ×64 clock modes
  - Break generation and detection
  - Parity, overrun and framing error detection



Z80-SIO BLOCK DIAGRAM

the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active, but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information as specified by C/$\overline{D}$. As mentioned previously, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the Z80-SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{RD}$. *Read Cycle Status.* (input from CPU, active Low). If $\overline{RD}$ is active, a memory or I/O read operation is in progress. $\overline{RD}$ is used with B/$\overline{A}$, $\overline{CE}$ and $\overline{IORQ}$ to transfer data from the Z80-SIO to the CPU.

$\overline{RESET}$. *Reset* (input, active Low). A Low $\overline{RESET}$ disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the Z80-SIO is reset and before data is transmitted or received.

IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80-SIO. 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 the Z80-SIO is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{W/RDYA}$, $\overline{W/RDYB}$. *Wait/Ready A, Wait/Ready B* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80-SIO data rate. The reset state is open drain.

$\overline{CTSA}$, $\overline{CTSB}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime inputs. The Z80-SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger inputs do not guarantee a specified noise-level margin.

$\overline{DCDA}$, $\overline{DCDB}$. *Data Carrier Detect* (inputs, active Low). These signals are similar to the CTS inputs, except they can be used as receiver enables.

RxDA, RxDB. *Receive Data* (inputs, active High).

TxDA, TxDB. *Transmit Data* (outputs, active High).

$\overline{RxCA}$, $\overline{RxCB}$.* *Receiver Clocks* (inputs). See the following section on bonding options. The Receive Clock may be 1, 16, 32 or 64 times the data rate in asynchronous modes. Receive data is sampled on the rising edge of RxC.

*See footnote on next page.

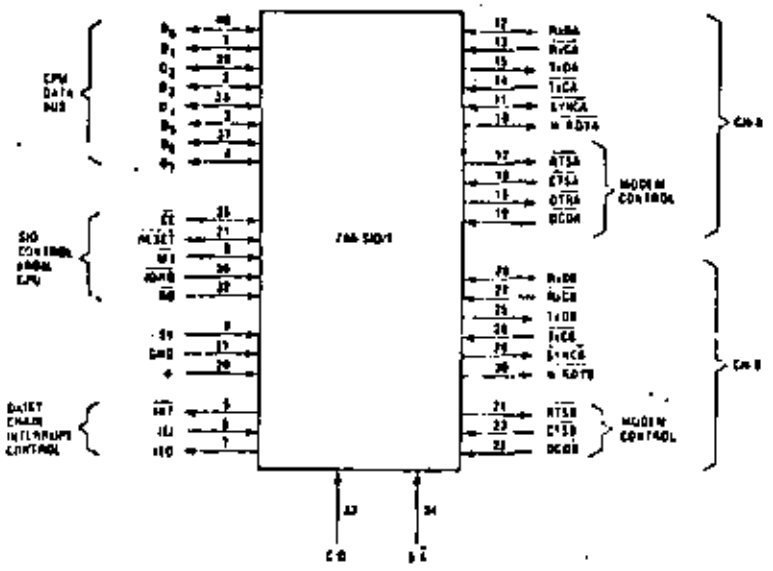


Figure 2. Z80-SIO/1 Pin Configuration

The device internal structure includes a Z80-CPU interface, internal control and interrupt logic, and two full-duplex channels. Associated with each channel are read and write registers, and discrete control and status logic that provides the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers, two sync-character registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through Read Register 2 in Channel B. The registers for both channels are designated in the text as follows:

WR0-WR7 — Write Registers 0 through 7
RR0-RR2 — Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process. Table 1 illustrates the functions assigned to each read or write register.

WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and controls
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

(a) Write Register Functions

RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

(b) Read Register Functions

Table 1. Functional Assignments of Read and Write Registers

The logic for both channels provides formats, synchronization and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send (CTS) and Data Carrier Detect (DCD) are monitored by the discrete control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/ Status interrupts are prioritized in that order within each channel.

Data Path

The transmit and receive data path for each channel is shown in Figure 4. The receiver has three 8-bit buffer registers in a FIFO arrangement (to provide a 3-byte delay) in addition to the 8-bit receive shift register. This arrangement creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. The receive error FIFO stores parity and framing errors and other types of status information for each of the three bytes in the receive data FIFO.

Incoming data is routed through one of several paths depending on the mode and character length. In the Asynchronous mode, serial data is entered in the 3-bit buffer if it has a character length of seven or eight bits, or is entered in the 8-bit receive shift register if it has a length of five or six bits.

In the Synchronous mode, however, the data path is determined by the phase of the receive process currently in operation. A Synchronous Receive operation begins with the receiver in the Hunt phase, during which the receiver searches the incoming data stream for a bit pattern that matches the preprogrammed sync characters (or flags in the SDLC mode). If the device is programmed for Monosync Hunt, a match is made with a single sync character stored in WR7. In Bisync Hunt, a match is made with dual sync characters stored in WR6 and WR7.

In either case the incoming data passes through the receive sync register, and is compared against the programmed sync character in WR6 or WR7. In the Monosync mode, a match between the sync character programmed into WR7 and the character assembled in the receive sync register establishes synchronization.

In the Bisync mode, however, incoming data shifted to the receive shift register while the next eight bits of the message are assembled in the receive sync register. The match between the assembled character in the receive sync registers with the programmed sync character in WR6 and WR7 establishes synchronization. Once synchronization is established, incoming data by-

Asynchronous data in the transmit shift register is formatted with start and stop bits and is shifted out to the transmit multiplexer at the selected clock rate. Synchronous (Monosync or Bisync) data is shifted out to the transmit multiplexer and also to the CRC generator at the ×1 clock rate.

SDLC/HDLC data is shifted out through the zero insertion logic, which is disabled while the flags are being sent. For all other fields (address, control and frame check) a 0 is inserted following five contiguous 1's in the data stream. The CRC generator result for SDLC data is also routed through the zero insertion logic.

Interrupts. The Z80-SIO offers an elaborate interrupt scheme to provide fast interrupt response in real-time applications. As mentioned earlier, Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To serv operations in both channels and to eliminate the nece sity of writing a status analysis routine, the Z80-SIO can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D_2) in Channel B called "Status Affects Vector." When this bit is set, the interrupt vector in WR2 is modified according to the assigned priority of the various interrupting conditions. The table in the Write Register 1 description (Z80-SIO Programming section) shows the modification details.

Functional Description

The functional capabilities of the Z80-SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of various data communications protocols; as a Z80 family peripheral, it interacts with the Z80-CPU and other Z80 peripheral circuits, and shares their data, address and control busses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the Z80-SIO offers valuable features such as non-vectored interrupts, polling and simple handshake capabilities.

The first part of the following functional description describes the interaction between the CPU and Z80-SIO; the second part introduces its data communications capabilities.

I/O CAPABILITIES

The Z80-SIO offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

Polling. The Polled mode avoids interrupts. Status registers RR0 and RR1 are updated at appropriate times for each function being performed (for example, CRC Error status valid at the end of the message). All the interrupt modes of the Z80-SIO must be disabled to operate the device in a polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D_0 and D_2 indicate that a receive or transmit data transfer is needed. The status also indicates Error or other special status conditions (see "Z80-SIO Programming"). The Special Receive Condition status contained in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

Transmit Interrupts, Receive Interrupts and External/Status Interrupts are the main sources of interrupts (Figure 5). Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit Interrupt is enabled, the CPU is interrupted by the transmit buffer *becoming* empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on first receive character
- Interrupt on all receive characters
- Interrupt on a Special Receive condition

Interrupt On First Character is typically used with the Block Transfer mode. Interrupt On All Receive Characters has the option of modifying the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character or message basis (End Of Frame interrupt in SDLC, for example). The Special Receive condition can cause an interrupt only if the Interrupt On First Receive Character or Interrupt On All Receive Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except Parity Error) after the first receive character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{CTS}$, $\overline{DCD}$ and $\overline{SYNC}$ pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition or by the detection of a Break (Asynchronous mode) or Abort (SDLC mode) sequence in the data stream. The interrupt caused by the Break/Abort sequence has a special feature that allows the Z80-SIO to interrupt when the Break/Abort sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break/Abort condition in external logic.

To receive or transmit data in the Asynchronous mode, the Z80-SIO must be initialized with the following parameters: character length, clock rate, number of stop bits, even or odd parity, interrupt mode, and receiver or transmitter enable. The parameters are loaded into the appropriate write registers by the system program. WR4 parameters must be issued before WR1, WR3 and WR5 parameters or commands.

If the data is transmitted over a modem or RS232C interface, the REQUEST TO SEND (RTS) and DATA TERMINAL READY (DTR) outputs must be set along with the Transmit Enable bit. Transmission cannot begin until the Transmit Enable bit is set.

The Auto Enables feature allows the programmer to send the first data character of the message to the Z80-SIO without waiting for CTS. If the Auto Enables bit is set, the Z80-SIO will wait for the CTS pin to go Low before it begins data transmission. CTS, DCD and SYNC are general-purpose I/O lines that may be used for functions other than their labeled purposes. If CTS is used for another purpose, the Auto Enables Bit must be programmed to 0.

Figure 6 illustrates asynchronous message formats; Table 2 shows WR3, WR4 and WR5 with bits set to indicate the applicable modes, parameters and commands in asynchronous modes. WR2 (Channel B only) stores the interrupt vector; WR1 defines the interrupt modes and data transfer modes. WR6 and WR7 are not used in asynchronous modes. Table 3 shows the typical program steps that implement a full-duplex receive/transmit operation in either channel.

Asynchronous Transmit

The Transmit Data output (TxD) is held marking (High) when the transmitter has no data to send. Under program control, the Send Break (WR5, D_4) command can be issued to hold TxD spacing (Low) until the command is cleared.

The Z80-SIO automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits to the data character to be transmitted. When the character length is six or seven bits, the unused bits are automatically ignored by the Z80-SIO. If the character length is five bits or less, refer to the table in the Write Register 5 description (Z80-SIO Programming section) for the data format.

Serial data is shifted from TxD at a rate equal to 1, 1/16th, 1/32nd or 1/64th of the clock rate supplied to the Transmit Clock input (TxC). Serial data is shifted out on the falling edge of (TxC).

If set, the External/Status Interrupt mode monitors the status of DCD, CTS and SYNC throughout the transmission of the message. If these inputs change for a period of time greater than the minimum specified pulse width, the interrupt is generated. In a transmit operation, this feature is used to monitor the modem control signal CTS.

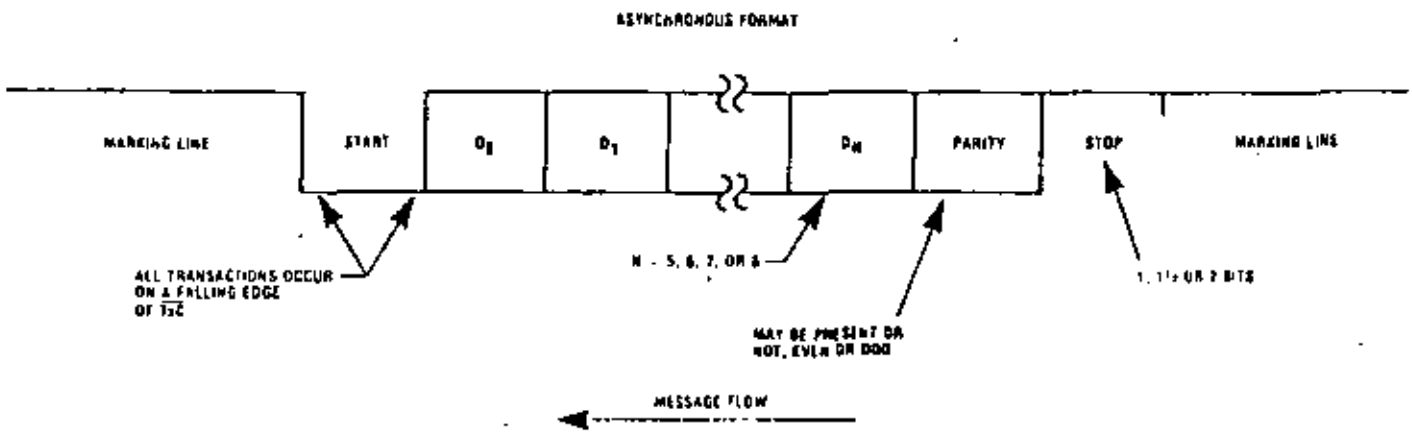


Figure 6. Asynchronous Message Format

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
	REGISTER	*INFORMATION LOADED:*	
INITIALIZE	WR0	CHANNEL RESET	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4, RESET EXTERNAL/STATUS INTERRUPT	
	WR4	ASYNCHRONOUS MODE, PARITY INFORMATION, STOP BITS INFORMATION, CLOCK RATE INFORMATION	Issue parameters
	WR0	POINTER 3	
	WR3	RECEIVE ENABLE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	
	WR0	POINTER 5	
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, TRANSMIT CHARACTER LENGTH, DATA TERMINAL READY	Receive and Transmit both fully initialized. Auto Enables will enable Transmitter if $\overline{CTS}$ is active and Receiver if $\overline{DCD}$ is active.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	TRANSMIT INTERRUPT ENABLE, STATUS AFFECTS VECTOR, INTERRUPT ON ALL RECEIVE CHARACTERS, DISABLE WAIT/READY FUNCTION, EXTERNAL INTERRUPT ENABLE	Transmit/Receive interrupt mode selected. External Interrupt monitors the status of the $\overline{CTS}$, $\overline{DCD}$ and $\overline{SYNC}$ inputs and detects the Break sequence. Status Affects Vector in Channel B only.
	TRANSFER FIRST DATA BYTE TO SIO		This data byte must be transferred; no transmit interrupts will occur.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		Program is waiting for an interrupt from the SIO.
DATA TRANSFER AND ERROR MONITORING	Z80 INTERRUPT ACKNOWLEDGE CYCLE TRANSFERS RR2 TO CPU		When the interrupt occurs, the interrupt vector is modified by: 1. Receive Character Available; 2. Transmit Buffer Empty; 3. External/Status change; and 4. Special Receive condition.
	IF A CHARACTER IS RECEIVED: • TRANSFER DATA CHARACTER TO CPU • UPDATE POINTERS AND PARAMETERS • RETURN FROM INTERRUPT		Program control is transferred to one of the eight interrupt service routines.
	IF TRANSMITTER BUFFER IS EMPTY: • TRANSFER DATA CHARACTER TO SIO • UPDATE POINTERS AND PARAMETERS • RETURN FROM INTERRUPT		If used with processors other than the Z80, the modified interrupt vector (RR2) should be returned to the CPU in the Interrupt Acknowledge sequence.
	IF EXTERNAL STATUS CHANGES: • TRANSFER RR0 TO CPU • PERFORM ERROR ROUTINES (INCLUDE BREAK DETECTION) • RETURN FROM INTERRUPT		
	IF SPECIAL RECEIVE CONDITION OCCURS • TRANSFER RR1 TO CPU • DO SPECIAL ERROR (E.G. FRAMING ERROR) ROUTINE • RETURN FROM INTERRUPT		
TERMINATION	REDEFINE RECEIVE/TRANSMIT INTERRUPT MODES		When transmit or receive data transfer is complete.
	DISABLE TRANSMIT/RECEIVE MODES		
	UPDATE MODEM CONTROL OUTPUTS (E.G. RTS OFF)		In Transmit, the All Sent status indicates transmission is complete.

Table 3. Asynchronous Mode

Synchronous Operation

Before describing synchronous transmission and reception, the three types of character synchronization—Monosync, Bisync and External Sync—require some explanation. These modes use the ×1 clock for both Transmit and Receive operations. Data is sampled on the rising edge of the Receive Clock input ($\overline{RxC}$). Transmitter data transitions occur on the falling edge of the Transmit Clock input ($\overline{TxC}$).

The differences between Monosync, Bisync and External Sync are in the manner in which initial character synchronization is achieved. The mode of operation must be selected before sync characters are loaded, because the registers are used differently in the various modes. Figure 7 shows the formats for all three of these synchronous modes.

Monosync. In a Receive operation, matching a single sync character (8-bit sync mode) with the programmed sync character stored in WR7 implies character synchronization and enables data transfer.

Bisync. Matching two contiguous sync characters (16-bit sync mode) with the programmed sync characters stored in WR6 and WR7 implies character synchronization. In both the Monosync and Bisync modes, $\overline{SYNC}$ is used as an output, and is active for the part of the receive clock that detects the sync character.

External Sync. In this mode, character synchronization is established externally; $\overline{SYNC}$ is an input that indicates external character synchronization has been achieved. After the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{SYNC}$ input. The SYNC input must be held Low until character synchronization is lost. Character assembly begins on the rising edge of $\overline{RxC}$ that precedes the falling edge of $\overline{SYNC}$.

In all cases after a reset, the receiver is in the Hunt phase, during which the Z80-SIO looks for character synchronization. The hunt can begin only when the receiver is enabled, and data transfer can begin only when character synchronization has been achieved. If character synchronization is lost, the Hunt phase can be re-entered by writing a control word with the Enter Hunt Phase bit set (WR3, D4). In the Transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16). In the Monosync mode, the transmitter transmits from WR6; the receiver compares against WR7.

In the Monosync, Bisync and External Sync modes, assembly of received data continues until the Z80-SIO is reset, or until the receiver is disabled (by command or by $\overline{DCD}$ in the Auto Enables mode), or until the CPU sets the Enter Hunt Phase bit.

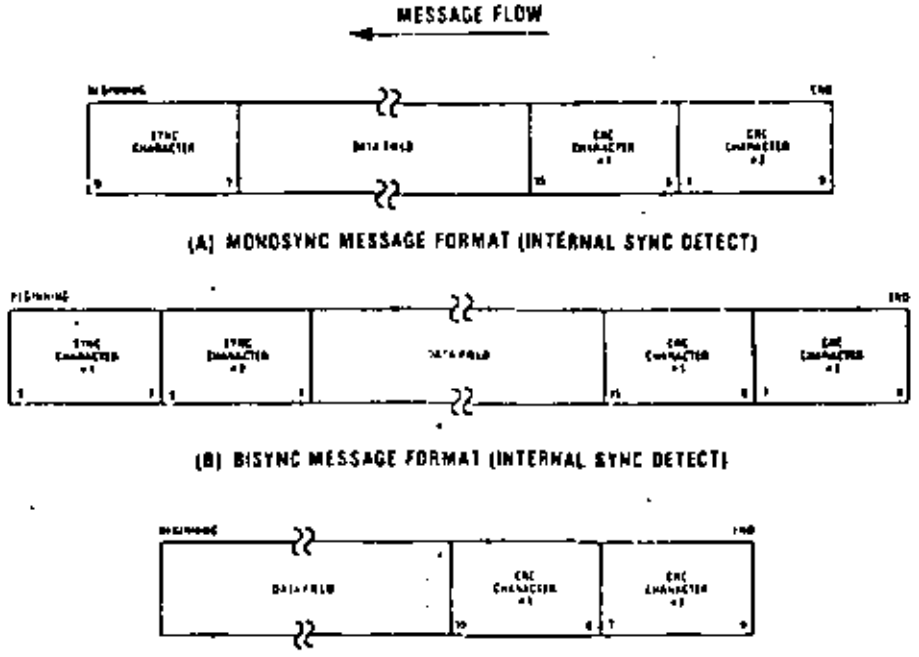


Figure 7. Synchronous Formats

Data Transfer Using WAIT/READY. To the CPU, the activation of WAIT indicates that the Z80-SIO is not ready to accept data and that the CPU must extend the output cycle. To a DMA controller, READY indicates that the transmit buffer is empty and that the Z80-SIO is ready to accept the next data character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition.

Bisync Transmit Underrun. In Bisync protocol, filler characters are inserted to maintain synchronization when the transmitter has no data to send (Transmit Underrun condition). The Z80-SIO has two programmable options for solving this situation: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters.

These options are under the control of the Reset Transmit Underrun/EOM command in WR0. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0, D_6) is in a set condition and allows the insertion of sync characters when there is no data to send. CRC is not calculated on the automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data. In this case, the Z80-SIO sends CRC, followed by sync characters, to terminate the message.

There is no restriction as to when in the message the Transmit Underrun/EOM bit can be reset. If Reset is issued after the first data character has been loaded the 16-bit CRC is sent and followed by sync characters the first time the transmitter has no data to send. Because of the Transmit Underrun condition, an External/Status interrupt is generated whenever the Transmit Underrun/EOM bit becomes set.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded. The status indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5, D_3).

Pad characters may be sent by setting the Z80-SIO to 8 bits/transmit character and writing FF to the transmitter while CRC is being sent. Alternatively, the sync characters can be redefined as pad characters during this time. The following example is included to clarify this point.

The Z80-SIO interrupts with the Transmit Buffer Empty bit set.

The CPU recognizes that the last character (ETX) of the message has already been sent to the Z80-SIO by examining the internal program status.

To force the Z80-SIO to send CRC, the CPU issues the Reset Transmit Underrun/EOM Latch command (WR0) and satisfies the interrupt with the Reset Transmit Interrupt Pending command. (This command prevents the Z80-SIO from requesting more data.) Because of the transmit underrun caused by this command, the Z80-SIO starts sending CRC. The Z80-SIO also causes an External/Status interrupt with the Transmit Underrun/EOM latch set.

The CPU satisfies this interrupt by loading pad characters into the transmit buffer and issuing the Reset External/Status interrupt command.

With this sequence, CRC is followed by a pad character instead of a sync character. Note that the Z80-SIO will interrupt with a Transmit Buffer Empty interrupt when CRC is completely sent and that the pad character is loaded into the transmit shift register.

From this point on the CPU can send more pad characters or sync characters.

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5, D_0) initiates CRC accumulation when the program sends the first data character to the Z80-SIO. Although the Z80-SIO automatically transmits up to two sync characters (16-bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded from the transmit data buffer into the transmit shift register. To ensure this bit is in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the Z80-SIO.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16-bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the Z80-SIO.

In the case of a Transmit Underrun condition in the Transparent mode, a pair of DLE-SYN characters are sent. The Z80-SIO can be programmed to send the DLE-SYN sequence by loading a DLE character into WR6 and a sync character into WR7.

Transmit Termination. The Z80-SIO is equipped with special termination feature that maintains data integrity and validity. If the transmitter is disabled while a data or sync character is being sent, that character is sent as usual, but is followed by a marking line rather than CRC or sync characters. When the transmitter is disabled, a

character in the buffer remains in the buffer. If the transmitter is disabled while CRC is being sent, the 16-bit transmission is completed, but sync is sent instead of CRC.

A programmed break is effective as soon as it is written into the control register; characters in the transmit buffer and shift register are lost.

In all modes, characters are sent with the least significant bits first. This requires right-hand justification of transmitted data if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 discussion (Z80-SIO Programming section) must be used for the data format. The states of any unused bits in a data character are irrelevant, except when in the Five Bits Or Less mode.

If the External/Status Interrupt Enable bit is set, transmitter conditions such as "starting to send CRC characters," "starting to send sync characters," and $\overline{CTS}$ changing state cause interrupts that have a unique vector if Status Affects Vector is set. This interrupt mode may be used during block transfers.

All interrupts may be disabled for operation in a Polled mode, or to avoid interrupts at inappropriate times during the execution of a program.

Synchronous Receive

INITIALIZATION

The system program initiates the Synchronous Receive operation with the following parameters: odd or even parity, 8- or 16-bit sync characters, ×1 clock mode, CRC polynomial, receive character length, etc. Sync characters must be loaded into registers WR6 and WR7. The receivers can be enabled only after all receive parameters are set. WR4 parameters must be issued before WR1, WR3, WR5, WR6 and WR7 parameters or commands.

After this is done, the receiver is in the Hunt phase. It remains in this phase until character synchronization is achieved. Note that, under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit bit in WR3.

DATA TRANSFER AND STATUS MONITORING

After character synchronization is achieved, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer the data and its associated status to the CPU.

No Interrupts Enabled. This mode is used for a purely polled operation or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a polling loop or a Block Transfer instruction using $\overline{WAIT/READY}$ to synchronize the CPU or the DMA device to the incoming data rate. In this mode the Z80-SIO interrupts on the first character and thereafter interrupts only if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character command to allow the next character received to generate an interrupt. Parity errors do not cause interrupts in this mode, but End Of Frame (SDLC mode) and Receive Overrun do.

If External/Status interrupts are enabled, they may interrupt any time $\overline{DCD}$ changes state.

Interrupt On Every Character. Whenever a character enters the receive buffer, an interrupt is generated. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected. Optionally, a Parity Error may be directed not to generate the special interrupt vector.

Special Receive Condition Interrupts. The Special Receive Condition interrupt can occur only if either the Receive Interrupt On First Character Only or Interrupt On Every Receive Character modes is also set. The Special Receive Condition interrupt is caused by the Receive Overrun error condition. Since the Receive Overrun and Parity error status bits are latched, the error status—when read—reflects 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. These status bits can only be reset by the Error reset command.

CRC Error Checking and Termination. A CRC error check on the receive message can be performed on a per character basis under program control. The Receive CRC Enable bit (WR3, D_3) must be set/reset by the program before the next character is transferred from the receive shift register into the receive buffer register. This ensures proper inclusion or exclusion of data characters in the CRC check.

To allow the CPU ample time to enable or disable the CRC check on a particular character, the Z80-SIO calculates CRC eight bit times after the character has been transferred to the receive buffer. If CRC is enabled before the next character is transferred, CRC is calculated on the transferred character. If CRC is disabled before the time of the next transfer, calculation proceeds on the word in progress, but the word just transferred to the buffer is not included. When these requirements are satisfied, the 3-byte receive data buffer is, in effect, unusable in Bisync operation. CRC may be enabled and disabled as many times as necessary for a given calculation.

In the Monosync, Bisync and External Sync modes, the CRC/Framing Error bit (RR1, D_6) contains the comparison result of the CRC checker 16 bit times (eight bits delay and eight shifts for CRC) after the character has been transferred from the receive shift register to the buffer. The result should be zero, indicating an error-

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
INITIALIZE (CONTINUED)	WR0	POINTER 3 ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Resetting this interrupt mode provides simple program loopback entry for the next transaction.
	WR3	RECEIVE ENABLE, SYNC CHARACTER LOAD INHIBIT, ENTER HUNT MODE, AUTO ENABLE, RECEIVE WORD LENGTH	WR3 is reissued to enable receiver. Receive CRC Enable must be set after receiving SOH or STX character.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		Receive mode is fully initialized and the system is waiting for interrupt on first character.
DATA TRANSFER AND STATUS MONITORING	*WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING:* • TRANSFERS DATA BYTE TO CPU • DETECTS AND SETS APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU) • INCLUDES/EXCLUDES DATA BYTE IN CRC CHECKER • UPDATES POINTERS AND OTHER PARAMETERS • ENABLES WAIT/READY FOR DMA OPERATION • ENABLES DMA CONTROLLER • RETURNS FROM INTERRUPT		During the Hunt mode, the SIO detects two contiguous characters to establish synchronization. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller. The controller is also programmed to capture special characters (by examining only the bits that specify ASCII or EBCDIC control characters) and interrupt the CPU upon detection. In response, the CPU examines the status or control characters and takes appropriate action (e.g. CRC Enable Update).
	WHEN WAIT/READY BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: • TRANSFERS DATA BYTE TO MEMORY • INTERRUPTS CPU IF A SPECIAL CHARACTER IS CAPTURED BY THE DMA CONTROLLER • INTERRUPTS THE CPU IF THE LAST CHARACTER OF THE MESSAGE IS DETECTED		
	FOR MESSAGE TERMINATION, THE CPU DOES THE FOLLOWING: • TRANSFERS RR1 TO THE CPU • SETS ACK/NAK REPLY FLAG BASED ON CRC RESULT • UPDATES POINTERS AND PARAMETERS • RETURNS FROM INTERRUPT		The SIO interrupts the CPU for error condition, and the error routine aborts the present message, clears the error condition, and repeats the operation.
TERMINATION	REDEFINE INTERRUPT MODES AND SYNC MODES UPDATE MODEM CONTROLS DISABLES RECEIVE MODE		

Table 8. Bisync Receive Mode (Continued)

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit is set, an interrupt is generated each time the buffer becomes empty. The interrupt may be satisfied either by writing another character into the transmitter or by resetting the Transmit Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CNTD5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there are no further transmitter interrupts. The result is a Transmit Underrun condition. When another character is written and sent out, the transmitter can again become empty and interrupt the CPU. Following the flags in an SDLC operation, the 8-bit address field, control field and information field may be sent to the Z80-SIO using the Transmit Interrupt mode. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

When the transmitter is first enabled, it is already empty and obviously cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

When the transmitter is first enabled, it is already empty and cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

Data Transfer Using Wait/Ready. If the Wait/Ready function has been selected, WAIT indicates to the CPU that the Z80-SIO is not ready to accept the data and the CPU must extend the I/O cycle. To a DMA controller, READY indicates that the transmitter buffer is empty and that the Z80-SIO is ready to accept the next character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition. Address, control and information fields may be transferred to the Z80-SIO with this mode using the Wait/Ready function. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

SDLC Transmit Underrun/End Of Message. SDLC-like protocols do not have provisions for fill characters within a message. The Z80-SIO therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. does this by first sending the two bytes of CRC and following these with one or more flags. This technique allows very high-speed transmissions under DMA or CPU block I/O control without requiring the CPU to respond quickly to the end of message situation.

The action that the Z80-SIO takes in the underrun situation depends on the state of the Transmit Underrun/EOM command. Following a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Consequently, flag characters are sent. The Z80-SIO begins to send the frame as data is written into the transmit buffer. Between the time the first data byte is written and the end of the message, the Reset Transmit Underrun/EOM command must be issued. Thus the Transmit Underrun/EOM status bit is in the reset state at the end of the message (when underrun occurs), which automatically sends the CRC characters. The sending of CRC again sets the Transmit/Underrun/EOM status bit.

Although there is no restriction as to when the Transmit Underrun/EOM bit can be reset within a message, it is usually reset after the first data character (secondary address) is sent to the Z80-SIO. Resetting this bit allows CRC and flags to be sent when there is no data to send which gives additional time to the CPU for recognizing the fault and responding with an abort command. By resetting it early in the message, the entire message has the maximum amount of CPU response time in an unintentional transmit underrun situation.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS CHAR 10 = Rx 6 BITS CHAR 01 = Rx 7 BITS CHAR 11 = Rx 8 BITS CHAR		AUTO ENABLES	ENTER HUNT MODE (IF INCOMING DATA NOT NEEDED)	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1	0 SELECTS SDLC MODE	0	0	0	0
WR5	DTR	00 = Tx 5 BITS (OR LESS) CHAR 10 = Tx 6 BITS CHAR 01 = Tx 7 BITS CHAR 11 = Tx 8 BITS CHAR		0	Tx ENABLE	0 SELECTS SDLC CRC	RTS	Tx CRC ENABLE

Table 7. Contents of Write Registers 3, 4 and 5 in SDLC Modes

Z80 CTC
Z80A-CTC

Technical Manual

1.0 INTRODUCTION

The Z80-Counter Timer Circuit (CTC) is a progammable component with four independent channels that provide counting and timing functions for microcomputer systems based on the Z80-CPU. The CPU can configure the CTC channels to operate under various modes and conditions as required to interface with a wide range of devices. In most applications, little or no external logic is required. The Z80-CTC utilizes N-channel silicon gate depletion load technology and is packaged in a 28-pin DIP. The Z80-CTC requires only a single 5 volt supply and a one-phase 5 volt clock. Major features of the Z80-CTC include:

- All inputs and outputs fully TTL compatible.
- Each channel may be selected to operate in either Counter Mode or Timer Mode.
- Used in either mode, a CPU-readable Down Counter indicates number of counts-to-go until zero.
- A Time Constant Register can automatically reload the Down Counter at Count Zero in Counter and Timer Mode.
- Selectable positive or negative trigger initiates time operation in Timer Mode. The same input is monitored for event counts in Counter Mode.
- Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
- Interrupts may be programmed to occur on the zero count condition in any channel.
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.

2.2 STRUCTURE OF CHANNEL LOGIC

The structure of one of the four sets of Counter/Timer Channel Logic is shown in figure 2.0-2. This logic is composed of 2 registers, 2 counters and control logic. The registers are an 8-bit Time Constant Register and an 8-bit Channel Control Register. The counters are an 8-bit CPU-readable Down Counter and an 8-bit Prescaler.

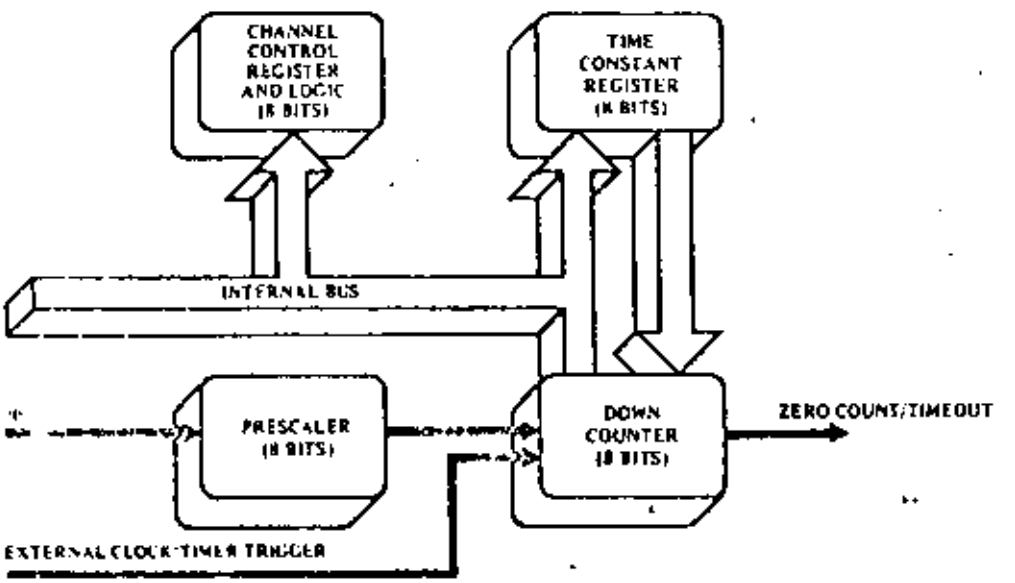


FIGURE 2.0-2
CHANNEL BLOCK DIAGRAM

2.2.1 THE CHANNEL CONTROL REGISTER AND LOGIC

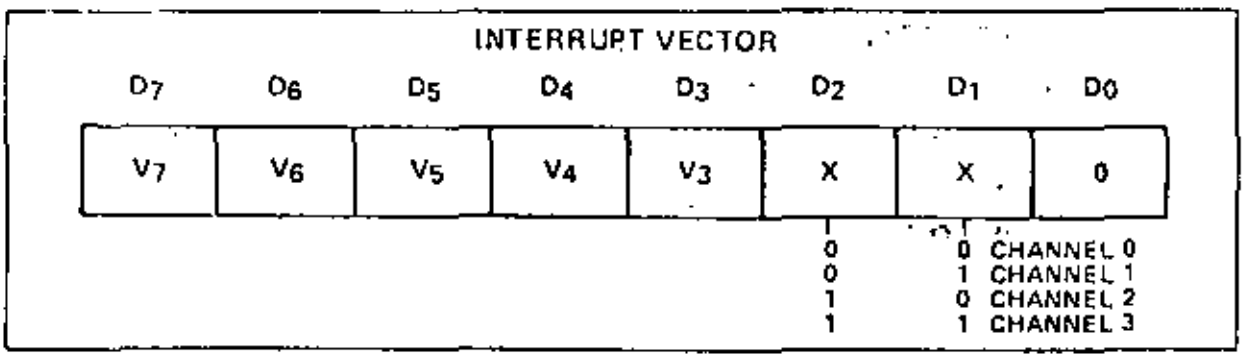
The Channel Control Register (8-bit) and Logic is written to by the CPU to select the modes and parameters of the channel. Within the entire CTC device there are four such registers, corresponding to the four Counter/Timer Channels. Which of the four is being written to depends on the encoding of two channel select input pins: CS0 and CS1 (usually attached to A0 and A1 of the CPU address bus). This is illustrated in the truth table below:

	CS1	CS0
Ch 0	0	0
Ch 1	0	1
Ch 2	1	0
Ch 3	1	1

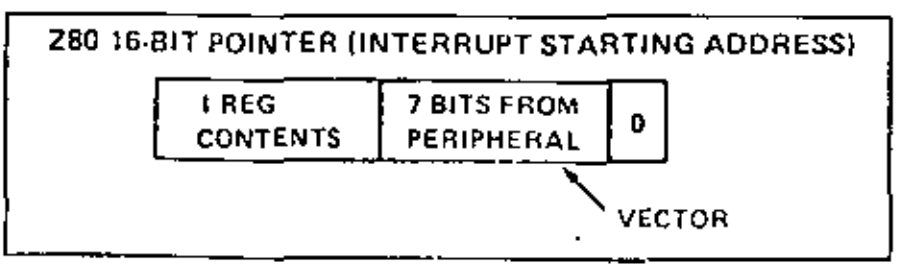
2.3 INTERRUPT CONTROL LOGIC

The Interrupt Control Logic insures that the CTC acts in accordance with Z80 system interrupt protocol for nested priority interrupting and return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in CTC devices to form this system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority down to channel 3 which has the lowest priority. The purpose of a CTC-generated interrupt, as with any other peripheral device, is to force the CPU to execute an interrupt service routine. According to Z80 system interrupt protocol, lower priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and have not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels.

A CTC channel may be programmed to request an interrupt every time its Down Counter reaches a zero count of zero. (To utilize this feature requires that the CPU be programmed for interrupt mode 2.) Some time after the interrupt request, the CPU will send out an interrupt acknowledge, and the CTC's Interrupt Control Logic will determine the highest-priority channel which is requesting an interrupt within the CTC device. Then if the CTC's IEI input is active, indicating that it has priority within the system daisy chain, it will place an 8-bit Interrupt Vector on the system data bus. The high-order 5 bits of this vector will have been written to the CTC earlier as part of the CTC initial programming process; the next two bits will be provided by the CTC's Interrupt Control Logic as a binary code corresponding to the highest-priority channel requesting an interrupt; finally the low-order bit of the vector will always be zero according to a convention described below.



This interrupt vector is used to form a pointer to a location in memory where the address of the interrupt service routine is stored in a table. The vector represents the least significant 8 bits, while the CPU reads the contents of the I register to provide the most significant 8-bits of the 16-bit pointer. The address in memory pointed to will contain the low-order byte, and the next highest address will contain the high-order byte of an address which in turn contains the first opcode of the interrupt service routine. Thus in mode 2, a single 8-bit vector stored in an interrupting CTC can result in an indirect call to any memory location.



There is a Z80 system convention that all addresses in the interrupt service routine table should have their low-order byte in an even location in memory, and their high-order byte in the next highest location in memory, which will always be odd so that the least significant bit of any interrupt vector will always be even. Hence the least significant bit of any interrupt vector will always be zero.

The RETI instruction is used at the end of any interrupt service routine to initialize the daisy chain enable line IEO for proper control of nested priority interrupt handling. The CTC monitors the system data bus and decodes this instruction when it occurs. Thus the CTC channel control logic will know when the CPU has completed servicing an interrupt, without any further communication with the CPU being necessary.

$\overline{RD}$

Read Cycle Status from the CPU (input, active low)

The $\overline{RD}$ signal is used in conjunction with the $\overline{IORQ}$ and $\overline{CE}$ signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, $\overline{IORQ}$ and $\overline{CE}$ must be true and $\overline{RD}$ false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid $\overline{RD}$ signal. In a CTC Read Cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ must be active to place the contents of the Down Counter on the Z80 Data Bus.

IEI

Interrupt Enable In (input, active high)

This signal is used to help form a system-wide interrupt daisy chain which establishes priorities when more than one peripheral device in the system has interrupting capability. A high level on this pin indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z80-CPU.

IEO

Interrupt Enable Out (output, active high)

The IEO signal, in conjunction with IEI, is used to form a system-wide interrupt priority daisy chain. IEO is high only if IEI is high and the CPU is not servicing an interrupt from any CTC channel. Thus this signal blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced by the CPU.

$\overline{INT}$

Interrupt Request (output, open drain, active low)

This signal goes true when any CTC channel which has been programmed to enable interrupts has a zero-count condition in its Down Counter.

$\overline{RESET}$

Reset (input, active low)

This signal stops all channels from counting and resets channel interrupt enable bits in all control registers, thereby disabling CTC-generated interrupts. The ZC/TO and $\overline{INT}$ outputs go to their inactive states, IEO reflects IEI, and the CTC's data bus output drivers go to the high impedance state.

CLK/TRG3 – CLK/TRG0

External Clock/Timer Trigger (input, user-selectable active high or low)

There are four CLK/TRG pins, corresponding to the four independent CTC channels. In the Counter Mode, every active edge on this pin decrements the Down Counter. In the Timer Mode, an active edge on this pin initiates the timing function. The user may select the active edge to be either rising or falling.

ZC/TO2 – AC/T00

Zero Count/Timeout (output, active high)

There are three ZC/TO pins, corresponding to CTC channels 2 through 0. (Due to package pin limitations channel 3 has no ZC/TO pin.) In either Counter Mode or Timer Mode, when the Down Counter decrements to zero an active high going pulse appears at this pin.

4.0 CTC OPERATING MODES

At power-up the Z80-CTC state is undefined. Asserting RESET puts the CTC in a known state. Before any channel can begin counting or timing, a Channel Control Word and a time constant data word must be written to the appropriate registers of that channel. Further, if any channel has been programmed to enable interrupts, an Interrupt Vector word must be written to the CTC's Interrupt Control Logic. (For further details, refer to section 5.0: "CTC Programming.") When the CPU has written all of these words to the CTC all active channels will be programmed for immediate operation in either the Counter Mode or the Timer Mode.

4.1 CTC COUNTER MODE

In this mode the CTC counts edges of the CLK/TRG input. The Counter Mode is programmed for a channel when its Channel Control Word is written with bit 6 set. The Channel's External Clock (CLK/TRG) input is monitored for a series of triggering edges; after each, in synchronization with the next rising edge of Φ (the System Clock), the Down Counter (which was initialized with the time constant data word at the start of any sequence of down-counting) is decremented. Although there is no set-up time requirement between the triggering edge of the External Clock and the rising edge of Φ, (Clock), the Down Counter will not be decremented until the following Φ pulse. (See the parameter ts(CK) in section 8.3: "A.C. Characteristics.") A channel's External Clock input is pre-programmed by bit 4 of the Channel Control Word to trigger the decrementing sequence with either a high or a low going edge.

In any of Channels 0, 1, or 2, when the Down Counter is successively decremented from the original time constant until finally it reaches zero, the Zero Count (ZC/TO) output pin for that channel will be pulsed active (high). (However, due to package pin limitations, channel 3 does not have this pin and so may only be used in applications where this output pulse is not required.) Further, if the channel has been so pre-programmed by bit 7 of the Channel Control Word, an interrupt request sequence will be generated. (For more details, see section 7.0: "CTC Interrupt Servicing.")

As the above sequence is proceeding, the zero count condition also results in the automatic reload of the Down Counter with the original time constant data word in the Time Constant Register. There is no interruption in the sequence of continued down-counting. If the Time Constant Register is written to with a new time constant data word while the Down Counter is decrementing, the present count will be completed before the new time constant will be loaded into the Down Counter.

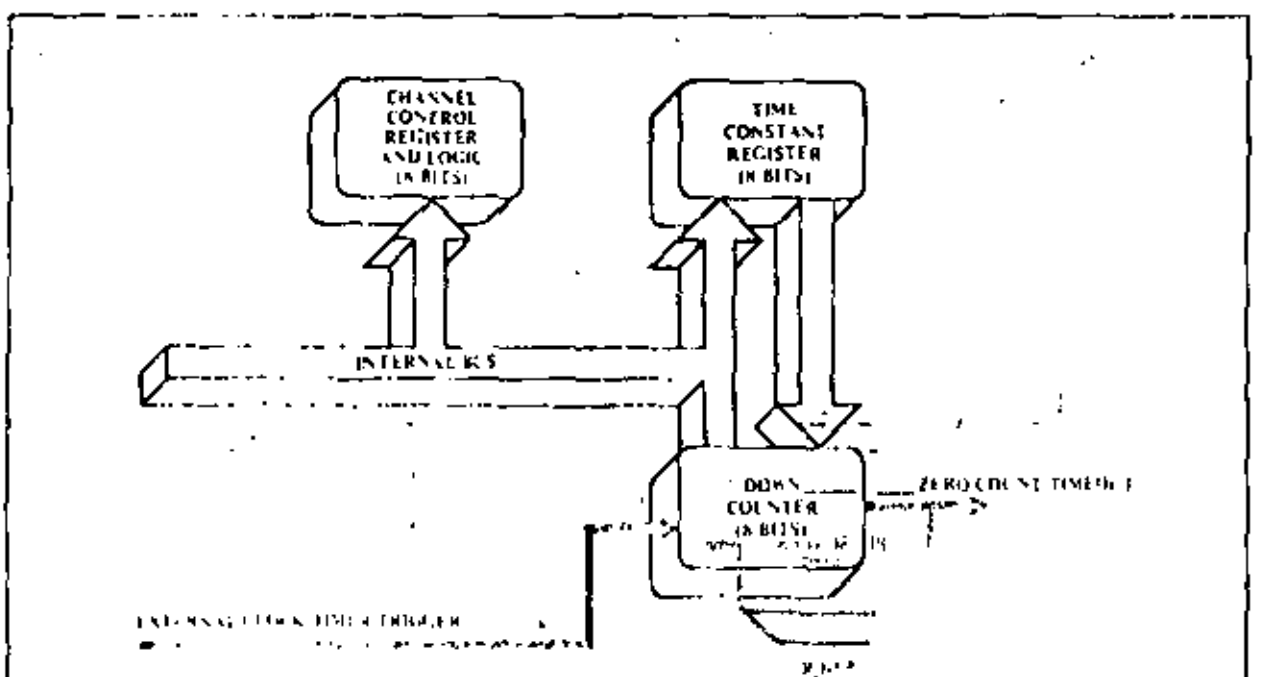


FIGURE 4.1-0
CHANNEL COUNTER MODE

5.0 CTC PROGRAMMING

Before any CTC channel can begin counting or timing operations, a Control Word and a Time Constant data word must be written to it by the CPU. These words will be stored in the Channel Control Register and the Time Constant Register of that channel. In addition, if any of the four channels have been programmed with bit 7 of their Channel Control Words to enable interrupts, an Interrupt Vector must be written to the appropriate register in the CTC. Due to automatic features in the Interrupt Control Logic, one pre-programmed Interrupt Vector suffices for all four channels.

5.1 LOADING THE CHANNEL CONTROL REGISTER

To load a Channel Control Word, the CPU performs a normal I/O Write sequence to the port address corresponding to the desired CTC channel. Two CTC input pins, namely CS0 and CS1, are used to form a 2-bit binary address to select one of four channels within the device. (For a truth table, see section 2.2.1: "The Channel Control Register and Logic".) In many system architectures, these two input pins are connected to the Address Bus lines A0 and A1, respectively, so that the four channels in a CTC device will occupy contiguous I/O port addresses. A word written to a CTC channel will be interpreted as a Channel Control Word, and loaded into the Channel Control Register, its bit 0 is a logic 1. The other seven bits of this word select operating modes and conditions as indicated in the diagram below. Following the diagram the meaning of each bit will be discussed in detail.

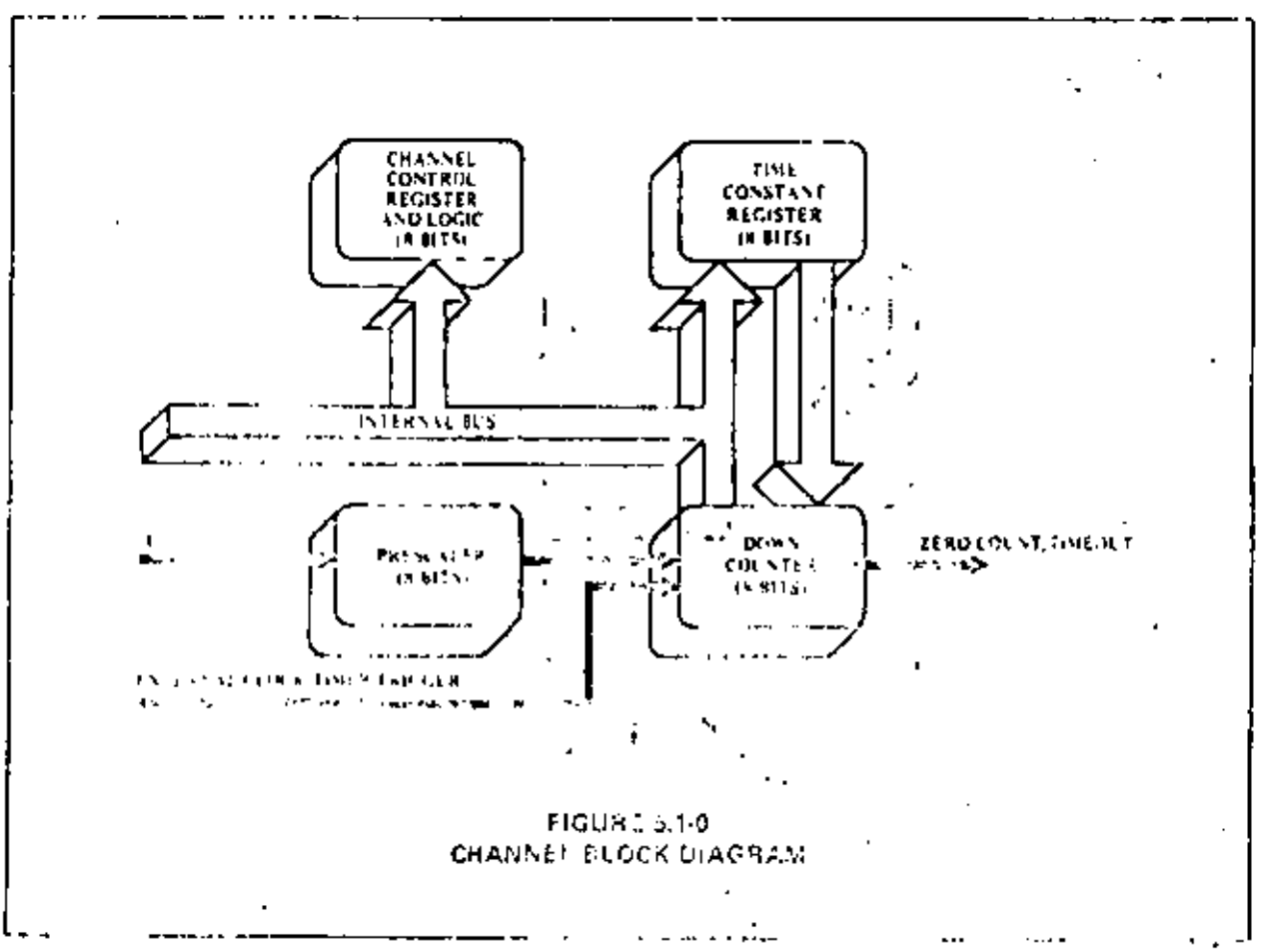
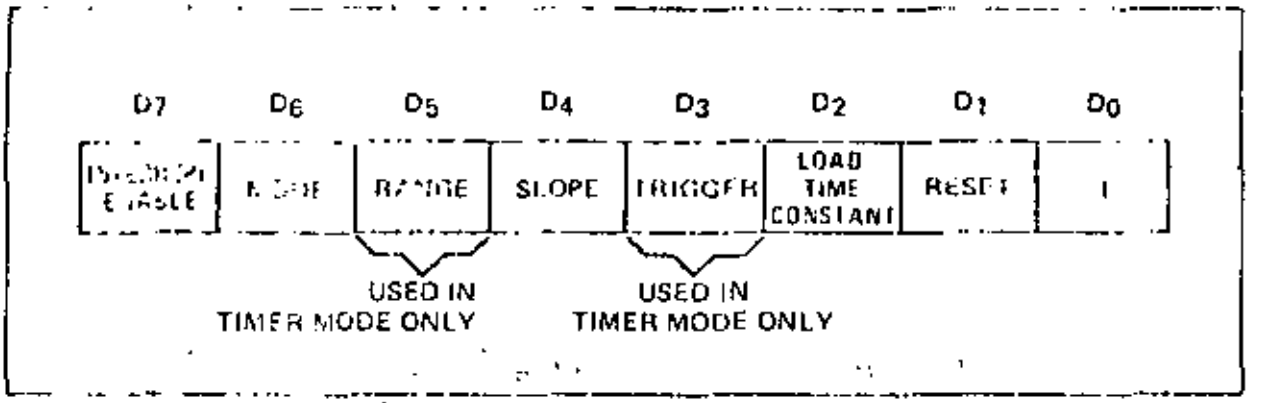


FIGURE 5.1-0
CHANNEL BLOCK DIAGRAM

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)



Bit 4 = 1

TIMER MODE -- positive edge trigger starts timer operation.

COUNTER MODE - positive edge decrements the down counter.

Bit 4 = 0

TIMER MODE - negative edge trigger starts timer operation.

COUNTER MODE - negative edge decrements the down counter.

Bit 3 = 1

Timer Mode Only – External trigger is valid for starting timer operation after rising edge of T_2 of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles.

Bit 3 = 0

Timer Mode Only – Timer begins operation on the rising edge of T_2 of the machine cycle following the one that loads the time constant.

Bit 2 = 1

The time constant data word for the Time Constant Register will be the next word written to this channel. If an updated Channel Control Word and time constant data word are written to a channel while it is already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded into it.

Bit 2 = 0

No time constant data word for the Time Constant Register should be expected to follow. To program bit 2 to this state implies that this Channel Control Word is intended to update the status of a channel already in operation since a channel will not operate without a correctly programmed data word in the Time Constant Register, and bit 2 in this Channel Control Word provides the only way of writing to the Time Constant Register.

Bit 1 = 1

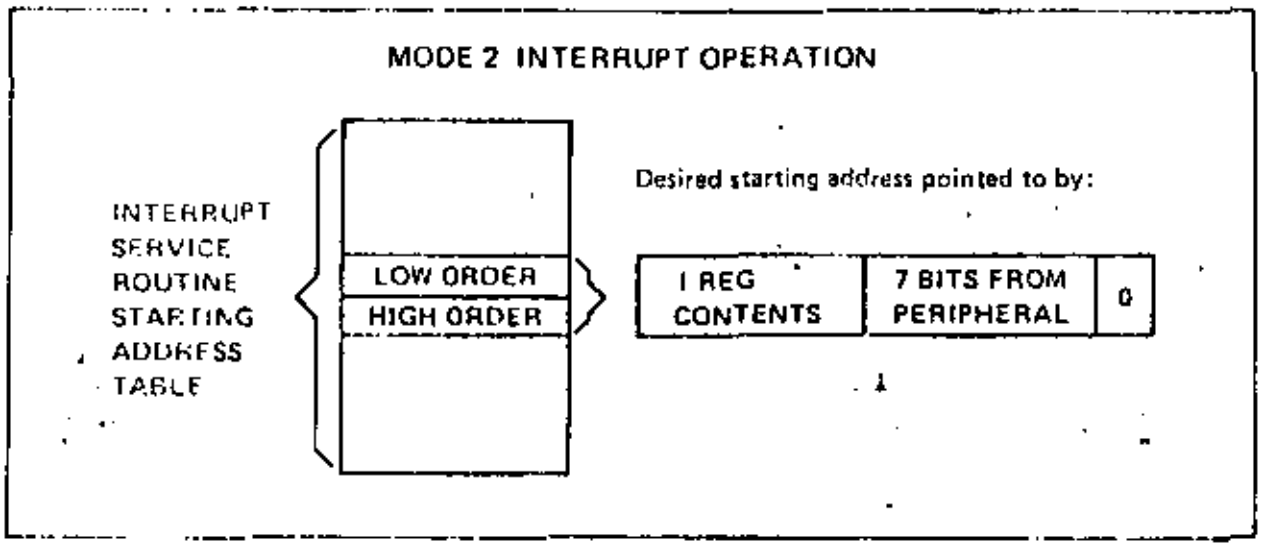
Resets channel operation after finishing. This is not a stored condition. Upon writing into this bit a reset is initiated to current channel operation, however, none of the bits in the channel control register are changed. If bit 0 = 1 and bit 1 = 1 the channel will resume operation upon loading a time constant.

Bit 1 = 0

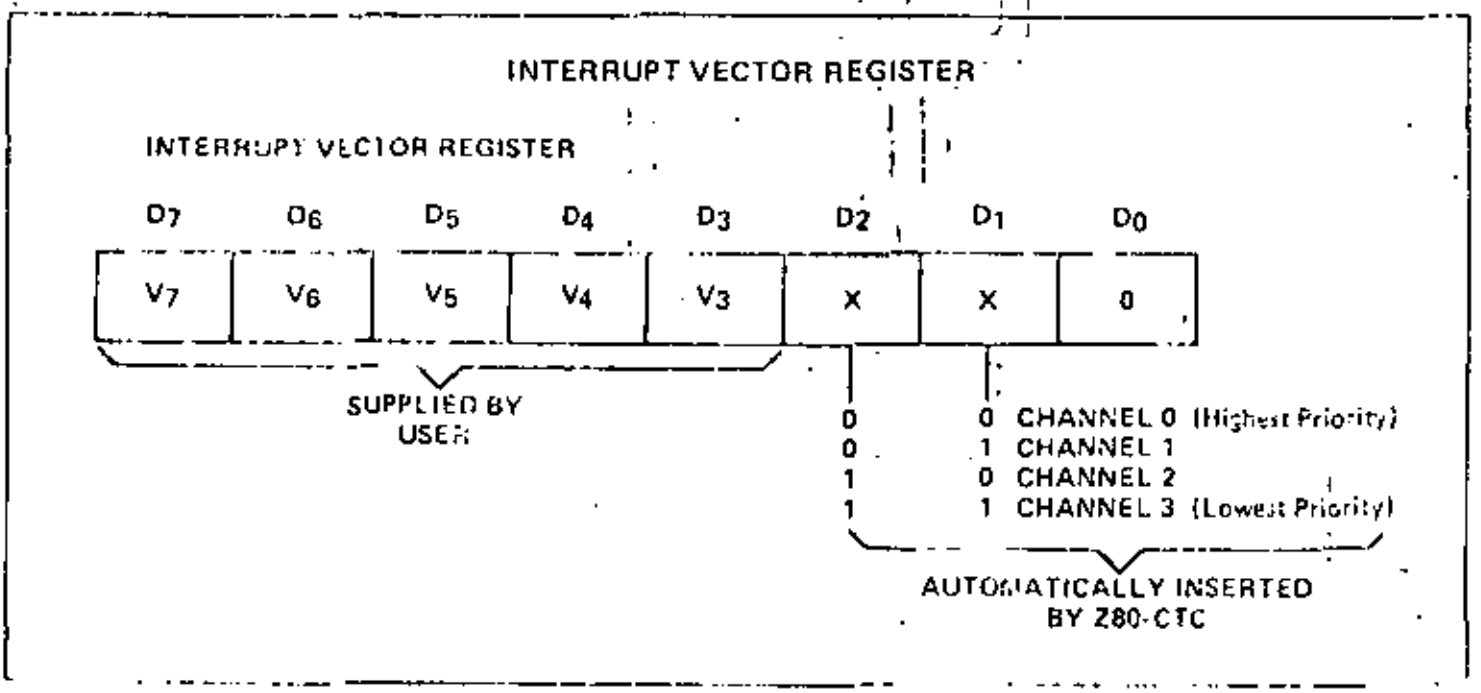
Channel continues current operation.

5.3 LOADING THE INTERRUPT VECTOR REGISTER

The Z80 CTC has been designed to operate with the Z80-CPU programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The upper 8 bits of this pointer are provided by the CPU's I register, and the lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, see section 7.0: "CTC Interrupt Servicing".)

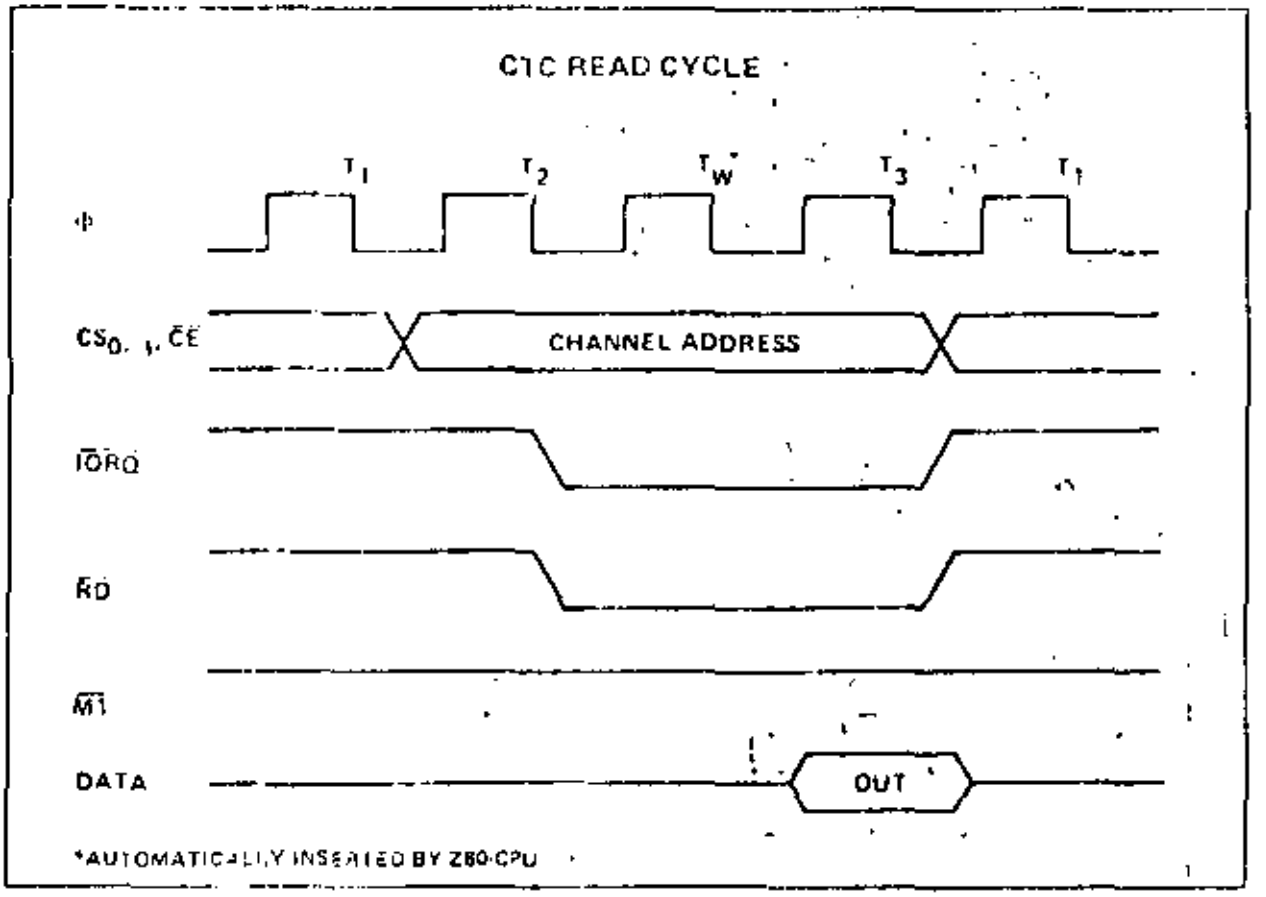


The high order 5 bits of this Interrupt Vector must be written to the CTC in advance as part of the initial programming sequence. To do so, the CPU must write to the I/O port address corresponding to the CTC channel 0, just as it would if a Channel Control Word were being written to that channel, except that bit 0 of the word being written must contain a 0. (As explained above in section 5.1, if bit 0 of a word written to a channel were set to 1, the word would be interpreted as a Channel Control Word, so a 0 in bit 0 signals the CTC to load the incoming word into the Interrupt Vector Register.) Bits 1 and 2, however, are not used when loading this vector. At the time when the interrupting channel must place the Interrupt Vector on the Z80 Data Bus, the Interrupt Control Logic of the CTC automatically supplies a binary code in bits 1 and 2 identifying which of the four CTC channels is to be serviced.



6.2 CTC READ CYCLE

Figure 6.0-2 illustrates the timing associated with the CTC Read Cycle. This sequence is used any time the CPU reads the current contents of the Down Counter. During clock cycle T_1, the Z80-CPU initiates the Read Cycle with true signals at input pins RD (Read), IORQ (I/O Request), and CE (Chip Enable). Also at this time a 2-bit binary code appears at CTC inputs CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being read from. (Note: M1 must be false to distinguish the cycle from an interrupt acknowledge.) On the rising edge of the cycle T_3 the valid contents of the Down Counter as of the rising edge of cycle T_2 will be available on the Z80 Data Bus. No additional wait states are allowed.



7.0 CTC INTERRUPT SERVICING

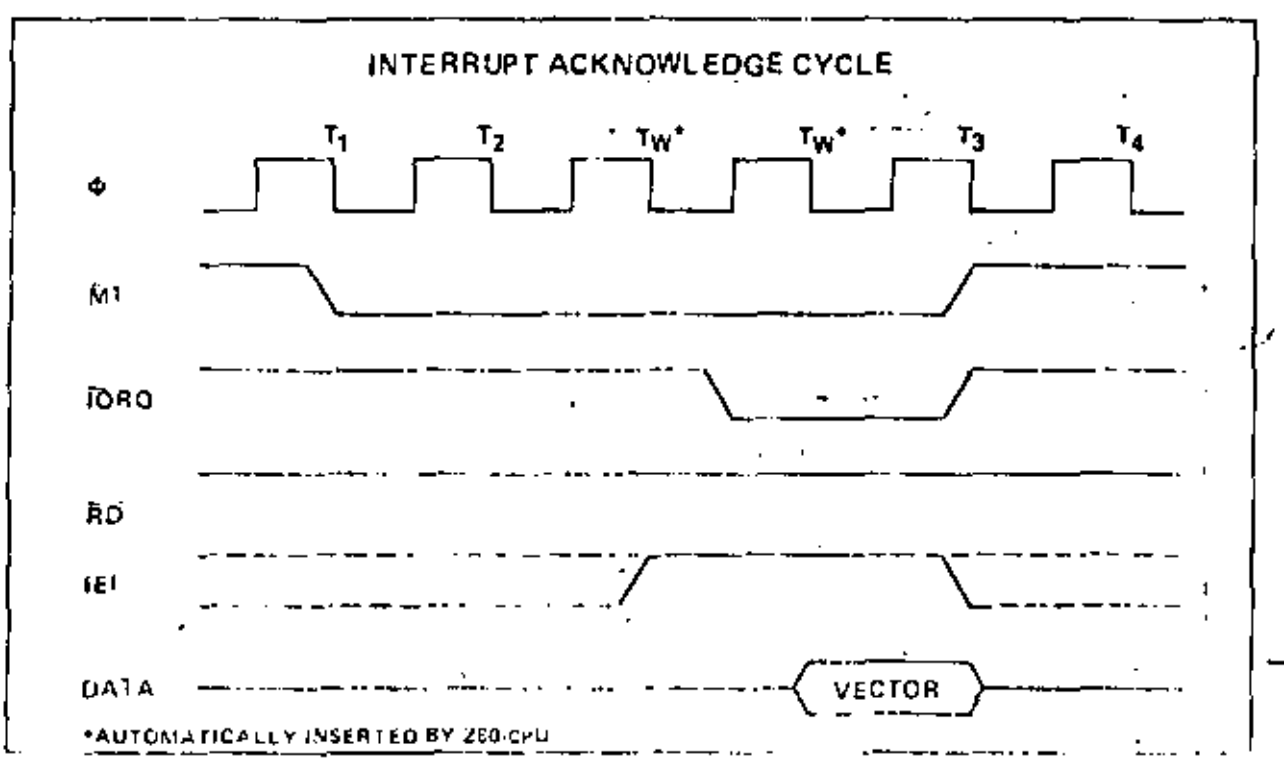
Each CTC channel may be individually programmed to request an interrupt every time its Down Counter reaches a count of zero. The purpose of a CTC-generated interrupt, as for any other peripheral device, is to force the CPU to execute an interrupt service routine. To utilize this feature the Z80-CPU must be programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, refer to chapter 8.0 of the Z80-CPU Technical Manual.)

The CTC's Interrupt Control Logic insures that it acts in accordance with Z80 system interrupt protocol for nested priority interrupt and proper return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in the CTC and all Z80 peripheral devices to form the system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority. According to Z80 system interrupt protocol, low priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels. (For further details, see section 2.3: "Interrupt Control Logic".)

Sections 7.1 and 7.2 below describe the nominal timing relationships of the relevant CTC pins for the Interrupt Acknowledge Cycle and the Return from Interrupt Cycle. Section 7.3 below discusses a typical example of daisy chain interrupt servicing.

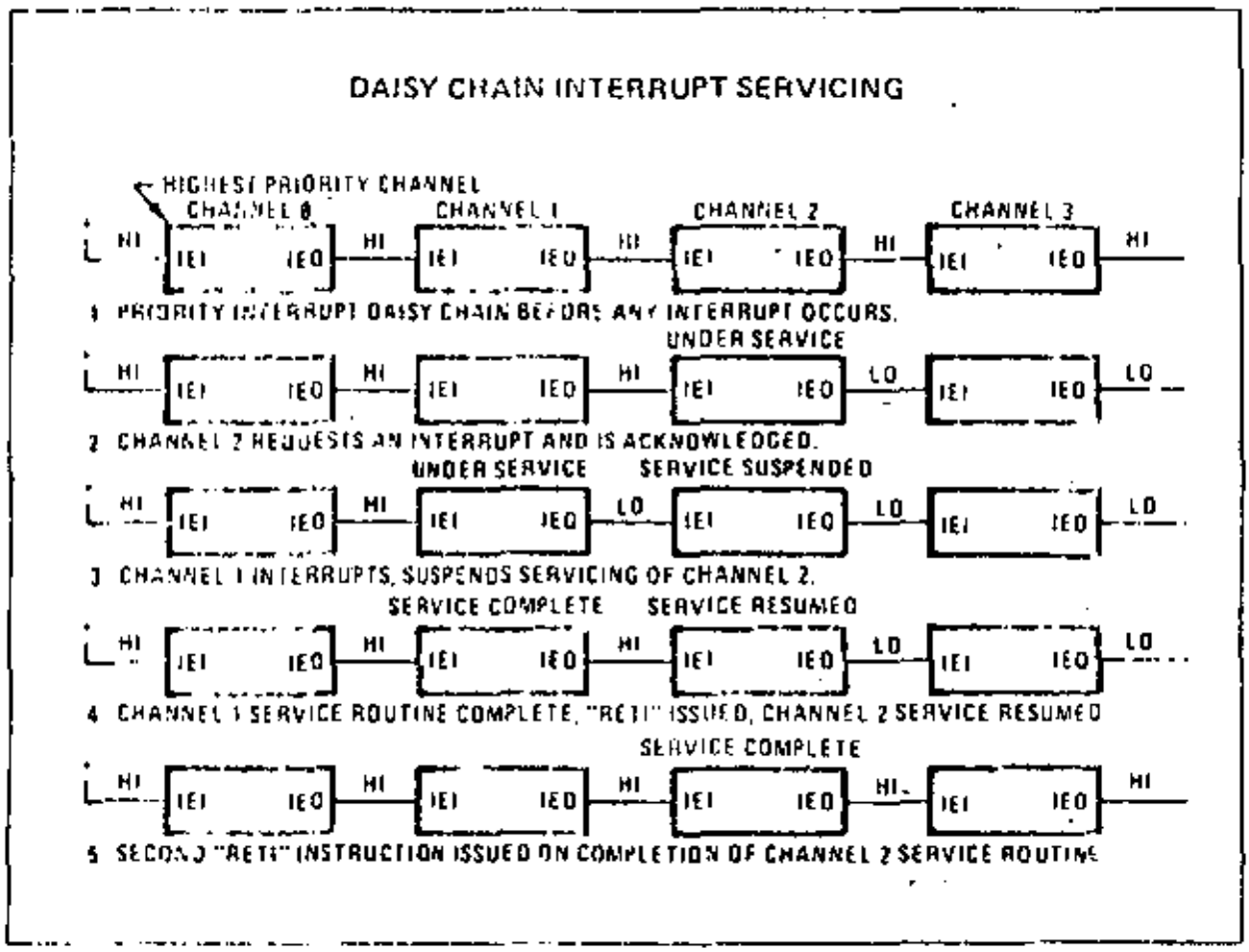
7.1 INTERRUPT ACKNOWLEDGE CYCLE

Figure 7.0-1 illustrates the timing associated with the Interrupt Acknowledge Cycle. Some time after an interrupt is requested by the CTC, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). To insure that the daisy chain enable lines stabilize, channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active. $\overline{M1}$ is active about two clock cycles earlier than $\overline{IORQ}$, and $\overline{RD}$ is false to distinguish the cycle from an instruction fetch. During this time the interrupt logic of the CTC will determine the highest priority channel requesting an interrupt. If the CTC Interrupt Enable Input (IEI) is active, then the highest priority interrupting channel within the CTC places its Interrupt Vector onto the Data Bus when $\overline{IORQ}$ goes active. Two wait states (T_W*) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.



7.3 DAISY CHAIN INTERRUPT SERVICING

Figure 7.0-3 illustrates a typical nested interrupt sequence which may occur in the CIC. In this example, channel 2 interrupts and is granted service. While this channel is being serviced, higher priority channel 1 interrupts and is granted service. The service routine for the higher priority channel is completed, and a RETI instruction (see sect. 7.2 for further details) is executed to signal the channel that its routine is complete. At this time, the service routine of the lower priority channel 2 is resumed and completed.



$T_A = 0^\circ C$ to $70^\circ C$, $V_{CC} = +5\ V \pm 5\%$, unless otherwise noted

Signal	Symbol	Parameter	Min	Max	Unit	Com
Φ	t_C	Clock Period	400	[1]	ns	
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	170	2000	ns	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	170	2000	ns	
	t_r, t_f	Clock Rise and Fall Times		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		ns	
CS, $\overline{CE}$, etc	$t_{S\Phi}(CS)$	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	160		ns	
D_0–D_7	$t_{DH}(D)$	Data Output Delay from Rising Edge of $\overline{RD}$ During Read Cycle		480	ns	[2]
	$t_{S\Phi}(D)$	Data Setup Time to Rising Edge of Φ During Write or M1 Cycle	60		ns	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of IORQ During INTA Cycle		340	ns	[2]
	$t_F(D)$	Delay to Floating Bus (Output Buffer Disable Time)		230	ns	
IEI	$t_S(IEI)$	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	200		ns	
IEO	$t_{DH}(IEO)$	IEO Delay Time from Rising Edge of IEI		220	ns	[3]
	$t_{DI}(IEO)$	IEO Delay Time from Falling Edge of IEI		190	ns	[3]
	$t_{DS}(IEO)$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		300	ns	[3]
$\overline{IORQ}$	$t_{S\Phi}(IR)$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	250		ns	
$\overline{M1}$	$t_{S\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or M1 Cycle	210		ns	
$\overline{RD}$	$t_{S\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or M1 Cycle	240		ns	
$\overline{INT}$	$t_{DCK}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(H) + 200$		Counter Mode
	$t_{D\Phi}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(H) + 200$		Timer Mode
CLK/TRG0–3	$t_C(CK)$	Clock Period	$2t_C(\Phi)$			Counter Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		50		
	$t_S(CK)$	Clock Setup Time to Rising Edge of Φ for Immediate Count	210			Counter Mode
	$t_S(TR)$	Trigger Setup Time to Rising Edge of Φ for Enabling of Prescaler on Following Rising Edge of Φ	210			Timer Mode
	$t_{W(CTH)}$	Clock and Trigger High Pulse Width	200			Counter and Timer Modes
	$t_{W(CTL)}$	Clock and Trigger Low Pulse Width	200			Counter and Timer Modes
ZC/TO0–2	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		190		Counter and Timer Modes
	$t_{DL}(ZC)$	ZC/TO Delay Time from Falling Edge of Φ, ZC/TO Low		190		Counter and Timer Modes

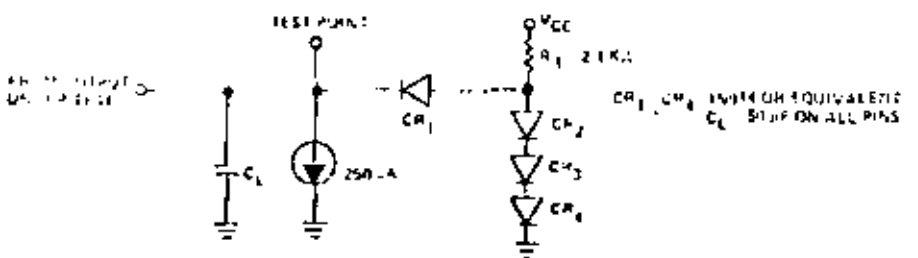
Notes: [1] $t_C < t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$.

[2] Increase delay by 10 nsec for each 50 pF increase in loading, 200 pF maximum for data lines and 100 pF for control lines.

[3] Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum.

[4] $\overline{RESET}$ must be active for a minimum of 3 clock cycles.

OUTPUT LOAD CIRCUIT



$T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 5\%$, unless otherwise noted

Signal	Symbol	Parameter	Min	Max	Unit	Comments
Φ	t_C	Clock Period	250	[1]	ns	
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	105	2000	ns	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	105	2000	ns	
	t_r, t_f	Clock Rise and Fall Time		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		ns	
CS, $\overline{CE}$, $R/\overline{W}$	$t_{S\Phi(CS)}$	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	60		ns	
D_0–D_7	$t_{DR(D)}$	Data Output Delay from Falling Edge of $\overline{RD}$ During Read Cycle		380	ns	[2]
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Φ During Write or M1 Cycle	50		ns	
	$t_{D(D)}$	Data Output Delay from Falling Edge of IORQ During INTA Cycle		160	ns	[2]
	t_F	Delay to Floating Bus (Output Buffer Disable Time)		110	ns	
IEI	$t_{S(IEI)}$	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	140		ns	
IEO	$t_{DH(IO)}$	IEO Delay Time from Rising Edge of IEI		160	ns	[3]
	$t_{DL(IO)}$	IEO Delay Time from Falling Edge of IEI		130	ns	[3]
	$t_{DM(IO)}$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		190	ns	[3]
$\overline{IORQ}$	$t_{S\Phi(IR)}$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	115		ns	
$\overline{M1}$	$t_{S\Phi(M1)}$	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or M1 Cycle	90		ns	
$\overline{RD}$	$t_{S\Phi(RD)}$	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or M1 Cycle	115		ns	
$\overline{INT}$	$t_{D(CLK)(IT)}$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(\Phi) + 140$		Counter Mode
	$t_{D\Phi(IT)}$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(\Phi) + 140$		Timer Mode
CLK/TRG_{0-3}	$t_{C(CK)}$	Clock Period	$2t_C(\Phi)$			Counter Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		30		
	$t_{S(CK)}$	Clock Setup Time to Rising Edge of Φ for Immediate Count	130			Counter Mode
	$t_{S(TR)}$	Trigger Setup Time to Rising Edge of Φ for enabling of Prescaler on Following Rising Edge of Φ	130			Timer Mode
	$t_{W(CTH)}$	Clock and Trigger High Pulse Width	120			Counter and Timer Modes
	$t_{W(CTL)}$	Clock and Trigger Low Pulse Width	120			Counter and Timer Modes
ZC/TO_{0-2}	$t_{DH(ZC)}$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		120		Counter and Timer Modes
	$t_{DL(ZC)}$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO Low		120		Counter and Timer Modes

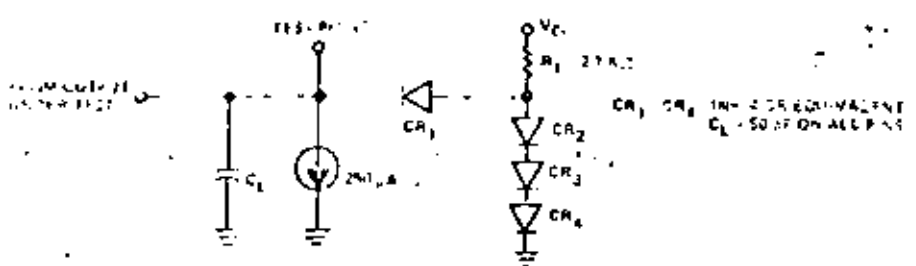
Notes: [1] $t_C = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

[2] Increase delay by 10 nsec for each 50 pF increase in loading, 200 pF maximum for data lines and 100 pF for control lines.

[3] Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum.

[4] $\overline{RESET}$ must be active for a minimum of 3 clock cycles.

OUTPUT LOAD CIRCUIT



INTRODUCCION A LOS MICROPROCESADORES (Z-80)

SISTEMAS EN UNA SOLA TABELTA (SBC)

Marzo, 1981

- Z - 80 MBC

- iSBC - 86/12

INTEGRATED COMPUTER SYSTEMS™

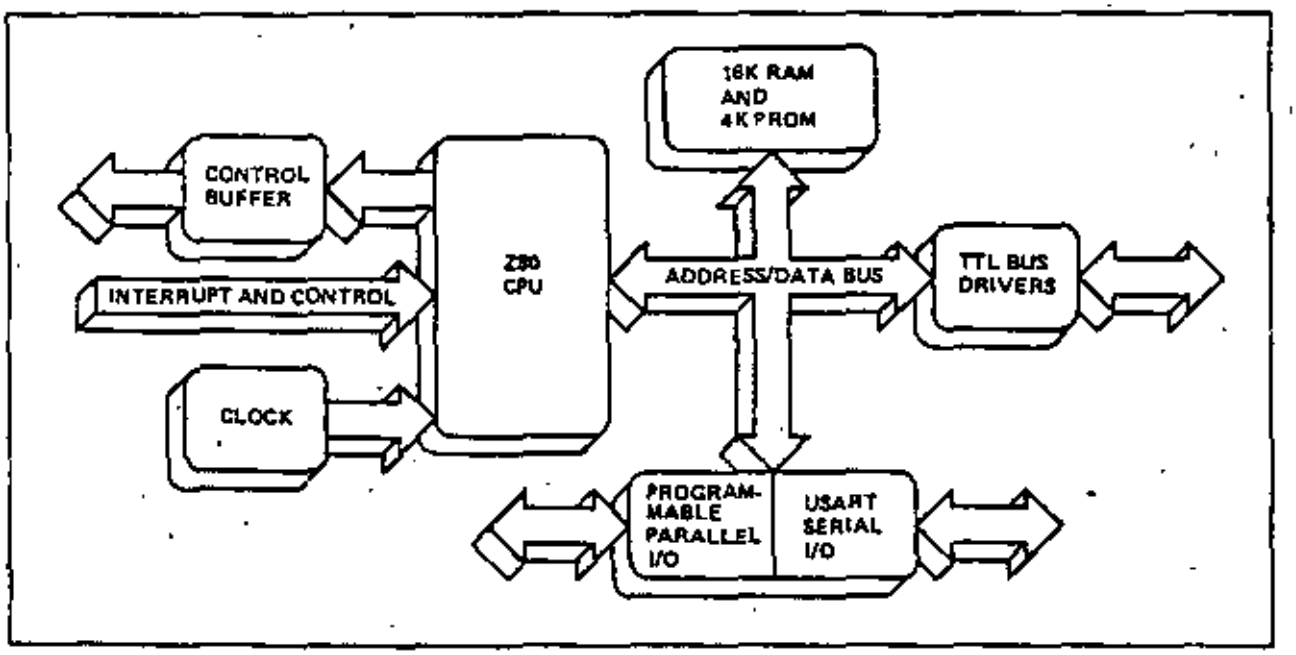
THIS PAGE INTENTIONALLY LEFT BLANK.

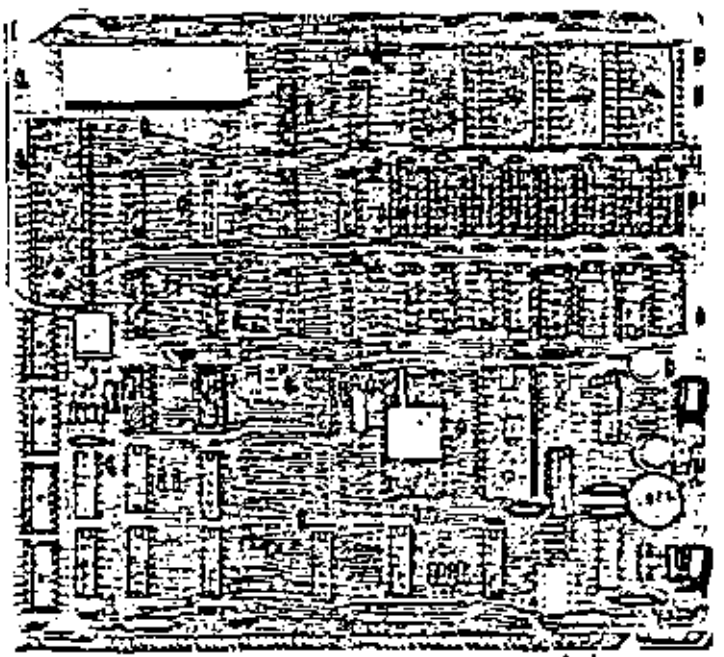
Z80®-MCB Microcomputer Board

Product Specification

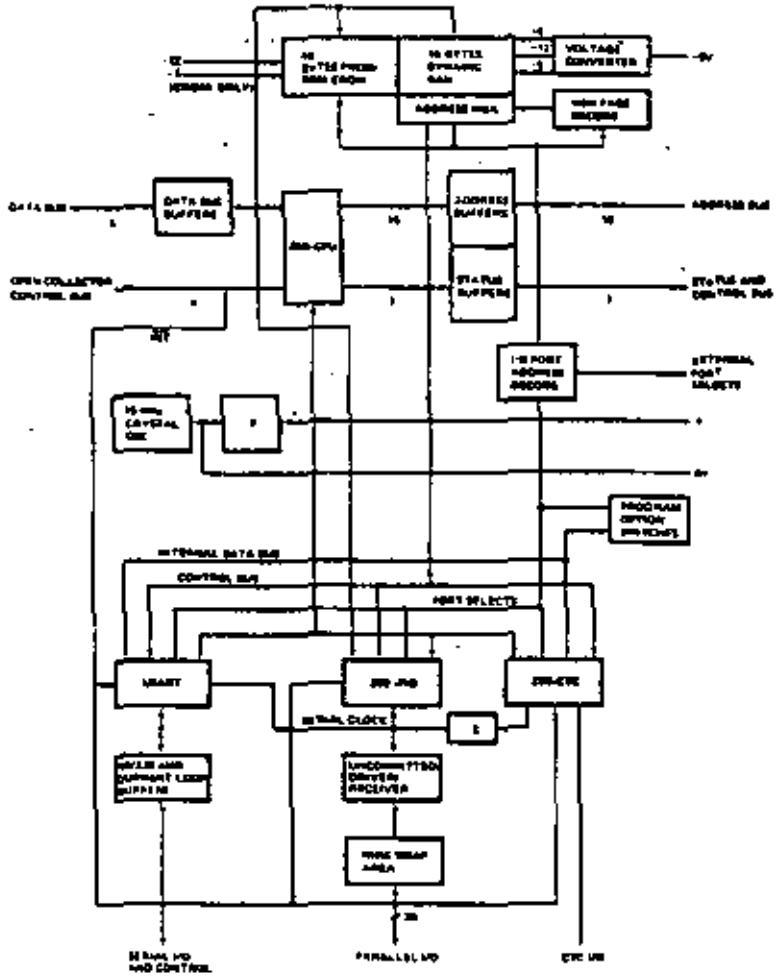
APRIL 1978

The Z80-MCB Microcomputer Board is a complete single board computer with its own self-contained memory plus serial and parallel I/O ports. It features the use of the Z80-CPU, Z80-CTC, Z80-PIO, and Z-6116 devices that have become standard components in the microcomputer industry.





Z80-MCB Block Diagram



Summary of Z80-CPU Instruction Set

ADD REGISTER/MEMORY/VALUE TO A/WITH CARRY
ADD REGISTER PAIR TO HL/IX/IY
ADD REGISTER PAIR TO HL WITH CARRY
SUBTRACT REGISTER/MEMORY/VALUE FROM A/ WITH BORROW
SUBTRACT REGISTER PAIR FROM HL WITH BORROW
DECIMAL ADJUST

AND REGISTER/MEMORY/VALUE WITH A
OR REGISTER/MEMORY/VALUE WITH A
EXCLUSIVE OR REGISTER/MEMORY/VALUE WITH A
COMPARE REGISTER/MEMORY/VALUE WITH A
COMPARE/INCREMENT/DECREMENT/HL, DECREMENT BC/REPEAT
INCREMENT/DECREMENT REGISTER/REGISTER PAIR MEMORY
COMPLEMENT A/2'S/1'S/

LOAD REGISTER/REGISTER PAIR/MEMORY WITH REGISTER/REGISTER PAIR/MEMORY/VALUE
LOAD (DE) WITH (HL), INC/DEC/HL AND DE, DEC BC/REPEAT

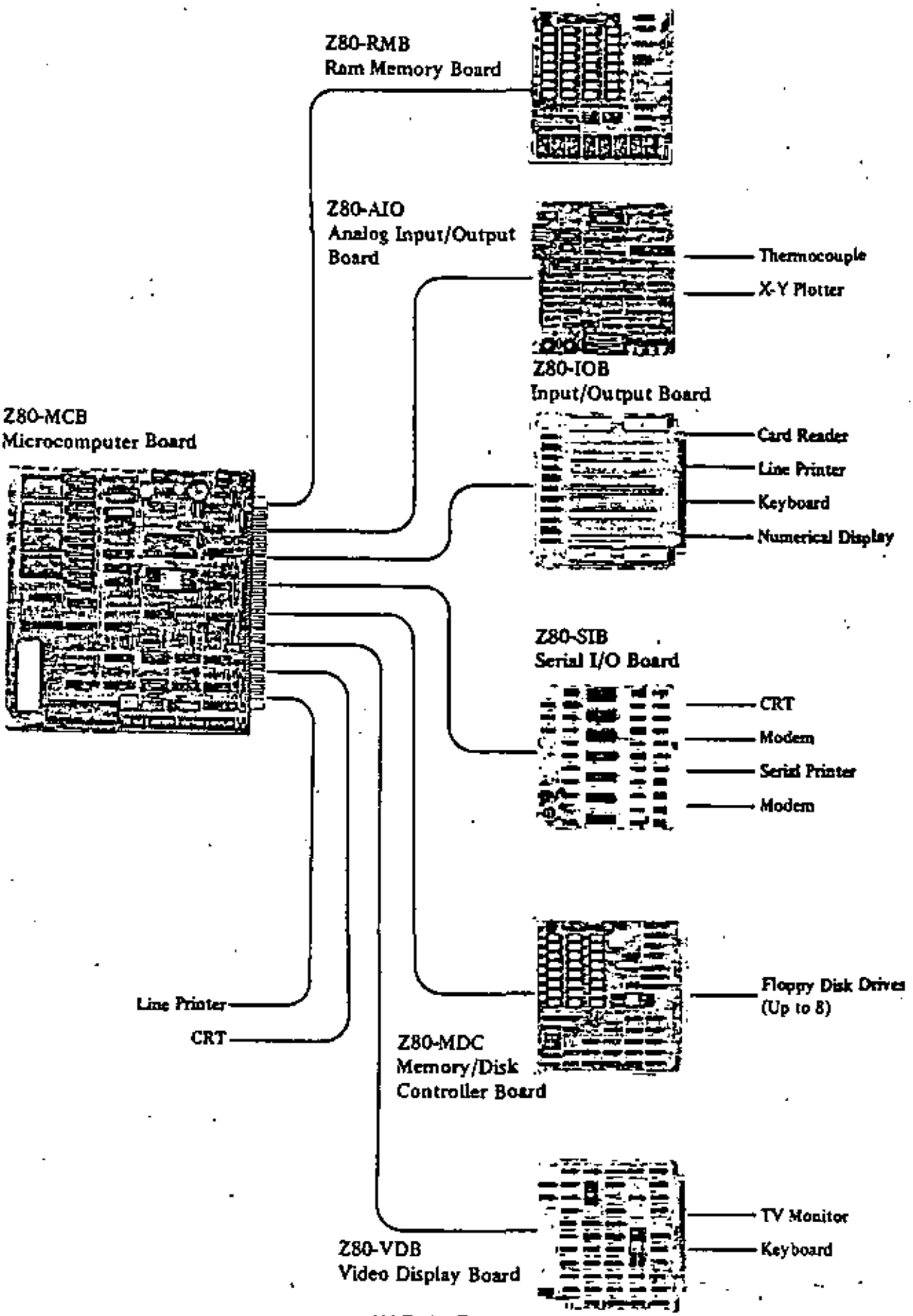
ROTATE RIGHT/LEFT, CIRCULAR/THRU CARRY/ REGISTER/MEMORY
ROTATE RIGHT/LEFT DIGIT
SHIFT REGISTER/MEMORY, LEFT ARITHMETIC
SHIFT REGISTER/MEMORY, RIGHT LOGICAL

SET/RESET/TEST BIT IN REGISTER/MEMORY

CALL SUBROUTINE IF CONDITION TRUE/UNCONDITIONAL CALL
JUMP IS CONDITION TRUE/UNCONDITIONAL JUMP
JUMP RELATIVE IF CONDITION TRUE/UNCONDITIONAL RELATIVE JUMP
JUMP TO (HL), (IX), OR (IY)
RESTART CALL
EXCHANGE REGISTERS
EXCHANGE REGISTER PAIR AND STACK
RETURN FROM SUBROUTINE IF CONDITION TRUE/ UNCONDITIONAL RETURN
POP/PUSH REGISTER PAIR FROM/TO STACK
SET INTERRUPT MODE 0/1/2
ENABLE/DISABLE INTERRUPTS
RETURN FROM INTERRUPT/NMI INTERRUPT
DECREMENT B, JUMP IF B = 0

INPUT/OUTPUT REGISTER
INPUT/OUTPUT, (HL), INC (DEC) HL, DEC B/REPEAT

SET/COMPLEMENT CARRY FLAG
NO OPERATION
HALT


Z80-RMB
Ram Memory Board
Z80-AIO
Analog Input/Output
Board
Thermocouple
X-Y Plotter
Z80-IOB
Input/Output Board
Card Reader
Line Printer
Keyboard
Numerical Display
Z80-MCB
Microcomputer Board
Z80-SIB
Serial I/O Board
CRT
Modem
Serial Printer
Modem
Line Printer
CRT
Z80-MDC
Memory/Disk
Controller Board
Floppy Disk Drives
(Up to 8)
Z80-VDB
Video Display Board
TV Monitor
Keyboard

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 86/12 is Intel's 8086, a powerful 16-bit H-MOS device. The 225 sq. mil chip contains 29,000 transistors and has a clock rate of 5MHz. The architecture includes four (4) 16-bit byte addressable data registers, two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

Instruction Set — The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. The instruction set of the 8086 is a superset of the 8080A/8085A family and with available software tools, programs written for the 8080A/8085A can be easily converted and run on the 8086 processor.

Architectural Features — A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.2μsec minimum instruction cycle to 400 nsec by having the instruction already in the queue. The stack oriented architecture facilitates nested subroutines and co-routines, reentrant code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K-bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The iSBC 86/12 has an internal bus for communicating with on-board memory and I/O options, a system bus (the MULTIBUS) for referencing additional memory and I/O options, and the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system

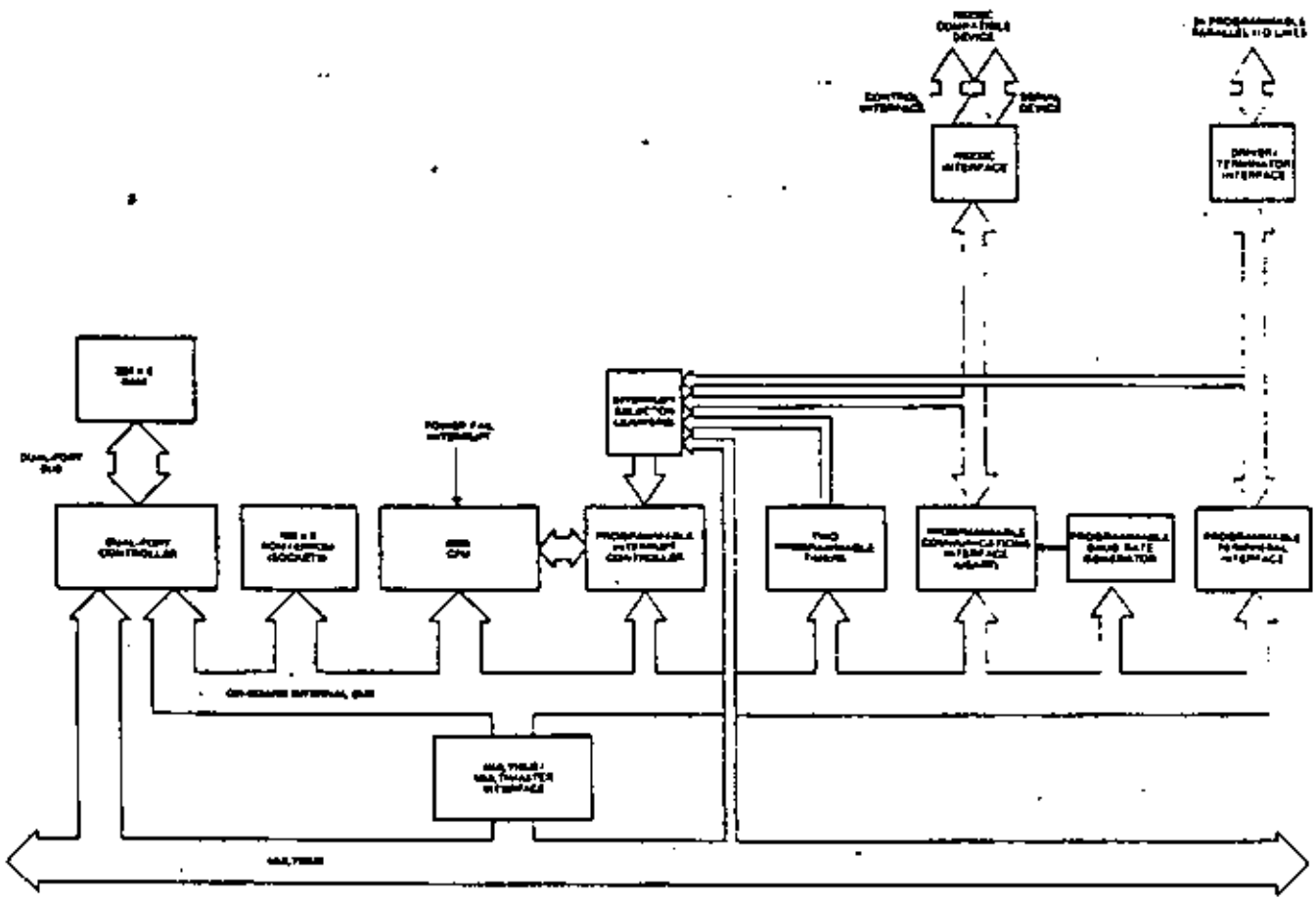


Figure 1. iSBC 86/12 Single Board Computer Block Design

direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26 pin edge connector that mates with RS232C compatible flat or round cable. The iSBC 530 Teletypewriter Adapter provides an optically isolated interface for those systems requiring a 20 mA current loop. The iSBC 530 may be used to interface the iSBC 86/12 to teletypewriters or other 20 mA current loop equipment.

Programmable Timers

The iSBC 86/12 provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O line drivers associated with the 8255A Programmable Peripheral Interface, or may be routed as inputs to the 8255A chip. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/12 RS232C USART serial port. In utilizing the iSBC 86/12, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given time delay or count is needed, software commands to the programmable timers/event counters select the desired function. Seven functions are available, as shown in Table 2. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents can be read "on the fly".

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occuring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

Table 2. Programmable Timer Functions

MULTIBUS and Multimaster Capabilities

The MULTIBUS features asynchronous data transfers for the accommodation of devices with various transfer rates while maintaining maximum throughput. Twenty address lines and sixteen separate data lines eliminate the need for address/data multiplexing/demultiplexing logic used in other systems, and allow for data transfer rates up to 5 megawords/sec. A failsafe timer is included in the iSBC 86/12 which can be used to generate an interrupt if an addressed device does not respond within 6 msec.

Multimaster Capabilities — The iSBC 86/12 is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 86/12 provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/12's or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers, to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters to share the MULTIBUS with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 86/12 or optionally provided directly from the MULTIBUS) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct memory access (DMA) operations, and high speed peripheral control, but are by no means limited to these three.

Interface and Execution Package — The iSBC 957 Interface and Execution Package allows the Intellec user to interface an iSBC 86/12 system to the development system. Included with the package are the necessary cables and software to allow transfer of files between the Intellec system and the iSBC 86/12. Additionally, the Intellec user can access a system monitor program (supplied on ROMs) resident in the iSBC 86/12 which allows access to programs loaded into the iSBC 86/12. The system monitor includes commands to examine and modify to memory, registers and I/O ports. Additionally, breakpoints, searches, and other useful operations are included to simplify software debug. Used in conjunction with the iSBC 957 Package, iSBC 86/12 execution packages are offered which include additional hardware such as the iSBC 660 system chassis to mount and power the iSBC 86/12 for program development.

PL/M-86 — Intel's high level programming language, PL/M-86, is also available as an Intellec microcomputer development system option. PL/M-86 provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M-86 programs can be written in a much shorter time than assembly language programs for a given application. PL/M-86 includes byte and word, integer, pointer and floating point (32-bit) data types and also includes conditional compilation and macro features.

SPECIFICATIONS

Word Size

Instruction — 8, 16, 24, or 32 bits
Data — 8, 16 bits

Cycle Time

Basic Instruction Cycle — 1½ μsec
— 400 nsec (assumes instruction in the queue)

Note:
Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles)

Memory Addressing

On Board ROM/EPROM — FF000–FFFFF$_H$ (using 2758 EPROM's); FE000–FFFFF$_H$ (using 2316E ROM's or 2716 EPROM's); and FC000–FFFFF$_H$ (using 2332 ROM's).

On-board RAM — 32k bytes of dual port RAM.CPU Access: 00000–07FFF$_H$. MULTIBUS Access is jumper selectable for any 8K-byte boundary, but not crossing a 128K byte boundary. Access for 8K, 16K, 24K, or 32K bytes may be selected for CPU use only.

Memory Capacity

On-Board Read Only Memory — 16K bytes (sockets only)

On-Board RAM — 32K bytes

Off-Board Expansion — Up to 1 megabyte in user specified combinations of RAM, ROM, and EPROM.

Note:
Read only memory may be added in 2K, 4K, or 8K-byte increments.

I/O Addressing

On-Board Programmable I/O

Port	8255A				USART	
	1	2	3	Control	Data	Control
Address	C8	CA	CC	CE	D8 or DC	DA or DE

I/O Capacity

Parallel — 24 programmable lines using one 8255A.
Serial — 1 programmable line using one 8251A.

Serial Communications Characteristics

Synchronous — 5—8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous — 5—8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Baud Rates

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)		
	Synchronous	Asynchronous	
		÷ 16	÷ 64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
2.4	2400	150	—
1.76	1760	110	—

Note:
Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Interrupts

Addresses for 8259A Registers (Hex notation I/O address space)

C0 or C4 — Write: Initialization Command Word 1 (ICW1) and Operation Control Words 2 and 3 (OCW2 and OCW3)
Read: Status and Poll Registers

C2 or C6 — Write: 1CW2, 1CW3, 1CW4, 0CW1 (Mask Register)
Read: 0CW1 (Mask Register)

Note:
Several registers have the same physical address: sequence of access and one data bit of control word determine which register will respond.

Interrupt Levels — 8086 CPU includes a non-maskable interrupt (NMI) and a maskable interrupt (INTR). NMI interrupt is provided for catastrophic events such as power failure. NMI vector address is 00008. INTR inter-

Electrical Characteristics

DC Power Requirements

Configuration	Current Requirements			
	V_{CC} = +5V ±5% (max)	V_{DD} = +12V ±5% (max)	V_{BB} = −5V ±5% (max)	V_{AA} = −12V ±5% (max)
Without EPROM[1]	5.2A	350 mA	—	40 mA
RAM Only[2]	390 mA	40 mA	1.0 mA	—
With iSBC 530[4]	5.2A	450 mA	—	140 mA
With 4K EPROM[5] (using 2758)	5.8A	450 mA	—	140 mA
With 8K ROM[5] (using 2316E)	6.1A	450 mA	—	140 mA
With 8K EPROM[5] (using 2716)	5.8A	450 mA	—	140 mA
With 16K ROM[5] (using 2332)	5.4A	450 mA	—	140 mA

Notes:

1. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.
2. Does not include power required for optional ROM/EPROM, I/O drivers and I/O terminators.
3. RAM chips powered via auxiliary power bus.
4. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators. Power for iSBC 530 is supplied via serial port connector.
5. Includes power required for four ROM/EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

Environmental Characteristics

Operating Temperature — 0°C to 55°C
Relative Humidity — to 90% (without condensation)

Reference Manual

9800645A — iSBC 86/12 Single Board Computer Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 86/12	Single Board Computer with 32K bytes RAM

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) [illegible] 987-8080
TWX: 910-338-0026
TELEX: 34-6372

U.S. REGIONAL SALES OFFICES

WESTERN

CALIFORNIA
Intel Corp.*
1651 East 4th Street
Suite 150
Santa Ana 92701
Tel: (714) 835-9642
TWX: 910-595-1114

MID-WESTERN

ILLINOIS
Intel Corp.*
900 Jorie Boulevard
Suite 220
Oakbrook 60521
Tel: (312) 325-9510
TWX: 910-651-5881

EASTERN

OHIO
Intel Corp.*
8312 North Main Street
Dayton 45415
Tel: (513) 890-5350
TELEX: 288-004

ATLANTIC

MASSACHUSETTS
Intel Corp.*
187 Billerica Road, Suite 14A
Chelmsford 01824
Tel: (617) 256-6567
TWX: 710-343-6333

OVERSEAS MARKETING OFFICES

ORIENT

JAPAN
Intel Japan Corporation*
Flower Hill-Shinmachi East Bldg.
1-23-9, Shinmachi, Setagaya-ku
Tokyo 154
Tel: (03) 426-9261
TELEX: 781-28428

EUROPE

BELGIUM
Intel International*
Rue du Moulin à Papier
51-Boite 1
B-1160 Brussels
Tel: (02) 660 30 10
TELEX: 24814

**Note New Telephone Number

*Field Application Location

MSE D-16

Printed in U.S.A./SP2/0778/40K/SG/BL

intel

MODEL 240
INTELLEC® SERIES II
MICROCOMPUTER DEVELOPMENT SYSTEM

Complete microcomputer development center for Intel MCS-80™, MCS-85™, MCS-86™, and MCS-48™ microprocessor families

64K bytes RAM memory

Self-test diagnostic capability

Built-in interfaces for high speed paper tape reader/punch, printer, and universal PROM programmer

Integral CRT with detachable upper/lower case typewriter-style full ASCII keyboard

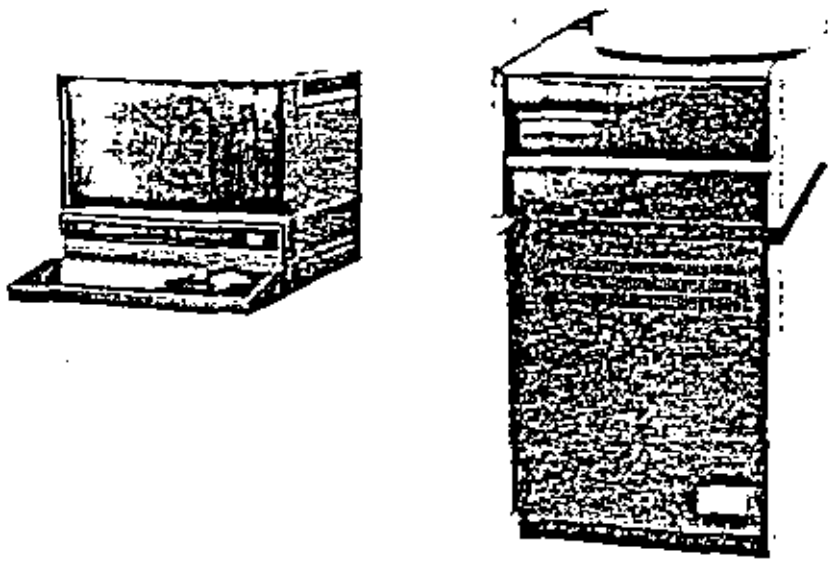
Integral 250K-byte floppy disk plus 7.3 million bytes (expandable to 15.6M bytes) of hard disk storage

Powerful ISIS-II Diskette Operating System software with relocating macroassembler, linker, and locater

Supports PL/M, FORTRAN, BASIC and COBOL high level languages

Software compatible with previous Intellec systems

The Model 240 Intellec Series II Microcomputer Development System is a complete center for the development of microcomputer-based products. It includes a CPU, 64K bytes of RAM, 4K bytes of ROM memory, a 2000-character CRT, a detachable full ASCII keyboard, and 250K-byte floppy diskette drive. A hard disk subsystem provides over 7 million bytes of on-line data storage. Powerful ISIS-II Diskette Operating System software allows the Model 240 to be used quickly and efficiently for assembling and/or compiling and debugging programs for Intel's MCS-80, MCS-85, MCS-86, or MCS-48 microprocessor families without the need for handling paper tape. ISIS-II performs all file handling operations, leaving the user free to concentrate on the details of his own application. When used in conjunction with an optional in-circuit emulator (ICE™) module, the Model 240 provides all the hardware and software development tools necessary for the rapid development of a microcomputer-based product.



MSE E-3

the CRT, keyboard, and standard Intellec peripherals including printer, high speed paper tape reader/punch, and universal PROM programmer. The IOC contains its own independent microprocessor, also an 8080A-2. The CPU controls all I/O operations as well as supervising communications with the IPB. 8K bytes of ROM contain all I/O control firmware. 8K bytes of RAM are used for CRT screen refresh storage. These do not occupy space in Intellec Series II main memory since the IOC is a totally independent microcomputer subsystem.

Integral CRT

Display — The CRT is a 12-inch raster scan type monitor with a 50/60 Hz vertical scan rate and 15.5 kHz horizontal scan rate. Controls are provided for brightness and contrast adjustments. The interface to the CRT is provided through an Intel 8275 single-chip programmable CRT controller. The master processor on the IPB transfers a character for display to the IOC, where it is stored in RAM. The CRT controller reads a line at a time into its line buffer through an Intel 8257 DMA controller and then feeds one character at a time to the character generator to produce the video signal. Timing for the CRT control is provided by an Intel 8253 interval timer. The screen display is formatted as 25 rows of 80 characters. The full set of ASCII characters is displayed, including lower case alphas.

Keyboard — The keyboard interfaces directly to the IOC processor via an 8-bit data bus. The keyboard contains an Intel UPI-41 Universal Peripheral Interface, which scans the keyboard, encodes the characters, and buffers the characters to provide N-key rollover. The keyboard itself is a high quality typewriter style keyboard containing the full ASCII character set. An upper/lower case switch allows the system to be used for document preparation. Cursor control keys are also provided.

Peripheral Interface

A UPI-41 Universal Peripheral Interface on the IOC board provides interface for other standard Intellec peripherals including a printer, high speed paper tape reader, high speed paper tape punch, and universal PROM programmer. Communication between the IPB and IOC is maintained over a separate 8-bit bidirectional data bus. Connectors for the four devices named above, as well as the two serial channels, are mounted directly on the IOC itself.

Control

User control is maintained through a front panel, consisting of a power switch and indicator, reset/boot switch, run/halt light, and eight interrupt switches and indicators. The front panel circuit board is attached directly to the IPB, allowing the eight interrupt switches to connect to the primary 8259, as well as to the Intellec Series II bus.

Disk System

The Intellec Series II Hard Disk System provides direct access bulk storage, intelligent controller, and a disk drive containing one fixed platter and one removable cartridge. Each provides 3.5 million bytes of storage with a data transfer rate of 2.5 Mbits/second. The controller is implemented with Intel's powerful Series 3000 Bipolar Microcomputer Set. The controller provides an interface to the Intellec Series II system bus, as well as supporting up to 2 disk drives. The disk system records all data in Double Frequency (FM) on 2 surfaces per platter. Each platter can be write protected by a front panel switch. The disk system is capable of performing six different operations: recalibrate, seek, format track, write data, read data, and verify CRC.

Disk Controller Boards — The disk controller consists of two boards, the channel board and the interface board. These two PC boards reside in the Intellec Series II system chassis and constitute the disk controller. The channel board receives, decodes and responds to channel commands from the 8080A-2 CPU in the Model 240. The interface board provides the disk controller with a means of communication with the disk drives and with the Intellec system bus. The interface board validates data during reads using a cyclic redundancy check (CRC) polynomial and generates CRC data during write operations. When the disk controller requires access to Intellec system memory, the channel board requests and maintains DMA master control of the system bus, and generates the appropriate memory command. The channel board also acknowledges I/O commands as required by the Intellec bus. In addition to supporting a second drive, the disk controller may coreside with the Intel double-density controller to allow up to 17 million bytes of on-line storage.

Floppy Disk Drive

The floppy disk drive is controlled by an Intel 8271 single chip, programmable floppy disk controller. It transfers data via an Intel 8257 DMA controller between an IOC RAM buffer and the diskette. The 8271 handles reading and writing of data, formatting diskettes, and reading status, all upon appropriate commands from the IOC microprocessor.

MULTIBUS™ Capability

All Intellec Series II models implement the industry standard MULTIBUS. MULTIBUS enables several bus masters, such as CPU and DMA devices, to share the bus and memory by operating at different priority levels. Resolution of bus exchanges is synchronized by a bus clock signal derived independently from processor clocks. Read/write transfers may take place at rates up to 5 MHz. The bus structure is suitable for use with any Intel microcomputer family.

ORDERING INFORMATION

23

Part Number	Description
MDS-240*	Intellec Series II Model 240 Microcomputer Development System (110V/60 Hz)
MDS-241*	Intellec Series II Model 240 Microcomputer Development System (220V/50 Hz)

*"MDS" is an ordering code only, and is not used as a product name or trademark. MDS® is a registered trademark of Monawk Data Sciences Corp.

24

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080

Printed in U.S.A./HS2/0879/5K/DA BL

The 8001 Microprocessor Lab is a total hardware debugging system for the design of microprocessor-based products. A key feature is its ability to support many microprocessor chips, including the Intel 8080A, 8085A, Motorola 6800, Texas Instruments TMS9900, 3870/72, F8 and Zilog Z-80A. In addition to multiple microprocessor support, the 8001 offers three emulation modes for software debugging, partial and full emulation, as well as a real time prototype analyzer option with all the capabilities of a microprocessor analyzer.

Three Emulation Modes

In a typical design sequence, software is first developed independently using time-sharing, a minicomputer, or some other means. It is then downloaded to the 8001. At this point the in-prototype emulation and software/hardware integration capabilities of the 8001 come into play.

In emulation mode 0, the software runs only on the emulator processor. This enables the program to be debugged on a microprocessor virtually identical to the one that will ultimately be used in the completed product. In emulation modes 1 and 2, the prototype control probe is connected to the emulator processor at one end and plugged into the empty microprocessor socket in the prototype circuitry at the other.

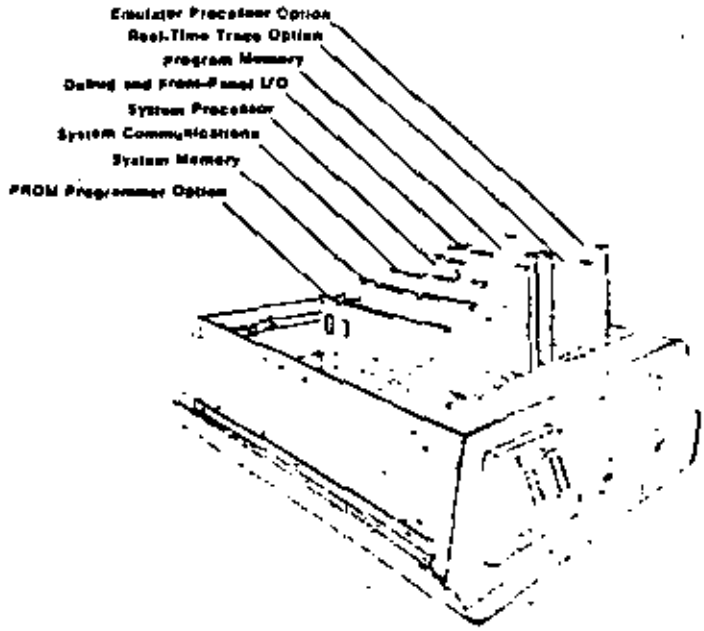
Partial emulation (mode 1) lets the user release control in methodical steps from the 8001 to the prototype. The developmental software runs using 8001 memory space and prototype I/O and clock. The 8001 memory mapping feature allows memory to be gradually mapped over to the prototype by address blocks. Throughout partial emulation, the user has access to prototype circuitry via the powerful 8001 debugging system, which enables him to trace, set breakpoints, examine and change memory and register contents.

Full emulation (mode 2) lets the user exercise the program on the now stand-alone prototype while still maintaining complete control through the Microprocessor Lab. All I/O and timing functions are directed by the prototype; all memory has been mapped over to the prototype; and only the prototype control probe is still in place, emulating the target microprocessor. Although the prototype is effectively freestanding, then, the user may still direct program activity, at the prototype end of the probe, from the 8001.

Hardware Trace Option

The optional Real-Time Prototype Analyzer enables the user to dynamically monitor the prototype address bus, data bus, and up to eight other locations on the prototype circuit board. Prototype activity is monitored at full speed, without stopping or slowing up the microprocessor in mode 2. This enables the designer to locate critical timing problems and hardware/software sequence problems. Full speed emulation is also possible in mode 1 with most of the emulators. Refer to the specific emulator data sheet for exact performance specifications.

In summary, then, each of the three emulation modes supports a specific phase of the product development cycle. Beginning with assembled source language, the designer proceeds from software debugging (mode 0), to the sequential integration of program and circuit (mode 1), to the final integration and test of the stand-alone product (mode 2). The Real-Time Prototype Analyzer enhances modes 1 and 2 by allowing the user to monitor and access prototype activity.



8001 Characteristics

The 8001 Microprocessor Lab is a modular system whose mainframe houses up to 20 plug-in circuit boards. The system includes System Processor, Debug and Front-Panel I/O, Program Memory, System Communications, and System Memory modules. Emulator Processor module for the selected microprocessor is optional, its associated prototype control probe, and a ROM-based software module is included. Additional emulator processor packages are available as options for each microprocessor the system supports.

The Real-Time Prototype Analyzer module, additional 16K byte Program Memory modules, and EPROM Programmer modules for the 1702A or 2704/2708 are available as options. Optional peripherals include: the TEKTRONIX CT8100 Crt Terminal, CT8101 TTY Terminal, 4024/4025 Computer Display Terminal, and the LP8200 Line Printer.

System Processor Module

The System Processor communicates directly with the system console and functions as the control for the 8001. It performs input/output functions to all system peripherals through its own I/O port and ports located on the System Communications module; and it directs all other modules, such as the Debug module and the Emulator Processor through the system bus. All power requirements are supplied from the system power supply.

Debug and Front-Panel I/O Module

The Debug and Front-Panel I/O module performs three functions: 1) It controls the interaction and bus time-sharing of the System Processor and Emulator Processor modules. 2) It supports all software debug features such as break, trace, and emulator program start at any location. 3) It provides interface to the front panel.

System Memory Module

System Memory contains the operating firmware for the 8001. The module consists of 6K bytes of main program memory ROM; 2K bytes of RAM; and space for additional ROM. System Memory can be accessed only by System Processor.

SPECIFICATIONS

8001 PHYSICAL CHARACTERISTICS

Dimensions	cm	in
Height	24.86	9.8
Width	46.31	18.8
Length	57.3	22.3
	kg	lb
Weight	20	44

8001 ENVIRONMENTAL CHARACTERISTICS

Temperature	
Operating	0°C to +35°C (+32°F to 95°F)
Humidity	To 80% relative noncondensing.
Altitude	
Operating	To 15,000 feet max.
Storage	To 50,000 feet max.

8001 ELECTRICAL CHARACTERISTICS

AC Input Voltages	115 VAC ±10% or 230 VAC ±10%
Frequency Range	60 Hz (50 Hz special order).
Outputs	
Dual Supply VDC	−12 VDC/+12 VDC ±5% at 3.4 A.
Single Voltage	+5.2 VDC ±5% at 25 A
Single Current	25 A
Line Regulation	
Dual Supply	±0.05% for a 10% line change.
Single Supply	±0.01% for a 10% line change.
Load Regulation	
Dual Supply	±0.05% for a 50% load change.
Single Supply	±0.05% for a 50% load change.
Output Ripple	
Dual Supply	1.5 mV (p-p) 0.4 mV (rms)
Single Supply	1.5 mV (p-p) 0.4 mV (rms)
Transient Response	
Dual Supply	30 µs for 50% load change.
Single Supply	30 µs for 50% load change.
Overload Protection	Automatic current limit foldback.
Adjustments	
Dual Supply	For −12 VDC to −15 VDC] Factory For +12 VDC to +15 VDC] Set
Single Supply	None.

Fuses	Amps	at	Volts AC
Primary	6		115
	3		230
60 VAC Second.	0.6		
60 VAC Second.	0.3		
±12 VDC	2		115
	1		230

ORDERING INFORMATION

8001 Microprocessor Lab $4810

Option	Description	Factory Price	Field Number	Field Price
Option 01	8080A Microprocessor Support Package*	3690	8001F01	3740
Option 02	6800 Microprocessor Support Package*	3690	8001F02	3740
Option 03	Z80A Microprocessor Support Package*	3690	8001F03	3740
Option 04	TMS9900 Microprocessor Support Package*	4120	8001F04	4180
Option 05	8085A Microprocessor Support Package*	3690	8001F05	3740
Option 06	3870/72 Microcomputer Support Package	4190	8001F06	4240
Option 07	F8 Microprocessor Support Package	4190	8001F07	4240
Option 45	32K Program Memory Modules	3100		
Option 46	Real-Time Prototype Analyzer	2500	8001F46	2700
Option 47	1702 PROM Programmer	650	8001F47	700
Option 48	2704/2708 PROM Programmer	650	8001F48	700
Option 49	16K Program Memory Modules	1550	8001F49	1710

*A Microprocessor Support Package consists of an emulator PROM, an emulator processor board, and a prototype control probe. One support package is selected from the factory options with purchase of an 8001. Additional support packages may be selected from the field options.

Option Peripherals

CT8100 Crt Terminal $3495
CT8101 TTY Terminal $1395
LP8200 Line Printer $3765
4024 Computer Display Terminal with Option 20* $3245
4025 Computer Display Terminal with Option 20* $3845

*Option 20 (8K bytes Display Memory) is required for proper 8001 operation.

STANDARD ACCESSORIES

8001 Installation Guide 070-2717-00
8001 Systems Users Manual 070-2464-00
8001 System Reference Card 070-2471-01

OPTIONAL ACCESSORIES

8001 Service Manual*
RS232 Interconnector Cable 10 ft (300 cm) 013-0757-00
Null-Modem Interconnecting Cable 5 ft (150 cm) 013-0820-00

*Available January, 1979.

The TEKTRONIX Z80A Emulator Processor and Prototype Control Probe aid in the development of prototype systems built around the Zilog Z80 and Z80A microprocessors.* Operable with either the 8002A or 8001 Microprocessor Lab, these options provide the architectural capabilities necessary to emulate the Z80A in a prototype system.

Three progressive modes of emulation are available to support the designer during prototype development. Emulation is controlled at all times by the Microprocessor Lab's powerful debugging system software.

Emulation mode 0 is used strictly for software development. The designer executes the test program on the Z80A emulator processor, which is identical to the processor that will be used in the completed system. The designer is able to set breakpoints; examine and modify memory contents; and trace through the program to see the contents of the registers and the values of the status word, workspace pointer, and program counter.

In this emulation mode the Z80A and its mobile processor card can be located either in the emulator processor module or the prototype control probe.

Emulation mode 1 is used to begin software/hardware integration. The test program executes on the Z80A emulator processor, and all I/O functions are performed by prototype hardware. The program itself may execute both from 8002A/8001 program memory and from prototype memory with the Microprocessor Lab memory mapping capability.

In this mode the mobile microprocessor card is installed in the prototype control probe interface assembly, where it can now be used in any of the three emulation modes; the probe's 40 pin plug is inserted in the prototype's microprocessor socket. This configuration moves the emulator processor closer to the prototype, thus alleviating delay and propagation problems that might occur at the Z80A's 4 MHZ maximum operating speed.

Emulation mode 2 is used to complete system integration. The test program still executes on the emulator processor, but all programmed functions and I/O functions are now performed by the prototype via the control probe under the control of the Microprocessor Lab.

*Z80 and Z80A are trademarks of Zilog for a particular type of microprocessor. Tektronix, Inc., does not guarantee that other vendors' versions of the Z80 will be compatible with the TEKTRONIX Microprocessor Labs.

The Real-Time Prototype Analyzer, available as a Microprocessor Lab option, functions in all emulation modes. Operating as a microprocessor analyzer, this option enables the designer to monitor 43 channels of data simultaneously, including the prototype address bus, data bus, control signals such as R/W, M/IO, and Fetch, and up to 8 other locations acquired via the option's general-purpose logic probe.

Z80A EMULATOR SUPPORT PACKAGE CHARACTERISTICS

PHYSICAL CHARACTERISTICS

Length	5 ft. of cable from the emulator processor to the interface assembly. 1 ft. of cable from the interface assembly to the 40 pin plug.
Cable Configuration	
5 ft.	2 40 conductor ribbon cables with chassis ground plane and signal paths.
1 ft.	2 40 conductor twisted pair cables.
Termination	
5 ft.	The interface assembly contains resistors for data, address, and control lines from the Z80A emulator processor module.
1 ft.	Not terminated.

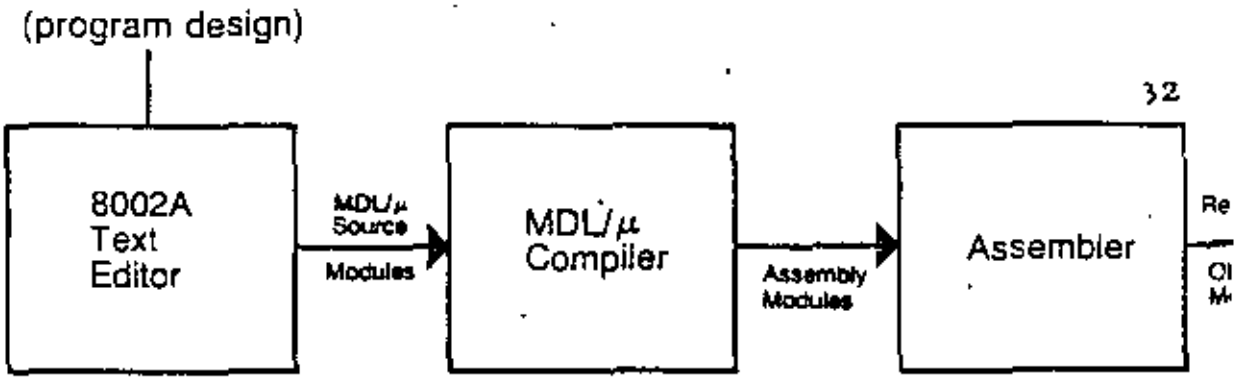
TIMING CHARACTERISTICS

The Z80A emulator processor was designed to match the AC characteristics of the Z80A microprocessor with two exceptions:

Prototype Clock	The prototype clock may not be stretched over a total of 10 μs during any one memory or I/O request when a Microprocessor Lab memory access may occur in the next cycle. This exception is valid only if the prototype clock runs at speeds of 1 MHZ.
NMI	NMI (Non-Maskable Interrupt) must occur one-half cycle earlier than in a standard Z80A configuration. This means the NMI must occur before the next to last trailing edge of the M cycle just prior to M1.

ORDERING INFORMATION

Option Description	Factory Price	Field Number	Field Price
8001 Microprocessor Lab			
Option 03 Z80A Microprocessor Support Package		8001P03	
Option 44 Real-Time Prototype Analyzer		8001P44	
8002A Microprocessor Lab			
Option 03 Z80A Assembler Software Support		8002P03	
Option 19 Z80A Emulator Support		8002P19	
Option 33 Z80A Prototype Control Probe		8002P33	
Option 45 32K Program Memory Module			
Option 44 Real-Time Prototype Analyzer		8002P44	
Option 48 16K Memory Program Module*		8002P48	



8002A Modular Development

MDL/μ Applications

1. MULTI-PROGRAMMER SOFTWARE PROJECTS
   Modular software development that allows local and global reference control between MDL/μ or Assembler Modules.
2. MICROPROCESSOR SYSTEMS PROGRAMMING
   High level control of memory and I/O with all run time code re-entrant for interrupt driven systems.
3. ROM BASED PRODUCTS
   All run time code is ROM-able with stack and variable sections identified and grouped for ease in Link Time Location.
4. LARGE APPLICATION PROGRAMS
   Reliable, easy to maintain code.
5. HIGH VOLUME PRODUCTS
   Efficient code generation for minimum memory overhead.
6. HIGHLY COMPETITIVE PRODUCTS
   Shortened development time, low maintenance costs, and ease of new feature addition.

FEATURES OF THE MODULAR DEVELOPMENT LANGUAGE

1) A true compiler that generates executable (non interpretative) code:
   - —produces linkable, relocatable, and ROM-able code.
   - —compiler output (assembly code) may be edited, if desired, for crucial optimizations or code modifications.
   - —different sections of the output may be specified for ROM or RAM.
   - —run-time support routines:
     a) provide simple use of 8002A I/O facilities;
     b) are linked with program only if referenced by program;
     c) reduce size of object code (there's only one instance of that code sequence);
     d) provide code which is locally optimized for efficient use of space;
     e) provide re-entrant sub-routines;
     f) allow object code to be linked with 8080A/8085A/Z80 or 6800 assembly language modules.
2) Compiler, assembler, and linker form a modular programming facility:
   - —programs may be written in modular components (MDL/μ and assembly).
   - —modules may have any number of functions and procedures.
   - —procedures, functions, and variables defined in one module can be used by another.
   - —module interfaces are completely described.
   - —modules are separately compiled, then linked together.
   - —incremental development is possible with module as "increment".
3) User-defined, multi-statement functions and procedures.
4) High Level Language constructs for the 8002A and Microprocessor I/O:
   - —direct control of I/O ports and memory (accessed as variables).
   - —MDL/μ constructs for vectored interrupts and mask instructions.
5) Data handling capabilities:
   - —Integer capability for one byte (0 to 255) and two byte (−32,768 to +32,767) variables with arithmetic, logical relational and bit-manipulation operations — no real numbers or floating point.
   - —String capability with 0 to 255 byte length variables with concatenation and substring operations.
   - —One and two dimensional arrays of all data types.
6) Variable, procedure, function and modular identifiers may have one to six alphanumeric characters.

BENEFITS

Use of MDL/μ can:

- —reduce the design time for your application
- —improve the documentation of your programs
- —increase the reliability of your software
- —simplify and reduce the cost of program maintenance

MDL/μ KEYWORDS

LET	Assigns a value to a variable
REM	Allows program documentation
ENABLE	Enables interrupts
DISABLE	Disables interrupts
*MASK**	Sets the interrupt mask state
DIM	Specifies size and or dimensions and reserves space for string, array, integer variables
DEFFN	Begins a user-defined function
DEFPR	Begins a user-defined procedure
FNEND	Ends a user-defined function
PREND	Ends a user-defined procedure
MODULE	Specifies a module name, starting address and emulation mode
MAIN	Indicates program starting address is contained in this module
USES	Identifies externally defined functions, procedures, variables ports, and interrupts available for internal use
PROVIDES	Identifies internally defined functions, procedures and variables available for external use
FUNCTION	Specifies a function that is used or provided by a module
PROC	Specifies a procedure that is used or provided by a module
VAR	Specifies a variable that is used or provided by a module
PORT†	Defines an I/O port that is used by a module
ORIGIN	Specifies a memory location for a port or variable
MODE	Indicates the emulation mode in which the module runs
GO TO	Transfers control to a specified line
ON..GO TO	Transfers control to a selected line
GOSUB	Transfers control to a specified line that begins a subroutine
ON..GOSUB	Transfers control to a selected line that begins a subroutine

* 8085A. † 8080A 8085A Z80A

MINIMUM HARDWARE REQUIREMENTS

Tektronix 8002A Microprocessor Lab with system terminal and 64K Program Memory.
Emulator for 8080A, 8085A, Z80, or 6800 is required to debug and execute object code

SOFTWARE PREREQUISITE

8080A, 8085A, Z80, or 6800 TEKDOS Version 3.0 software support is required.

Z80 GENERATED CODE CHARACTERISTICS

—Generated code does not use any of the Z80 extension capabilities over 8080A instruction sets.
—Generated code does not use any index or alternate registers of the Z80
—Generated code does not use alternate interrupt modes.

SOFTWARE SUPPORT SERVICES

During the ninety (90) day period following installation of this Software Product (SOFTWARE), if the customer encounters a problem with the SOFTWARE which his diagnosis indicates is caused by a defect in the SOFTWARE, the customer may submit a Software Performance Report (SPR) to Tektronix. Tektronix will respond to problems reported in SPRs which are caused by defects in the current unaltered release of the SOFTWARE via the Maintenance Periodical for this SOFTWARE, which reports SPRs received, code corrections, temporary corrections, generally useful emergency bypasses and/or notices of the availability of corrected code. Software updates, if any, released by Tektronix during the ninety (90) day period, will be provided to the customer on Tektronix standard distribution media as specified in this Software Data Sheet. Charges will be based on the prevailing update price.

ORDERING INFORMATION

	Factory Price	Field Number	Field Price
8002A Microprocessor Lab			
Option Description			
Option 01 8080A Assembler Software Support		8002F01	
Option 1A MDL/8080A/8085A Software Support* (Requires 64K Program Memory & Option 01 or 05)		8002F1A	
Option 02 6800 Assembler Software Support		8002F02	
Option 2A MDL/6800 Software Support (Requires 64K Program Memory & Option 02)			
Option 03 Z80 Assembler Software Support		8002F03	
Option 3A MDL/8080A/Z80 Software Support** (Requires 64K Program Memory & Option 03)		8002F3A	
Option 05 8085A Assembler Software Support		8002F05	
Option 1A MDL/8080A/8085A BASIC Software Support* (Requires 64K Program Memory & Option 01 or 05)		8002F1A	

*The compiler generates 8080A assembly language. This output must be assembled with the 8080A assembler included in Option 3A and run on the Z80.

Printed in U.S.A. U.S.A. and Foreign Products of Tektronix, Inc., are covered by U.S.A. and Foreign Patents under Patents Pending. All specifications subject to change without notice.

Tektronix
COMMITTED TO EXCELLENCE

MSE E-18

10/79

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

ARTICULOS 'SPECTRUM' IEEE, sobre
Microprocesadores de 1bits.

Marzo, 1981

Architecture of a New Microprocessor

Stephen P. Peng
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.

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.) For 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.[3] 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 accumulator 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-

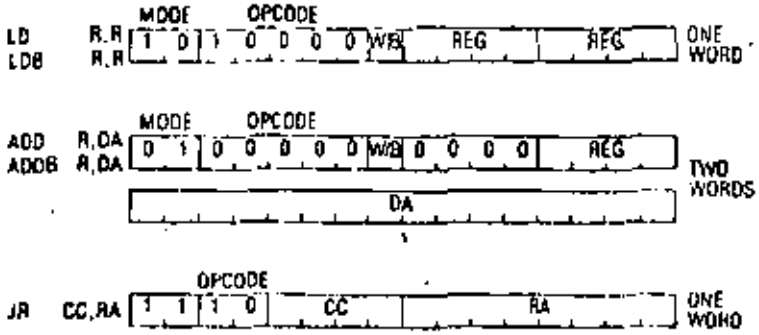


Figure 2. Examples of instruction formats (nonsegmented version).

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.[4] 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

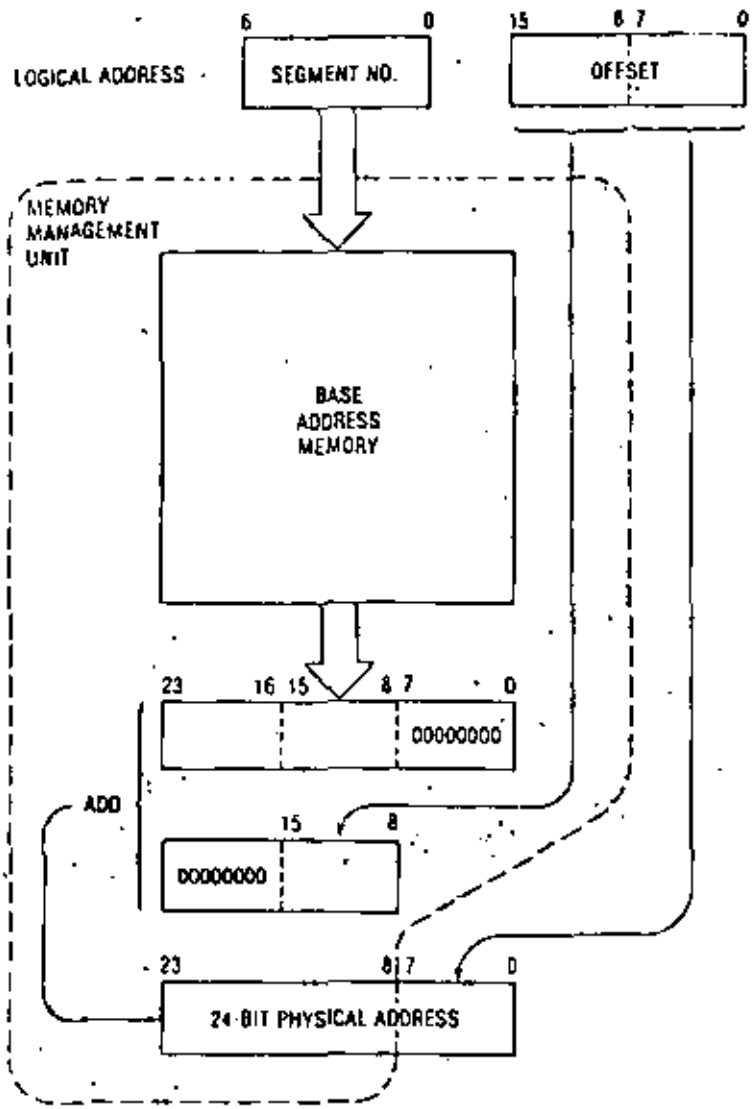


Figure 4. Logical to physical address translation.

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 associated stacks. A special class of "privileged" instructions is defined, which deals with I/O, 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. 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-

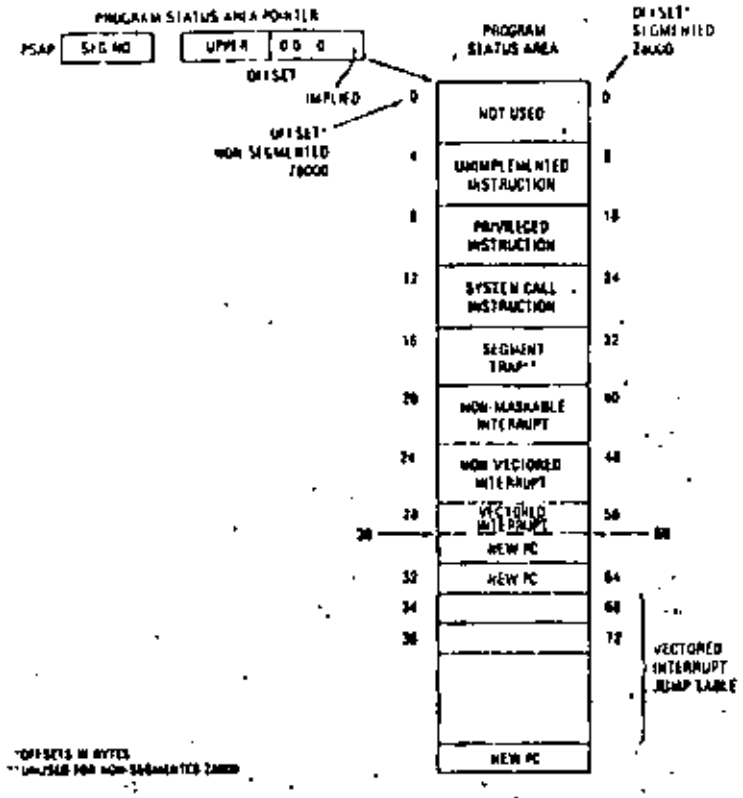


Figure 6. Program status area.

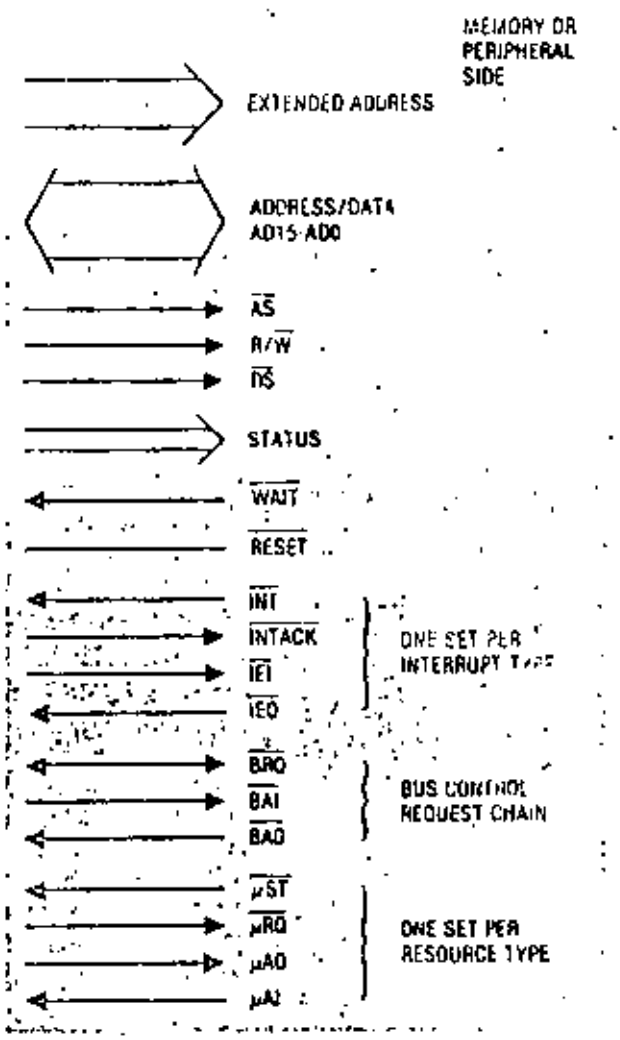


Figure 7. Z-bus signals.

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

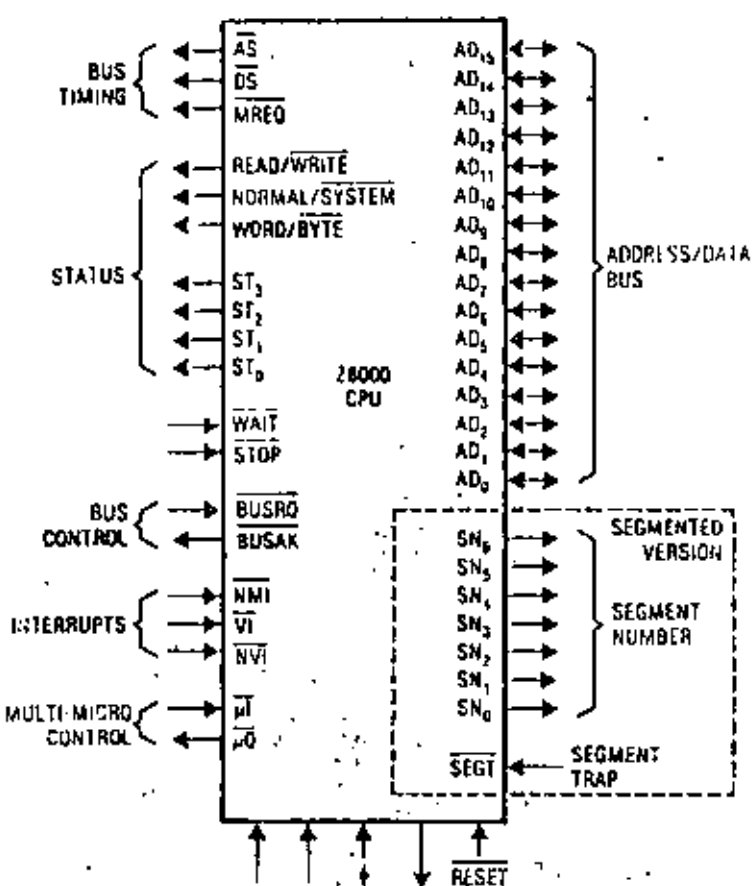


Figure 8. Z8000 pin functions.

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 seems 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 width 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 necessary to include the protection and relocation mechanisms 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

The Intel MCS-48 Microcomputer Family: A Critique

Daniel Weinbaum
Micro Systems Consultants

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.

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 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

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	22 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

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.

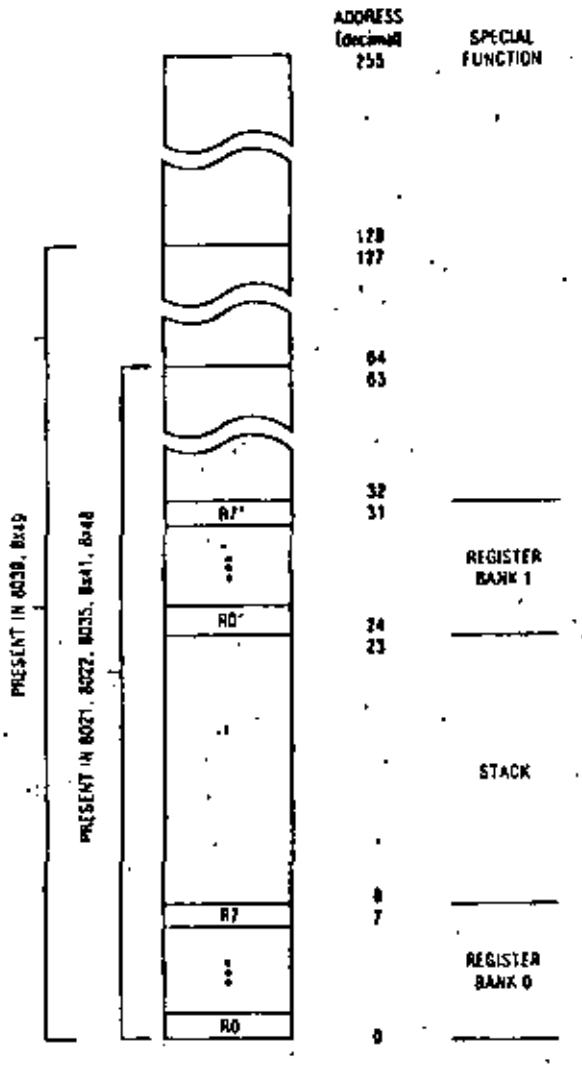


Figure 3. MCS 48 internal data memory.

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)

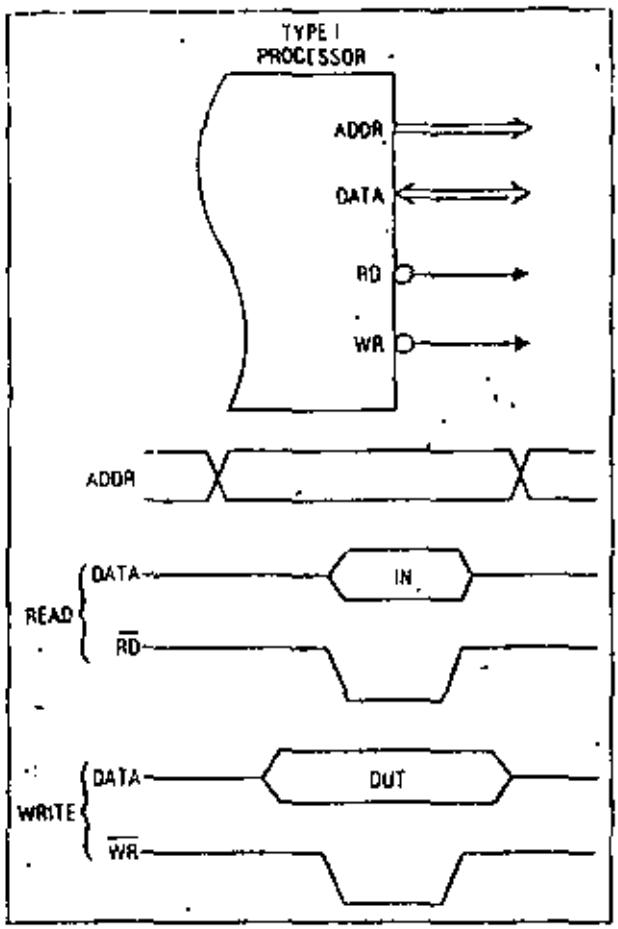


Figure 5. Type I bus control signals.

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[4-6] 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 8080, 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, $\overline{RD}$ and $\overline{WR}$, 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, $\overline{REQ}$, and a level signal R/$\overline{W}$ that indicates which type of operation is to take place. The timing of R/$\overline{W}$ 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 $\overline{RD}$ and $\overline{WR}$ from the processor nicely produces $\overline{REQ}$ for the Type Z peripheral. $\overline{RD}$ has the correct logic value to serve as R/$\overline{W}$, but its timing is a problem. The Type Z peripheral expects R/$\overline{W}$ timing to be similar in character to an address signal, that is, it should be valid long before the $\overline{REQ}$ pulse appears. The only way we could ensure this would be to artificially delay $\overline{REQ}$ long enough for $\overline{RD}$ to satisfy the setup time of R/$\overline{W}$. Unfortunately, such a delay (unless highly asymmetric) would also delay the trailing edge of REQ until long after R/$\overline{W}$ ($\overline{RD}$) 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 R/$\overline{W}$. For example, the least significant bit of the I/O port address could be reserved as R/$\overline{W}$. 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?

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
0144 01 3A BF 59
*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 four-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. ☐

MICROSYSTEMS

The first implementation of a new microprocessor architecture promises to narrow the gap between the power of very small and very large computers.

A Microprocessor Architecture for a Changing World: The Motorola 68000

Edward Stritter
Tom Gunter
Motorola Semiconductor

Microprocessor technology is entering a new and especially challenging era. While technology constraints have not completely disappeared, we are nearly to the point where the limiting factor in microprocessor design is not how much function can be included, but how imaginative and creative the designer can be.[1] As a result, several companies have introduced new-generation microprocessors. We describe how one of them, the Motorola 6800, responds to these unique conditions.

Motivations for a new microprocessor architecture

Previous generations of microprocessors were limited by the available technology. Brooks, in an overview article,[2] discusses how the technology constraints and the perceived microprocessor market motivated early microprocessor architecture. Microprocessors were limited in number of registers, data-path width, and instruction-set power primarily because technology could not support more features on a single chip. Other limitations of microprocessors, such as having too small an address space[3] and awkwardness of address computation,[4] may be attributed as much to prevailing perceptions of the potential market as to technology constraints.[5] Whatever the former sources of restraint, however, we are now in a period of technical innovation and spirited competition.

Technological advances. The basic microprocessor technology, MOS, has been steadily advanced in the last few years. The most noticeable improvement has been circuit density (Figure 1), which translates directly into the amount of capability that can be put on a single-chip microprocessor. Whereas earlier microprocessors contained from 5000 to 10,000 transistors per chip, current processors have from 25,000 to 70,000 transistors, which is less than an order of magnitude away from the number in many of the largest maxi-computers. Circuit density is not the only technology advance that has been made; corresponding improvements have been achieved in circuit speed and power dissipation.

Advances in technology have been more evolutionary than revolutionary. The major advance, increased circuit density, is the result of gradual improvements in processing techniques that permit smaller circuit dimensions. Density improvements are expected to continue, since they depend not on overcoming fundamental limitations but only on further evolutionary improvement of existing processes. New microprocessor architectures must be devised to take advantage of this future advancement.

Market demands. The demand for microprocessors in applications not foreseen just a few years ago is providing new opportunities for microprocessor manufacturers. Just as the original microprocessor designers could not predict the many uses that would be found for their devices, today's designers cannot hope to envision more than a few of the eventual applications of new microprocessors. The implication for the designer is that new designs must be flexible and general if they are to be useful in a large number of potential applications.

High software costs. The problem of software costs is even worse in microprocessor applications than it is with computers generally. Decreasing memory costs,

68000 internal architecture

Resources. The 68000 design provides an address space of 2^{32} bytes (limited to 2^{24} bytes in the initial implementation). Memory is byte addressable, with individual-bit addressing provided for bit-manipulation instructions. Memory may be accessed in units of 1, 8, 16, or 32 bits. CPU resources include sixteen 32-bit registers, a 32-bit program counter (24 bits in the initial implementation), and a 16-bit status register.

The registers (Figure 2) are divided into two classes. The eight data registers are used primarily for data manipulation; they may be operand sources or destinations for all operations but are used in addressing only as index registers. The eight remaining (address) registers are used primarily for addressing. The stack pointer is one of the address registers. The program counter and status word are separate registers.

Addressing. Memory is logically addressed in 8-bit bytes, 16-bit words, or 32-bit long words. The current implementation requires that word and long-word data be word aligned. Bits are individually addressable in the bit-manipulation instructions.

The architecture specifies an optimal memory-management scheme that implements and enforces variable-length segmentation of the address space with access rights specifiable for individual segments. The processor can be used with or without memory management.

Address calculations (Table 2) are specified by 6-bit fields of the instruction. The addressing specification is orthogonal to the operation specification of the instruction; that is, any addressing mode can be used in any instruction that uses addressing.

Addresses are 32-bit quantities (24 bits in the current implementation). The architecture efficiently supports small systems (those with fewer than 2^{16} addressable bytes) by allowing 16-bit address quantities to be specified, moved, or calculated in almost every addressing situation. For example, an absolute address carried in an instruction can use 16 or 32 bits, or an index calculation can use 16 bits (sign extended to 24 bits) or 32 bits of a register as input. This feature allows the architecture to support very large addresses without penalizing the efficiency of programs that require only small addresses. The address size (16 or 32 bits) is individually specified for each use, so that large and small addresses can be intermixed arbitrarily in a program.

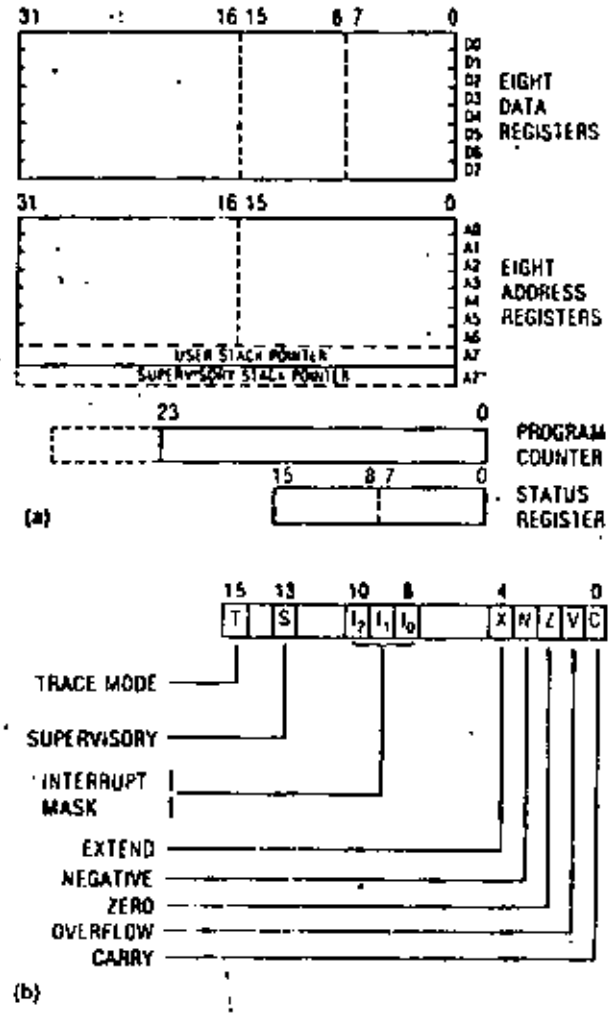


Figure 2. MC68000 programming model (a) and internal structure of status register (b).

Table 2.
MC68000 addressing modes.

REGISTER DIRECT ADDRESSING:	
data register direct	EA = Dn
address register direct	EA = An
status register direct	EA = SR
REGISTER DEFERRED ADDRESSING:	
register deferred	EA = (An)
register deferred post-increment	EA = (An); An <– An + N
register deferred pre-decrement	An <– An – N; EA = (An)
base relative	EA = (An) + d16
indexed	EA = (An) + (Xn) + d8
PROGRAM COUNTER RELATIVE:	
relative with offset	EA = (PC) + d16
relative indexed	EA = (PC) + (Xn) + d8
short PC relative branch	EA = (PC) + d8
long PC relative branch	EA = (PC) + d16
ABSOLUTE ADDRESSING:	
absolute short	EA = (next instruction word)
absolute long	EA = (next two instruction words)
IMMEDIATE DATA ADDRESSING:	
immediate	DATA = next instruction word(s)
quick immediate	DATA = subfield of instruction (4 bits)

DEFINITIONS:
EA = effective address
An = address register
Dn = data register
Xn = address or data register used as index register
SR = status register
PC = program counter
d8 = 8-bit displacement
d16 = 16-bit displacement
N = 1 for byte, 2 for word, and 4 for long word operands
( ) = contents of
<– = replaces

The 68000 data types and the operations that support them are:

Integer. The operations are ADD, SUBTRACT, MULTIPLY, DIVIDE, NEGATE, COMPARE, and ARITHMETIC SHIFT. Integers may be 1, 2, or 4 bytes. Shifts are multiple-bit shifts, either left or right, with shift count specified in the instruction or previously calculated in a data register, and indicate overflow as appropriate.

Multiprecision integer. ADD WITH EXTEND, SUBTRACT WITH EXTEND, NEGATE WITH EXTEND, UNSIGNED MULTIPLY, and UNSIGNED DIVIDE are the primitives supplied for easily implementing multiprecision integer arithmetic. Operands may be 1, 2, or 4 bytes, except for multiply and divide, which operate only on 2-byte quantities.

Logical. The operations are AND, OR, EXCLUSIVE OR, COMPLEMENT, COMPARE, SHIFT, and ROTATE (which allow multiple-position shifts and rotates, left or right, with or without extend bit). Logicals may be 1, 2, or 4 bytes.

Boolean. AND, OR, EXCLUSIVE OR, COMPLEMENT, IMPLICATION, and SET ACCORDING TO CONDITION CODES are provided. (SET ACCORDING TO CONDITION CODES is used to retrieve the logical value of any of the conditional tests that are available to the CONDITIONAL BRANCH instruction.) Boolean data are one-byte quantities.

Bit. The operations are SET, CLEAR, CHANGE, and TEST. Bits are individually addressable.

Decimal. ADD, SUBTRACT, NEGATE, and COMPARE are decimal operations. The decimal (BCD) instructions work on operands in memory (memory-to-memory) two digits (one byte) at a time. Combined with a looping instruction, the decimal instructions implement variable-length memory-to-memory decimal operations.

Character. Character instructions, MOVE and COMPARE, work on operands in memory (memory-to-memory).

Address. Address operations include INCREMENT (by 1, 2, or 4), DECREMENT (by 1, 2, or 4), ADD INTEGER, SUBTRACT INTEGER, COMPARE, and LOAD EFFECTIVE ADDRESS.

Real. Floating-point ADD, SUBTRACT, MULTIPLY, and DIVIDE are specified but not implemented in the first version.

String. STRING MOVE, STRING SEARCH, and TRANSLATE are specified but not implemented in the first version.

Program control. Program-control instructions include CONDITIONAL BRANCH (program counter relative), JUMP, JUMP TO SUBROUTINE, RETURN FROM SUBROUTINE, and RETURN FROM INTERRUPT, all of which are traditional instructions. Sixteen separate operating-system calls are specifiable with the TRAP instruction. Conditional traps, looping, and subroutine control are discussed below. The STOP instruction halts the processor, the RESET instruction reinitializes the system environment, and the MOVE instruction can manipulate the processor status word.

Privilege states. The 68000 processor can operate in user or supervisor state. In supervisor state, the entire instruction set is available. Indication of the current state is given to the external world so that, for instance, address translation can be inhibited when the processor is in supervisor state. In user state, certain instructions, such as STOP, RESET, and those that modify the status word, are not allowed; they cause a

Table 3.
MC6800 instruction set.

MNEMONIC	DESCRIPTION
ABCD	Add decimal with extend
ADD	Add
ADDX	Add with extend
AND	Logical and
ASL	Arithmetic shift left
ASR	Arithmetic shift right
BCC	Branch conditionally
BCHG	Bit test and change
BCLR	Bit test and clear
BRA	Branch always
BSET	Bit test and set
BSR	Branch to subroutine
BTST	Bit test
CHK	Check register against bounds
CLR	Clear operand
CMP	Arithmetic compare
DCNT	Decrement and branch non-zero
DIVS	Signed divide
DIVU	Unsigned divide
EOR	Exclusive or
EXG	Exchange registers
EXT	Signed extend
JMP	Jump
JSR	Jump to subroutine
LDM	Load multiple registers
LDQ	Load register quick
LEA	Load effective address
LINK	Link stack
LSL	Logical shift left
LSR	Logical shift right
MOVE	Move
MULS	Signed multiply
MULU	Unsigned multiply
NBCD	Negate decimal with extend
NEG	Two's complement
NEGX	Two's complement with extend
NOP	No operation
NOT	One's complement
OR	Logical or
PEA	Push effective address
RESET	Reset external devices
ROTL	Rotate left without extend
ROTR	Rotate right without extend
ROTXL	Rotate left with extend
ROTXR	Rotate right with extend
RTR	Return and restore
RTS	Return from subroutine
SBCD	Subtract decimal from extend
SCC	Set conditionally
STM	Store multiple registers
STOP	Stop
SUB	Subtract
SUBX	Subtract with extend
SWAP	Swap data register halves
TAS	Test and set operand
TRAP	Trap
TRAPV	Trap on overflow
TST	Test
UNLK	Unlink stack

ed by the CONDITIONAL BRANCH instructions. These instructions help implement Boolean-expression evaluation by avoiding extra conditional branches, especially in the case (as with Pascal) where "short-circuited" evaluation may be undesirable because of possible side effects.

Procedure calls. The 68000 uses a stack—pointed to by one of the address registers, called the stack pointer—to build the nested environments of called procedures. Three instructions (plus an additional one for each parameter) implement a high-level-language procedure call (Figure 4). The entire call mechanism uses only the stack and is completely reentrant (Figure 5). These instructions are described in more detail below.

Push parameter values or addresses onto the stack. The MOVE instruction pushes a value onto the stack, and the PUSH EFFECTIVE ADDRESS (see LOAD EFFECTIVE ADDRESS explained earlier) pushes the result of an arbitrary address calculation onto the stack for call by reference.

Call procedure. The JUMP TO SUBROUTINE instruction pushes the return address on the stack and jumps to the procedure entry point.

Establish new local environment. The LINK instruction does all of the following: saves the old contents of the frame pointer (an arbitrary address register) on the stack, points the frame pointer to the new top of stack, and subtracts the number of bytes of local storage required by the procedure from the stack pointer. This establishes local storage for the called procedure and a frame pointer (address register) for index addressing of local variables and parameters.

Save an arbitrary subset of the registers on the stack. The MOVE MULTIPLE REGISTERS instruction saves an arbitrary subset of the registers on the stack (or anywhere in memory) in a single instruction. The registers to be saved are indicated by setting the corresponding bits in a 16-bit field of the instruction.

A set of at most four instructions reverses the process for procedure return:

Reload saved registers. The MOVE MULTIPLE REGISTERS instruction is used here also.

Reestablish previous environment. The UNLINK instruction undoes the work of the LINK instruction.

Return from procedure. The RETURN instruction pops the return address from the stack and returns to the calling procedure.

Pop parameters from the stack. The ADD IMMEDIATE instruction used on the stack pointer pops any number of values off the stack.

The 68000 system architecture

A computer architecture specifies interactions between the processor and its environment by defining such things as interrupt structure, memory segmentation, bus interfaces, and input/output structure. The 68000 system architecture is designed to be as flexible as possible. For instance, I/O device registers are addressed as memory locations (memory-mapped I/O), as on other Motorola microprocessors. Memory-mapped I/O gives the programmer the flexibility and power of the entire instruction set for manipulating device control and data registers. Since no additional instructions are required for I/O, the processor is simpler, and the instruction set is easier to remember. The I/O space is protected by the same memory-management facilities that are used to protect critical areas of memory.

The 68000 bus structure is also designed for simplicity, speed, and flexibility. The address and data lines are separate; no multiplexing is needed. This avoids the need for any separate devices for demultiplexing, ensuring maximum performance for systems in which speed is important. The bus is asynchronous; transfers on the bus are controlled by accompanying handshake signals, so that no assumptions need be made about timing or system synchrony. The use of handshake signals allows devices and memories with large variations in response time to be used on the same processor bus. The processor waits an arbitrary amount of time until the accessed device or memory signals that the transfer is occurring.

A simple bus request/grant protocol is implemented on-chip so that processors and direct-memory-access devices can cooperatively share the system bus with no extra arbitration logic. Also, the chip has a bus-fault input pin that causes instruction execution to be terminated at any point and a trap to be taken if an illegal or faulty memory access is made. This facilitates memory protection.

The 68000 interrupt structure is like that of most minicomputers. Eight priority levels are implemented. Interrupts are vectored so that software has full control over the placement and execution of interrupt-handling routines. The current priority level of the processor is kept in its status word. Interrupts at or below the current priority are inhibited. Interrupts at higher levels may occur, so interrupt handling may be nested. When an enabled interrupt occurs, the processor sends an acknowledge signal. The interrupting device responds with a vector number. The vector number is used by the processor

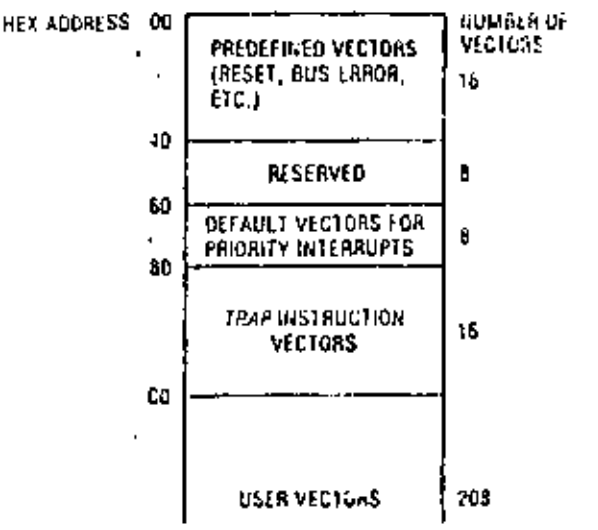


Figure 6. Trap and interrupt vector allocation

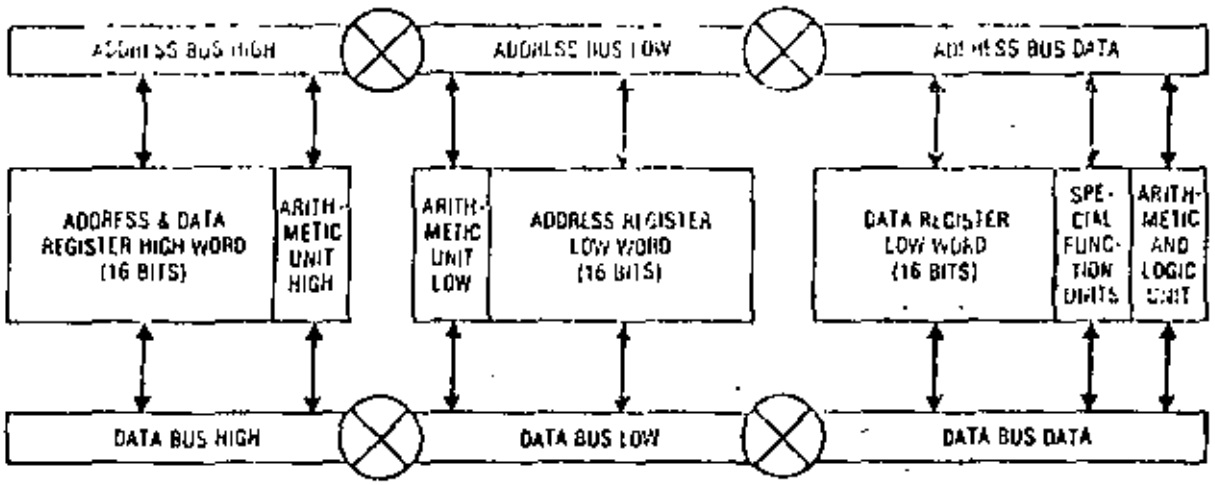


Figure 8. MC68000 execution unit configuration.

of circuit area and permits logic gates to be designed with a speed-to-power product of one picojoule. An advanced high-density *n*-channel silicon-gate MOS technology was selected for the design of the 68000. This technology supports three-micron device geometries and provides the designer with multiple MOS transistor threshold voltages. The technology allows the circuit designer to develop high-performance logic gates using minimum-size devices and to develop internal buffer circuits requiring little power.

The execution unit is a dual-bus structure that performs both address and data processing (Figure 8). The two buses are 16 bits wide, and each can be dynamically reconfigured into three independent sections as required by the microcode. Three independent arithmetic units are available to perform these calculations; also, special logic functions are provided to execute long shifts, priority encoding, and bit manipulation. Each of these units is connected to two internal buses and receives both input operands simultaneously from the registers. Each bus contains both the true and the complement logic values so that differential circuit design can be used for higher-speed operation. The execution unit directly interfaces to the external bus logic and buffers, but its operation is independent of the external timing requirements of the bus.

The control of the 68000 is implemented by microcode. The actual structure of the microprogrammed control structure is discussed in detail in another paper.[9] The microcontrol is implemented as a two-level structure. The first level contains sequences of microinstructions with short "vertical" format and complex branching capabilities. Microinstructions contain the addresses of nanoinstructions, wide "horizontal" control words, stored in the second level. The nanoinstructions directly control the execution unit. The use of microcode is motivated by the high design cost of new VLSI chips. The microcode's regularity of structure compared to combinatorial logic significantly decreases the design complexity. Microcode also permits some engineering decisions - for instance, details of specific instructions - to be delayed. In other words, once the micromachine architecture is determined, hardware implementation (circuit design) and firmware implementation (microprogramming) can be done in parallel.

Conclusion

The Motorola 68000 architecture combines advanced technology improvements with a better understanding of the architectural needs of microprocessor users and microprocessor applications. The 68000 is a step into an area previously occupied

APPENDIX E

Z80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	
8E	ADC	A,(HL)
DD8E05	ADC	A,(IX+d)
FD8E05	ADC	A,(IY+d)
8F	ADC	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
CE20	ADC	A,n
ED4A	ADC	HL,BC
ED5A	ADC	HL,DE
ED6A	ADC	HL,HL
ED7A	ADC	HL,SP
86	ADD	A,(HL)
DD8605	ADD	A,(IX+d)
FD8605	ADD	A,(IY+d)
87	ADD	A,A
80	ADD	A,B
81	ADD	A,C
82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
C620	ADD	A,n
09	ADD	HL,BC
19	ADD	HL,DE
29	ADD	HL,HL
39	ADD	HL,SP
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD29	ADD	IX,IX
DD39	ADD	IX,SP
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD29	ADD	IY,IY
FD39	ADD	IY,SP
A6	AND	(HL)
DDA605	AND	(IX+d)
FDA605	AND	(IY+d)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L

OBJ CODE	SOURCE STATEMENT	
E620	AND	n
CB46	BIT	0,(HL)
DDCB0546	BIT	0,(IX+d)
FDCB0546	BIT	0,(IY+d)
CB47	BIT	0,A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB4E	BIT	1,(HL)
DDCB054E	BIT	1,(IX+d)
FDCB054E	BIT	1,(IY+d)
CB4F	BIT	1,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB56	BIT	2,(HL)
DDCB0556	BIT	2,(IX+d)
FDCB0556	BIT	2,(IY+d)
CB57	BIT	2,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB5E	BIT	3,(HL)
DDCB055E	BIT	3,(IX+d)
FDCB055E	BIT	3,(IY+d)
CB5F	BIT	3,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB66	BIT	4,(HL)
DDCB0566	BIT	4,(IX+d)
FDCB0566	BIT	4,(IY+d)
CB67	BIT	4,A
CB60	BIT	4,B
CB61	BIT	4,C
CB62	BIT	4,D

OBJ CODE	SOURCE STATEMENT	
CB63	BIT	4,E
CB64	BIT	4,H
CB65	BIT	4,L
CB6E	BIT	5,(HL)
DDCB056E	BIT	5,(IX+d)
FDCB056E	BIT	5,(IY+d)
CB6F	BIT	5,A
CB68	BIT	5,B
CB69	BIT	5,C
CB6A	BIT	5,D
CB6B	BIT	5,E
CB6C	BIT	5,H
CB6D	BIT	5,L
CB76	BIT	6,(HL)
DDCB0576	BIT	6,(IX+d)
FDCB0576	BIT	6,(IY+d)
CB77	BIT	6,A
CB70	BIT	6,B
CB71	BIT	6,C
CB72	BIT	6,D
CB73	BIT	6,E
CB74	BIT	6,H
CB75	BIT	6,L
CB7E	BIT	7,(HL)
DDCB057E	BIT	7,(IX+d)
FDCB057E	BIT	7,(IY+d)
CB7F	BIT	7,A
CB78	BIT	7,B
CB79	BIT	7,C
CB7A	BIT	7,D
CB7B	BIT	7,E
CB7C	BIT	7,H
CB7D	BIT	7,L
DC8405	CALL	C,nn
FC8405	CALL	M,nn
D48405	CALL	NC,nn
C48405	CALL	NZ,nn
F48405	CALL	P,nn
EC8405	CALL	PE,nn
E48405	CALL	PO,nn
CC8405	CALL	Z,nn
CD8405	CALL	nn
3F	CCF	
BE	CP	(HL)
DDBE05	CP	(IX+d)
FDBE05	CP	(IY+d)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n
EDA9	CPD	
EDB9	CPDR	

OBJ CODE	SOURCE STATEMENT	
ED81	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
10FE	DJNZ	e
FB	EI	
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF,AF'
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM	0
ED56	IM	1
ED5E	IM	2
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY
2C	INC	L
33	INC	SP
DB20	IN	A,(n)

OBJ CODE	SOURCE STATEMENT	
EDAA	IND	
EDBA	INDR	
EDA2	INI	
EDB2	INIR	
C38405	JP	nn
E9	JP	(HL)
DDE9	JP	(IX)
FDE9	JP	(IY)
DA8405	JP	C,nn
FA8405	JP	M,nn
D28405	JP	NC,nn
C28405	JP	NZ,nn
F28405	JP	P,nn
EA8405	JP	PE,nn
E28405	JP	PO,nn
CA8405	JP	Z,nn
38FE	JR	C,e
30FE	JR	NC,e
20FE	JR	NZ,e
28FE	JR	Z,e
18FE	JR	e
02	LD	(BC),A
12	LD	(DE),A
77	LD	(HL),A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
3620	LD	(HL),n
DD7705	LD	(IX+d),A
DD7005	LD	(IX+d),B
DD7105	LD	(IX+d),C
DD7205	LD	(IX+d),D
DD7305	LD	(IX+d),E
DD7405	LD	(IX+d),H
DD7505	LD	(IX+d),L
DD360520	LD	(IX+d),n
FD7705	LD	(IY+d),A
FD7005	LD	(IY+d),B
FD7105	LD	(IY+d),C
FD7205	LD	(IY+d),D
FD7305	LD	(IY+d),E
FD7405	LD	(IY+d),H
FD7505	LD	(IY+d),L
FD360520	LD	(IY+d),n
328405	LD	(nn),A
ED438405	LD	(nn),BC
ED538405	LD	(nn),DE
228405	LD	(nn),HL
DD228405	LD	(nn),IX
FD228405	LD	(nn),IY
ED738405	LD	(nn),SP
0A	LD	A,(BC)
1A	LD	A,(DE)
7E	LD	A,(HL)

OBJ CODE	SOURCE STATEMENT	
DD7E05	LD	A,(IX+d)
FD7E05	LD	A,(IY+d)
3A8405	LD	A,(nn)
7F	LD	A,A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
ED57	LD	A,I
7D	LD	A,L
3E20	LD	A,n
ED5F	LD	A,R
46	LD	B,(HL)
DD4605	LD	B,(IX+d)
FD4605	LD	B,(IY+d)
47	LD	B,A
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
0620	LD	B,n
ED4B8405	LD	BC,(nn)
018405	LD	BC,nn
4E	LD	C,(HL)
DD4E05	LD	C,(IX+d)
FD4E05	LD	C,(IY+d)
4F	LD	C,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
4B	LD	C,E
4C	LD	C,H
4D	LD	C,L
0E20	LD	C,n
56	LD	D,(HL)
DD5605	LD	D,(IX+d)
FD5605	LD	D,(IY+d)
57	LD	D,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
1620	LD	D,n
ED5B8405	LD	DE,(nn)
118405	LD	DE,nn
5E	LD	E,(HL)
DD5E05	LD	E,(IX+d)
FD5E05	LD	E,(IY+d)
5F	LD	E,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D

OBJ CODE	SOURCE STATEMENT	
5B	LD	E,E
5C	LD	E,H
5D	LD	E,L
1E20	LD	E,n
66	LD	H,(HL)
DD6605	LD	H,(IX+d)
FD6605	LD	H,(IY+d)
67	LD	H,A
60	LD	H,B
61	LD	H,C
62	LD	H,D
63	LD	H,E
64	LD	H,H
65	LD	H,L
2620	LD	H,n
2A8405	LD	HL,(nn)
218405	LD	HL,nn
ED47	LD	I,A
DD2A8405	LD	IX,(nn)
DD218405	LD	IX,nn
FD2A8405	LD	IY,(nn)
FD218405	LD	IY,nn
6E	LD	L,(HL)
DD6E05	LD	L,(IX+d)
FD6E05	LD	L,(IY+d)
6F	LD	L,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
2E20	LD	L,n
ED4F	LD	R,A
ED7B8405	LD	SP,(nn)
F9	LD	SP,HL
DDF9	LD	SP,IX
FDF9	LD	SP,IY
318405	LD	SP,nn
EDA8	LDD	
EDB8	LDDR	
EDA0	LDI	
EDB0	LDIR	
ED44	NEG	
00	NOP	
B6	OR	(HL)
DDB605	OR	(IX+d)
FDB605	OR	(IY+d)
B7	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F620	OR	n
EDBB	OTDR	

OBJ CODE	SOURCE STATEMENT	
EDB3	OTIR	
ED79	OUT	(C),A
ED41	OUT	(C),B
ED49	OUT	(C),C
ED51	OUT	(C),D
ED59	OUT	(C),E
ED61	OUT	(C),H
ED69	OUT	(C),L
D320	OUT	(n),A
EDAB	OUTD	
EDA3	OUTI	
F1	POP	AF
C1	POP	BC
D1	POP	DE
E1	POP	HL
DDE1	POP	IX
FDE1	POP	IY
F5	PUSH	AF
C5	PUSH	BC
D5	PUSH	DE
E5	PUSH	HL
DDE5	PUSH	IX
FDE5	PUSH	IY
CB86	RES	0,(HL)
DDCB0586	RES	0,(IX+d)
FDCB0586	RES	0,(IY+d)
CB87	RES	0,A
CB80	RES	0,B
CB81	RES	0,C
CB82	RES	0,D
CB83	RES	0,E
CB84	RES	0,H
CB85	RES	0,L
CB8E	RES	1,(HL)
DDCB058E	RES	1,(IX+d)
FDCB058E	RES	1,(IY+d)
CB8F	RES	1,A
CB88	RES	1,B
CB89	RES	1,C
CB8A	RES	1,D
CB8B	RES	1,E
CB8C	RES	1,H
CB8D	RES	1,L
CB96	RES	2,(HL)
DDCB0596	RES	2,(IX+d)
FDCB0596	RES	2,(IY+d)
CB97	RES	2,A
CB90	RES	2,B
CB91	RES	2,C
CB92	RES	2,D
CB93	RES	2,E
CB94	RES	2,H
CB95	RES	2,L
CB9E	RES	3,(HL)
DDCB059E	RES	3,(IX+d)
FDCB059E	RES	3,(IY+d)

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

INTRODUCCION A LOS MICROPROCESADORES (Z-80)

PROGRAMA DE RELOJ DIGITAL

MARZO, 1981.

io de Minería Calle de Tacuba 5 primer piso México 1, D. F. Tel: 521-40-20 Apdo. Postal M-2285

Programa de Reloj Digital

Reloj del sistema = 1.9968 Mhz

1.9968 Mhz /2564800 Hz; contando 200 ciclos por interrupción y 39 interrupciones por segundo para incrementar segundos.

Programa Principal

```
2000  ED 5E      IM 2          Modo de interrupción 2
      3E 20      LDA, 20       Byte más significativo
      ED 47      LDI, A        Registro I
      3E 1A      LDA, 1A       Byte menos significativo
      D3 84      OUT (84),A    Canal 0 CTC
      3F A5      LDA,A5        Permitir int. reloj/256
      D3 85      OUT(85),A     Canal 1 CTC
      3E C8      LDA,C8        Constante de tiempo = 200
2010  D3 85      OUT (85), A
      D9         EXX           Intercambiar registros
      06 27      LDB,39        Número de interrupciones
      D9         EXX
      FB         EI            Se permiten interrupciones
      C3 9F 00   JP RESTAR     Monitor despliegue
201A  2020                     Vector de interrupción
```

Programa de servicio de interrupciones

```
201C                           Segundos en BCD
201D                           Minutos en BCD
201E                           Horas en BCD

2020  08         EX AF,A'F'    Intercambiar registros
      D9         EXX
      05         DEC B         Contar interrupciones
      20 5C      JR NZ RET     Continuar
      06 27      LD B,27       39 interrupciones por seg.
      3A 1C 20   LD A,(201C)   Segundos
      3C         INC A
      27         DAA           Ajuste decimal
      FE 60      CP 60         60 segundos.
      38 26      JR C          No
2030  AF         XOR A         Si-borrar A
```



```
      32 1C 20   LD (201C),A
      3A 1D 20   LDA,(201D)    Minutos
      3C         INC A
      27         DAA
      FE 60      CP 60         60 minutos ?
      38 14      JR C          No
      AF         XOR A         Si-borrar A
      32 1D 20   LD (201D),A
2041  3A 1E 20   LDA,(201E)    Horas
      3C         INC A
      27         DAA
      FE 13      CP 13         13 horas ?
      38 02      JR C          No.
204A  3E 01      LDA,01
      32 1E 20   LD (201E),A
      18 08      JR 08
2051  32 1D 20   LD (201D),A
      18 03      JR 03
      32 1C 20   LD (201C),A
      3A 1C 20   LDA,(201C)    Segundos
      DD 21 FB 23  LD IX,23FB  LED buffer No.5
2060  CD 86 20   CALL INT      Format para LED buffer
      3A 1D 20   LDA, (201D )  Minutos
      DD 21 F9 23  LD IX, 23F9 LED buffer No. 3
      CD 86 20   CALL INT
      3A 1E 20   LDA, (201E )  Horas
2070  DD 21 F7 23  LD IX,23E7  LED buffer No. 1
      CD 86 20   CALL INT
      3A 17 23   LDA,(2317)    LED buffer No. 2 = 0 ?
      20 05      JR NZ,05
      3E 10      LDA,10        Apagar LED No. 1
      32 F7 23   LD (23F7),A
2081  08         RET EX AF,A'F'  Intercambiar registros
      D9         EXX
      FB         EI            Permitir interrupciones
      ED 4D      RTI           Retorno
2086  4F         INT LD C,A    Subrutina para format
      E6 0F      AND 0F        4 bits menos significativo
```

```
DD 77 01        LD (IX+1),A     LED buffer
CB 39           SRL C           4 bits más significativo
CB 39           SRL C
CB 39           SRL C
CB 39           SRL C
DD 71 00        LD (IX+0),C     LED buffer
C9              RET             Retorno
```


Programa para Control de un Motor de Paso Superior MS-25

Cuenta Inicial: Registros BC Cuenta Final: Registros DE

Bobinas del motor conectadas a PIO PB0-PB3 por amplificador de potencia - (transistores Darlington).

```
2000  3E CF           LDA,CF          Modo Salida
      D3 83           OUT(83),A
      62        COMP  LD H,D
      6B              LD L,E
      AF              XOR A           Borrar acarreo
      ED 42           SBC HL,BC
      FA 11 20        JP M 2011       Dirección negativa?
      28 25           JR Z STOP       Igual?
      03              INC BC          Positiva
      18 01           JR 01
2011  0B              DEC BC          Negativa
      79              LD A,C          Byte menos significativo
      E6 03           AND 03          Dos bits
      62              LD H, D
      6B              LD L,E
      EB              EX DE,HL        Guardar cuenta final
      16 00           LD D,0
      5F              LD E,A
      DD 21 39 20     LD IX,TABLA     Rotación de bobinas
      DD 19           ADD IX,DE       Posición en la tabla
2021  DD 7E 00        LD A, (IX+0)    Codigo de bobinas
      D3 81           OUT (81),A      PIO lado B
      EB              EX DE, HL       Reponer cuenta en DE
      26 01           LD H,1          Byte más significativo
      2E 00           LD L,0          Byte menos significativo
      2D              DEC L           De espera
      20 ED           JR NZ ESP
      25              DEC H
      20 FB           JR NZ ESP
2031  18 D1           JR COMP         Otro paso
      AF        STOP  XOR A           Salidas Cero
      D3 81           OUT (81),A
      C3 AE 00        JP RESTRI       Retorno al monitor
2039  0A 06 05 09 TABLA
```

Programa para Contar Interrupciones

Interrupciones: pulsos hacia arriba en ASTRB. Despliegue número 1-99 en los LEDs.

```
2200    ED 5E          IM 2
        3E F8          LDA F8          Byte menos significativo
        D3 82          OUT (82),A      del vector de inter.
        3E 4F          LDA, 4F         Modo entradas
        D3 82          OUT(82),A       Lado A control
        3E 87          LDA,87          Se permiten interrupciones
        D3 82          OUT (82),A
        3E 07          LDA, 07         Byte más significativo
2210    ED 47          LDI,A           en registro I
        FB             EI              MPU acepta interrupciones
        C3 9F 00       JP RESTAR       Retorno al monitor
        ;
07F8    D6 23                          Vector en ROM

2306    C3 20 22       JP 2220
        ;
2220    08             EX AF,A'F'      Intercambio registros
        D9             EXX
        3A FC 23       LDA, (DISMEM + 5)   LED Buffer No. 6
        E6 0F          AND 0F
        3C             INC A
        FE 0A          CP A            Diez o más?
        38 0F          JR C, CONT      No.
        AF             XOR A           Si- borrar A
        32 FC 23       LD(DISMEM + 5),A
2230    3A FB 23       LDA, (DISMEM + 4)   Incrementar 10's
        E6 0F          AND 0F
        3C             INC A
        32 FB 23       LD(DISMEM + 4),A
        18 03          JR 03
        32 FC 23       LD(DISMEM +5) ,A
        21 00 80       LD HL,8000      Esperar desconexión
2241    2B        ESP  DEC L
        20 FD          JR NZ ESP
```



```
        25             DEC H
        20 FA          JR NZ ESP
        08             EX AF,A' F'     Reponer registros
        D9             EXX
        FB             EI              Permitir interrupciones
        ED 4D          RTL             Retorno
```

Programa para contar frecuencia de una señal externa con CTC

Contar ondas en Canal 1 CLK/TRG por I/O reg. usando canal 3 con 4 interrupciones, cada una de 195*256/1.9968x10^6 = .025 seg.
Para probarlo, usar canal 0 salida ZC.

Programa Principal

```
2060  3E 05        LDA,05          Reloj /16, no interrup.
      D3 84        OUT(84),A       Canal 0
      3E 80        LDA,80          128*16, ciclos =975 Hz
      D3 84        OUT (84),A
2068  ED 5E        IM 2
      3E 20        LDA,20          Byte más significativo
      ED 47        LDI,A           Registro I
      3E 68        LDA,68          Byte menos significativo
2070  D3 84        OUT (84),A      se escribe en canal 0
      3E A5        LDA,A5          Reloj/256, permitir  int.
      D3 87        OUT (87),A      Canal 3
      3E C3        LDA,C3          Constante de tiempo = 195
      D3 87        OUT (87),A
      3E 55        LDA,55          Modo contador, no int.
      D3 85        OUT (85),A      Canal 1
      AF           XOR A           Borrar A
      D3 85        OUT (85),A      Constante de tiempo
2081  32 FC 23     LD (DISMEM +5),A  Cero en LED No. 6
      D9           EXX             Cargar registros altern.
      0E 00        LDC,00          Cuenta inicial
      06 04        LD B,04         Cuatro interrupciones
      D9           EXX
      FB           EI              Permitir interrupciones
      C3 F4 00     JP DISUP        Monitor despliegue
208E  90 20                        Vector de int. canal 3
```

Programa de servicio de interrupciones

```
2090  08           EX AF,A'F'
      D9           EXX
      05           DEC B           Contar interrupciones
      20 41        JR NZ RET
      06 04        LD B, 04
      DB 85        IN A,(85)       Cuenta nueva
      5F           LD E, A         Guardar en E
      57           LD D, A         Contar con D
      AF           XOR A           Borrar A y DISMEM
      32 F9 23     LD (DISMEM +2),A
      32 FA 23     LD (DISMEM +3),A
2DA2  32 FB 23     LD (DISMEM +4),A
      18 2A        JR CMPR
      3A FB 23  REPT  LD A,(DISMEM +4)
      3C           INC A
      FE 0A        CP A            Diez o mayor?
      38 1E        JR C            No.-- continuar
      AF           XOR A           Si --borrar A
2DB0  32 FB 23     LD (DISMEM +4),A
2DB3  3A FA 23     LD A,(DISMEM +3)
      3C           INC A
      FE 0A        CP A            Diez o mayor?
      38 0D        JR C            No - continuar
      AF           XOR A           Si - borrar A
      32 FA 23     LD (DISMEM +3),A
      3A F9 23     LD A, (DISMEM +2)
2DC2  3C           INC A
      32 F9 23     LD (DISMEM +2),A
      18 08        JR 08
      32 FA 23     LD(DISMEM +3),A
      18 03        JR 03
      32 FB 23     LD (DISMEM +4),A
2DD0  14           INC D
      7A        CMPR  LDA,D
      B9           CP C            Igual a cuenta previa?
      20 D2        JR NZ,REPT      Repetir
      4B           LD C,E          Guardar nueva cuenta
      08        RET  EX AF,A'F'    Intercambiar registros
      D9           EXX
      FB           EI              Permitir interrupciones
      ED 4D        RTI             Retorno
```

Programa para Síntesis de Música con Z-80 Starter Kit

Salida: onda cuadrada en U14 pata 2 proveniendo del CTC canal 1.
La melodía se almacena en una tabla de frecuencias (períodos de tonos - en unidades de 16 micro segundos) y duraciones en unidades de 11 mi-→ lisegundos. Una frecuencia de 00 en la tabla indica silencio.

Programa Principal

```
2200   21 04 22    CON   LD  HL, TABLA     Dirección de la tabla
       7E          RECOM LD A, (HL)       Nota
       B7                OR A             Checar para cero
       CC 20 22          CALL Z, SILEN
       C4 28 22          CALL, NZ. NOTA
       23                INC HL
       56                LD D, (HL)       Duración
       CD 30 22          CALL ESP         Subrutina de espera
2210   CD 20 22          CALL SILEN       Pausa entre notas
       16 01             LD D, 01
       CD 30 22          CALL ESP
       23                INC HL           Siguiente nota
       18 E8             JR RECOM
```

Subrutina de silencio

```
2220   3E 03    SILEN LD A, 03
       D3 85          OUT (35),A         Apagar CTC
       AF             XOR A              Canal 1
       C9             RET                Regresar con cero
                                         Retorno
```

Subrutina de Nota

```
2228   3E 05    NOTA  LD A, 05           Dividir por 16
       D3 05          OUT (05),A         Canal 1
       7E             LDA, (HL)          Sacar período
       D3 05          OUT (05),A
       C9             RET                Retorno
```


Subrutina de Espera

```
2230   06 10     ESP   LD B, 10           16 veces por el bucle
       1E 00           LD E, 00
       1D        BUCLE DEC E
       20 FD           JR NZ BUCLE
       10 FB           DJNZ BUCLE         Registro B
       15              DEC D              Duración de la tabla
       20 F4           JR NZ ESP
       C9              RET                Retorno
```

Tabla de la melodía

```
2240   9F 08 B6 04 05 04 C8 08 D5 10 A9 08 9F 10 00 00 00 00
```

Programa para Salida Audio — PIO PAO

440 Hz (T/2) = 1.13636 mseg. por 1/2 ciclo
Reloj del Starter Kit= 1.9968 MHz
2269.1 ciclos = 1.13636 mseg.

```
2260  3E CF        LDA CF          Modo de salidas
      D3 82        OUT (82),A      Canal A control
      AF           XOR A           Registro A = 0
      D3 80  REP   OUT(80),A       Canal A datos
      06 8C        LD B,8C         140 veces por el bucle
      05           DEC B
      20 FD        JR NZ
      3C           INC A           Cambiar bit 0
      18 F6        JR REP
```

Cuenta : 16*139 + 11 + 11 + 7 +4 +12 =2269 ciclos.

Programa para Salida Audio —CTC Canal 1

Starter Kit 2C1 = pata 8 del CTC = pulsos (1 microseg.) a 878,9 Hz con constante de tiempo de 142*16 = 7272 ciclos del reloj. U14 pata 3= onda cuadrada a 439.4Hz.

```
2270  3E 05        LDA,05          Timer modo, ÷ 16, no int.
      D3 85        OUT(85),A       Canal 1 control
      3E 8E        LDA,8E          Constante de tiempo
      D3 85        OUT(85),A
      C3 F4 00     JP 00F4         Retorno al monitor
```


Programa para contar ángulos usando convertidor incremental

Rutina de servicio de interrupciones

```
2000  08           EX AF, A'F'     Interrupción lado A
      DB 81        IN A, (81)      Leer lado B
      18 03        JR 03
2005  08           EX AF, A'F'     Interrupción lado B
      DB 80        IN A,(80)       Leer lado A
      D9           EXX             Guardar registros
      E6 C0        AND C0          Examinar bits 6,7
      B8           CP B            B=valor previo
      FA 1420      JP M 2014       Negativo ?
      28 09        JR Z            Cero?
2011  13           INC DE          Ajustar ángulo +
      18 01        JR 01
2014  1B           DEC DE          Ajustar ángulo -
      47           LD B,A          Guardar bits 6,7
      ED 53 9C 20  LD(209C),DE     Guardar ángulos
      08           EX AF,A'F'      Intercambiar registros
      D9           EXX
      FB           EI              Permitir interrupciones
      ED 4D        RETI            Retorno
```

Programa Principal

```
2020  ED 5E        IM 2
      3E 98        LDA,98          Byte menos significativo
```

```
     D3 82          OUT(82),A       Vector lado A
     3E 9A          LDA, 9A         
     D3 83          OUT (93),A      Vector lado B
     3E 4F          LDA,CF          Todo entrada
     D3 82          OUT(82),A       Lado A
     D3 83          OUT(83),A       Lado B
2030 3E 87          LDA,87          Permitir interrupciones
     D3 82          OUT(82),A       Lado A
     D3 83          OUT(83),A       Lado B
     3E 20          LDA,20          Byte más significativo
     ED 47          LDI, A          Registro I
     FB             EI              Permitir interrupciones
     DD 21 A0 20 EMP LD Ix, 20A0    LED buffer
     3A 9D 20       LD A,(209D)     Byte más significativo
2042 F6 0F          AND 0F          4 bits menos signif.
     DD 77 01       LD (IX+1),A
     3A 9D 20       LD A,(209D)
204A CB 3F          SRL A           Correr bits 4-7
     CB 3F          SRL A
     CB 3F          SRL A
2050 CB 3F          SRL A
     DD 77 00       LD(IX),A        4 bits más significativo
     3A 9C 20       LDA,(209C)      Byte menos significativo
     E6 0F          AND 0F
     DD 77 03       LD (IX+3),A     Bits menos significativo
     3A 9C 20       LDA,(209C)
2060 CB 3F          SRL A
     CB 3F          SRL A
     CB 3F          SRL A
     CB 3F          SRL A
     DD 77 02       LD (IX+2),A     4 bits más signif.
     21 A0 20       LD HL,20A0      LED buffer
     06 08          LD B,08         LED No. 3
2070 5E      DCB    LD E,(HL)
     16 00          LD D,0
     3E 00          LD A,0
     D3 8C          OUT (DIGLH),A   Apagar LEDs
     DD 21 A6 07    LD IX, 07A6     Tabla de segmentos
     DD 19          ADD IX,DE       Posición en la tabla
```



```
     DD 7E 00       LD A,(IX+0)     Codigo de segmentos
2080 D3 88          OUT (SEGLH),A   LED segmentos
     78             LD A,B          B indica LED número
     D3 8C          OUT (DIGLH),A   LED catodos
     1E 2D          LD E,2D         0.6 mseg. espera
     1D             DEC E
     3E 00          LD A,0
     BB             CP E
     20 FA          JR NZ
     3E 01          LD A,01
     B8             CP B            Terminado?
2090 28 A9          JR Z,INF        Recomenzar
     23             INC HL          Continuar
     CB 38          SRL B           Siguiente LED
     18 D9          JR DES
     00
2098 00 20                          Puerto A vector
209A 05 20                          Puerto B vector
```

DIRECTORIO DE ASISTENTES AL CURSO INTRODUCCION A LOS MICROPROCESADORES Z-80
3,4 y 5 DE MARZO DE 1981

NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
1. ELEUTERIO BADILLO ANGELES Av. 531 No. 147 Unidad Aragón México 14, D. F. Tel: 5-51-83-17	PETROLEOS MEXICANOS Ingeniero Especialista Comunicaciones y/o Electrónica Marina Nacional No. 329 Col. Anáhuac México 17, D. F. Tel: 2-50-26-11
2. MIGUEL ANGEL BRENA BECERRIL Calzada de Tlalpan 1108-34 Col. Nativitas México 13, D. F. Tel: 6-74-01-40	COLEGIO DE CIENCIAS Y HUMANIDADES NAUCALPAN,UNAM Profesor Area de Matemáticas Calz. de los Remedios No. 10 Naucalpan, Edo. de México
3. RAMON CERDA PONS Nápoles 84-22 Col. Juárez México 6, D. F. Tel: 5-11-90-21	SECRETARIA DE AGRICULTURA Y RECURSOS HIDRAULICOS Reforma 69-5o. Piso Col. Tabacalera México 1, D. F. Tel: 5-46-17-55
4. SAUL ARTURO CERVANTES TEMELO Melero y Piña 607-3 Toluca, Edo de México	ELECTRONICA, S. A. de C.V. Ingeniero de Diseño Km. 52.5 Carretera México Toluca Tel: 6-13-00
5. IVONNE A.CISHEK HERRERA Paseo de la Reforma Norte No. 616 Apdo. 1806 Tlatelolco México 3, D. F. Tel: 5-29-90-80	
6. RENATO DESCHAMPS ESQUIVEL Calzada La Virgen No. 26-2 Col. Avante México 21, D. F. Tel: 5-44-52-69	FACULTAD DE INGENIERIA, UNAM Subcoordinador Computadoras y Programación Ciudad Universitaria México, 20, D F Tel: 5-50-52-15 Ext. 4606
7. SALVADOR ENCISO OROZCO San Bernardino No. 6 Col. del Valle México 12, D. F. Tel: 5-23-01-87	INTERNATIONAL ENTERPRISES ORGANIZATION,S.A. Analista Reforma 116-14o. Piso Col. Juárez México 6, D. F. Tel: 5-66-58-11

DIRECTORIO DE ASISTENTES AL CURSO INTRODUCCION A LOS MICROPROCESADORES Z-80
3,4 y 5 DE MARZO DE 1981

NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
15. JUAN ALBERTO MARQUEZ CAMPOS Tolvaneras No. 2 INFONAVIT Iztacalco México 8, D. F. Tel: 6-50-08-39	FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN Profesor Cuautitlán de Romero Rubio C.U.Norte Edo de México
16. ALEJANDRO JESUS MILLAN MONGE Av. División del Norte 2769 Coyoacan México 21, D. F. Tel: 5-49-21-61	ACEROS ECATEPEC, S.A. Jefe de Proyectos Km. 19.5 Carr. México-Laredo Tulpetlac, Edo. de México Tel: 5-69-32-00 Ext. 172
17. JESUS NUÑEZ VALADEZ Central Laboratorios No. 112 Col. Emilio Carranza México 2; D. F. Tel: 5-29-36-58	UNAM ENEP-ARAGON Jefe Laboratorio Area Electrica Av. Central y Rancho Seco s/n México 14, D F
18. FRANCISCO RAMIREZ GOMEZ. Axapusco No. 65 Col. Cumbria Cuautitlan, Izcallí, Edo de México Tel: 3-03-35	FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN Jefe de la Sección de Ingeniería Eléctrica Cuautitlán de Romero Rubio C.U.Norte Edo de México
19. JUAN PABLO RODRIGUEZ BARRON José Rodríguez González No. 9 Constitución de 1917 México 13, D. F. Tel: 6-81-08-48	DIVISION DE ESTUDIOS DE POSGRADO FACULTAD DE INGENIERIA, UNAM Asesoria de Servicios de Cómputo Ciudad Universitaria México 20, D. F. Tel: 5-50-52-15 Ext. 4486
20. SERGIO FRANCISCO RUIZ PALACIOS Av. Nueve No. 37-3 Col. Independencia México 13, D. F.	DIVISION DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERIA,UNAM Asesor de Servicios de Cómputo Ciudad Universitaria México 20, D. F. Tel: 5-50-52-15 Ext. 4486
21. FELIPE TOLEDO MIJANGOS Benito Fernández Arrieta No. 67 Los Cipreses México 21, D. F. Tel: 6-77-49-47	SECRETARIA DE ASENTAMIENTOS HUMANOS Y OBRAS PUBLICAS Jefe de Oficina Xola y Universidad Col. Narvarte México 12, D. F. Tel: 5-30-30-00 Ext. 471