DIRECTORIO DE PROFESORES DEL CURSO
LOS MICROPROCESADORES Y SUS APLICACIONES
DEL 14 DE AGOSTO AL 12 DE SEPTIEMBRE DE 1992.

ING. JOSE ANTONIO ARREDONDO GARZA
PROFESOR ASOCIADO A
DEPARTAMENTO DE CONTROL Y DEPARTAMENTO DE MECATRONICA
FACULTAD DE INGENIERIA DE LA UNAM
TEL. 593 10 71

ING. ABEL CLEMENTE REYES
JEFE DEL CENTRO DE SERVICIOS EDUCATIVOS DE LA
FACULTAD DE INGENIERIA
TEL. 550 52 15 EXT. 5715

ING. ALEJANDRO JIMENEZ HERNANDEZ
PROFESOR DE TIEMPO COMPLETO EN LA CAMARA DE INGENIERO EN
COMPUTACION,
MONROVIA 907, COL. PORTALES, DELEG. B. JUAREZ
TEL. 605 11 94

ING. RUBEN LIZARDI CERVERA
JEFE DEL DEPTO. DE TELEPROCESO DE LA SUBA. COMERCIAL DE
PETROLEOS MEXICANOS.
JEFE DEL CENTRO DE DISEÑO DE APLICACIONES PARA COMPUTADORA EN
LA FACULTAD DE INGENIERIA, UNAM.
TEL. 550 52 15 EXT. 3746 y 254 79 94

ING. ARTURO SALVA CALLEJA
PROFESOR DE CARRERA FACULTAD DE INGENIERIA UNAM
TEL. 539 07 79

INGENIERO JESUS SAVAGE CARMONA
ENCARGADO DEL LABORATORIO DE PROCESAMIENTO DE VOZ EN LA DEPFI
TEL. 549 87 91

ING. ALBERTO TEMPLOS CARBAJAL

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

14 DE AGOSTO AL 12 DE SEPTIEMBRE DE 1992

DIA	HORARIO	TEMA	PROFESORES
VIERNES 14 AGO.	17 A 21 H	INTRODUCCION ARQUITECTURA DE UN SISTEMA DE PROCESAMIENTO EN BASE A MICROPROCESADORES CRITERIOS DE SELECCION	ING. ABEL CLEMENTE REYES
SABADO 15 AGO.	09 A 14 H	INTRODUCCION ARQUITECTURA DE UN SISTEMA DE PROCESAMIENTO EN BASE A MICROPROCESADORES CRITERIOS DE SELECCION	ING. ABEL CLEMENTE REYES
VIERNES 21 AGO.	17 A 21 H	MICROCONTROLADORES	ING. ALEJANDRO JIMENEZ HDEZ.
SABADO 22 AGO.	09 A 14 H	MICROCONTROLADORES	ING. ALEJANDRO JIMENEZ HDEZ.
VIERNES 28 AGO.	17 A 21 H	SISTEMAS DE INSTRUMENTACION DIGITAL	ING. ANTONIO SALVA CALLEJA
SABADO 29 AGO.	09 A 14 H	SISTEMAS DE INSTRUMENTACION DIGITAL	ING. GLORIA CORREA PALACIOS ING. AGUSTIN SOTO URRUTIA
VIERNES 4 SEPT.	17 A 21 H	PROCESAMIENTO DIGITAL DE SEÑALES. APLICACIONES DIVERSAS	ING. JOSE ANTONIO ARREDONDO GARZA
SABADO 5 SEPT.	09 A 14 H	PROCESAMIENTO DIGITAL DE SEÑALES APLICACIONES DIVERSAS	ING. JOSE ANTONIO ARREDONDO GARZA
VIERNES 11 SEPT.	17 A 21 H	PROCESAMIENTO DIGITAL DE SEÑALES APLICACIONES DIVERSAS	ING. JOSE ANTONIO ARREDONDO GARZA
SABADO 12 SEPT.	09 A 14 H	COMPUTADORAS PERSONALES ORGANIZACION INTERNA	ING. ABEL CLEMENTE REYES

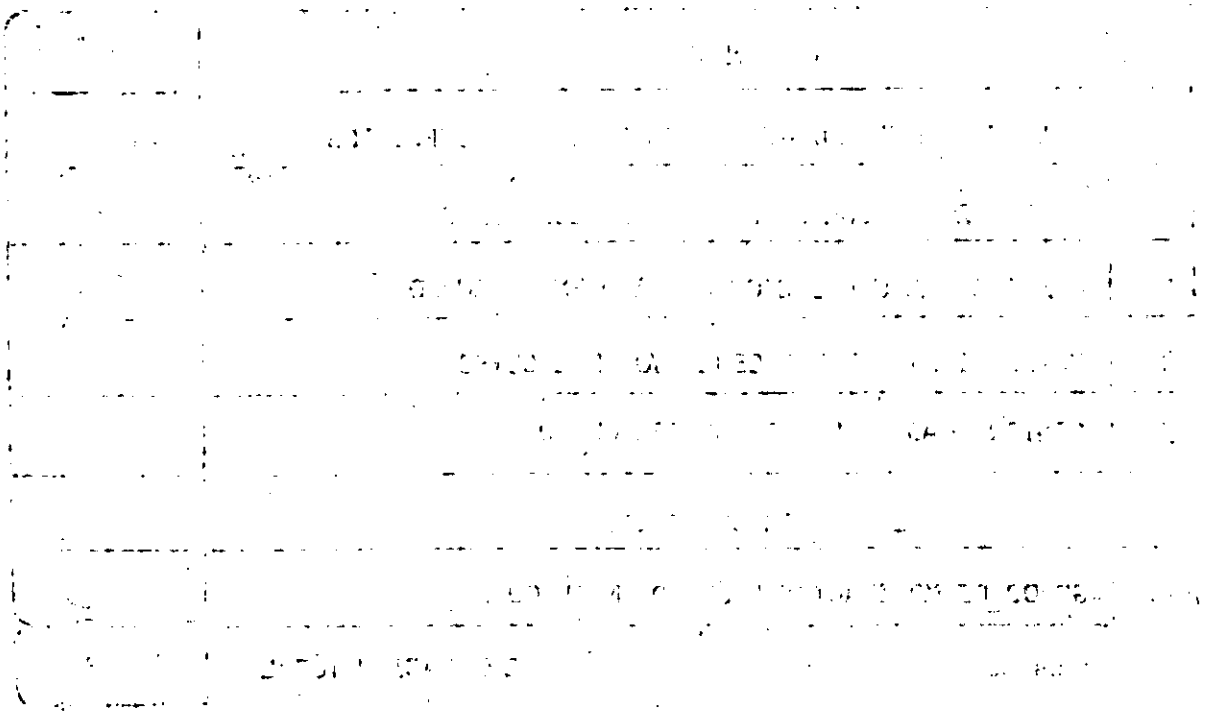
EVALUACION DEL PERSONAL DOCENTE

CURSO: LOS MICROPROCESADORES Y SUS APLICACIONES

FECHA: 14 DE AGOSTO AL 12 DE SEPTIEMBRE DE 1992

CONFERENCISTA	*DOMINIO DEL TEMA*	*EFICIENCIA EN EL USO DE AYUDAS AUDIOVISUALES*	*MANTENIMIENTO DEL INTERES (COMUNICACION CON LOS ASISTENTES, AMENIDAD, FACILIDAD DE EXPRESION)*	*PUNTUALIDAD*	
ING. ABEL CLEMENTE REYES					
ING. ALEJANDRO JIMENEZ HDEZ					
ING. ANTONIO SALVA CALLEJA					
ING. GLORIA CORREA PALACIOS					
ING. AGUSTIN SOTO URRUTIA					
ING. JOSE ANTONIO ARREDONDO GARZA					

ESCALA DE EVALUACION: 1 a 10



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

MUY AGRADABLE ( ) AGRADABLE ( ) DESAGRADABLE ( )

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

PERIODICO EXCELSIOR ANUNCIO TITULADO DIVISION DE EDUCACION CONTINUA ( )

PERIODICO NOVEDADES ANUNCIO TITULADO DIVISION DE EDUCACION CONTINUA ( )

FOLLETO DEL CURSO ( )

CARTEL MENSUAL ( )

RADIO UNIVERSIDAD ( )

COMUNICACION CARTA, TELEFONO, VERBAL, ETC. ( )

REVISTAS TECNICAS ( )

FOLLETO ANUAL ( )

CARTELERA UNAM "LOS UNIVERSITARIOS HOY" ( )

GACETA UNAM ( )

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

AUTOMOVIL PARTICULAR ( )

METRO ( )

OTRO MEDIO ( )

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

_____

_____

_____

5.- ¿Recomendaría el curso a otras personas? ( ) SI ( ) NO

5.a.¿Qué periódico lee con mayor frecuencia?

_____

_____

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

_____

_____

_____

7.- La coordinación académica fué:

EXCELENTE ☐ BUENA ☐ REGULAR ☐ MALA ☐

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

LUNES A VIERNES DE 9 a 13 H. Y DE 14 A 18 H. (CON COMIDAD) ☐

LUNES A VIERNES DE 17 a 21 H. ☐

LUNES A MIERCOLES Y VIERNES DE 18 A 21 H. ☐

MARTES Y JUEVES DE 18 A 21 H. ☐

VIERNES DE 17 A 21 H. SABADOS DE 9 A 14 H. ☐

VIERNES DE 17 A 21 H. SABADOS DE 9 A 13 H. DE 14 A 18 H. ☐

OTRO ☐

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

_____

_____

_____

10.- Otras sugerencias:

_____

_____

_____

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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

COMPUTADORAS PERSONALES. SU ORGANIZACION INTERNA

ING. ABEL CLEMENTE REYES

SEPTIEMBRE- OCTUBRE

1992.

AN INTRODUCTION TO PC ASSEMBLY-LANGUAGE PROGRAMMING

BY WILLIAM J. CLAFF

How to program the 8088

This is a guide for writing assembly-language programs or routines on the IBM PC. It is necessary to program in assembly language when an application must be as fast as possible or when the high-level language being used cannot perform the desired operation. High-level languages are also easier to comprehend when you understand assembly-language concepts. This article is not a primer. I am assuming that you have read a primer or manual and are left with unanswered questions. I hope this article will answer some of those questions and that, with one or more of the references listed at the end of the article, you will become a more polished assembly-language programmer. I will present specifics of the 8088 first, followed by a sample program and routine. The program is a useful utility illustrating DOS (disk operating system) function calls and string manipulation. The routine introduces a new technique for writing code to be called from BASIC.

I assume you have read a primer and are left with unanswered questions. I hope this article will answer some of those questions.

THE 8088 CENTRAL PROCESSING UNIT

Figure 1 shows an elementary block diagram of the 8088. This processor has two separate processing units: the execution unit (EU), which executes instructions, and the bus interface unit (BIU), which is responsible for the 8088's communication with the outside world. The BIU is capable of coordinating multiple EUs such as the 8087 numeric data processor (NDP) and the 8089 I/O processor. The only difference between the 8088 and the 8086 is their BIUs.

The BIU sits between the EU and the outside world. An EU provides a logical address to the BIU, which translates it into a physical address. This translation, called the physical-address computation, uses two 16-bit quantities: a segment register and an offset. The notation used for logical addresses is segment:offset. The segment registers (parts of the BIU) are code segment (CS), stack segment (SS), data segment (DS), and extra segment (ES). The offset is usually supplied by the EU.

The physical address is computed by shifting the segment register left 4 bits and adding the offset in the BIU's dedicated adder. Segments are 64K-byte relocatable pieces of the 1-megabyte physical-address space. They are located on 16-byte boundaries called paragraphs. Assembly-language programs are written in logical segments. Placement of these segments in memory is a function of the linker and the DOS. They can be overlapped, contiguous, or disjointed.

The address of the next instruction to be executed is CS:IP (code segment: instruction pointer). For increased efficiency, the BIU pipelines bytes (prefetches them and puts them into a queue). To facilitate this calculation, the offset instruction pointer is kept in the BIU.

The EU contains eight 16-bit registers, any of which can be used in computations. Four of these registers comprise the data group. They are the accumulator (AX), base (BX), count (CX), and data (DX) registers. The high and low 8 bits of each data register also can be accessed. The two halves of the accumulator register are AH (accumulator high) and AL (accumulator low). The halves of the base, count, and data

(continued)

William J. Claff (7 Roberts Rd., Wellesley, MA 02181) *is a member of the engineering department of Spinnaker Software Corporation, where he is the in-house expert on the IBM PC family. He also has run his own microcomputer retail and consulting business for the past six years. Mr. Claff holds a master's degree in applied mathematics from Harvard University.*

registers are similarly named.

Two other general registers, the source index (SI) and the destination index (DI), comprise the index group. These registers are used primarily in string operations. Two segment registers are required to perform moves or comparisons on memory more than 64K bytes apart. This is why there is an extra segment in addition to the data segment. The destination in a string operation is always ES:DI (extra segment: destination index).

The last two general registers, the stack pointer (SP) and the base pointer (BP), comprise the pointer group. These registers manipulate the stack that holds subroutine return information. When a subroutine is invoked, SS:SP (stack segment: stack pointer) is used to store the return address on the stack. Stack pointer points to the top of the stack and is automatically decremented by calls and incremented by returns. The stack also is used to pass subroutine parameters. The base-pointer register is used to access these parameters.

Many of the registers in the EU have special uses. Table 1 shows these registers and their uses.

ADDRESSING

An offset is also known as an effective address. The EU generates an effective address using one of several methods called addressing modes. An effective address has one or more of the following: base, index, and displacement. A base can be either a base register or base pointer; an index can be either a source index or destination index; and the displacement is a 16-bit signed number.

If no segment register is specified, the data-segment register will be used. If the base-pointer register is specified as the base, the stack-segment register is used. Supplying a segment register that is not the default is called a segment-override prefix. The segment cannot be overridden for an instruction pointer, stack pointer, or destination-index registers in string operations. Figure 2 shows how the various addressing modes in the EU and the BIU combine to form the physical address.

There are three types of addresses: Short, Near, and Far. Short addressing is used for looping, conditional jumps, and for some unconditional jumps. Near and Far addressing are used in calls and unconditional jumps that do not qualify for Short addressing. Short and Near addresses affect only the instruction-pointer register and are always relative. Far addresses affect code-segment and instruction-pointer registers and are absolute. When the value of the code-segment register does not change, you have intrasegment addressing. When

(continued)

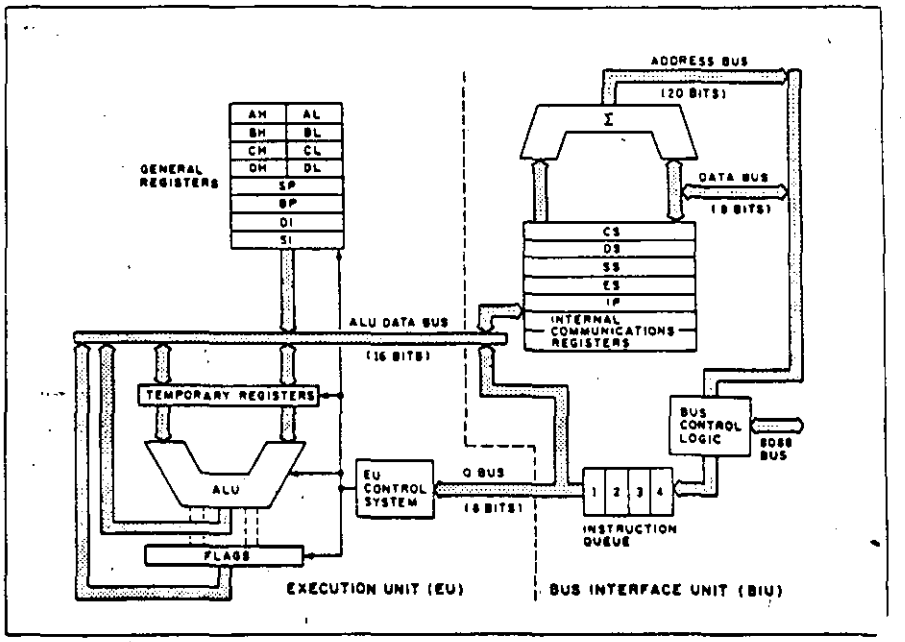


Figure 1: A *block diagram of the 8088's two processing units (see reference 3).*

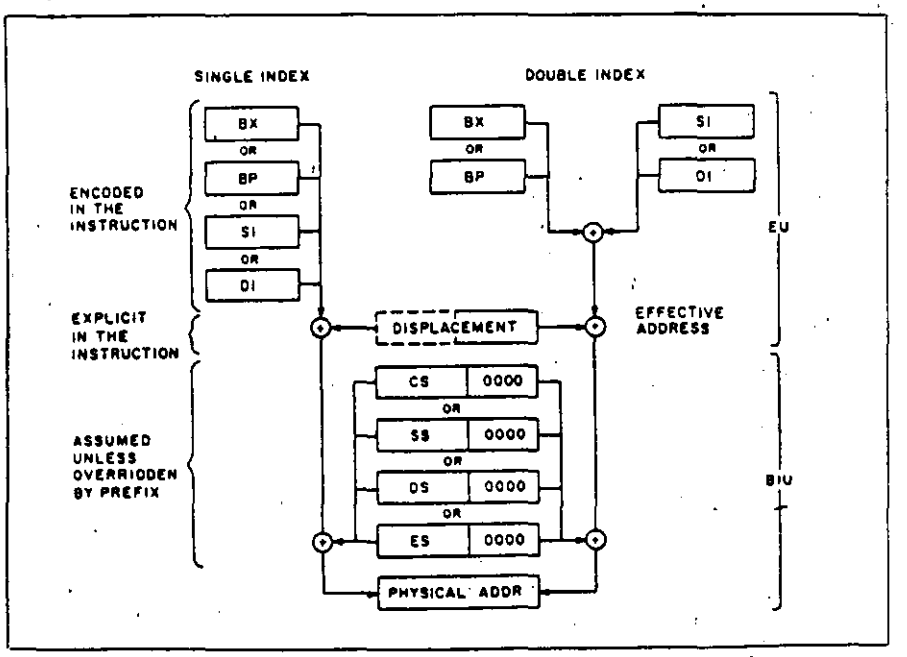


Figure 2: *Addressing modes in the execution unit and bus-interface unit combine to form the physical address (see reference 4).*

```
;--------GET DRIVE,FILENAME AND EXTENSION OF OUTPUT FILE
        MOV     CX,(SIZE DRIVE_NUMBER) + (SIZE FILE_NAME) + (SIZE
                FILE_EXTENSION)
        MOV     SI,OFFSET FORMATTED_AREA_1
        MOV     DI,OFFSET FCBOUT
        REP     MOVSB
;--------GET DRIVE,FILENAME AND EXTENSION OF SEARCH CRITERIA
        MOV     CX,(SIZE DRIVE_NUMBER) + (SIZE FILE_NAME) + (SIZE
                FILE_EXTENSION)
        MOV     SI,OFFSET FORMATTED_AREA_2
        MOV     DI,OFFSET FCBSEA
        REP     MOVSB
;--------DS = DSEG
        MOV     DS,CONDSEG
        ASSUME  DS:DSEG
;.........CREATE THE OUTPUT FILE
        MOV     AH,016H
        MOV     DX,OFFSET FCBOUT
        INT     021H
        OR      AL,AL
        JZ      CREATED
;--------FAILED
        MOV     AH,009H
        MOV     DX,OFFSET MSGCRE
        INT     021H
        JMP     EXIT
;--------SUCCESS
CREATED LABEL   NEAR
        MOV     FCBOUT.REC_SIZE,1
        MOV     FCBOUT.CURRENT_REC,0
        MOV     FCBOUT.RANDOM_REC_LO,0
        MOV     FCBOUT.RANDOM_REC_HI,0
;.........SEARCH FOR FILENAMES
NEXT    LABEL   NEAR
;--------SET DISK TRANSFER ADDRESS FOR RESULT OF SEARCH
        MOV     AH,01AH
        MOV     DX,OFFSET FCBRES
        INT     021H
;--------PERFORM THE SEARCH
        MOV     AH,VARFUN
        MOV     DX,OFFSET FCBSEA
        INT     021H
        OR      AL,AL
        JNZ     DONE
;.........BUILD OUTLINE FROM RESULT
;--------'%'
        MOV     DI,OFFSET OUTLINE
        MOV     AL,'%'
        STOSB
        MOV     AL,'1'
        STOSB
        MOV     AL,'
        STOSB
;--------DRIVE & ':'
        MOV     AL,FCBRES.DRIVE_NUMBER
        ADD     AL,'@'
        STOSB
        MOV     AL,':'
        STOSB
;.........FILENAME & '.'
;--------MOVE ENTIRE FILENAME
        MOV     CX,(SIZE FILE_NAME)
        MOV     SI,OFFSET FCBRES.FILE_NAME
        REP     MOVSB
;--------REMOVE TRAILING BLANKS
        CALL    TRAIL
;--------'.'
        MOV     AL,'.'
        STOSB
```

```
INTERRUPTS      SEGMENT AT 0H    ;This is where the keyboard interrupt
        ORG     9H*4             ;holds the address of its service routine
KEYBOARD_INT    LABEL   DWORD
INTERRUPTS      ENDS

SCREEN  SEGMENT AT 0B000H        ;A dummy segment to use as the
SCREEN  ENDS                     ;Extra Segment

ROM_BIOS_DATA   SEGMENT AT 40H   ;BIOS statuses held here, also keyboard buffer

        ORG     1AH
        HEAD DW     ?                    ;Unread chars go from Head to Tail
        TAIL DW     ?
        BUFFER      DW      16 DUP (?)           ;The buffer itself
        BUFFER_END  LABEL   WORD

ROM_BIOS_DATA   ENDS

CODE_SEG        SEGMENT
        ASSUME  CS:CODE_SEG
        ORG     100H             ;ORG = 100H to make this into a .COM file
FIRST:  JMP     LOAD_PAD         ;First time through jump to initialize routine

        CNTRL_N_FLAG    DW      0                ;Cntrl-N on or off
        PAD             DB      '_',499 DUP(' ')         ;Memory storage for pad
        PAD_CURSOR      DW      0                ;Current position in pad
        PAD_OFFSET      DW      0                ;Chooses 1st 250 bytes or 2nd
        FIRST_POSITION  DW      ?                ;Position of 1st char on screen
        ATTRIBUTE       DB      112              ;Pad Attribute -- reverse video
        SCREEN_SEG_OFFSET       DW      0        ;0 for mono, 8000H for graphics
        IO_CHAR         DW      ?                ;Holds addr of Put or Get_Char
        STATUS_PORT     DW      ?                ;Video controller status port
        OLD_KEYBOARD_INT.       DD      ?        ;Location of old kbd interrupt

N_PAD   PROC    NEAR             ;The keyboard interrupt will now come here.
        ASSUME  CS:CODE_SEG
        PUSH    AX               ;Save the used registers for good form
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    DI
        PUSH    SI
        PUSH    DS
        PUSH    ES
        PUSHF                    ;First, call old keyboard interrupt
        CALL    OLD_KEYBOARD_INT

        ASSUME  DS:ROM_BIOS_DATA         ;Examine the char just put in
        MOV     BX,ROM_BIOS_DATA
        MOV     DS,BX
```

(Figure 2 continues)

Figure 2: *Assembler code for NPAD.COM.*

```
        MOV     BX,TAIL                 ;Point to current tail
        CMP     BX,HEAD                 ;If at head, kbd int has deleted char
        JE      IN                      ;So leave
        SUB     BX,2                    ;Point to just read in character
        CMP     BX,OFFSET BUFFER        ;Did we undershoot buffer?
        JAE     NO_WRAP                 ;Nope
        MOV     BX,OFFSET BUFFER_END    ;Yes -- move to buffer top
NO_WRAP:MOV     DX,[BX]                 ;Char in DX now
        CMP     DX,310EH                ;Is the char a Cntrl-N?
        JNE     NOT_CNTRL_N             ;No
        MOV     TAIL,BX                 ;Yes -- delete it from buffer
        NOT     CNTRL_N_FLAG            ;Switch Modes
        CMP     CNTRL_N_FLAG,0          ;Cntrl-N off?
        JNE     CNTRL_N_ON              ;No, only other choice is on
CNTRL_N_OFF:
        MOV     ATTRIBUTE,7             ;Set up for normal video
        MOV     PAD_OFFSET,250          ;Point to 2nd half of pad
        LEA     AX,PUT_CHAR             ;Make IO call Put_Char as it scans
        MOV     IO_CHAR,AX              ;over all locations in pad on screen
        CALL    IO                      ;Restore screen
IN:     JMP     OUT                     ;Done
CNTRL_N_ON:
        MOV     PAD_OFFSET,250          ;Point to screen stroage part of pad
        LEA     AX,GET_CHAR             ;Make IO use Get_char so current screen
        MOV     IO_CHAR,AX              ;is stored
        CALL    IO                      ;Store Screen
        CALL    DISPLAY                 ;And put up the pad
        JMP     OUT                     ;Done here.
NOT_CNTRL_N:
        TEST    CNTRL_N_FLAG,1          ;Is Cntrl-N on?
        JZ      IN                      ;No -- leave
        MOV     TAIL,BX                 ;Yes, delete this char from buffer
        CMP     DX,5300H                ;Decide what to do -- is it a Delete?
        JNE     RUBOUT_TEST             ;No -- try Rubout
        MOV     BX,249                  ;Yes -- fill pad with spaces
DEL_LOOP:
        MOV     PAD[BX],' '             ;Move space to current pad position
        DEC     BX                      ;and go back one
        JNZ     DEL_LOOP                ;until done.
        MOV     PAD,'_'                 ;Put the cursor at the beginning
        MOV     PAD_CURSOR,0            ;And start cursor over
        CMP     BX,0                    ;Are we at beginning?
        JLE     NEVER_MIND              ;Yes -- can't rubout past beginning
        MOV     PAD[BX],' '             ;No -- move space to current position
        MOV     PAD[BX-1],'_'           ;And move cursor back one
        DEC     PAD_CURSOR              ;Set the pad location straight
NEVER_MIND:
        CALL    DISPLAY                 ;And put the result on the screen
        JMP     OUT                     ;Done here.
CRLF_TEST:
        CMP     DX,1C0DH                ;Is it a carriage return-line feed?
        JNE     CHAR_TEST               ;No -- put it in the pad
```

(Figure 2 continues)

```
        CALL    DISPLAY                 ;Put up the new pad on screen
        JMP     OUT                     ;And take our leave
RUBOUT_TEST:
        CMP     DX,0E08H                ;Is it a Rubout?
        JNE     CRLF_TEST               ;No -- try carriage return-line feed
        MOV     BX,PAD_CURSOR           ;Yes -- get current pad location
        CALL    CRLF                    ;Yes -- move to next line
        CALL    DISPLAY                 ;And display result on screen
        JMP     OUT                     ;Done.
CHAR_TEST:
        MOV     BX,PAD_CURSOR           ;Get current pad location
        CMP     BX,249                  ;Are we past the end of the pad?
        JGE     PAST_END                ;Yes -- throw away char
        MOV     PAD[BX],DL              ;No -- move ASCII code into pad
        MOV     PAD[BX+1],'_'           ;Advance cursor
        INC     PAD_CURSOR              ;Increment pad location
        PAST_END:
        CALL    DISPLAY                 ;Put result on screen
OUT:    POP     ES      ;Having done Pushes, here are the Pops
        POP     DS
        POP     SI
        POP     DI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET                    ;An interrupt needs an IRET
N_PAD   ENDP

DISPLAY PROC    NEAR                    ;Puts the whole pad on the screen
        PUSH    AX
        MOV     ATTRIBUTE,112           ;Use reverse video
        MOV     PAD_OFFSET,0            ;Use 1st 250 bytes of pad memory
        LEA     AX,PUT_CHAR             ;Make IO use Put-Char so it does
        MOV     IO_CHAR,AX
        CALL    IO                      ;Put result on screen
        POP     AX
        RET                             ;Leave
DISPLAY ENDP

CRLF    PROC    NEAR                    ;This handles carriage returns
        CMP     PAD_CURSOR,225          ;Are we on last line?
        JGE     DONE                    ;Yes, can't do a carriage return, exit
NEXT_CHAR:
        MOV     BX,PAD_CURSOR           ;Get pad location
        MOV     AX,BX                   ;Get another copy for destructive tests
EDGE_TEST:
        CMP     AX,24                   ;Are we at the edge of the pad display?
        JE      AT_EDGE                 ;Yes -- fill pad with new cursor
        JL      ADD_SPACE               ;No -- Advance another space
```

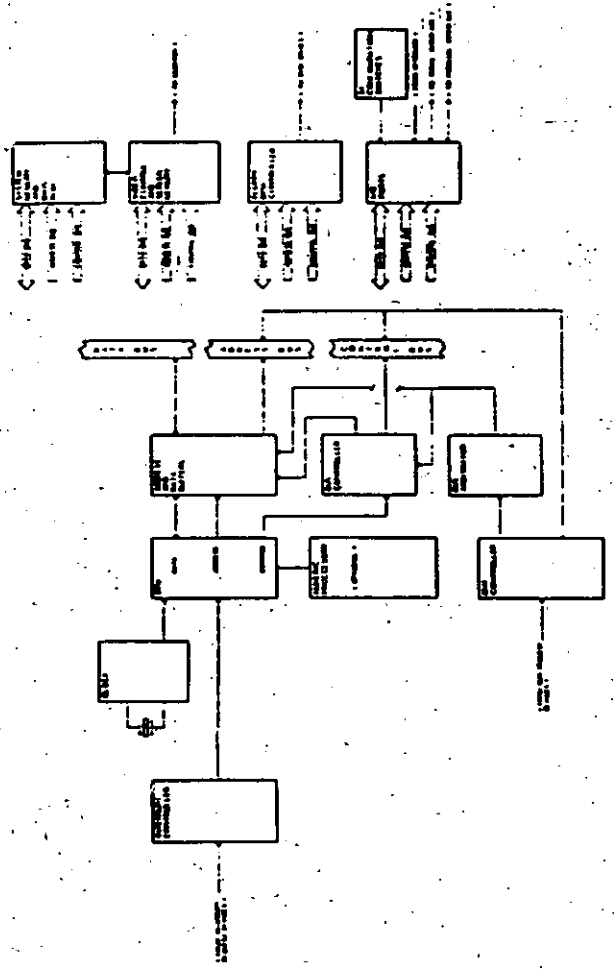
(Figure 2 continues)

```
        SUB     AX,25                   ;Subtract another line-width
        JMP     EDGE_TEST               ;Check if at edge now
ADD_SPACE:
        MOV     PAD[BX],' '             ;Add a space
        INC     PAD_CURSOR              ;Update pad location
        JMP     NEXT_CHAR               ;Check if at edge now
AT_EDGE:
        MOV     PAD[BX+1],'_'           ;Put cursor in next location
        INC     PAD_CURSOR              ;Update pad location to new cursor
DONE:   RET                             ;And out..
CRLF    ENDP

GET_CHAR        PROC    NEAR    ;Gets a char from screen and advances position
        PUSH    DX
        MOV     SI,2            ;Loop twice, once for char, once for attribute
        MOV     DX,STATUS_PORT  ;Get ready to read video controller status
G_WAIT_LOW:                     ;Start waiting for a new horizontal scan -
        IN      AL,DX           ;Make sure the video controller scan status
        TEST    AL,1            ;is low
        JNZ     G_WAIT_LOW
G_WAIT_HIGH:                    ;After port has gone low, it must go high
        IN      AL,DX           ;before it is safe to read directly from
        TEST    AL,1            ;the screen buffer in memory
        JZ      G_WAIT_HIGH
        MOV     AH,ES:[DI]      ;Do the move from the screen, one byte at a time
        INC     DI              ;Move to next screen location
        DEC     SI              ;Decrement loop counter
        CMP     SI,0            ;Are we done?
        JE      LEAVE           ;Yes
        MOV     PAD[BX],AH      ;No -- put char we got into the pad
        JMP     G_WAIT_LOW      ;Do it again
LEAVE:  INC     BX              ;Update pad location
        POP     DX
        RET
GET_CHAR        ENDP

PUT_CHAR        PROC    NEAR    ;Puts one char on screen and advances position
        PUSH    DX
        MOV     AH,PAD[BX]      ;Get the char to be put onto the screen
        MOV     SI,2            ;Loop twice, once for char, once for attribute
        MOV     DX,STATUS_PORT  ;Get ready to read video controller status
P_WAIT_LOW:                     ;Start waiting for a new horizontal scan -
        IN      AL,DX           ;Make sure the video controller scan status
        TEST    AL,1            ;is low
        JNZ     P_WAIT_LOW
P_WAIT_HIGH:                    ;After port has gone low, it must go high
        IN      AL,DX           ;before it is safe to write directly to
        TEST    AL,1            ;the screen buffer in memory
        JZ      P_WAIT_HIGH
        MOV     ES:[DI],AH      ;Move to screen, one byte at a time
        MOV     AH,ATTRIBUTE    ;Load attribute byte for second pass
        INC     DI              ;Point to next screen postion
        DEC     SI              ;Decrement loop counter
        JNZ     P_WAIT_LOW      ;If not zero, do it one more time
        INC     BX              ;Point to next char in pad
        POP     DX
        RET                     ;Exeunt
PUT_CHAR        ENDP
```

(Figure 2 continues)



CENTRAL PROCESSING UNIT (CPU) AND CLOCK GENERATOR (Schematic 1)

The IBM PC-compatible computer uses an Intel 8088 microprocessor (U111) as its CPU. The 8088 is a 16-bit microprocessor with an 8-bit data bus. It has 16-bit internal architecture, and its 20 address lines make it possible to address one megabyte of memory directly. The design of the CPU thus makes it possible to achieve 16-bit hardware speeds while maintaining an economical package design.

In this computer, the 8088 is operated in the maximum mode. In this mode, the CPU can share its control functions with an external bus controller (8288) and an optional co-processor (8087). This configuration provides the flexibility needed for implementing extended large system features, such as high resolution graphics and fast computations.

Communication with the bus controller and bus arbitration logic is accomplished by means of the CPU status signals S0, S1, S2, and LOCK. S0, S1, and S2 provide coded information that is used by the bus controller to generate the control signals. This coded information is also used by the bus controller to monitor the CPU status. (The bus controller generates the interrupt acknowledge signal, INTA, that goes to the interrupt controller when an interrupt request has been made.)

The LOCK signal prevents control of the system buses from being transferred from one component to another at the incorrect time.

Timing for the CPU comes from the 8284A clock generator (U128). This chip produces the clock frequency (CLK) needed for proper operation of the 8088 and 8087. (This is one-third of the crystal frequency, or 4.77 MHz.) It also provides an output (PCLK) for the programmable interval timer (on Schematic 5). This signal, whose frequency is one-half that of the CLK signal, is used for timing peripherals.

In addition, the clock generator produces the RESET signals that are used to stop and restart the processor and to initialize the logic throughout the PCB.

Switch S2 is used to manually force a system RESET to occur. When this switch is pressed, all current processing activity stops and the system reboots.

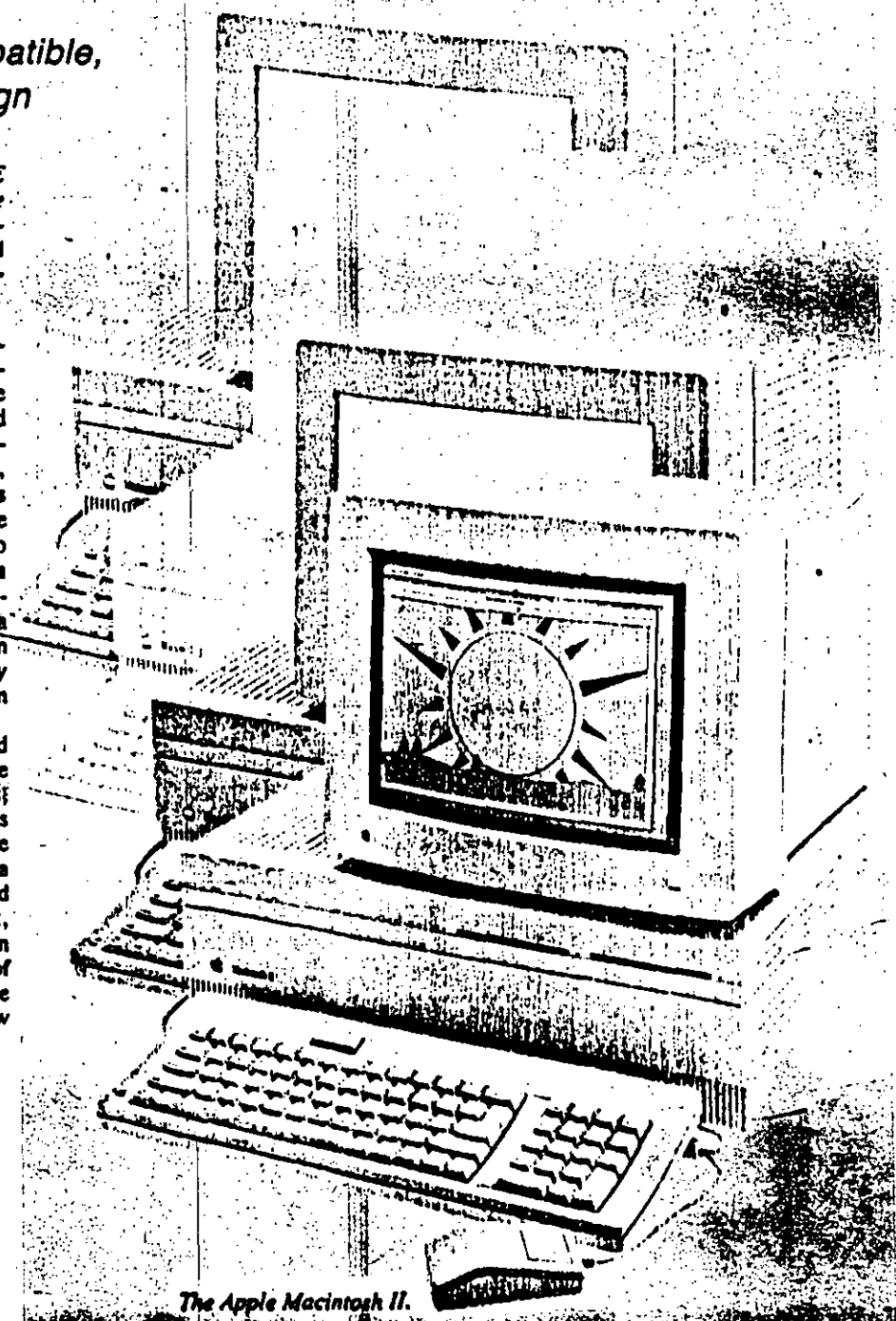
The Apple Macintosh II

The Mac II's improvements include hardware slots, color, speed, and a compatible, open-ended system design

Editor's note: *The following is a BYTE product preview. It is not a review. We provide an advance look at this new product because we feel it is significant. A complete review will follow in a subsequent issue.*

Innovation and compatibility don't always go together easily. Some companies, when asked to improve their computer, go for a bigger-is-better approach, like the man who tried to breed his horses larger so they could pull a carriage faster—the strategy works, but only so far. Other companies, like Apple, take a think-then-act approach that looks beneath the surface of a problem to deliver a broader, more fundamental answer—in contrast to the horse breeder above, they retain the carriage but power it with an automobile engine.

Apple has combined innovation and compatibility in the Macintosh II, the Mac with color and peripheral-card slots; and it has been worth the wait. Apple has added the roman numeral "II" in homage to the Apple II, a product that has had a supernaturally long life span so far, and Apple's action in doing this is one that, for once, contains more substance than hyperbole. The original Mac's lack of slots stunted its growth and forced Apple to expand the machine by offering new



The Apple Macintosh II.

models. With the Mac II, Apple—and, more important, third-party developers—can expand the machine radically without forcing you to buy a new computer. One thing is obvious: This is the design on which Apple plans to build its Macintosh empire.

About the only valid complaint that comes to mind—its lack of multitasking—will probably be remedied once Motorola's 68851 memory management chip becomes available. Even its under-$5000 price is defensible. As is the case with many other new computers, you are buying it partially for its *potential*—but never before have we seen a computer in which that surcharge is so reasonable.

System Description

Here are the most important features of the Macintosh II (see also the system block diagram in figure 1 and the circuit boards in photo 1):

• *68020 and 68881 processing power*: The Mac II comes with a Motorola 68020 processor running at 15.6672 megahertz and a 68881 floating-point coprocessor. The inclusion of the latter chip as standard gives system software and *any* application access to hardware-assisted number crunching and the speed boost that comes with it. Existing applications that use SANE (standard Apple numerics environment) run 3 to 30 times faster automatically, but applications that directly access the 68881 will be 30 to 200 times faster.

• *Six NuBus slots*: These six slots will let you extend the Mac II's hardware with coprocessors, LAN cards, and other add-ins. NuBus is a 96-pin card used until now in minicomputers and adapted for microcomputer use by Apple. Any card can become the "master," and the machine can be configured to start from any card. Because the 68020 motherboard acts like a NuBus card, it is possible for an add-in NuBus card to "take over" the system.

• *Growth within the same footprint*: The Mac II box, about the size of an IBM PC AT, has room inside it for the options most people want. The Mac II comes with 1 megabyte of memory (expandable to 8 megabytes on-board and up to 2 gigabytes using NuBus slots), one 800K-byte 3½-inch floppy disk drive, two Mini-8 serial (RS-232/RS-422) ports, a DB-25 SCSI hard disk interface, and two Apple Desktop Bus (ADB) connectors (for mouse and keyboard). The box also contains room for a second floppy disk drive; a 20-, 40-, or 80-megabyte internal hard disk; and six NuBus expansion cards. All this can be added without increasing the amount of space the Mac II takes on your desk.

• *Backward compatibility*: The Mac II supports most existing monochrome Mac programs (more than 95 percent of them) and the few programs that use QuickDraw's fixed-color capability. It does this because of the similarity of the 68000 processor (in the old Mac) and the 68020 (in the Mac II) and the heavy use in both machines' software of high-level libraries that let the same software run on vastly different machines.

• *Color support*: The Mac II supports color through Color QuickDraw and various other extensions to the Mac II Toolbox. Application programs manipulate 48-bit "absolute" colors, then translate them to the nearest approximations available through the attached video or printer cards.

• *No standard video output*: Though this sounds strange at first, it's really an advantage. All Mac II hardware and both old and new Mac software are designed to work with any Mac II video card, *present or future*. (Your video card will take up one NuBus slot.) Because you will be able to add *any* card and use it with *all* your software, hardware designers are

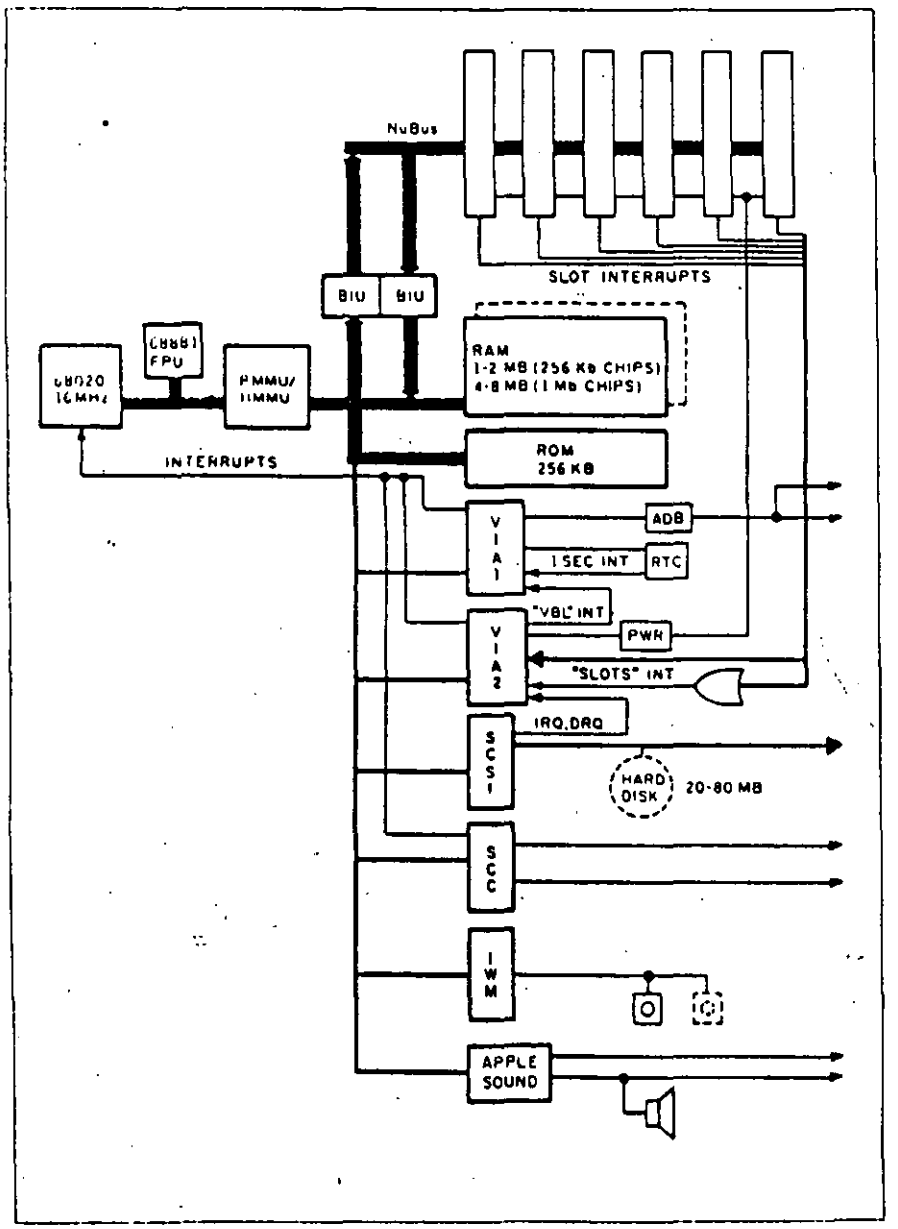


Figure 1: *A block diagram of the Macintosh II. The dashed components are optional.*

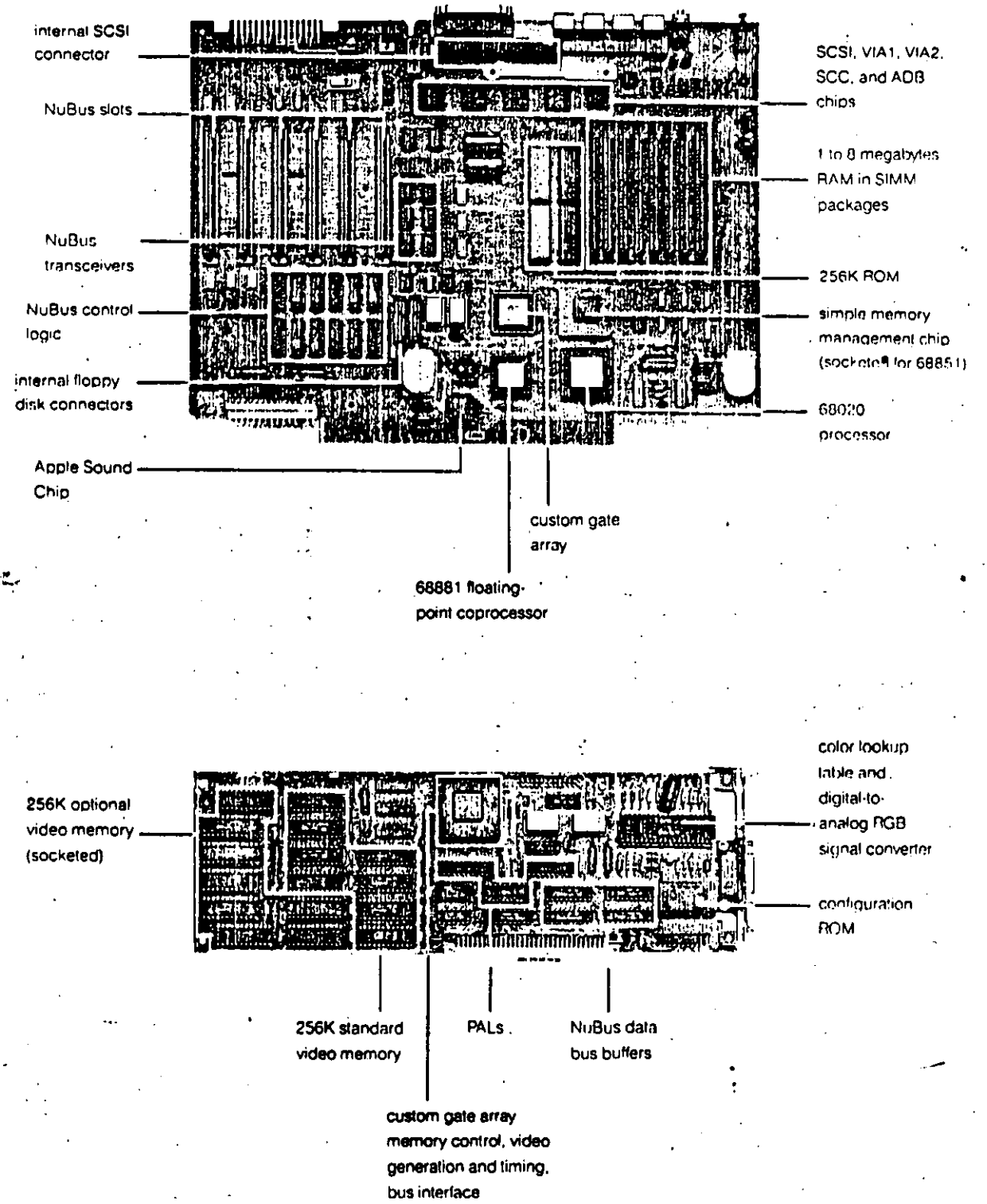


Photo 1: *The motherboard (top) and the NuBus Graphics Card (bottom).*

more likely to build custom video cards—which means you will eventually have numerous video output options from which to choose.

• *640- by 480-pixel video*: Apple has currently announced only one video board that will drive either a color or monochrome board at the 640 by 480 resolution. The standard board has 256K bytes of memory and displays up to 16 colors (or shades of gray) on the screen at one time, from a palette of more than 16 million colors. By adding an extra 256K bytes of memory to the board, you can increase this to 256 on-screen colors (or shades) at one time.

• *Multiple-screen desktop*: One consequence of the video design is that multiple video displays can be combined to create a "desktop" that spans two or more monitors. Even combined color and monochrome displays draw their contents correctly, and a window can span multiple displays.

• *MS-DOS and UNIX capability*: Apple says that a third-party company will offer an 80286 coprocessor card. Apple is also working on a version of UNIX, but that will have to wait for the availability of the Motorola 68851 memory management chip.

• *Sound support in hardware*: The Mac II contains a custom Apple Sound Chip (ASC) that replaces the old Macintosh sound-generating software with hardware and adds stereo capability, four-voice synthesized sound, and arbitrary sound sampled at up to 44.1 kilohertz. (Sound output is limited to 7.5 kHz, up from 5 kHz on old Macs.) Because these functions are now in hardware, sound can be used freely in applications without noticeable system degradation.

• *Other enhancements*: The Toolbox ROM, containing Color QuickDraw and other libraries of system software, is now 256K bytes long. An improved TextEdit allows the retention of text attributes like color, style, font, vertical spacing, and size during cut-and-paste operations. Use of the ADB frees the processor from much of the routine servicing of the keyboard and mouse and makes the Mac II more accessible to future input peripherals. SCSI data transfers are now faster because of hardware handshaking. Apple also offers detachable 81- and 105-key keyboards.

NuBus

NuBus is a 32-bit high-performance bus that emphasizes independence of any particular system architecture and a simple yet sophisticated transaction protocol. A card's address space is determined by the slot it occupies on the bus, and a "strictly fair" arbitration protocol allows every card a chance at bus access. Multiprocessing is possible by allowing multiple bus-master cards and restricting access to shared resources through bus locking. We'll take a closer look at each of these characteristics in turn to see how the NuBus design accomplishes this.

NuBus is a multiplexed bus (i.e., address information and data share the same lines at different intervals) operating synchronously at 10 MHz. The bus reads and writes data to a 32-bit address space of 4 gigabytes. Bus addressing is accomplished by driving all 32 bits of the address onto the multiplexed lines. Data transfers can be 8-, 16-, or 32-bit quantities, a facility that complements the dynamic bus-sizing capabilities of the 68020 processor. NuBus explicitly defines data sizes of a byte (8 bits), halfword (16 bits), and word (32 bits) and their addressing relationship. (When a halfword is broken into 2 bytes, the most significant byte is in the lower memory address. Words are broken into halfwords similarly.) These sizes will be referred to as NuBus byte, NuBus halfword, and NuBus word when it is necessary to distinguish between a NuBus quantity and a 68020 quantity. NuBus also defines block transfers of 2, 4, 8, or 16 NuBus words. However, we won't cover this capability since it's not used by the Mac II.

Two important points must be made about the NuBus address space. First, all addressable resources occupy a single address space whether it's a memory chip or a control register. There isn't an "I/O space" or "CPU space" or other entity requiring additional control signals and logic. Second, since the addressing relationship of data sizes has been defined from the bus's point of view, you know precisely where byte *x* lies on NuBus. This provides a common ground where processors that address bytes differently can share information. If a processor's bus-interface circuitry is wired so that reading or writing a byte corresponds to reading or writing a NuBus byte, dissimilar processors can share data through NuBus byte transfers.

Each slot on the bus is hard-wired with a unique 4-bit ID number that identifies it to a peripheral card inserted into the slot and limits NuBus to a maximum of 16 slots. This ID serves to set the address range that the card will respond to and also figures in the arbitration scheme, which will be described later.

The upper sixteenth of the 4-gigabyte address space (256 megabytes) is termed *slot space*. This slot space is partitioned into 16 regions of 16 megabytes each. Slot addresses are of the form FSxxxxxx, where S (bits 27 through 24) is assigned by the slot ID. This assignment of a fixed address space based on a card's position on the bus is called *geographic addressing*. No jumpers or DIP switches are required to configure a card into the system since a card's address range is determined by the ID of the slot it occupies. The remaining portion of the NuBus address space is unreserved and can be allocated to devices as needed.

The NuBus specification makes two requirements of a card on the bus. First, the card must respond with the appropriate control signals to reads of the NuBus word located at the top of its allocated slot space (address FSFFFFFC). This is required to indicate that the bus slot is occupied. Accesses to an unoccupied slot will be handled by a bus time-out mechanism. Second, a card must have a configuration ROM located at the top of its slot space. The purpose of this ROM is not defined by the NuBus specification. The presence of a configuration ROM does happen to satisfy the first NuBus card requirement: indicating slot occupancy.

NuBus also specifies the physical dimensions, or form-factor, of a card. Two types of cards are defined: a triple-height form-factor and a PC-style form-factor. The PC form-factor is defined for microcomputer use and describes a 4- by 13-inch card that uses a 96-pin Eurocard type C connector.

NuBus Lines

NuBus is composed of 96 lines: 51 signal lines and 45 power and ground lines. All signals are active low except for the address/data lines that use tristate drivers. The signal lines can be divided into three types: Utility, Bus Data Transaction, and Arbitration System Signals. All signal names ending in an asterisk are active low.

The power lines supply voltages of +5 V, −5.2 V, +12 V, and −12 V for every card on the bus. The Utility lines carry signals that are supplied to the backplane by the computer system. Some of these signals are the Clock (CLK*), Power Fail Warning (PFW*), Card Slot Identification (ID3*-ID0*), and Reset (RESET*). The Bus Data Transaction lines handle addressing and data (AD31*-AD0*), parity signals (SPV* and SP*), and two lines that manage the start and end of a data transfer (START* and ACK*). The Arbitration System Signals handle the arbitration of several cards contending for ownership of the bus. The Arbitration Signals (ARB3*-ARB0*) are used to determine the next bus master, and Bus Re-

continued

quest (RQST*) is used to indicate that a card wants bus ownership.

To use NuBus, a card normally obtains ownership of the bus. It accomplishes this by requesting the bus and waits until this request is granted. A card that owns the bus can initiate a data transfer and is called a *master*. A card becomes a *slave* when it is addressed by a master and responds to the data transfer. A read or write between a master and a slave card begins with a START* cycle, followed by multiple bus cycles to address and transfer the data, and ends with an ACK* cycle. Such a data transfer is called a *transaction*. *Tenure* is the period of time a card continuously owns the bus. The NuBus specification does not require a card to become a master. A special line, the Non-Master Request (NMRQ*), allows this type of card to signal a need for service.

Two of the Bus Data Transaction lines, termed Transfer Mode, serve double duty during a NuBus transaction. At the start of a transaction, these two lines (TM1* and TM0*) carry a transaction code that indicates the type of transfer (read or write and data size) taking place. At the end of a transaction, they carry a response status code that indicates whether the data transfer was successful. See tables 1 and 2 for more information on the transaction codes and their response codes. See figures 2 and 3 for a detailed look at complete NuBus read and write transactions.

NuBus also defines an Event Transaction, which is a special form of a write transaction. Its purpose is to post interrupts to a slave card. The Mac II does not use Event Transactions, so they will not be discussed further.

Table 1: *NuBus Transfer Mode signals.*

TM1*	TM0*	AD1*	AD0*	Type of Cycle
0	0	0	0	Write byte 3
0	0	0	1	Write byte 2
0	0	1	0	Write byte 1
0	0	1	1	Write byte 0
0	1	0	0	Write halfword 1
0	1	0	1	Block write
0	1	1	0	Write halfword 0
0	1	1	1	Write word
1	0	0	0	Read byte 3
1	0	0	1	Read byte 2
1	0	1	0	Read byte 1
1	0	1	1	Read byte 0
1	1	0	0	Read halfword 1
1	1	0	1	Block read
1	1	1	0	Read halfword 0
1	1	1	1	Read word

Table 2: *NuBus Transaction Response signals.*

TM1*	TM0*	Type of Acknowledge	Comment
0	0	Bus Transfer Complete	The transaction was successful.
0	1	Error	During a read, the data may be corrupted. During a write, the transaction may not have completed successfully.
1	0	Bus Timeout Error	Slave failed to respond in 256 cycles. The bus time-out logic has generated an ACK* to terminate the transaction.
1	1	Try Again Later	Slave cannot complete transaction at this time. The slave may be able to complete the transaction at a future request.

Table 3: *NuBus Attention Cycle signals.*

TM1*	TM0*	Type of Attention Cycle	Comment
0	0	Attention-Null	Used to reinitiate arbitration or end a locked-resource transaction.
0	1	reserved	
1	0	Attention-Resource-Lock	Start of a locked-resource transaction.
1	1	reserved	

NuBus Arbitration

When many cards are on the bus, it's possible that two or more of them may request bus ownership on the same clock cycle. NuBus provides distributed arbitration logic (so called because the components implementing the arbitration mechanism are present on every card) to handle this situation. The arbitration protocol is called "fair" because cards attempting to own the bus at the same moment will eventually obtain access to the bus and obtain access before any of the competing cards get access a second time. Because there isn't a special priority scheme embedded in the NuBus arbitration logic, it is said to be "strictly fair." The NuBus design avoids a preemptive or priority arbitration protocol that can produce conditions where higher-priority cards continue to own the bus and "starve" a lower-priority card's access to the bus.

A card requests use of the bus by asserting the Bus Request line (RQST*). It will not assert RQST*, however, if this line was asserted on the last clock cycle. If the card is able to assert RQST*, it will continue to do so until it gains ownership of the bus and begins a transaction by asserting START*. Once it has asserted RQST*, a card drives its slot ID onto the arbitration lines ARB3*-ARB0*. The card will unassert these lines if it finds higher IDs present. This results in the arbitration lines holding the ID of the highest-numbered card competing for the bus. The ID present on these lines indicates the next bus master. As you can see, when several cards request the bus on the same clock cycle, the arbitration contest will be won by the card with the highest slot ID.

he winning card has access to the bus mediately if the bus is not busy, or at completion of a transaction if the bus busy. Once this card owns the bus, the maining cards again contend for the bus d undergo the arbitration contest to se- ct the next bus master. This process re- ats until all the cards that requested the s have been granted bus access. Fair- ess is implemented by the requirement at a card can request the bus only if QST* is not already asserted. Other ards will be blocked from competing for e bus until all the cards that requested e bus simultaneously on a previous lock cycle have owned the bus.

A bus master may continue to own the us as long as RQST* is unasserted (that , no other card is requesting use of the us). The master is said to be *parked* on e bus and can continue to use the bus ithout undergoing an arbitration con- st. Bus parking reduces the time nor- ally required to gain access to the bus in computer system with few active cards. lowever, once RQST* is asserted, the us master won't start another transac- ion, and a new arbitration contest egins.

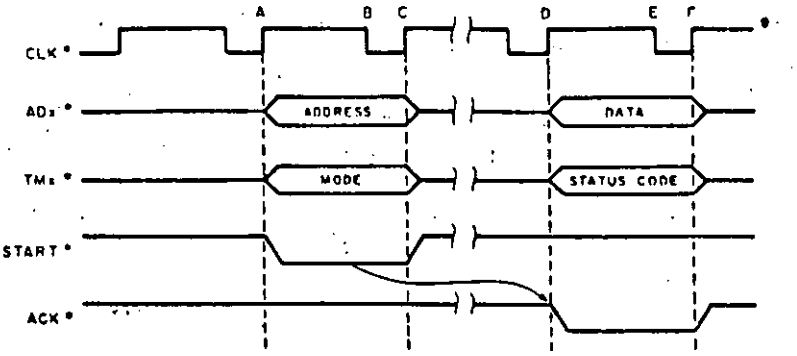
Multiprocessing on NuBus

The NuBus specification also lets a mas- r lock the bus. This is necessary for cer- in operations that must be allowed to mplete in a multiprocessor environ- ent (e.g., a test-and-set operation on a maphore).

The NuBus specification defines two pes of locking: bus and resource. Bus cking is used by a master to ensure an nbroken bus tenure. A master can also ck the bus to gain performance for a rge data transfer involving many bus ansactions, although this is not recom- ended. Resource locking is used to in- rm a slave card to lock out all local ac- ess routes on the card to a resource being ddressed by NuBus. For example, a ultiprocessor card might have dual- orted RAM that the processor could ac- ess during a NuBus transaction on the me RAM. Resource locking informs e card to lock out the local CPU port hile a locked NuBus transaction is in progress. More than one card can be cked during a resource lock. Note that source locking accompanies bus lock- g; that is, a continuous bus tenure oc- rs during a resource lock.

Bus locking occurs when the master ntinues to assert RQST*. The master, aving won the arbitration contest previ- sly, is still the highest ID card in the mpetition and thus continues to own e bus.

Resource locking requires the master



NuBus Read Transaction

(A) The master begins a transaction by asserting START*. The master drives lines AD31*-AD0* with the target read address and lines TM1*-TM0* with the proper transfer mode information. The master ensures that ACK* is unasserted.

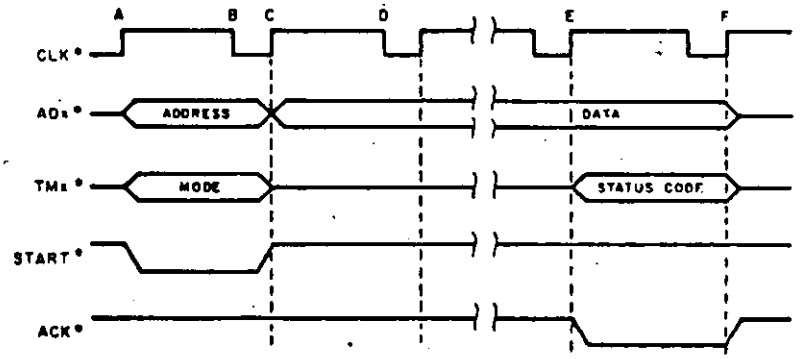
(B) All bus cards sample the AD31*-AD0* and TM1*-TM0* lines.

(C) The master stops driving AD31*-AD0*, TM1*-TM0*, ACK*, and unasserts START*. The master waits for an ACK*.

(D) The slave card (determined by the target address) drives the desired data onto lines AD31*-AD0*. The slave also drives TM1*-TM0* with the proper transaction response status. The slave asserts ACK*, signaling the end of the transaction.

(E) The master samples the lines AD31*-AD0*, TM1*-TM0*, and ACK*. The master drives ACK* to the appropriate state, possibly the start of a new transaction.

Figure 2: *Timing diagram for a NuBus read transaction.*



NuBus Write Transaction

(A) The master begins a transaction by asserting START*. The master drives lines AD31*-AD0* with the target write address and lines TM1*-TM0* with the proper transfer mode information. The master drives bus ACK* unasserted.

(B) All bus cards sample the AD31*-AD0* and TM1*-TM0* lines.

(C) The master drives the data onto the AD31*-AD0* lines and releases the TM1*-TM0* and ACK* lines. The master unasserts START* and looks for an ACK*.

(D) The slave card (determined by the target address) samples the AD31*-AD0* and TM1*-TM0* lines, receiving the data.

(E) The slave drives the proper transaction response status onto TM1*-TM0* ACK* is asserted, signaling the end of the transaction.

(F) The master releases the AD31*-AD0* lines, and the slave releases ACK* and the TM1*-TM0* lines. The master drives ACK* to the appropriate state, possibly the start of a new transaction.

Figure 3: *Timing diagram for a NuBus write transaction.*

to issue certain signals to inform cards on the bus that a locked transaction is occurring. The master begins the lock by issuing an Attention-Bus-Lock cycle. An Attention cycle is generated by asserting both START* *and* ACK* at the beginning of a bus transaction. The master also drives an Attention-Resource-Lock code onto the Transfer Mode lines (see table 3). At the end of the locked transaction, the master issues an Attention-Null cycle (START* and ACK* asserted with the corresponding code on the Transfer Mode lines) to signal the end of the bus lock. All cards with lockable resources sample the bus for an Attention-Bus-Lock cycle and note it. If a card happens to be addressed by the master during this interval and before an Attention-Null cycle is issued, it will lock its resources. NuBus does not require a card to lock its local resources, but its use in a multiprocessor environment won't be reliable if it doesn't.

Apple NuBus

The Macintosh II comes equipped with six NuBus slots. These slots are hardwired with IDs from 9 to 14 (9 to E hexadecimal), and each follows the PC form-factor as described in the NuBus specification. The motherboard is treated as slot 0, and ID 15 (which has no corresponding physical slot) is reserved. One of the slots will be occupied with a video card.

Apple has worked with the IEEE NuBus proposal group and has followed the specification closely. Not all of the NuBus features are supported, however. Apple calls this subset of the NuBus specification Apple NuBus, which differs from the specification in one area: Apple NuBus does not supply −5.2-V power to the NuBus backplane.

Some parts of the NuBus specification are "open"; that is, certain parts are undefined or optional and can be implemented as the designer sees fit. One of these is the use of the NMRQ* line: It can be bused, or each slot can have its own dedicated interrupt line. Apple has chosen the latter method, feeding each line to the VIA2 (versatile interface adapter) chip. Although bus parity is described in the NuBus specification, its use is not required. Bus parity is not generated by Apple NuBus, and the NuBus lines SP* and SPV* are not used.

The processor on a card is not required to communicate to the bus by NuBus byte addressing, but it is convenient if several processors are sharing the bus. The Macintosh II is designed to support NuBus byte addressing. The bus transceivers are wired to place 68020 data bytes onto the bus in NuBus byte order. This wiring does not affect addressing. Finally, NuBus doesn't specify the contents of the NuBus configuration ROM. Apple describes specific information for the configuration ROM that enables the Macintosh II to install a driver for the card, run machine or card initialization code, and load bootstrap code if the card can be booted. All these code blocks in the configuration ROM are loaded into main memory on the Mac II's motherboard before being executed.

Apple NuBus has some limitations brought about by the Macintosh system architecture. For some time-critical operations (for example, a data transfer to the IWM), the 68020 must prevent NuBus from interfering with its local bus. It does this by performing a *local bus lock*. This is accomplished by asserting a line (BUSLOCK*) to the VIA2. This informs the NuBus interface to lock the motherboard RAM from NuBus access. The NuBus interface will respond with a Try-Again-Later transaction response code to any access attempt.

Current Macintosh software uses an address space of 4 megabytes, compared to the Macintosh II's 4 gigabytes. This came about because the 68000 processor is limited to 24-bit addresses and the location of the Mac ROMs in this address space. This poses a problem for the Macintosh II, since it must support the existing base of Mac software. The Macintosh II uses a mode bit on the VIA1 to indicate if it is running in a 32-bit mode or a 24-bit "compatibility" mode. In the 24-bit mode, logical addresses of the form Sxxxxx hexadecimal are mapped to physical addresses of the form FS0xxxxx hexadecimal. The 24-bit mode restricts the Mac II to six NuBus slots, and each slot is limited to 1 megabyte.

A vendor wishing to support either addressing mode should design a card's NuBus address decode to ignore AD23*-AD20* and use AD19*-AD0*. The card must be able to produce a 32-bit address to access resources on the motherboard.

From Bits to Pixels

The old Macintoshes use a bit map to represent the screen; one bit represents one pixel, and only two colors are possible: black and white. The Mac II generalizes this to 1, 2, 4, 8, 16, or 32 bits per pixel. Apple's first video board will use either

4- or 8-bit pixels, thereby allowing 16 or 256 different colors, respectively. On the high end, a 32-bit pixel gives a theoretical limit of 4,294,867,296 different colors on-screen at one time—from a 48-bit-wide palette representing more than 280 *trillion* colors. (These numbers far exceed other system constraints.)

The design of Color QuickDraw allows the support of three different layouts of video memory. In *planar* layout, the video display comprises one or more *bit planes*, where the number of colors or shades of gray possible equals 2^n, where n is the number of bit planes. Here, adjacent bits in a bit plane contribute to the definition of different pixels, but the n bits that define a given pixel are scattered throughout memory. Color QuickDraw supports the monochrome one-plane graphics and the eight-fixed-color graphics supported by previous Macintoshes.

The second layout is the one Apple supports completely: *chunky* pixels. In this layout, all the bits for one pixel are adjacent and are followed by all the bits for the next pixel. Each pixel is defined by 1, 2, 4, 8, 16, or 32 adjacent bits in memory. This layout works well with Apple's preferred design of graphics output devices, which use *color lookup tables*. These cards use the numeric value stored in the pixel's memory to index into a known table of colors from a much larger color spectrum. In the case of Apple's first video card, the actual (Apple calls it *concrete*) color is 24 bits wide, giving 16,777,216 colors from which to choose.

The last layout is a hybrid of the first two, *chunky-planar*. It has separate memory areas for the red, green, and blue components of its pixels, with the components being chunky, that is, 1, 2, 4, or 8 adjacent bits describing a given component. This layout might be used someday to drive a very high-resolution color device that would use three slots for its three bit planes. The current implementation of Color QuickDraw does not support this, but the overall design permits it.

Color on the Mac II

One of the most impressive additions to the Macintosh II is its use of color. But how did the Apple designers do this, while still allowing the machine to run most existing Macintosh applications? The answer lies in the parts of the Mac II's Toolbox ROM code called Color QuickDraw; its supporting package, the Color Manager; and several other sections of the Toolbox. Here are the most important aspects of Color QuickDraw:

Backward compatible: According to Apple, "All changes are designed to be fully backward-compatible with the older Macintosh ROM." The designers of the new code do this in several ways. First, some QuickDraw routines have the same name but have been extended to take care of color and other enhancements (e.g., CopyBits). Some routines and data structures are new but are color equivalents of their monochrome counterparts (e.g., NewCWindow, bkPixPat). Some data structures replace the data in a given field with a handle to a larger pixel-oriented data structure. Of these, some flag the color orientation of the data structure by setting the top 1 or 2 bits of a given field to 1s. Overall, the designers said that about 80 percent of the Color QuickDraw code—mostly high-level routines—is the same as it was in the older QuickDraw; the rest is low-level routines that have been enhanced or changed to deal with colored pixel structures.

Generalized: Many data structures have been made more versatile. For example, the mouse cursor is still 16 by 16 pixels, but it can now be in color. Similarly, the patterns that QuickDraw uses to "paint" areas are no longer limited to being an 8 by 8 monochrome image. Color QuickDraw supports colored rectangular patterns with each side being a power of 2.

Adaptive: Color QuickDraw adapts to the display hardware that it is currently using. For example, Color QuickDraw drawing routines look at the current configuration of the graphics output device (video screen, printer, etc.) and adapt accordingly. Also, both the color cursor and color icons have two images: a color image for normal use and a monochrome one for use when the screen is either monochrome or 4-colored (i.e., 1- or 2-bit pixels). Color QuickDraw uses the appropriate image automatically.

Room to grow: The designers have created data structures with future expansion in mind. Several data structures have a field or fields reserved for future Apple use and a single field available for the application's use. In addition, Apple's first video display card can grow from 4 to 8 bits per pixel (increasing the possible number of colors from 16 to 256). Color QuickDraw is designed to use pixels up to 32 bits wide from a color palette with 48-bit-wide entries.

Color QuickDraw does not do all this work by itself. The Color Manager routines and data structures manage the use and mapping of color through a data structure called a gDevice (graphics output device) that describes the display (or printer) device being used.

Absolute and Concrete Colors

One of the most unusual features of the Mac II is that it was not designed with a standard video output. Although only one video card is available now, Apple expects that several video cards of differing color capabilities and resolutions will be available eventually. With such diversity possible, how can a software developer know what to put on the Mac II's video display?

Just as Apple hardware engineers designed the Mac II to one of several video output cards, the system software engineers envisioned a way of representing video images that any video output card can use. They decided that all applications should work with colors in an *absolute* form, represented internally as three 16-bit values, one each for the red, green, and blue components of the color. As we will see below, the Color QuickDraw software and the Mac II video hardware will work together to translate an absolute color to the closest concrete color the video card can supply.

The Color Lookup Table

As we mentioned before, the Mac II is most comfortable with printer interface and video output cards that use a color lookup table. In this way, even the standard video card, which can display only 16 colors, can offer that many colors from a much larger palette. When you start up the Mac II, system software initializes each graphics output device with its closest approximation to the values for a standard color table. When an application requests an absolute color, certain routines (described below) use the current device's color table to supply the best approximation that device can supply.

The Color Manager provides several routines that allow running programs to interact with the current graphics output device. Among them are

Color2Index and *Index2Color*: These translate between the absolute color and the index number of the closest concrete color the device can supply.

InvertColor: This routine translates an absolute color to the closest concrete representation of its inverse.

GetSubTable: This routine is given a table of absolute colors and calculates their nearest concrete equivalents in the color table of the current device.

Since both desk accessories and multiple application programs (through Apple's Switcher program now and, perhaps someday, multitasking programs on a future Apple machine) must share the same color table, the Color Manager includes routines to change and protect the current device's color table;

continued

Color QuickDraw adds six modes that are equivalent to the modes of TI's TMS34010 chips.

SetEntries: This routine lets the application change an entry in the current device's color lookup table.
ProtectEntry: This "locks" a table entry so other applications (running under Switcher) or desk accessories cannot change it (or it can also unlock an entry).
ReserveEntry: This reserves or unreserves a given entry for exclusive use by the current application; other programs will not be able to "see" or use it.

Color Drawing Modes

Color QuickDraw supports the source/destination drawing modes of old QuickDraw (copy, OR, XOR, BIC [black-is-changed], and their negative counterparts), but all but the copy modes don't make sense when they are used with colored pixels. Color QuickDraw adds six modes that are equivalent to the modes of the Texas Instruments TMS34010 chip. The modes are replace-with-transparency (allows one image to overlay another), additive (which is like combining colored lights), subtractive (like mixing paints), maximum and minimum (for overlapping aliased objects), and blend (combines source and destination pixels in a fixed ratio).

Inside gDevice

So far, we've talked only about the interaction between Mac II software and the current graphics output device hardware. (This device is usually a video display but can also be a printer or an off-screen pixel map.) The data structure that bridges the software and the hardware is called the *gDevice*, or graphics device; this data structure is created when the system software opens the device driver for a given device.

In general, the gDevice record gives the system software access to certain necessary information about the current device. Here are some of the most important fields:

gdType is an integer that tells the software the type of the current device, for example, fixed-color, color lookup table, or direct RGB.
gdSearchProc points to a linked list of one or more routines that translate an absolute color to a concrete color. This routine can be called by higher-level routines like Color2Index, and different applications can install their own search routines for use by them alone.
gdCompProc is similar to gdSearchProc except that it points to a linked list of routines that map an absolute color to its concrete complement. This routine is called by InvertColor.
gdPMap is a handle to the device's pixel map.

Other Color-related Changes

The Apple software engineers have added several new window-related data structures and Toolbox routines that include color support. The GrafPort structure of the old Macintosh has CGrafPort as its counterpart. Similarly, the color equivalent of the old WindowRecord is CWindowRecord. Also, NewCWindow and GetNewCWindow create a new window. The Mac II's Window Manager routines have been expanded to work correctly with both WindowRecords and CWindowRecords.

The CWindowRecord is identical in size and most of its fields to the old WindowRecord (at least in the first implementation of the Mac II Toolbox ROM). Color-related information is stored in that window's *auxiliary window record*. This record points to a color window table, which determines the colors used for the window background, border, text, close and zoom box highlighting, and title bar background.

When the Apple engineers decided to preserve the congruence between the monochrome and color window records, this meant there was no space for the CWindowRecord to point to its auxiliary window record. Instead, the global variable AuxWindowList points to a linked list of auxiliary window records, each of which points to the color window record that owns it. Also, a window can do without one if it uses the system software's default window colors.

Controls (buttons, check boxes, etc.) have *auxiliary control records* that are analogous to auxiliary window records. The routines SetDeskColor and SetDeskPixPut allow software to add color and patterns to the desktop itself.

The following menu components can also contain their own independent colors: the menu title and the menu item background, text, check mark, and command key. The Mac II has systemwide default menu colors if the System file contains a *menu color information table* (an 'mctb' resource). An application can override these values if it contains its own 'mctb' resource.

(The engineers also described two other changes to menus that are not related to color. First, menu bars now have their own *definition procedure*, which controls how they are drawn. Although the Apple-supplied procedure will restrict menu bars to the top of the main screen, it will be possible to write a different definition procedure that can allow, for example, a menu title and its body to be "torn off" and moved around the desktop or attached to a window's drag bar. Second, Mac II menu items can have secondary menus that pop up to the right side of the item [as in the Commodore Amiga]. These let you make several related choices with one mouse movement.)

Macintosh II Video Card

The Macintosh II does not come with a built-in monitor. Video is "broken out" into a graphics card that plugs into a NuBus slot. This eliminates some processing constraints coupled to the video display that are a fact of life with the old Macintosh.

Apple will offer a Macintosh II Video Card that can provide up to 256 colors or shades of gray on a 640- by 480-pixel display. This card has a user-selectable color depth of 1, 2, 4, or 8 bits. It features a color lookup table that can be adjusted to display any 256 colors selected from a palette of 16.8 million. The video card will come equipped with 256K bytes of RAM, providing video memory that supports a display of up to 4 bits per pixel, or 16 colors. As an option, the video memory can be expanded to 512K bytes, allowing a display of 8 bits per pixel, for the maximum of 256 colors.

The heart of the Macintosh II Video Card is a custom chip called the TFB, named after its designer. This chip handles video timing and generation and on-card memory control. The TFB uses two clocks on the card for generating video signals. The first is a 30.24-MHz clock used with color monitors. The second is a 12.27-MHz clock used to generate RS170 video (RS170 video is an RGB signal with NTSC timing that is used with projection TVs or film recorders). A software-controlled interlace bit is used to select which clock signal the TFB uses to generate the display.

The video card features an adjustable color lookup table (CLUT). This is a chip with a memory array and three 8-bit D/A converters that generate the red, green, and blue analog signals. A color pixel that is to be displayed is first presented to the CLUT. This byte (we're assuming 8-bit color depth) is used as an offset into the memory array that is composed of 256 values that are 24 bits wide. This 24-bit value drives the three 8-bit D/A converters. A copy of the CLUT values is

continued

maintained in Macintosh II memory. If you happen to alter a color, this table in RAM is updated. The RAM table is then loaded into the CLUT, preserving your new selection. The CLUT mechanism offers ample flexibility in color selection without complex color-generation hardware. Applications or desk accessories that used the VBL interrupt will still function—a "fake" VBL interrupt is provided by the Mac II.

Enhancements to TextEdit

One of the shortcomings of the original Macintosh is that the standard text-editing package, TextEdit, cannot handle tabs or any change in font, size, or styling. (Obviously, it can't handle color information, either.) In addition, TextEdit has its own scrap, distinct from the Scrap Editor's desk scrap (used to implement the Macintosh's cut-and-paste Clipboard).

The TextEdit code in the Macintosh II uses the same edit record as the old Macintosh ROM to store a unit of text but interprets some of the fields differently to deliver new features. The most important of these are

txSize: If txSize > = 0, the edit record is of the old type and nothing changes. If txSize < 0, this is a new edit record, and the following fields are interpreted as described below.

txFont and *txFace*: If this is a new edit record, combine these two fields to make a handle; its value will point to this edit record's style record (discussed below).

lineHeight: If lineHeight > = 0, use it and the fontAscent field as normal. If lineHeight < 0, use the style record's pointer to the line-height table (LHTable), which contains information on the spacing between any given line and the one that follows it.

Using Styled Text

The old edit record has room for the text of the record and nothing more. The style table and style run data structures contain the additional information that the old edit record lacks.

Apple defines a substring of text with the same font/size/styling/color/vertical spacing attributes as a *run*. The *style table* contains one record for each distinct font/size/styling/color combination; the stCount field within a record contains the number of runs in the edit record text that use this combination.

The edit record points through the *style record* to a table called the *style run table*. The first record points to the first character of text and to the style table entry that describes its font/size/styling/color/vertical spacing attributes. The second record points to the first character of the second run and its attributes, and so on sequentially through the runs of the edit record's text.

The enhanced TextEdit routines cut and paste directly to the desk scrap (instead of the internal scrap the old TextEdit routines used). In addition, the Mac II designers have defined a new scrap type, styl, to go with the TEXT scrap type used by old Macintosh applications. The styl scrap type, StScrpRec, contains a table that is almost identical to the style table in that it describes the spacing, font, size, style, and color of a certain run of text. However, it is like the style run table in that it describes the text linearly, with one record for each of the runs as they appear. The stCount field has a new meaning: It gives the starting character position for the run.

Though the enhanced TextEdit is still not versatile enough to meet the needs of all applications (word processors, for example), it will be useful in many others. Also, applications' use of the styl record will make the cutting and pasting of styled text more commonplace among programs.

The result of all this is a set of routines and data structures that is backward-compatible with existing Macintosh software but has the ability to save a string of text with its spacing, font, size, style, and color attributes and pass it (via the standard Macintosh Clipboard) to another program that can use it. For compatibility with existing Macintosh applications, Apple recommends that a software developer save text directly to the desk scrap (both TEXT and styl records) and the old TextEdit private scrap that older programs expect to see.

Sound

The Macintosh II has decoupled sound generation from the hardware that limited the sound capabilities on the Macintosh. Some of the sound generation has been implemented in a custom chip, reducing the CPU overhead required to make complex stereophonic sounds. Also, the sound drivers have been expanded and are now incorporated into a Sound Manager.

The old Macintosh tied sound generation to the video display's Vertical Blanking interrupt and a buffer of 370 bytes interleaved with the buffer for disk speed control. The restriction imposed by this time interval (the blanking interrupt) limits digital sound reproduction on a Mac to a maximum frequency of 11 kHz. This is an ideal condition at best: Actually, the Mac has a practical frequency range of 5 kHz, due to the sound hardware and filtering. The Mac's hardware limits digital sound sampling rates to 22 kHz.

The Mac II has independent sound-generation circuitry. It can sample at 44.1 kHz, the same as a compact disk player's sample rate. However, a CD uses 16 bits of information to encode sounds, while the Mac II uses 8 bits. Finally, the Mac II's sound-reproduction circuitry, although improved, yields a practical frequency range of only 7.5 kHz.

The sound hardware consists of an Apple Sound Chip (ASC) and two Sony power-amplifier chips. The ASC has two pulse-width-modulated outputs, each routed to its own Sony chip to produce stereophonic sound. Only one of the Sony chips drives the Mac II's internal speaker, producing monophonic sound. The Sony chips drive an external stereo jack with the uppropriate voltage levels for Walkman-style headphones and booster amplifiers, which simplifies interfacing the Mac II to stereo sound equipment (the Mac overdrives this type of equipment). The Mac II senses whether something is plugged into the external stereo jack and generates stereophonic or monophonic sound as appropriate.

A four-voice Wave Table Synthesizer is built into the ASC. Repetitive waveforms can be loaded and played continuously without CPU intervention. The ASC reduces CPU overhead in waveform synthesis from 50 percent to less than 10 percent. Sound generation now requires so little processor overhead that complex sound generation can be performed concurrently. For example, you could have a favorite waltz sound file read off a hard disk and play while you worked with a spreadsheet or word processor. You can't play a sound file from a floppy disk concurrently because the IWM disk-controller chip requires too much processor intervention.

As stated earlier, the sound software has been improved. All old Sound Driver and Synthesizer calls are supported. The programmer also has four new synthesizers available for use.

The first is the Note Synthesizer. This software plays a simple melody of notes, one at a time. The software is equivalent to the Mac Square Wave Synthesizer if the Note Synthesizer is programmed to play square waves.

The second is the Wave Table Synthesizer. It plays sounds using wave table lookup synthesis. A *wave table* is one complete oscillation of a waveform stored in a table of 512 or 256 8-bit samples encoded in an offset binary format. A wave table can be loaded or modified at any time during play. The Synthesizer can

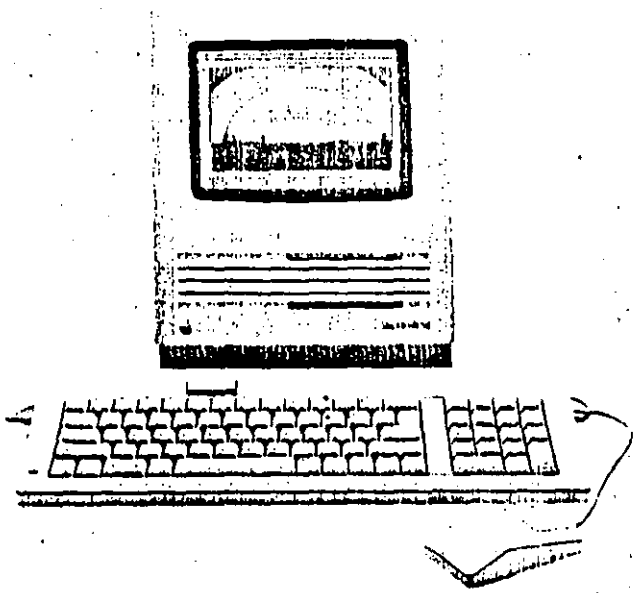
continued

play a single tone or several. A one-shot mode plays a wave table upon command, rather than continuously. This software corresponds to the old Four Voice Synthesizer if four waveforms are played.

The third is the Musical Instrument Digital Interface Synthesizer. It provides a convenient software interface to play an external synthesizer attached to the Mac II using the MIDI music standard. You'll need a vendor's MIDI interface unit to complete the connection between the Mac II's serial port and the music equipment. The MIDI Synthesizer conforms to the current MIDI specification. The Sound Manager has 16 channels that correspond to MIDI channels.

Finally, there is the Sampled Sound Synthesizer, which plays prerecorded or sampled sounds. As with the Wave Table Synthesizer, the sounds are encoded in offset binary. The sounds can be played at the original or different sampling rates. Different sampling rates change the pitch

The Mac SE



The Mac SE is one of the two new members of the Macintosh line. It will be priced from $2600 to $3600, depending on the system configuration, and should be available by the time you read this.

At first glance, the outside of the Mac SE resembles a Mac Plus: It has a built-in 9-inch monochrome monitor with a 512- by 342-pixel display, a single 3½-inch 800K-byte floppy disk drive, two serial ports, a connector for an external drive, and an SCSI port as standard. The housing looks almost the same, but then you note small differences. The cooling vents on the top of the Mac Plus are moved to the front of the machine. The compartment for the clock/calendar battery is missing—so is the mouse port! A new access panel for reaching a single peripheral card has appeared. The plug-in jack for the keyboard is gone because the Mac SE keyboard uses the Apple Desktop Bus, whose connector (one of two) is located on the back of the machine. Finally, when you turn the Mac SE on, you hear the purr of a cooling fan.

Internally, the Mac SE basically resembles the Mac Plus. The Mac SE uses the same 68000 microprocessor running at 7.83 MHz. It has the same SIMM (single in-line memory modules) holding a megabyte of RAM, expandable to 4 megabytes. It uses the same sound-generation circuitry as the Mac Plus. However, the similarities to the Mac Plus end here. The Mac SE has 256K-byte system ROMs. The power supply has been beefed up: It has a maximum output of 100 watts, and we've already mentioned the cooling fan. Many of the discrete components that populated the Mac Plus motherboard have been combined into a large gate-array chip on the SE's motherboard. The clock/calendar chip is powered by a seven-year lithium battery, also mounted on the motherboard, and a 50-pin SCSI connector is mounted next to the NCR 5380 SCSI controller chip.

There's enough room in the upper housing and adequate power to mount an internal SCSI 20-megabyte hard disk or an extra 3½-inch floppy disk drive. Both hardware options are available from Apple for the Mac SE. Last but not least, a single 96-pin slot using a Eurocard type C connector is mounted on the side of the Mac SE's motherboard (see photo A). This connector makes unbuffered processor signals and power available to vendor cards that can be plugged into the slot. The card must lie parallel to the motherboard in the cramped space at the bottom of the Mac SE.

The most interesting thing about the Mac SE, of course, is its expandability. Apple has announced that it will supply a 5¼-inch 360K-byte floppy disk controller card, with software to translate Macintosh text files to an MS-DOS file format and back. An Apple spokesperson indicated that a third-party company will announce an 8086-based expansion card that will give the Mac SE IBM PC compatibility. The card will emulate both the IBM monochrome and CGA cards in software and will give approximately the same performance as an IBM PC. It will contain an Intel 8086 processor but has no provision for the use of an 8087 numeric coprocessor. The board will do no multiprocessing: At all times, either the 8086 or the 68000 will be in control of the machine. The board will use Mac SE memory for its computation. Other possible uses for the slot are a coprocessor card (68020 or 68881, for example), a local area network card, and an interface board to an external expansion box.

The amount of code and data in ROM has doubled, from 128K bytes for the Macintosh Plus to 256K bytes for the Mac SE. About 160K bytes of this is actual code and resources, comprising the code from the Mac Plus ROM, considerable enhancements to that code, and all the code libraries that were formerly stored in RAM except the International Utilities Package (which handles time, date, currency, and other country-specific items). The rest of the space is taken up by the Macintosh system fonts (Chicago 12, Monaco and Geneva 9 and 12 for roman-language-based systems, and kanji for Japanese systems). The

2-J

of the sounds and can be used for special effects. This synthesizer corresponds to the old Free-Form Synthesizer.

SCSI

The Macintosh II uses the same NCR 5380 SCSI controller chip as does the Mac Plus. However, a number of changes to the Mac II hardware have improved performance and reliability of the SCSI interface.

The first of these changes is that the Mac II SCSI interface now supports an SCSI interrupt. This interrupt signal (IRQ) is connected to the VIA2 chip. The 5380 DMA (direct memory access) Request signal, which indicates that the data register is ready to be read or written, is also brought out to the VIA2 as an interrupt. This setup prevents the SCSI bus RESET* signal from causing a permanent interrupt to the 68020, since RESET* is not maskable through the 5380.

continued

Mac SE and Mac II ROMs share some of the same code. Some of the routines that use identical code are Appletalk drivers, TextEdit routines, SCSI Manager, ADB drivers, and the Script Manager.

The video circuits in the earlier Macintoshes spend 50 percent of the RAM access time available during display of a horizontal line, leaving the other 50 percent of that time for doing everything else (they spend all of the time for normal computation during the vertical and horizontal retrace intervals, when the video screen is not drawing anything). As mentioned earlier, the Mac SE integrates 19 discrete chips into one custom gate array and a PAL. Because this gate array can transfer twice as much data (collects two words instead of one word) at a time into the video circuitry, the Mac SE now spends only one quarter of the RAM access time servicing the video display. This provides a theoretical performance boost of approximately 25 percent for applications running in RAM. (The actual increase varies from 10 percent to 20 percent.)

The SCSI hardware in the Mac SE now does its signal handshaking in hardware; this allows it to run faster than it would under software that polls the SCSI port periodically and more accurately than it would under software that does "blind" (i.e., no handshaking) reads and writes. Like the Mac II, the SCSI hardware also provides an SCSI interrupt to the Mac SE. This, combined with the rewritten SCSI Manager, should provide a performance boost for SCSI hard disk operations.

The Mac SE, like the Apple IIGS, uses the new Apple Desktop Bus to connect the keyboard, mouse, and other human-input devices to the computer. (See "The Apple IIGS" by Gregg Williams and Richard Grehan in the October 1986 BYTE for more details.) This scheme decreases the amount of time the 68000 must take to service these devices and makes the design of new input devices much easier (and these devices can be used on both Macintoshes and Apple IIGSs).

Finally, the Mac SE has increased the amount of parameter RAM (used to maintain user preferences, time, and other data even when the machine is turned off) from 20 bytes to 256 bytes. Apple has not decided what to use this extra memory for, but we're sure it will eventually be put to good use.

There will be no upgrade path from the Mac Plus to the Mac SE. There are so many changes to the Mac SE's housing, motherboard, and analog board that it would be more economical to purchase a new computer rather than attempt an upgrade.

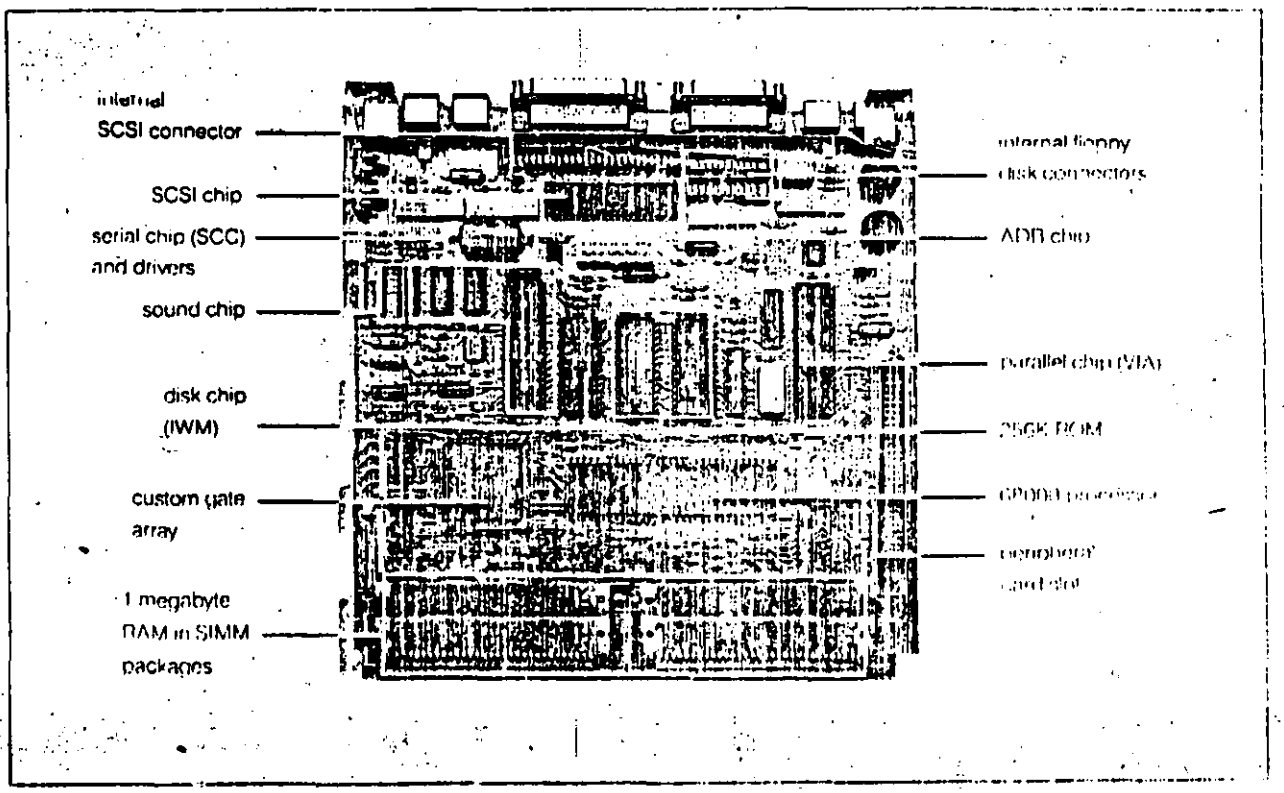


Photo A: *The Mac SE motherboard.*

This means that a slow device on the SCSI bus can now interrupt the processor when it has completed an operation. For example, suppose you have a hard disk and a tape backup unit attached to the Mac II SCSI bus. You order the tape unit to find the logical end of the tape and copy a file to it. The tape unit disconnects from the SCSI bus while executing the seek to end of tape. You can continue to manipulate files on the hard disk, since the SCSI bus is free for use while the tape unit is disconnected. When the tape unit reaches the tape end, it attempts to reconnect to the SCSI bus and generates an interrupt. Software then starts the process of copying the file from the disk drive to the tape unit. The important thing to note is that the interrupt mechanism prevents slow devices from tying up the Mac II processor or SCSI bus.

The second change is that the SCSI interface supports hardware handshaking during an SCSI bus transfer. The Mac Plus SCSI interface has two modes of data transfer: pseudo-DMA and blind reads or writes. The pseudo-DMA mode allows the 5380 to perform the SCSI bus handshake, but the 68000 polls the chip to check on the status of the transfer. The processor fetches or writes a byte when the 5380 indicates that the transfer operation is complete. Each byte moved through the SCSI interface has to be checked in this manner. It's obvious that SCSI transfers consume some CPU overhead and lower the effective transfer rate.

The alternative for Mac Plus SCSI transfers is to utilize blind reads or writes. These operations simply pass data bytes through the SCSI bus with no handshaking, nearly tripling the data-transfer rate. If the SCSI device is fast enough to handle this data flow, this isn't a problem. If the device isn't fast enough, however, the processor can write invalid data by overrunning the chip during a write operation, or it can read invalid data during a read operation by accessing the chip before it has received a valid byte. The Mac II's hardware handshake eliminates polling by allowing the 68020 to access the 5380 only when valid data is available. This is accomplished by suppressing the DSACK0* line, which holds the processor off the chip. The Mac II's handshake DMA eliminates the CPU overhead required to perform reliable high-speed SCSI transfers. It must be noted that these hardware handshakes must occur within 16 microseconds or a bus error will be issued in an attempt to end a presumed deadlock. Therefore, handshake DMA should be used only with high-speed devices.

A 50-pin SCSI connector is located internally in the Mac II. On the outside, a DB-25 SCSI connector—identical to the Mac Plus's—lets you connect other SCSI peripherals. The SCSI Manager now has the capability to partition hard disks and boot from a particular partition.

Slot Manager

The NuBus specification spares you from knowing intimate hardware details to configure a new card into the Mac II: The address space is set when the card is plugged into a slot. In a similar manner, something should spare you from knowing intimate software details to install a device driver for a new card or set an interrupt vector for the driver. This is the Slot Manager's job.

At start-up, the Slot Manager detects the presence of a configuration ROM on a NuBus card. If the card is defined as a boot device, the Slot Manager will read the boot code in the configuration ROM into memory and transfer control to it. If the card is not bootable, information is read in the ROM that describes the device driver or drivers for the card. The start-up code next attempts to read a driver with the same resource name in the System File and install it in the Mac II's main memory. If a resource with this name can't be found in the System File, the named driver is read from the configuration ROM and installed into memory.

This method of driver installation provides two benefits. First, the device driver embedded in the card's ROM is installed automatically into the system without user intervention. Second, should the device driver code need fixing, the vendor can provide the new code on a disk that can be inserted into the System File using a simple install program. Since the System File is searched first for the card's device driver, this replaces the old driver in the configuration ROM.

Interrupts using NMRQ* are posted to the VIA2 chip. The Slot Manager determines which slot requested service by reading a register in the VIA2 and dispatches the appropriate interrupt routine. Interrupt routines are also in the configuration ROM as part of the device driver.

Script Manager

One of the most significant concepts of the original Macintosh was its division of a file into *resources*, where each resource is a certain type of data used by the file. In the case of program files, the code resource (which contains the executable code of the program) is separate from, say, the information relating to dialog boxes in the DITL resource. This makes it possible, for example, for a developer to change the text contained in dialog boxes without having to change the code of the program itself. Thus, a developer can easily change a program to French, Spanish, or another Romance language (i.e., one that uses the Roman alphabet).

But what about other languages, like Arabic, which reads right to left and alters the shape of its letters based on its surrounding letters? What about Japanese, which has far more than the 256 characters allowed in a normal Macintosh font? To meet these needs, Apple has added another library of code, the Script Manager, to isolate language differences from the rest of an application program, thereby making it far more portable among different human languages.

(In addition to the features of the Script Manager, the TextEdit editing package and the International Utilities Package have been extended to work correctly with the Script Manager. Also, the system software described here will work with any Macintosh with enough memory to hold the needed programs and data. Because of this, references will be made to routines and data structures from the old Toolbox ROM, not the newer Mac II Toolbox ROM.)

According to Apple, a *script* is a "writing system" that includes a character set; a writing direction (left to right or vice versa); keyboard mapping(s) and text input method(s) (e.g., multiple keystrokes per character); text drawing, measuring, and editing methods; sorting methods; and time, date, and number formats. To use a given script, the Script Manager must have an associated *script interface file* for that script. Apple now has RIS (roman interface system), KIS (kanji interface system, also called KanjiTalk) for Japanese, and AIS (Arabic interface system) for the major Arabic languages. (AIS has been available since October 1986.)

Not all applications will need to use the Script Manager. (But those applications that do use it have access to routines they would normally have to provide themselves.) The enhanced TextEdit can handle text selection, highlighting, word selection, dragging, and word-wrapping of a given script automatically; only applications that do extensive text manipulation or that don't follow the *Inside Macintosh* guidelines will need to use Script Manager routines. But for those applications that do need to use them, the following paragraphs describe some of the major routines, what they do, and why they are needed.

The Macintosh finds which script to use by looking at the font associated with the current GrafPort. The routine FontScript returns the value of the current script. Applications can cause the keyboard to change in accordance with the

continued

user's font by using the KeyScript routine.

Each font has a direction associated with the drawing of its words; for example, the Arabic font places its characters on the screen right to left. The low-memory global variable teSysJust determines the direction of text justification (e.g., right justified for Arabic) and the direction of successive words in whatever font. For example, a mixed sequence of Arabic and English words is placed right to left if teSysJust is on and left to right if it is off. In both cases though, individual English words are written left to right and Arabic words are written right to left.

In languages with more than 256 characters (Japanese, for example), individual characters are represented by 16-bit words, but some roman characters are still represented by 8-bit bytes. This presents problems when searching a text string for a character that is represented in one byte—it may mistakenly match the second byte of a 16-bit character. To prevent this, the application should call the routine CharByte, which determines whether a given byte in a string is a 1-byte character or the first or second byte of a 16-bit character.

Similarly, the complexities of certain languages (including Japanese and Arabic) make it difficult to associate a certain pixel location on-screen (where the mouse button was clicked, for example) with the character it corresponds to in the associated text string. To help solve this, the routines Pixel2Char and Char2Pixel associate a given character in the text string with the number of pixels the drawn character is from the beginning edge of its representation on-screen.

Some languages do not use spaces to separate words. Because of this, the Script Manager uses a *break table* to define where text can be correctly broken. The break table is actually a collection of rules or templates, called *continuation sequences*, that define what character sequences shouldn't be broken. The routine FindWord uses the break table to determine where word breaks occur.

Finally, languages differ in the way they add "blank" space to make a certain text string justified. The routine DrawJust draws text fully justified, using a method particular to the given script to fill the line of text out so that it fills the entire space between both margins.

Multiple-Screen Desktop

One of the most amazing things the Mac II can do is to treat the images from two separate monitors as if they were both part of one large desktop area. As you can see from photo 2, the devices don't even have to be of the same type. The paragraphs that follow give a brief explanation of how this is accomplished.

Each graphic output device (gDevice) connected to the Mac II defines a rectangular area, the gdRect, it is responsible for. When the gDevices are connected properly, the Mac II system software considers the desktop (the region called GrayRgn) to be the union of all the devices' gdRects. By referencing GrayRgn, system software can move objects among all the display areas of the different output devices without any limitations; also, "well-behaved" Macintosh applications can use the larger desktop without having to do anything special to use it.

All the gDevices are connected together in a linked list of handles; a program gets the first device by calling the GetDeviceList routine, or it can get the primary device (the one that contains the menu bar) by calling GetMainDevice. The Color QuickDraw routines have been extended to draw to multiple gDevices. When a Color QuickDraw routine receives a drawing command, it checks to see if the drawing is intended for the screen (as opposed to an off-screen bit map). If it is, the routine compares the rectangle in which the drawing is to take place with the gdRects of all the screen devices and issues a drawing command to each device where there is some intersection between the two.

A window can even span multiple screens because the Mac II Window Manager has been modified to the dragging boundaries enforced by the older Macintosh ROM (thus enabling the mouse pointer and the objects it is dragging to move among screens). Also, since the various screens may be of different resolutions and color depths, the cursor-drawing routines must keep track of which screen contains the cursor.

MS-DOS Compatibility

Apple sees several ways in which to give its users the levels of MS-DOS (and IBM PC) compatibility. Depending on the user's needs, it may be enough to provide file transfer capability through a network of Macs and IBM PCs or through a file conversion utility called PasSport that Apple plans to announce later this year. (Apple offers a 5¼-inch disk drive that, when connected to a Macintosh, allows the appropriate software to read and write IBM PC-compatible files.)

Apple told us that an unnamed third-party company will provide an 80286 multiprocessor NuBus card and software for the Mac II that will give approximately the performance of a 6-MHz IBM PC AT. This combination will let you work on MS-DOS applications in a Macintosh window, with full access to the desktop and desk accessories and some cut-and-paste capability between applications on the different machines. They will also be able to use a Macintosh hard disk in such a way that both IBM and Macintosh programs will be able to read and write each others' files directly.

The 80286 card will be capable of true multiprocessing with the Mac's 68000. It also will have a socket for the addition of

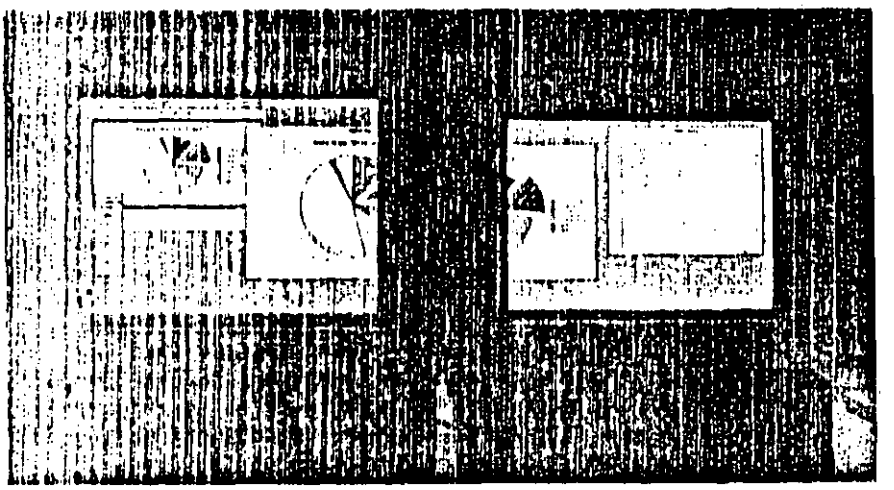


Photo 2: *The Macintosh II's multiple-screen desktop. The Mac II automatically configures the desktop to be the union of all active video devices. Windows and icons can be dragged between screens and still display properly. "Well-behaved" application programs will be able to use the extra desktop area. The program in use is an unmodified version of Cricket Graph 1.0, and as you can see on the color monitor, the Mac II supports the old QuickDraw fixed-color scheme. The ScrapBook shows an earlier version of the chart and demonstrates the new color PICT format.*

an 80287 math coprocessor chip, and it will include 1 megabyte of memory for the 80286's exclusive use (with the possibility of expanding that to 4 megabytes).

This card will emulate both the IBM monochrome and CGA (color) boards and the Hercules monochrome text/graphics board in software. It will also be able to use the Macintosh SE mouse to control mouse-based IBM PC applications.

UNIX on the Macintosh II

Since UNIX is multitasking, the Mac II will need the Motorola 68851 memory management chip (which Apple will make available to Mac II owners). Apple made no announcement regarding a UNIX product, but Jean-Louis Gassée, vice president of product development at Apple, spoke of a version of UNIX that would contain 4.2 BSD (Berkeley Standard Distribution) features. Such a version, he said, would have to boot up the machine and would take over the system, turning the Macintosh into a "vanilla" UNIX machine; a future version might give programs access to the Macintosh ROM as a library.

Pricing and Availability

Although Apple disclosed no prices, one spokesperson quoted a price of between $4200 and $4300 for the basic Macintosh II (one floppy disk, 1 megabyte of memory) with the video card and a monochrome display and a price of less than $6000 for the basic Mac II and the color display. Apple gave us no indication of how much an internal hard disk or a video board upgrade would cost. (A list of the official prices will be on BIX by the time you read this.)

Apple plans to ship the Mac II sometime in the second quarter of this year. Because of the dissimilarity between the Mac II and earlier Macintoshes, no upgrades are possible. Also, Apple does not plan to reduce the price of either the Macintosh 512KE or the Macintosh Plus because of the Mac II's introduction.

Caveats

We wrote this article after two visits to Apple in December 1986 and January 1987 (this included discussions with the hardware and software design team), about a day's worth of hands-on experience with the Mac II, study of three binders full of technical documents, and several follow-up calls to the Apple staff.

The design team described both the hardware and software as "late alpha." This means that the final hardware (and especially the software) may differ somewhat from the details of what we've described here; but the overall design will be the same. In particular, any performance times measured in seconds should be taken as "ballpark" estimates.

Observations

Overall, we were very impressed with the machine. The Macintosh II is generally 3 to 4 times faster than a Mac Plus, except when it is dealing with a lot of transcendental math, in which case it is between 30 and 40 times faster (see the text box "Comparing the Mac Plus, the Mac SE, and the Mac II"). The normal actions of a Macintosh user—opening, moving, resizing, and scrolling—were effortless because they were very quick. One of the slowest operations, scrolling a window of color information, was noticeable but not objectionable—somewhere around one second to scroll the contents of a full-screen color window about an inch.

Both the monitors have an area of 640

Comparing the Mac Plus, the Mac SE, and the Mac II

Although these timings shouldn't be considered as gospel (after all, we worked on machines that weren't finished yet), we think we can draw some conclusions about how fast the three machines are relative to each other.

Here are the overall conclusions: The Mac SE is about 10 percent to 20 percent faster than the Mac Plus, and the Mac II is roughly 3 to 4 times faster than the Mac Plus (except in applications where heavy number crunching is done, in which case the Mac II can be 30 to 40 times faster). The paragraphs that follow will explain these conclusions.

Figure A shows the results of running some standardized tests on the three machines. A bar's height shows how much faster a machine is than a Mac Plus, and the number on top of the bar is the number of seconds it took to run the test.

The first three programs are standalone programs compiled by Lightspeed C, version 2.01. "Quicksort" is the standard BYTE sort benchmark. "Dhrystone" is the Dhrystone benchmark, version 1.1; this test simulates an average program by executing a known mix of control, assignment, and proce-

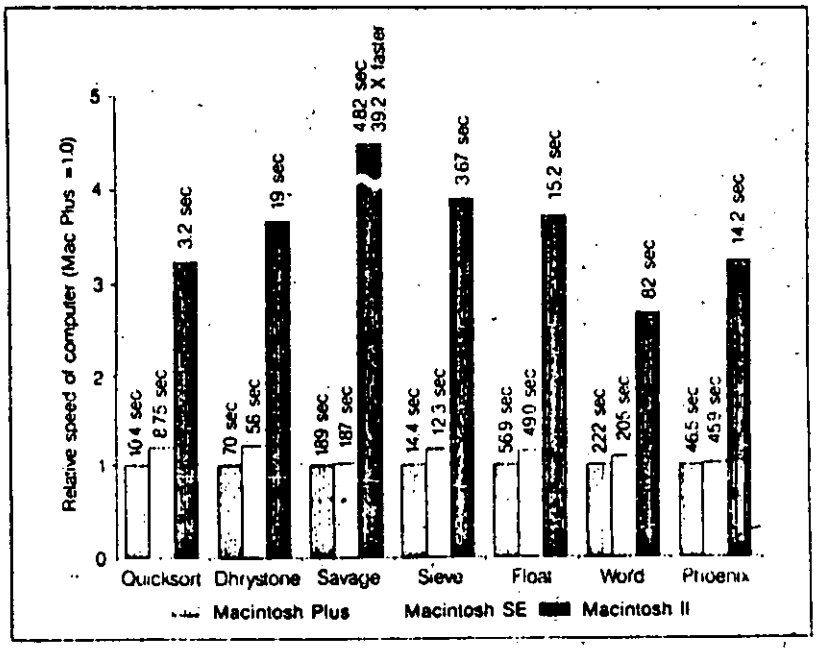


Figure A: *Relative performance of Macintosh family computers. The length of a bar tells how much faster a given computer is than a Macintosh Plus when running a given program. The number over each bar tells how long a program took to run on a given computer.*

by 480 pixels, which is 1.75 times the size of the Macintosh screen. Since most Macintosh applications will let you use this extra area, some of them were easier to use just because of the larger screen. The monochrome screen measures 12 inches diagonally, and the color RGB screen measures 13 inches. "Wonderful," you may say, "finally, a larger screen for my Macintosh!" Well, yes and no—it *is* larger, but it's also showing more pixels in each direction. Actually, both Mac II monitors have about the same pixel per inch density as the old Macintosh.

The precision of the color monitor is remarkable. When it was displaying a monochrome image, we found ourselves thinking we were looking at the monochrome monitor—which means that the monitor can display true black-and-white dots, even at the edge of the screen, without color fringes. This is an important factor when you remember you may be reading text, often as small as 9 or 10 points, on this screen for long periods of time.

Another remarkable quality of the Mac IIs we tested was their stability. Even though these were late-alpha prototypes, most of the software we tried out—software designed for the older Macin

continued

dure statements. Its result is usually expressed in dhrystones per second, but here we used the number of seconds needed to execute the test. "Savage" does a large number of transcendental functions (e.g., sine and exponentiation). The Mac II's relative performance of 39.2 times faster is due to its use of its 68881 floating-point coprocessor.

The next two tests are stand-alone programs compiled using Microsoft Compiled BASIC, version 1.0. "Sieve" is one iteration of the Sieve of Eratosthenes. "Float" is a test of multiplication and division.

"Word" gives an indication of a machine's video display performance by smooth scrolling through a 63K-byte Microsoft Word document (we used Word 1.0). The time given in figure A is for the Mac II using a monochrome display (1 bit per pixel).

Finally, "Phoenix" exercises the graphics and the SANE numerics package heavily by mensuring the time that a machine takes to transform a wire-image approximation of a sphere to a drawing smoothed with unframed faces; we used Dreams of the Phoenix's Phoenix 3D, version 1.2, three-dimensional program.

Remember, one key concept of the Mac II is that it lets the user decide how many colors to use on the screen at a time, with fewer colors giving better performance. Figure B bears this out. For the Mac II, it plots the number of bits per pixel versus the time required to smooth-scroll through a 63K-byte Microsoft Word document. As you can see, the relationship is almost linear. Though the Mac II using 1 bit/pixel performs 2.71 times faster than the Mac Plus, that performance degrades to 1.79 when using 4 bits/pixel (16 colors) and 1.28 when using 8 bits/pixel (256 colors).

One final set of figures comes from looking at the role the 68881 floating-point coprocessor plays in the performance of the Mac II. Normally, this can't be estimated because the 68881 is used automatically by system software. However, Apple had a demonstration program that did three-dimensional plots using no 68881 support, using the 68881 driven by the SANE numerics package (most applications will use it this way) and using the 68881 directly. The figure we used, Sombrero, is a common one that uses the sine function a great deal to create a figure of concentric ripples.

The times (and their respective ratios) are no support, 1165 seconds (1.0); SANE support, 261 seconds (4.46 times faster); and direct support, 37 seconds (31.5 times faster). The most important figure is the middle one because many existing applications use the SANE package; so when they run on the Mac II, they will get this level (up to 5 times faster) of improvement—and the more time the application spends using SANE, the greater the improvement.

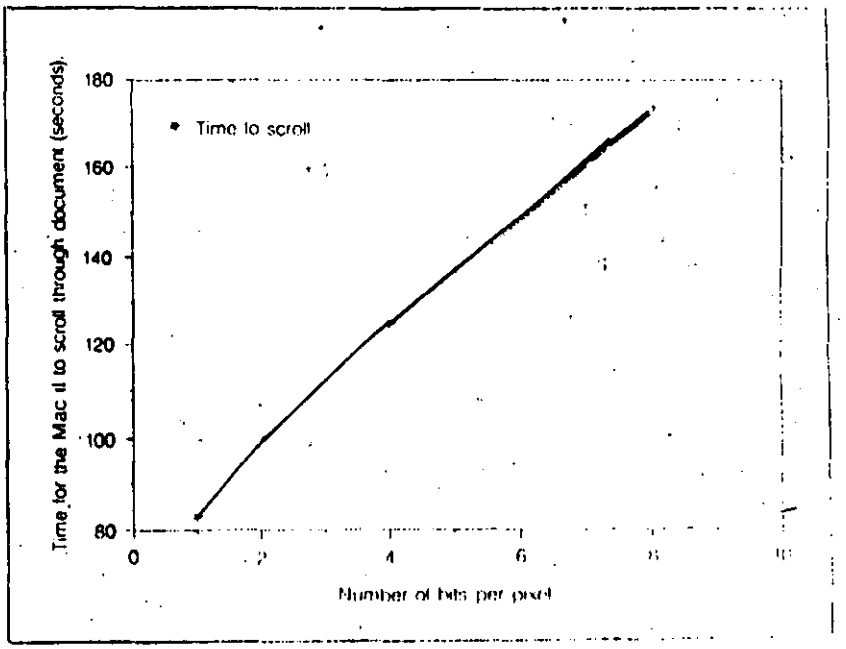


Figure B: *Macintosh II video display times as a function of the number of bits per pixel on the video display. As this figure shows, the more bits per pixel (or, equivalently, the more colors that can be displayed on-screen at one time), the longer the Macintosh II takes to scroll through a document of a given size.*

toshes—worked fine. The prototypes crashed only once or twice in several hours' usage (which is far more stable than most prerelease machines we see). We feel this is a testament to the stability of the machine's hardware and software architectures and is indicative of the quality of the hardware and software to come.

The bottom line on software compatibility is this: Mac software developers have had for quite some time now a list of guidelines to follow to ensure that their software would be compatible with future machines. Most companies have followed those guidelines, and their software will run on the Mac II (Apple claims "greater than 95 percent" compatibility). So the more recent your software and the more conscientious the software company, the more likely your old software will run correctly on the Mac II.

One concern we have involves the price and usefulness of any kind of MS-DOS compatibility card. In our experience, such cards always sacrifice some performance in the emulation of MS-DOS and cost as much as or more than an equivalent IBM PC clone. Another concern is that Apple told us that the third-party Mac SE and Mac II cards would use software to simulate and display the PC color and monochrome screens. The Commodore Amiga 2000's MS-DOS card uses hardware to maintain the screen's contents and software to display them and still cannot completely update the display. We feel that the all-software approach (for both the contents *and* the display of the MS-DOS cards) will either be too slow or will eat up too much of the processor's time. However, we will all have to wait for the products to come out before we can make any final judgments.

Though NuBus peripheral cards will be slow in coming, they will nevertheless be extremely important to the future of the Mac II. Many of the add-ins to the old Mac that required "major surgery"—memory upgrades, internal hard disks, and 68020 processors—are either included in the Mac II or are already planned for. Still other enhancements, like a full-page video display and other ideas we haven't thought of yet, will be much easier (and therefore much more likely) to be developed.

Conclusions

Apple has come a long way since the first 128K-byte Macintosh (with 64K ROMs) was introduced about three years ago. Now Apple has a true product line with the Macintosh 512KE, the Macintosh Plus, the Macintosh SE, and the Macintosh II, with prices from less than $2000 to more than $6000. The Macintosh is even affordable, with the bottom-of-the-line Mac 512KE having four times as much memory and twice as much ROM and disk storage, all for a street price slightly more than half the original Mac's $2500 price tag.

Often, a new machine with new features has an uncertain future. Will enough people buy new machines to prompt software developers to create new software that brings out its potential? Will enough programs come out to prompt the public to buy new machines? The Mac II will have some of this inevitable chicken-and-egg problem, but not as much as other machines. Mac II applications are not so much built-from-scratch implementations as they are bells and whistles added to a product that already has a large, established market. Also, both developers and buyers have been—no other word will do—*lusting* so much after these added capabilities that the only limitation, we think, will be the time it takes developers to learn how to use them. Just as the Macintosh has literally changed the way we compute, we're looking forward to the innovations the Macintosh II will make possible. ■

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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

14 DE AGOSTO AL 12 DE SEPTIEMBRE 1992

INTRODUCCION
ARQUITECTURA DE UN SISTEMA DE PROCESAMIENTO EN BASE A MICROPROCESADORES CRITERIOS DE SELECCION

ING. ABEL CLEMENTE REYES

AGOSTO-SEPTIEMBRE

1992

LOS MICROPROCESADORES Y SU APLICACION.

I. Introducción .

En la vida cotidiana actual es frecuente el ver como una gran parte de nuestros problemas los resuelven las computadoras, sin embargo, rara vez nos detenemos a reflexionar sobre ellas, tomando en cuenta que están pensadas y diseñadas para ejecutar una función en particular, la cual satisfaga alguna necesidad del ser humano, como podría serlo el comunicarlo telefónicamente o bien, el permitirle efectuar alguna operación bancaria desde su hogar, etc..

Por otra parte, cada computadora que existe actualmente es algo más que un montón de transistores y bulbos que sólo se prenden y apagan misteriosamente. Podemos considerar que una computadora es todo un sistema de procesamiento, el cual opera por la eficiente interrelación de sus elementos constitutivos, los cuales están centrados en un microprocesador.

En muchas ocasiones los diferentes sistemas de procesamiento están desarrollados alrededor de algún tipo de microprocesador en particular, sin embargo, generalmente dentro de un mismo sistema se encuentran presentes varias clases de microprocesadores, dependiendo de las características que deba tener una computadora en particular.

El principal objetivo de este estudio es analizar las principales características que deben ser tomadas en cuenta al momento de seleccionar o diseñar un sistema en base a microprocesadores, que esté orientado a resolver alguna necesidad específica de un centro de trabajo ya sea industrial o de otra índole.

II. Definiciones de Computadora y Microprocesador.

En ciencias de la computación es frecuente sobreentender ciertos conceptos comunes, sin embargo es importante que definamos con precisión que entendemos por una computadora, ya que de lo contrario al momento de estudiar las principales arquitecturas de sistemas de procesamiento, podríamos perder de vista que función debe realizar una computadora. Por otra parte, cuando se menciona el término "computadora" casi en forma instintiva pensamos en una terminal del tipo CRT, la cual debe estar desplegando los datos de un cierto programa; cuando pensamos en esto estamos todavía un poco lejanos del concepto que define a una computadora, mismo que es más fácil de entender desde el punto de vista de sistemas, considerando que una computadora es un cierto sistema de cómputo.

Si tomamos como base lo anterior, podemos definir que un sistema de cómputo es un conjunto de elementos íntimamente relacionados, los cuales reciben datos de entrada (información), mismos que serán modificados de acuerdo a una secuencia ordenada de pasos de un cierto algoritmo prefijado, para obtener datos de salida o resultados útiles que le permiten al ususario tomar una decisión. En otras palabras lo anterior nos quiere decir que un sistema de cómputo tiene como finalidad el recibir información que será procesada de acuerdo a un algoritmo fijo, con el propósito de permitir al usuario tomar decisiones.

Por otra parte, un sistema de cómputo puede ser de índole analógica, digital o híbrido, dependiendo de las principales características de sus elementos constitutivos.

En un sistema digital de cómputo, se encuentran presentes los siguientes elementos:

* Un medio de entrada por el que ingresa la información al sistema (UNIDAD DE ENTRADA).

* Un medio de salida por el que egresa la información ya procesada por el sistema. (UNIDAD DE SALIDA).

* Un medio que permite retener temporalmente la información dentro del sistema. (MEMORIA).

* Un medio que permita la ejecución de acciones aritméticas o lógicas necesarias para desarrollar un algoritmo dentro del sistema. (ALU).

* Un medio que permita el control de todas las acciones a realizar por el sistema . (UNIDAD DE CONTROL).

A los sistemas de cómputo que presentan estos cinco elementos se les llama clase Harvard, si aunado a lo anterior la instrucciones pueden ser tratadas en algún instante como datos, o los datos pueden ser interpretados como instrucciones, se dice entonces que el sistema en cuestión es clase Princeton o Von Neumann.

En estos casos notamos que estamos hablando de cinco elementos constitutivos, pero en un momento dado podemos asociarlos según ciertas características comunes, quedando los tradicionales tres elementos como se indica a continuación:

i) Un conjunto de elementos de entrada/salida, por los que se efectúan las transferencias de información entre el medio exterior y el sistema,dicho conjunto generalmente recibe el nombre de puertos de entrada/salida.

ii) Un conjunto de elementos que retienen temporalmente la información dentro del sistema; este conjunto comúnmente es llamado memoria.

iii) Un conjunto de elementos que procesan la información mediante la acción de ciertas operaciones lógicas o aritméticas. Por otra parte este mismo conjunto debe ejercer una acción de control sobre los demás elementos presentes en el sistema. A este conjunto en particular se le asocia el nombre de Unidad Central de Proceso o comúnmente CPU.

De lo anterior tenemos como conclusión importante el saber que los mínimos elementos constitutivos esenciales de todo CPU son la unidad de lógica aritmética y la unidad de control.

Por otra parte, cuando en un CPU se logran tener todos sus elementos integrados en un solo circuito de semiconductor, se esta hablando entonces de un microprocesador.

Existen comercialmente diferentes tipos de procesadores, sin embargo la gran mayoría son diseñados de acuerdo a lo que se conoce como paradigma de Von Neumann, el cual es un modelo basado en el concepto de una sola unidad de proceasmiento que accesa a un arreglo lineal de localidades de memoria de un tamaño fijo; pudiendo contener estas localidades ya sean datos o instrucciones.

Lo más relevante del modelo de Von Neumann radica en el hecho que el control del programa es secuencial y se encuentra centralizado en una sola unidad. Lo anterior se refleja fisicamente en el hecho de contar siempre en el procesador central, con un registro apuntador de programa, un registro que hace las veces de un acumulador y una unidad de control, la cual generalmente es un secuenciador.

Por otra parte, en una computadora diseñada es necesario tener mezclados los datos y las instrucciones en un mismo sector de la memoria, con el fin de que sean igualmente accesibles para el procesador; esta situación es ventajosa en algunos casos, pero no siempre es lo más adecuado en un sistema digital de cómputo.

Por último, es importante hacer notar que independientemente del tamaño de un cierto procesador, incluso particularizando en los que se encuentran integrados en un solo circuito, podemos hablar de microprocesadores de propósito general y microprocesadores de propósito particular.

Un microprocesador de propósito general está pensado y diseñado para poder ejecutar cualquier tipo de algorítmo que el

usuario le indique mediante un programa en lenguje ensamblador, en este caso no es necesario que el usuario analice la microprogramación del procesador en uso, basta únicamente que sepa la arquitectura interna (cuántos y cuáles son los registros internos presentes en el microprocesador). Por su parte, un microprocesador de propósito particular está diseñado para sólo poder ejecutar muy eficientemente un cierto algoritmo orientado a resolver una tarea especifica, resultando difícil ejecutar con este dispositivo algún otro algoritmo diferente par el cual fue diseñado. Es importante hacer notar que en esta clase de procesadores es frecuente el tener que programarlos mediante la escritura de algún patrón binario en particular, esto se debe a que su arquitectura interna requiere completar algún tipo de microprograma.

A continuación estudiaremos las principales características de las arquitecturas más comunes en sistemas desarrollados en base a microprocesadores.

III. Organización de Computadoras.

En sistemas desarrollados alrededor de microprocesadores, independientemente del tamaño, es común el oir los términos arquitectura y organización de computadoras, los cuales se refieren a la manera en que se relacionan física y lógicamente los diferentes elementos constitutivos de un sistema de procesamiento.

Existen actualmente diferentes técnicas bajo las cuales se organiza una computadora, sin embargo todas ellas tienen como principal objetivo el respaldar eficientemente el procesamiento de datos, con la finalidad de hacerlo más expedito.

Por otra parte, podemos decir que la organización de un sistema de procesamiento depende esencialmente de la aplicación que pretende satisfacer y, generalmente, dicha organización se refleja físicamente tanto en el diseño del mapa de memoria, como en la forma de interactuar de sus elementos constitutivos ya sean de programación, o bien electrónicos.

Si partimos desde un punto de vista operativo, podemos decir que el desarrollo cronológico de los sistemas de cómputo ha estado orientado como sigue:

i) Procesamiento en bloque.

   En éste caso los trabajos (jobs) del usuario son ejecutados de una manera secuencial, considerando que las operaciones de entrada, cálculo y salida de cada trabajo se efectúan sin sobreposición.

ii) Multiprogramación.

Este caso es similar al anterior, excepto que las operaciones de entrada/salida del i-ésimo trabajo se pueden superponer al cálculo en el CPU del j-ésimo trabajo.

iii) Tiempo compartido.

Para mejorar la eficiencia sobre las técnicas anteriores, las porciones de entrada/salida y cálculo, de un programa del usuario se parten en pequeños incrementos y sus ejecuciones van entrelazadas en base a un esquema de tiempo compartido.

iv) Multiprocesamiento.

El sistema operativo asigna programas del usuario a un determinado número de procesadores, los cuales ejecutan los programas de una manera cooperativista a todos los niveles de procesamiento.

Es frecuente el pensar que la multiprogramación y el multiprocesamiento son sinónimos, sin embargo de lo anterior notamos que estrictamente hablando no lo son ya que son dos técnicas diferentes en el desarrollo de sistemas de cómputo .

Por otra parte, en cada uno de los cuatro enfoques vistos se tiende a optimar el tiempo de procesamiento al ejecutar físicamente en forma más ágil un trabajo del usuario. Generalmente en este tipo de sistemas interviene uno o varios procesadores contando siempre con alguno en particular que ejecuta una acción de control según el paradigma de Von Neumann.

Si reflexionamos con más detenimiento en cada uno de los cuatro casos anteriores podemos llegar a la conclusión que en todos se presenta en mayor o menor grado lo que se llama procesamiento en paralelo, mismo que podemos entender como una forma de procesar información haciendo énfasis en la ejecución de eventos concurrentes en el desarrollo de un cierto trabajo o una cierta tarea asignada por el usuario. La concurrencia de eventos implica paralelismo y simultaneidad en la ejecución de acciones, referentes a un procedimiento.

Junto con la idea de paralelismo surge el concepto de lo que llamaremos "PIPELINE", que se refiere a la ejecución de eventos translapando el tiempo de realización de cada uno de ellos.

De lo anterior podemos sacar como conclusión importante el hecho que un procesamiento en paralelo requiere de la ejecu-

ción concurrente de varios programas en una cierta computadora, lo cual contrasta con el tradicional procesamiento secuencial; además la computadora que se oriente al procesamiento en paralelo debera cubrir algunas caracteristicas particulares en su circuiteria interna, principalmente en los microprocesadores que emplee.

Si ahora partimos desde un punto de vista organizativo, el procesamiento en paralelo puede ser clasificado a su vez en cuatro niveles diferentes que son:

a) A nivel trabajo.

   Diferentes procesadores ejecutan de una manera concurrente varios trabajos separados.

b) A nivel tarea.

   Algunos trabajos se puden particionar en varias tareas, mismas que pueden ser ejecutadas concurrentemente.

c) A nivel interinstrucción

   Más de una instrucción se puede ejecutar en paralelo, en caso de no existir alguna dependencia entre los datos de esas instrucciones.

d) A nivel intrainstrucción.

   Dentro de la ejecución de una instrucción se pueden efectuar ciertas fases de una manera superpuesta o paralela, en este punto se hace más patente el efecto pipeline .

En cada uno de estos niveles se establece una fuerte interacción entre el software y el hardware, por lo que es necesario contar con algún criterio que nos permita seleccionar adecuadamente los diferentes elementos que deben estar presentes en un sistema de cómputo.

Cuando se está desarrollando un sistema en base a microprocesadores y se desea lograr el paralelismo, principalmente en un sistema monoprocesador, se puede proceder en la siguiente manera:

* Buscar que nuestro procesador cuente con múltiples unidades funcionales, encargadas cada una de acciones diferentes.

* Buscar el paralelismo y el pipeline dentro del mismo procesador. Actualmente diferentes procesadores comerciales hacen uso internamente de múltiples unidades funcionales, por lo cual comparten internamente sus recursos empleando ele-

mentos de alta velocidad.

III. Arquitectura de Computadoras.

Como ya se mencionó, las computadoras en paralelo son sistemas que tienden a enfatizar el procesamiento en paralelo; a continuación se muestra una clasificación de éste tipo de sistemas de acuerdo a su configuración física, teniendose:

* Computadoras tipo pipeline.
  Esta clase de computadoras efectúan diferentes cálculos en forma superpuesta o con un cierto translape de acciones explotando el paralelismo temporal

* Arreglos de procesadores (Array processors)
  En éste caso,un arreglo de procesadores emplea, en forma sincronizada, múltiples unidades de lógica-aritmética para explotar el paralelismo espacial. A cada una de las unidades empleadas se les denomina Elemento de Procesamiento (PE).

* Sistemas multiprocesador.
  Esta clase de sistemas explotan el paralelismo asíncrono mediante la operación interactiva de diferentes procesadores que comparten recursos tales como memoria, dispositivos de entrada/salida, etc..

Estos tres enfoques de sistemas de cómputo no deben ser considerados como mutuamente excluyentes, ya que en ocasiones se entremezclan para lograr un cierto diseño en particular.

En culaquiera de estas clases de computadoras es importante hacer notar que se pueden tener diseños en base a procesadores del tipo pipeline, o también procesadores del tipo de arreglos, asi como también se puede escoger entre procesadores en paralelo o coprocesadores especiales.

Hasta ahora sólo nos hemos referido a la teoría en la que se basan los diferentes sistemas de cómputo comerciales actuales, sin embargo es necesario describir brevemente la principal tendencia que se está desarrollando entorno a lo que serán las computadoras de la quinta generación, es decir los sistemas del mañana. Debemos considerar que las máquinas convencionales del tipo Von Neumann también se denomian como computadoras de control de flujo, puesto que las instrucciones que componen un cierto programa se ejecutan en forma secuencial, siguiendo él orden establecido por un apuntador de programa (también llamado apuntador de instrucción en algunos casos); como podemos notar, la ejecución de cualquier tarea

efectuada en forma secuencial es inherentemente lenta, lo cual vuelve ineficiente a los sistemas de cómputo.

Con la idea de hacer más eficientes los sistemas de cómputo futuros, surgen algunas arquitecturas de computadoras que pretenden explotar al máximo los conceptos de paralelismo y translape de acciones, tales como las arquitecturas sistólicas, de frente de onda de procesamiento, de flujo de datos entre otras, sin embargo de dichas arquitecturas se estudia principalmente el modelo desarrollado en la Universidad de Manchester (Inglaterra), referente al control de un sistema mediante el flujo de datos a procesar; ésta clase de sistemas recibe el nombre "Computadoras de Flujo de Datos".

El principio fundamental en el cual se basan las computadoras de flujo de datos consiste en activar la ejecución de una instrucción hasta que todos los operandos que intervienen en ella se encuentren listos para ser procesados. Hay que considerar que sólo se utilzarán los operandos requeridos en cada fase del trabajo a realizar, y si en una instrucción se quieren repetir ciertos datos se puedé hacer en ese momento, ya que el control del programa se realiza según el flujo o la necesidad de procesar los datos y no según el flujo de las instrucciones de acuerdo al paradigma de Von Neumann ; por lo tanto, en ésta clase de sistemas no es necesario contar con un contador de programa, o en su defecto con un apuntador de instrucción.

En las computadoras de flujo de datos podemos imaginar que se tienen ciertos nodos de información en los cuales se tienen presentes los datos, y un sector de dichos nodos indica que instrucción se pretende realizar. Estos nodos o paquetes de información se encuentran ligados por la dependencia de datos, por lo tanto podemos decir que las instrucciones son independientes de su ubicación física en un cierto programa. Teoricamente esta clase de sistemas explota la máxima concurrencia de eventos, limitandose unicamente por los aspectos técnicos de la implementación electrónica de la computadora.

Por su parte, los programas que se escriban para este tipo de sistemas en donde se tiene un control por datos, pueden ser efectuados mediante una carta o gráfica de flujo de datos; ésta gráfica en general es diferente de un diagrama de flujo, ya que dicha técnica se desarrolló pensando en las máquinas tradicionales. A continuación se muestra un ejemplo de una parte de un programa escrito para las computadoras de flujo de datos.

La principal ventaja de trabajar con éste tipo de sistemas está en el poder visulizar más fácilmente las dependencias de datos entre varios procesos, situación que se pretende buscar también en los esquemas de programación estructurada

ampliamente utilizados en otras clases de computadoras.

En el caso de los programas empleados en los sistemas de flujo de datos, el escribir rutinas paralelas es mucho más simple que en otros esquemas, como se puede notar en la figura III.1, considerando a manera de ejemplo los cálculos siguientes:

A = B + C - F

D = B * C - F

E = A - D

Como podemos apreciar, los datos de entrada se repiten tantas veces como es necesario para éste caso en particular; la cantidad de veces que se repite un dato depende únicamente de la cantidad de operaciones que requieren ese dato.

En la figura III.2 podemos apreciar cómo se implementaría esta misma secuencia de cálculos, empleando alguno de los lenguajes de alto nivel comunes, por ejemplo BASIC.

En la figura III. 3 se muestra el mecanismo básico para la ejecución de este tipo de algoritmos.

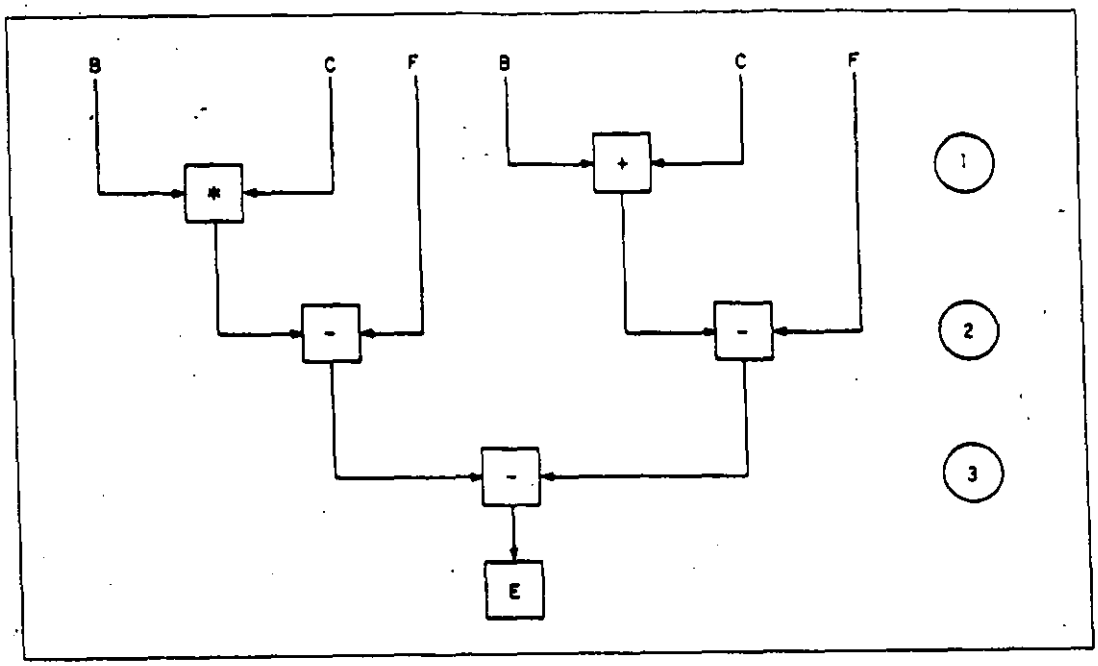
IV. Esquema de Clasificación de Arquitecturas.

Existen varios esquemas para clasificar los diferentes tipos de computadoras digitales que han sido comentadas en este estudio, sin embargo nos basaremos en el propuesto por Michael Flynn, el cual toma como referencia la multiplicidad de secuencias de instrucciones y datos en un sistema de cómputo.

Es importante tener en mente que el procesamiento de datos tiene como finalidad el aplicar un conjunto de instrucciones a un conjunto de datos. Por otra parte, las diferentes organizaciones propuestas se caracterizan también por el tipo de dispositivos que se emplean para soportar las secuencias de datos e instrucciones en uso.

A continuación se muestra la clasificación de las computadoras digitales, según el esquema de Flynn, considerando que una o varias instrucciones pueden actuar sobre uno o varios datos, según el grado de paralelismo desarrollado en cada sistema.

* Organización tipo SISD.
  (Instrucción Simple-Dato Simple o Single Instruction - Single Data)



Fifura III.1 Ejemplo de un algorítmo de flujo de datos.

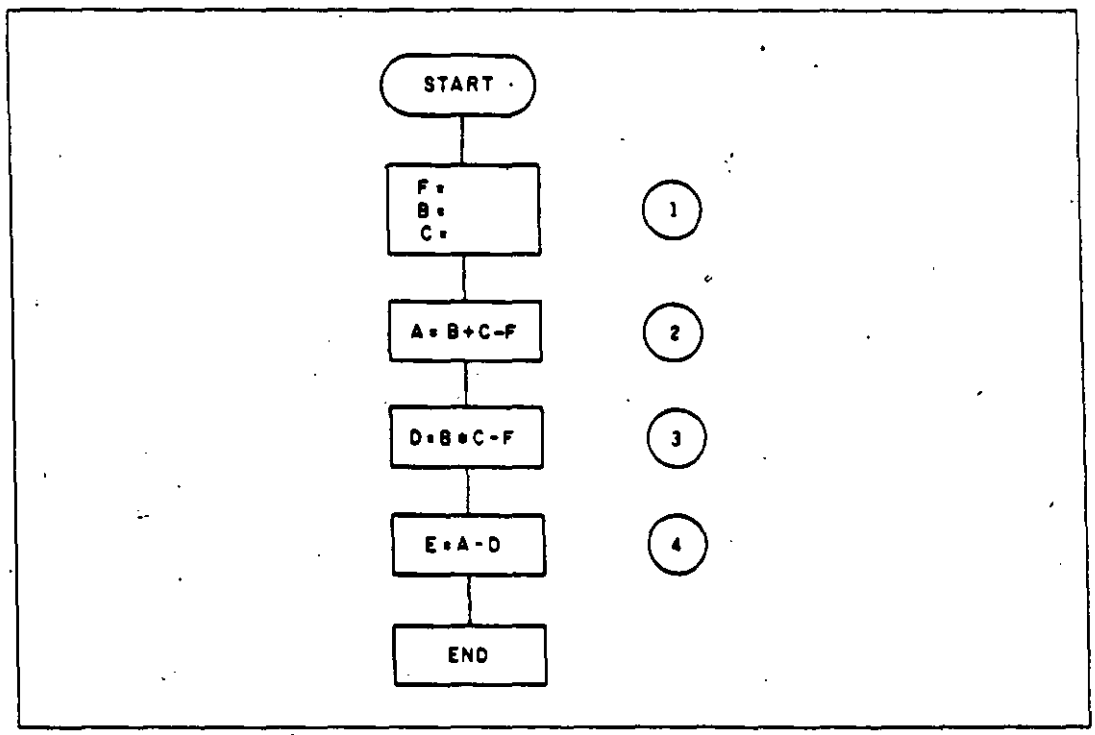


Figura III.2 Como se implementa el mismo algorítmo esquemas convencionales.

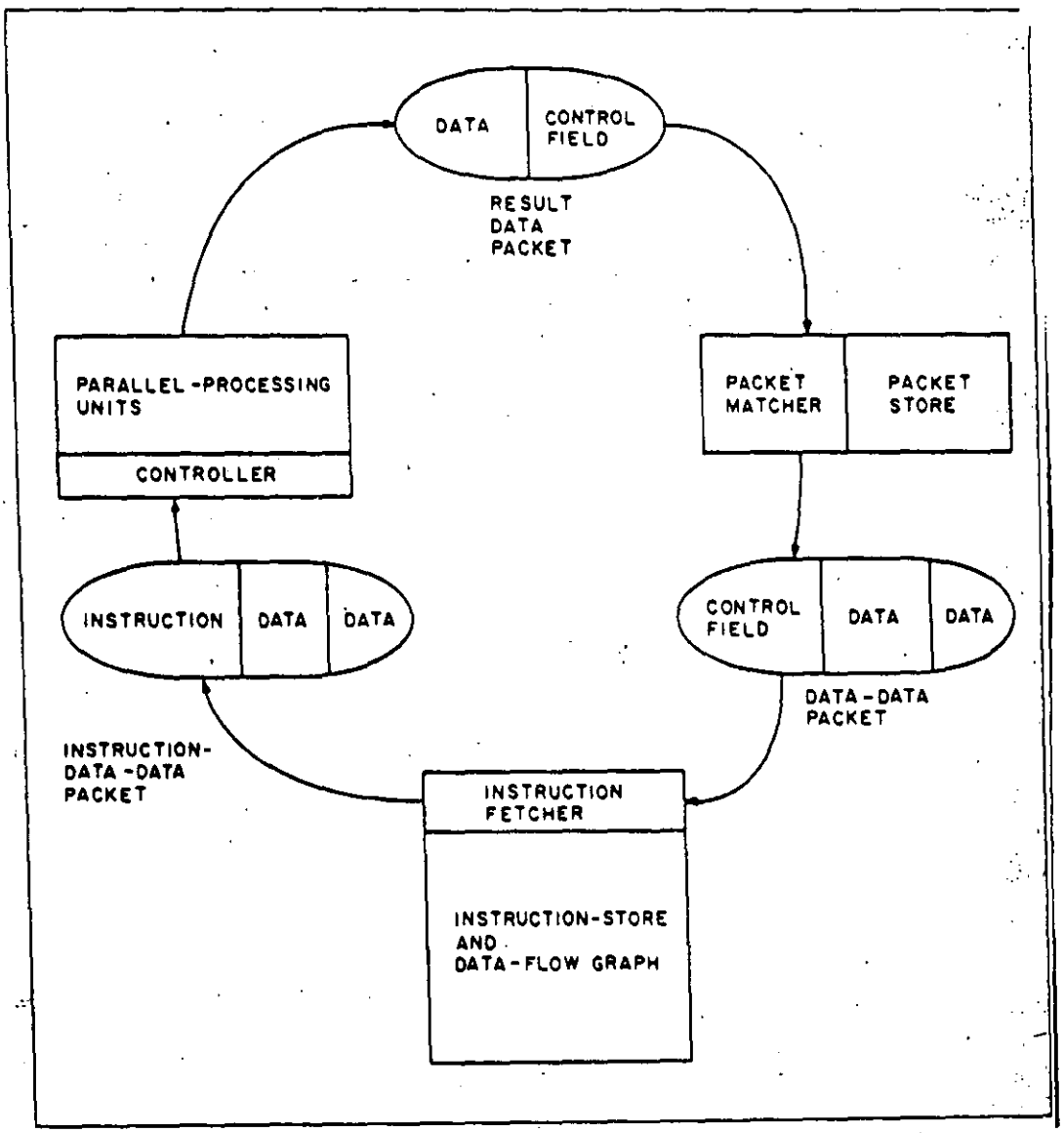


Figura III.3 Mecanísmo básico de una computadora de del tipo de flujo de datos.

* Organización tipo SIMD.
  (Instrucción Simple-Dato Múltiple o Single Instruction Multiple Data)

* Organización tipo MISD.
  (Instrucciones Múltiples-Dato Simple o Multiple Instruction-Single Data)

* Organización tipo MIMD.
  (Múltiples Instrucciones- Múltiples Datos o Multiple Instruction-Multiple Data)

Cada una de las categorías anteriores depende ,como sus nombres lo indican, de la multiplicidad simultánea de eventos que ocurren en los componentes del sistema.

Es necesario considerar que tanto los datos como las instrucciones son capturadas de ciertas celdas o localidades de memoria. Por otra parte, las instrucciones son siempre decodificadas por una unidad de control, la cual le indica a una unidad de procesamiento las acciones que debe ejecutar. Se considera además que una secuencia de instrucciones es generada par sólo por una unidad de control.

A continuación se discute brevemente cada una de las cuatro organizaciones antes propuestas.

* Organización SISD.

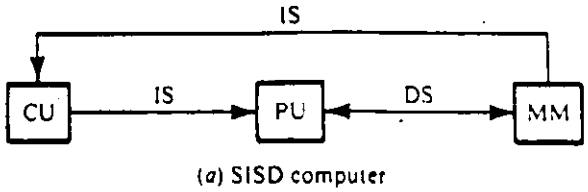
Este tipo de organización se muestra en la figura IV.1, donde notamos que prácticamente se representa a las computadoras actuales tipo serie. En éste caso,las instrucciones son ejecutadas secuencialmente, sin embargo es posible translapar la ejecución de las mismas al emplear el concepto de pipeline por instrucción.

* Organización SIMD.

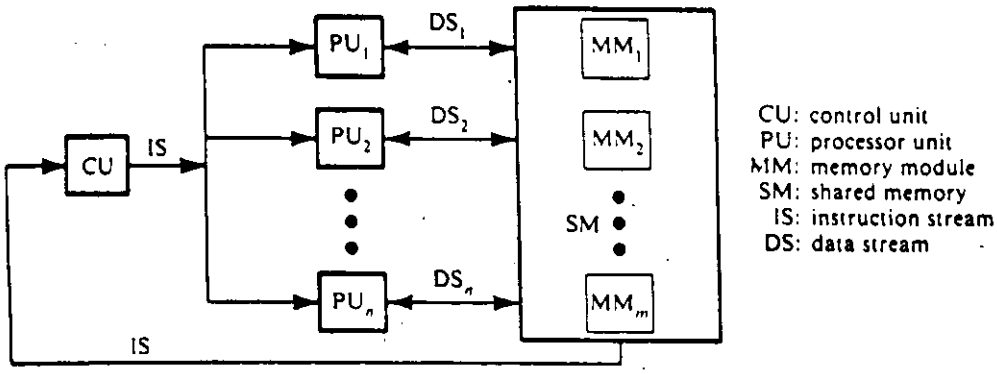
La figura IV.2 muestra la disposición lógica de esta organización, la cual corresponde a los procesadores en arreglo, mismos que ya se comentaron antes, donde notamos que pueden existir múltiples elementos de procesamiento, todos ellos supervisados por la misma unidad de control. Es importante hacer notar que todos los elementos de procesamiento reciben la misma instrucción, pero operan en diferentes datos de diferentes secuencias de información .

Es posible subdividir las máquinas SIMD en los tipos Word-Slice y Bit-Slice.

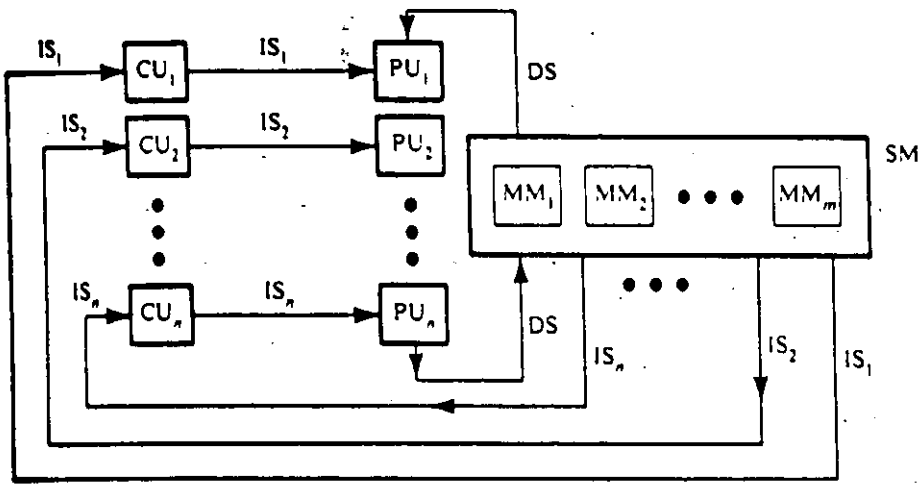
FIguras IV.1, a IV.2 Referentes a las arquitecturas SISD,SIMD, MISD y MIMD.



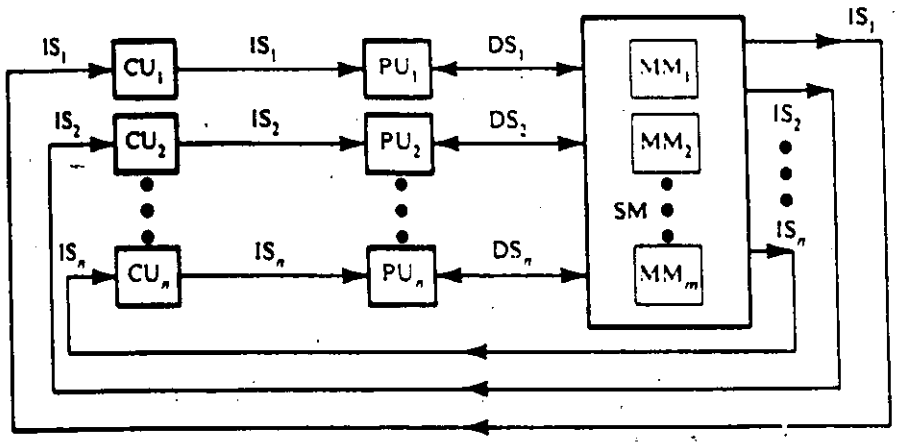
(*a*) SISD computer



(*b*) SIMD computer



(*c*) MISD computer



(*d*) MIMD computer

* Organización MISD.

Esta organización se muestra en la figura IV.3, en este caso se tienen N unidades de procesamiento, donde cada una recibe diferentes instrucciones pero todas ellas actúan sobre el misma secuencia de datos. Esta organización se juzga generalmente como impráctica y por lo tanto es remota su implementación en algún tipo de máquina actual.

* Organización MIMD.

En la figura IV.4 podemos apreciar ésta organización, la cual es muy popular en los sistemas actuales, ya sea a nivel de sistemas multiprocesador o bien, a nivel de un sistema con multiples computadoras como podría ser una red de computadoras.

Una computadora del tipo MIMD implica la interacción de los N microprocesadores que la constituyen, ya que los datos generalmente se obtienen de memorias de acceso compartido.

V. Sistemas en base a Microprocesadores.

Como ya se ha comentado, los sistemas de cómputo actuales hacen un uso intensivo y eficiente de diferentes técnicas de programación, procurando en todos los casos basarse en una arquitectura y en una organización robustas, las cuales se desarrollan en algún entorno de ciertos microprocesadores. Debemos hacer hincapié en que existen diferentes tipos de procesadores, ya sean integrados o no, sin embargo todos ellos llegan a presentar características comunes.

El objetivo de esta sección es analizar ciertos conceptos que generalmente son necesarios para desarrollar o estudiar ciertos sistemas basados en microprocesadores.

Como es fácil de entender, dependiendo de la arquitectura interna del procesador, podemos agrupar a los microprocesadores en dos grandes categorías que son:

i) Microprocesadores de propósito general.
Esta clase de dispositivos está disñada para poder efectuar casi cualquier clase de algoritmo, sin importar mucho la aplicación; cuentan generalmente con un buen repertorio de tanto de instrucciones como de registros internos. En estos procesadores son importantes el número de registros internos, el total de instrucciones posibles, el total de los modos de direccionamiento, la frecuencia de operación y las ca-

pacidades de sus canales de datos y direcciones principalmente.

ii) Microprocesadores de propósito particular.
Los microprocesadores de propósito particular son dispositivos del tipo VLSI, que cuentan con una arquitectura altamente especializada para soportar ciertos algoritmos en forma altamente eficiente, generalmente a ésta clase de circuitos también se les llama coprocesadores, su principal función es la de liberar al procesador central de ciertas tareas en particular. Cuando se emplean esta clase de dispositivos se está tendiendo a explotar el paralelismo, mediante algún tipo de arquitectura como podría serlo un sistema SIMD.

Es importante hacer notar que mientras en un microprocesador de propósito general se programa dependiendo de una secuencia de pasos indicada por el usuario mediante un programa, un microprocesador de propósito particular es un esclavo del anterior, y éste es quien lo programa generalmente eviandole un cierto código muy particular.

A grandes rasgos podemos clasificar al los microprocesadores de propósito particular en la siguiente manera:

* Coprocesadores de comunicaciones.

* Coprocesadores numéricos.

* Coprocesadores gráficos.

* Coprocesadores de soporte.

* Coprocesadores de procesamiento digital de señales.

En todos estos casos se establece una comunicación bidireccional entre el o los procesadores centrales y cada coprocesador, considerando que los coprocesadores ejecutan las instrucciones que les sean indicadas por el procesador maestro. Por otra parte la atención a cada coprocesador se puede efectuar empleando algunas de las técnicas populares de poleo o interrupciones.

Los coprocesadores de comunicaciones se encargan generalmente de controlar las comunicaciones del sistema, las cuales pueden ser en serie o paralelo, en forma síncrona o asíncrona.

Los coprocesadores numéricos se emplean generalmente para efectuar operaciones matemáticas a gran velocidad, siendo estas en la mayoría de los casos de tipo aritmético, sin embargo hay algunos que pueden generar alguna función como podría

serlo sen x, log x, etc. Estos coprocesadores pueden manejar cualquier técnica de representación, tales como el punto fijo y el punto flotante, según el modelo propio para cada aplicación.

Los coprocesadores gráficos han sido diseñados principalmente para encargarse de un manejo eficiente de las gráficas que pueden ser creadas en la pantalla de un sistema de cómputo, generalmente también se les conoce como controladores de video.

Los coprocesadores orientados al procesamiento digital de señales son similares a los anteriores, sin embargo estan diseñdos para poder efectuar muy ágilmente algunos de los algoritmos para el análisis de señales, como podría serlo la Transformada Rápida de Fourier (FFT).

Los coprocesadores de soporte pueden ser entendidos como todos aquellos microprocesadores esclavos que pueden ayudar al mejor funcionamiento del sistema, como podrían serlo por ejemplo los manejadores de memoria (MMU), los controladores de interrupciones, (en cualquier nivel, como por ejemplo, DMAC, controladores de interrupciones, etc.), contadores y convertidores programables, generadores y verificadores de códigos (CRC, SDLC, etc.), entre otros.

En general, en un sistema basado en microprocesadores, existe uno o varios procesadores centrales y cada uno cuenta con una familia de coprocesadores, mismos que han sido diseñados y fabricados para optimar la operación del sistema mediante la eficiente comunicación entre procesadores.

Por otra parte, en lo referente a los microprocesadores de propósito general, podemos decir que se clasifican según el número de bits que pueden ser procesados en un instante dado, dicho número corresponde generalmente a la capacidad del registro que haga las veces de acumulador. Actualmente existen microprocesadores con una capacidad de procesamiento de 4, 8, 16, y 32 bits, fabricados bajo varias tecnologías pero principalmente es empleado el tipo pipeline; a excepción de la categoría de 4 bits, ya que ésta es generalmente del tipo bit-slice. En la categoría de 8 bits existen varios modelos comerciales, sin embargo se pueden centrar en dos tendencias, aquellos orientados al procesamiento de información en un sistema con varios usuarios como el popular 6800, y aquellos orientados al manejo de dispositivos de entrada/salida, como podría ser el famoso Z-80.

Actualmente los microprocesadores más empleados son de 16 y 32 bits, sin embargo, en ambos casos los fabricantes han efectuado diseños compatibles entre tecnologías, basándose principalmente en los procesadores de 16 bits, por lo tanto a continuación mostramos una comparación entre los micropro-

cesadores de 16 bits más populares.

De lo anterior podemos notar que los microprocesadores de Motorola se orientan principalmente a soportara sistemas del tipo MIMD destinados a la multiprogramación, mientras que los microprocesadores de INTEL se orientan a soportar sistemas MIMD destinados principalmente al multiprocesamiento por vía de sus técnicas de multiprocesadores. Por su parte ZILOG se orienta principalmente a soportar los sistemas que interactúan grandemente con dispositivos de entrada /salida.

VI. Criterios de Selección de Microprocesadores.

En lo referente a los criterios de selección de microprocesadores se puede decir que no hay una técnica muy desarrollada, ya que en ocasiones se escoge un microprocesador porque sólo ese se conoce, sin embargo cuando se piensa diseñar un sistema digital de procesamiento es conveniente seguir estas recomendaciones:

1. Analizar detenidamente la aplicación que se pretende resolver.

2. Determinar la resolución y la velocidad con la que se desea procesar la información, esto nos determinará si empleamos 8, 16 o 32 bits y a que frecuencia de trabajo.

3. Determinar el tipo de interfaces que deben ser utilizadas con la finalidad de poder seleccionar los coprocesadores adecuados.

4. Determinar la capacidad y el tipo de memorias (caché y primaria) que se desean emplear.

5. Determinar el grado de paralelismo que deseamos emplear, lo cual nos indicará qué tipo de arquitectura se debe emplear, ya sea SISD, SIMD o MIMD.

6. Con la información anterior podemos definir si se desea principalmente un sistema orientado a la multiprogramación, al multiprocesamiento o a la interacción de entrada/salida.

En resumen, al aplicar adecuadamente esta secuencia a nuestro problema, podemos tener un criterio que nos permita una correcta selección del microprocesador para un sistema digital de cómputo.

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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

14 DE AGOSTO AL 12 DE SEPTIEMBRE 1992

ANEXO

ING. ABEL CLEMENTE REYES

AGOSTO-SEPTIEMBRE
1992

as possible. The implications of any constraining parameters introduced by the customer must be thoroughly discussed, to prevent alterations in the course of the undertaking.

In addition to the project's functional aims, other considerations must be detailed, as follows.

Accuracy

The expense and complexity of a system tends to be an exponential function of the demanded accuracy. Over-enthusiasm by the customer (or even designer!) may well radically alter the cost, and therefore viability, of the project.

Range and Resolution

This covers consideration of maximum and minimum values: how often events occur, and to what precision the end result is displayed. Their influence on cost is not so radical as that due to accuracy considerations.

Environmental

Ambient temperature and humidity are covered by this category. For example, in a greenhouse, an installation may have to withstand sprinkling by automatic watering systems. The electrical environment must also be included. Considerations should include battery or mains supply; and if the latter, the problem of noise and voltage fluctuation.

Ergonomic

System operation should be straightforward, without requiring frequent resort to an operating manual. This is particularly important where nonskilled personnel are involved. Frequently the look of an instrument is sufficient to sell it.

Failure

This involves considerations for dealing with malfunction, such as a mains failure (e.g. should a battery backup be employed?) or a temporary absence of an expected signal. Obviously these and similar factors are more important in critical applications; such as in the biomedical field.

Maintenance

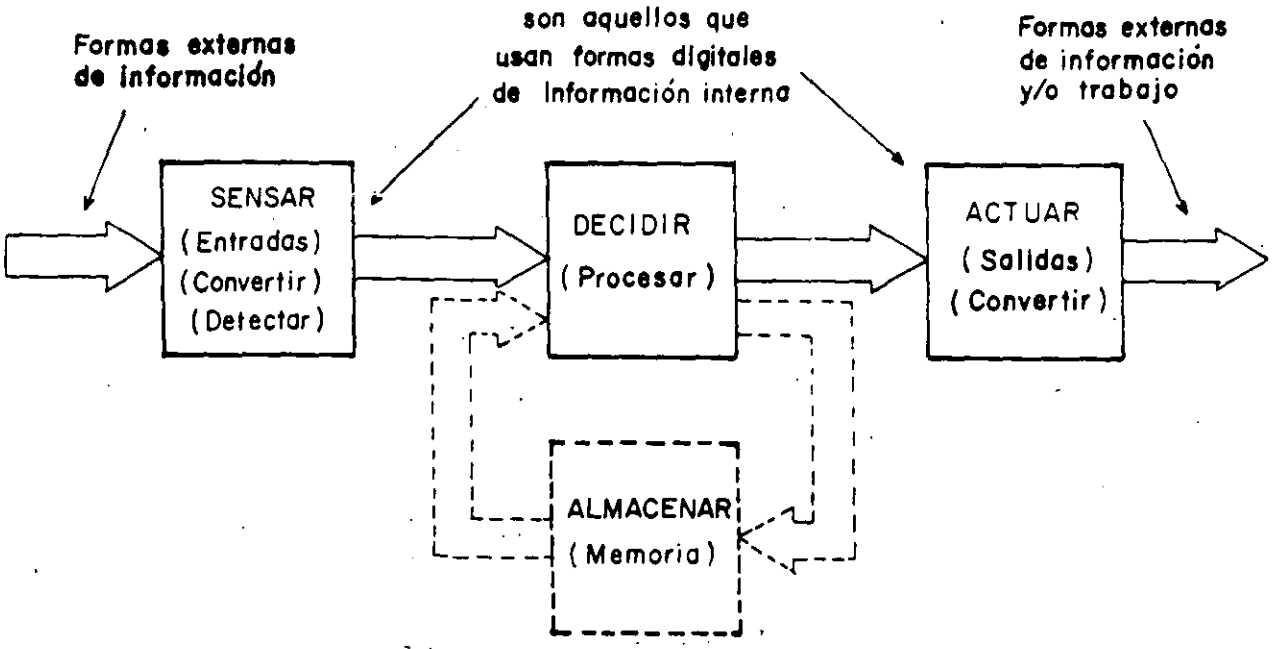
Documentation, service contracts and information are covered under this heading.

Once the overall picture is clear, the architecture of the system must be formulated. To a large extent this depends on the transducers, since these form the interface between the electronics and the real world. The types selected will depend on the range, accuracy, environmental and cost considerations. The choice is not unduly influenced by the technology used for the central processing electronics.

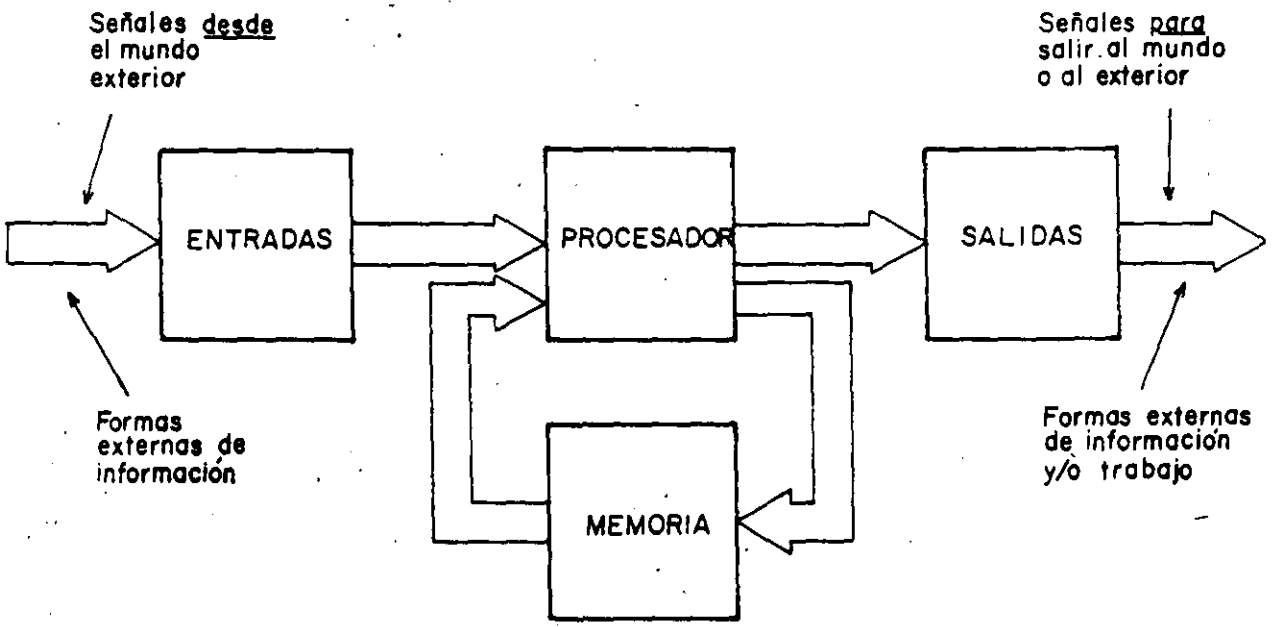
With the transducers selected, we can now turn our attention to the processing circuitry. In a digital system, this may be microprocessor or wholly conventional logic-based. Generally microprocessors do everything sequentially, one at a time, and are unsuitable for microsecond response times. However, in many cases a

SISTEMAS DIGITALES

son aquellos que usan formas digitales de información interna



ORGANIZACION UNIVERSAL DE SISTEMAS DIGITALES



CONSTRUCCION A BLOQUES DE COMPUTADORAS

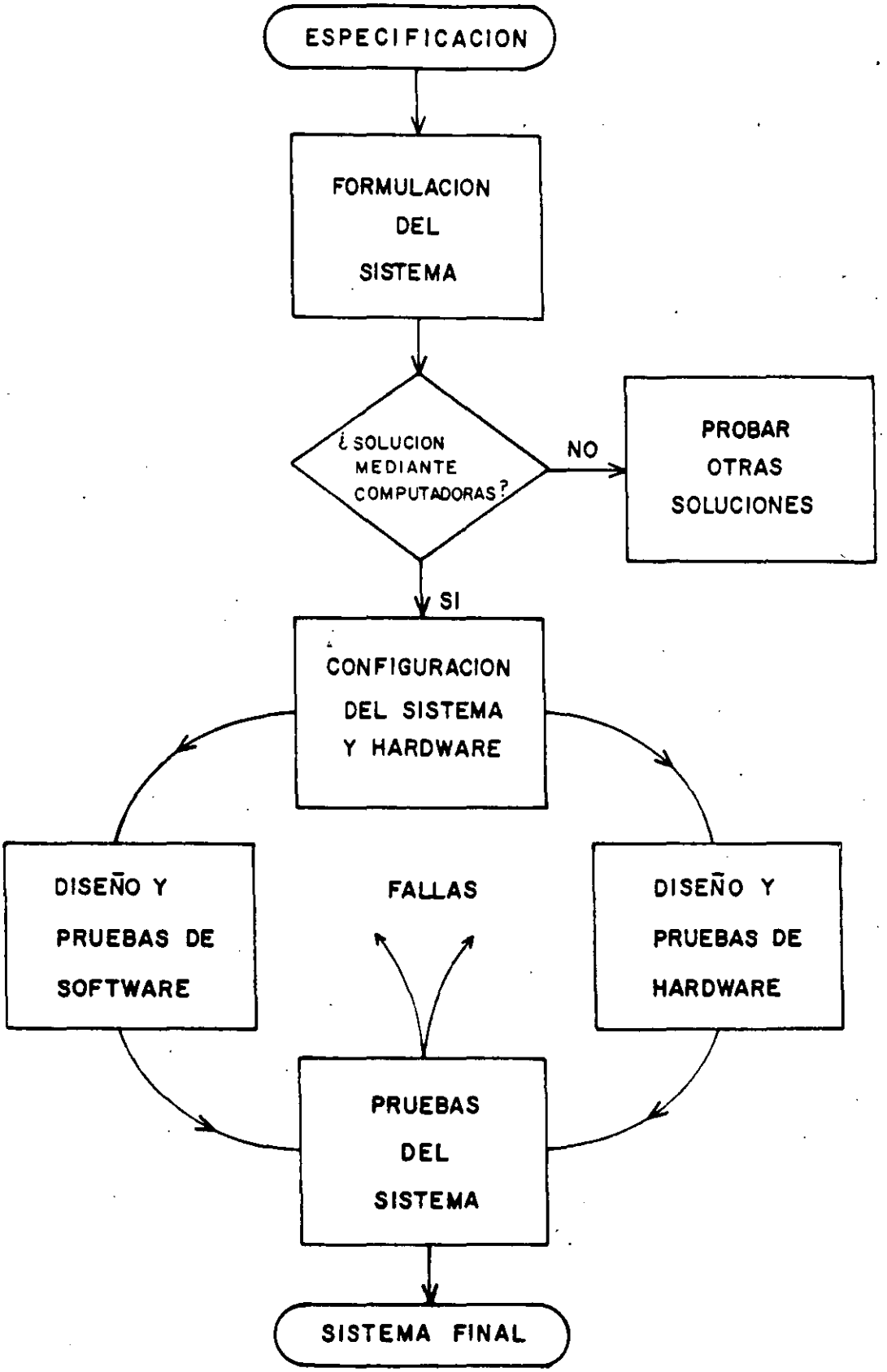
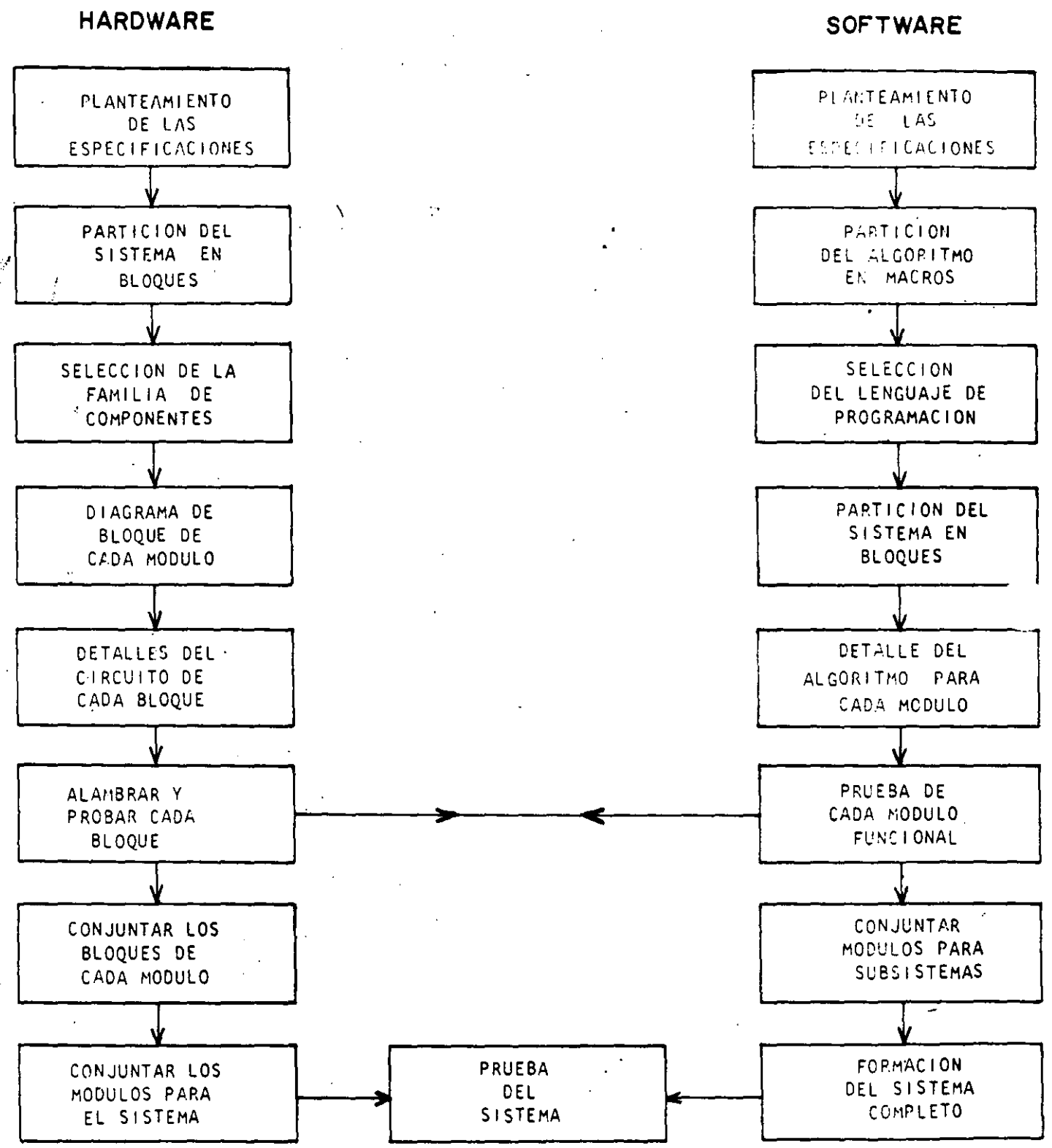
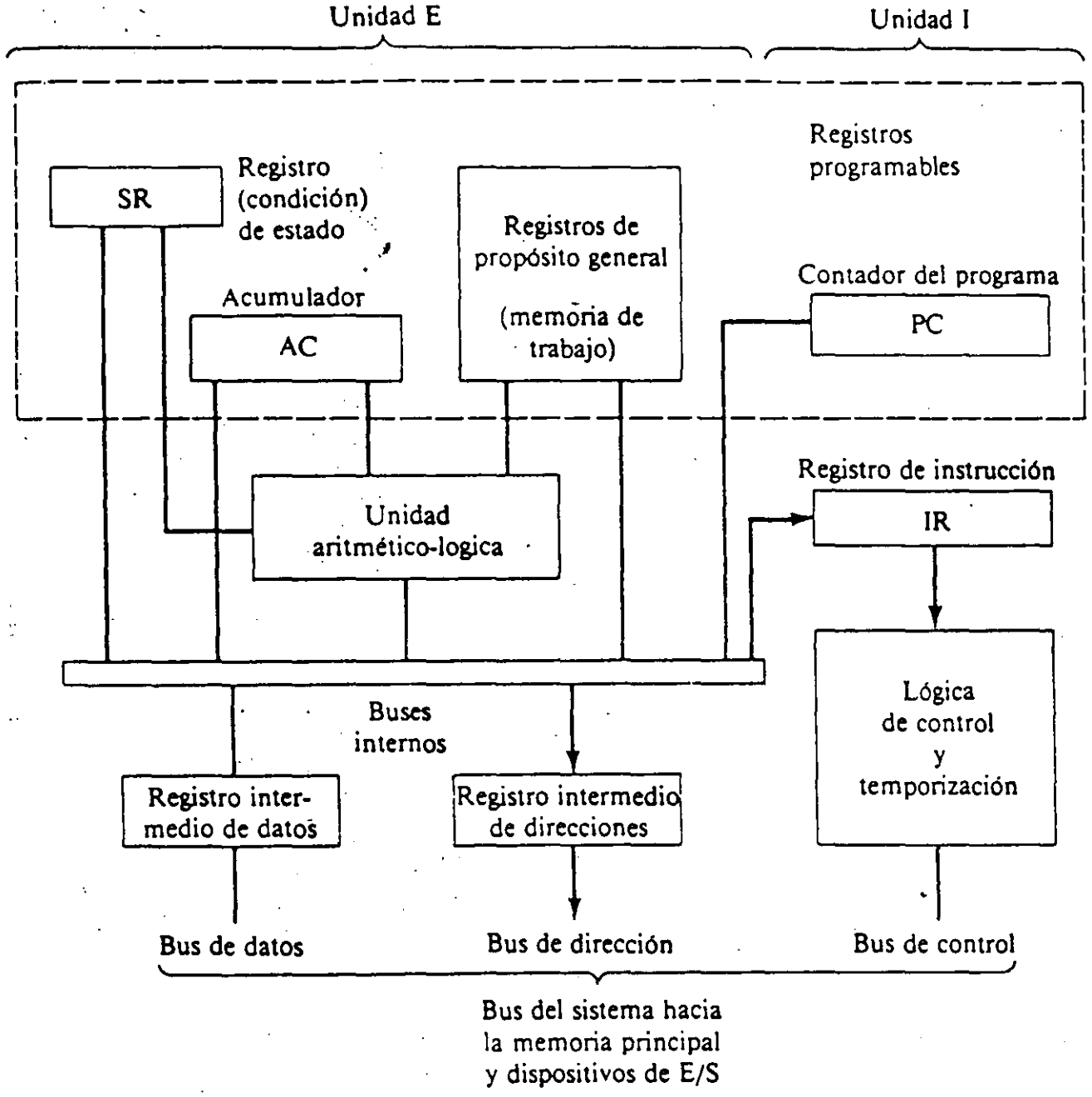


DIAGRAMA DE FLUJO GENERAL DE DISEÑO

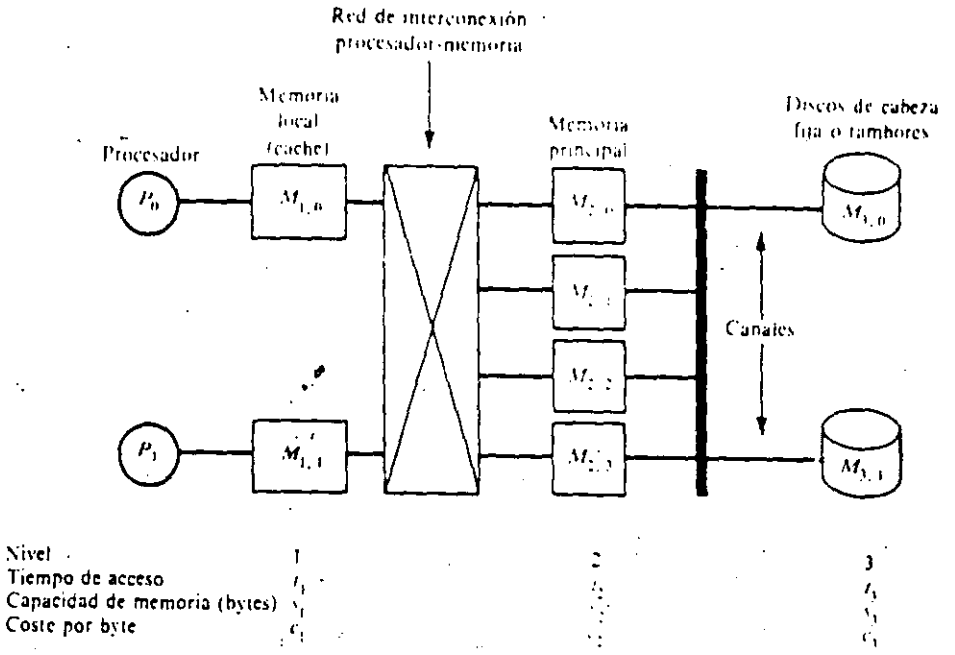


TECNICA DE DISEÑO TOP-DOWN

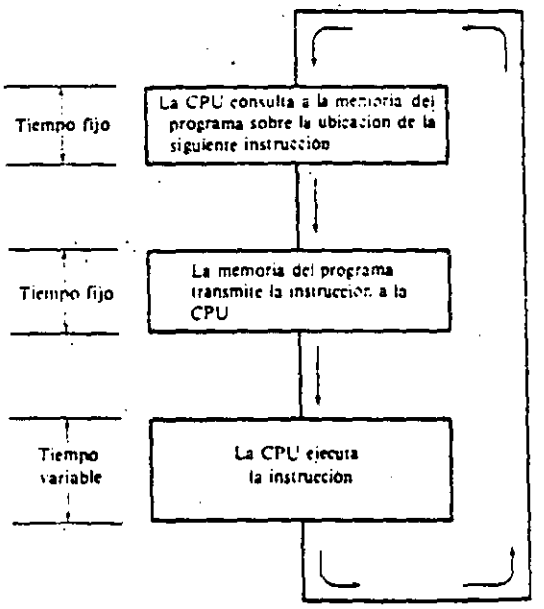
(DISEÑO FUNCIONAL DESCENDENTE)



Organización interna de un microprocesador típico.



Jerarquía de memoria de tres niveles.



Etapas básicas en un ciclo de instrucción.

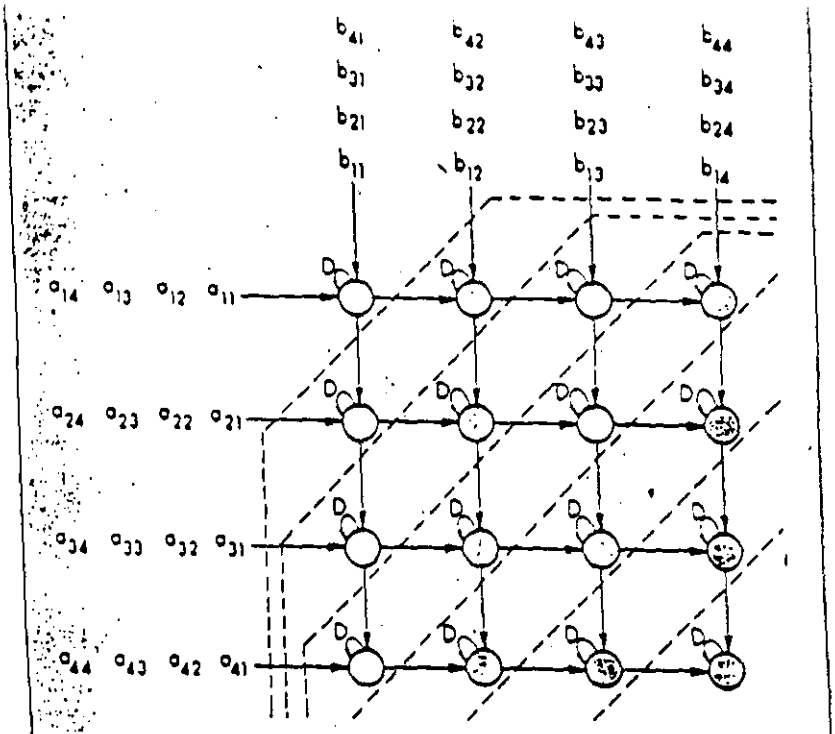


Figure 2-3 An SFG Array for Matrix Multiplication. A straightforward SFG array design is to broadcast the columns A and rows B (cf. Eqns. (2) and (3)) instantly along a square array, such as the 4 × 4 array shown in the figure. Multiply the two data meeting at node (i,j) and add the product to c_ij, the data value currently residing in a register in node (i,j). Finally, the new result will update the register via a feedback with a delay D and get ready to interact with the new arriving operands. As all the column and row input data continue to arrive at the nodes, all the outer products will be sequentially summed. Although this design is not directly suitable for a VLSI circuit design due to the use of global communication, it may be converted to a systolic array, as shown in Figure 2-4, or a wavefront array, shown in Figure 2-5. A simple conversion strategy will be provided in Section 3.

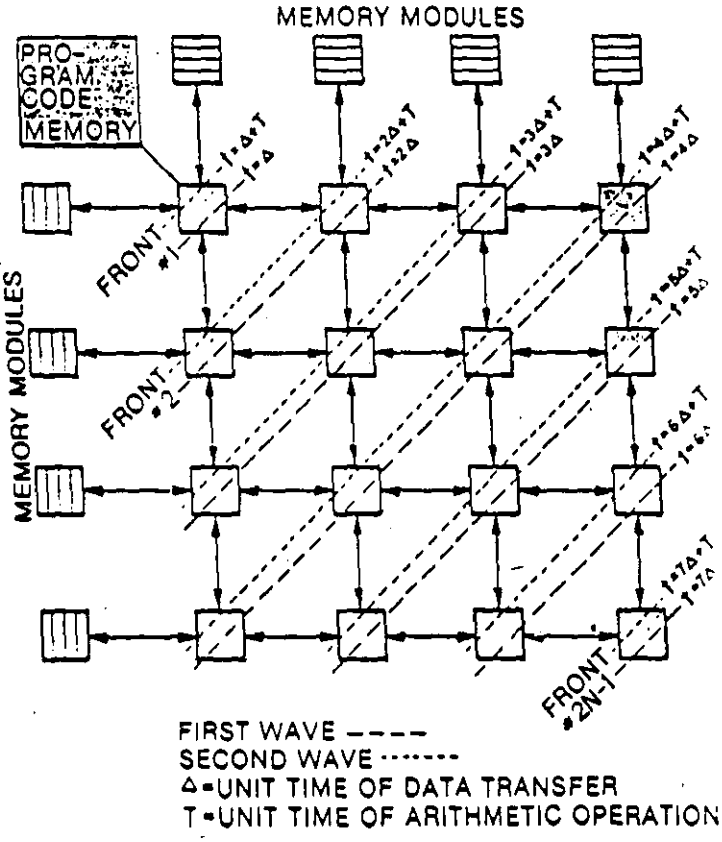


Figure 2-5 Wavefront processing for Matrix Multiplication. In this example, the wavefront array consists of N × N processing elements with regular and local interconnections. The figure shows the first 4 × 4 processing elements of the array. The computing network serves as a (data) wave propagating medium. Hence the hardware will have to support data tagging of the computational wavefronts as fast as resource and data availability allow. This can often be accomplished simply by means of a handshaking protocol, such as that proposed in [11]. The (average) time interval (T) between two separate wavefronts is determined by the availability of the operands and accordingly in this case. This is due to the time needed for the arithmetic operations, multiply-and-add. The speed of wavefront propagation is determined by the time interval Δ, which in this case is equivalent to the data transfer time. In general, the major characteristics of wavefront arrays are (i) self-timed, data-driven computation, meaning that no global clock is needed, (ii) regularity, modularity and spatial locality of interconnections, and (iii) effective pipelining

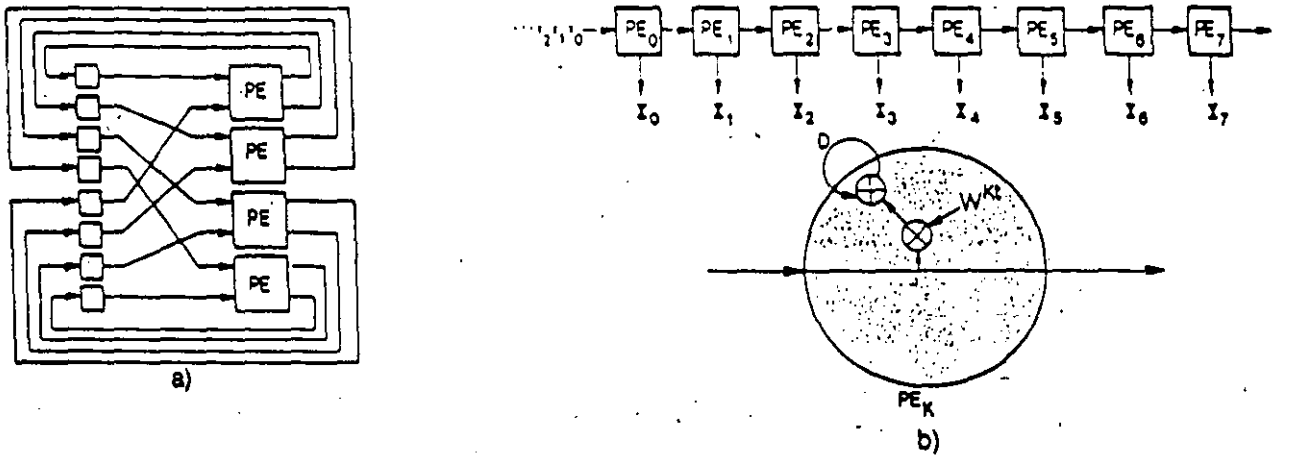


Figure 4—1: Array Processor Architectures for (a) FFT Algorithm; and (b) DFT Algorithm. Note that, in terms of total processing times, the ratio is log N vs. N in favor of FFT. On the other hand, the FFT array requires a *global* (Perfect-Shuffle) communication network. In contrast, the DFT array can be easily systolized and implemented in a modular processor array with *local* communication. Note that the weighting factors W^{kt} are time-varying; and the Fourier transform output $\{X_k\}$ stays in the node and will eventually be pumped out.

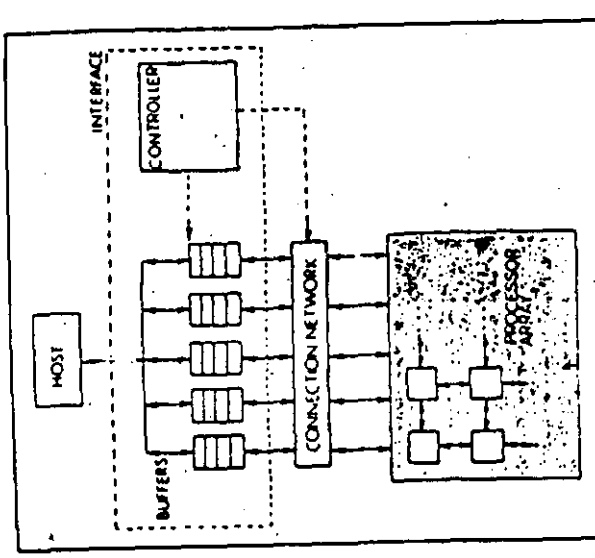


Figure 5-1: An example of Array Processing System Configuration. An array processor is often used as an attached processor linked with a host through an interface system.

Host Computer. The host computer should provide batch data storage, management, and formatting; determine the schedule programs that controls the interface system and connection network; and generate and load object codes to the PE's. A very challenging task to the system designer is to identify a suitable host machine for interfacing with high speed array processor units.

Interface System. The interface system, connected to the host via the host bus, has the functions of downloading and up-loading data. Based on the schedule program, the controller monitors the interface system and array processor. The interface system should also furnish an adequate hardware support for many common data management operations. Other challenging tasks for the system designer are managing blocks of data and making sure the memory (buffer) unit is able to balance the low bandwidth of system I/O and the high bandwidth of array processors.

Connection Network. Connection networks provide a set of mappings between processors and memory modules to accommodate certain common global communication needs. Incorporating certain structured interconnections may significantly enhance the speed performance of the processor arrays.

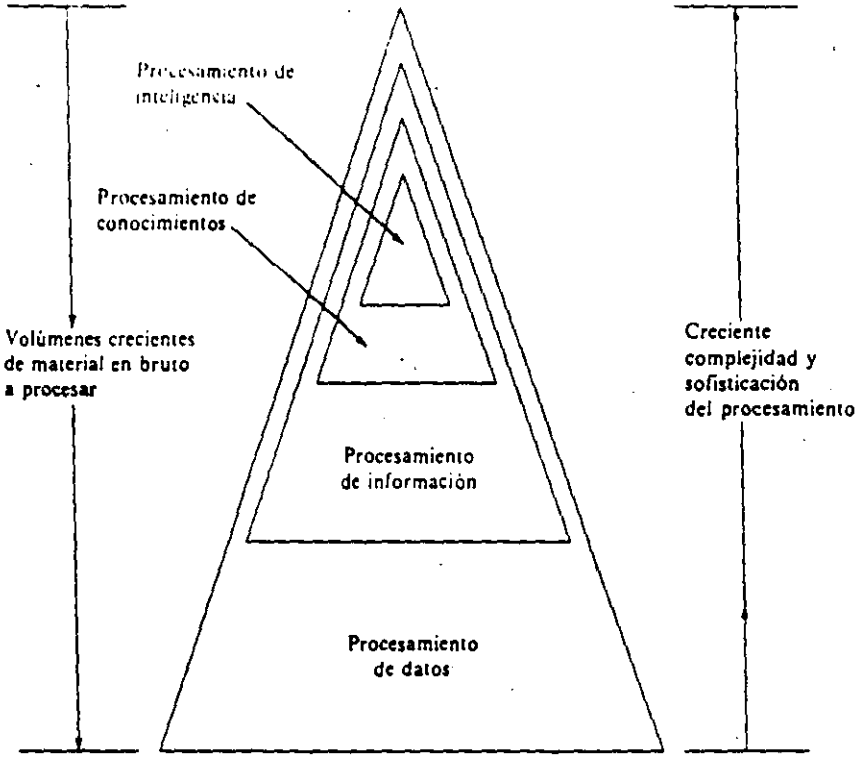
Processor Arrays. For simplicity, only one processor array is physically depicted in the figure. However, the concept of networking several processor arrays has now attracted a good deal of attention. For example, when a problem is reducible to several subproblems that can be executed one after the other, it will be useful to have each subproblem executed in its own processor array, while utilizing the network to facilitate the data pipelining between the arrays. This suggests a pipelining scheme at the array-level, which may increase the processing speed-up by one more order of magnitude.

PROCESAMIENTO DE CONOCIMIENTOS

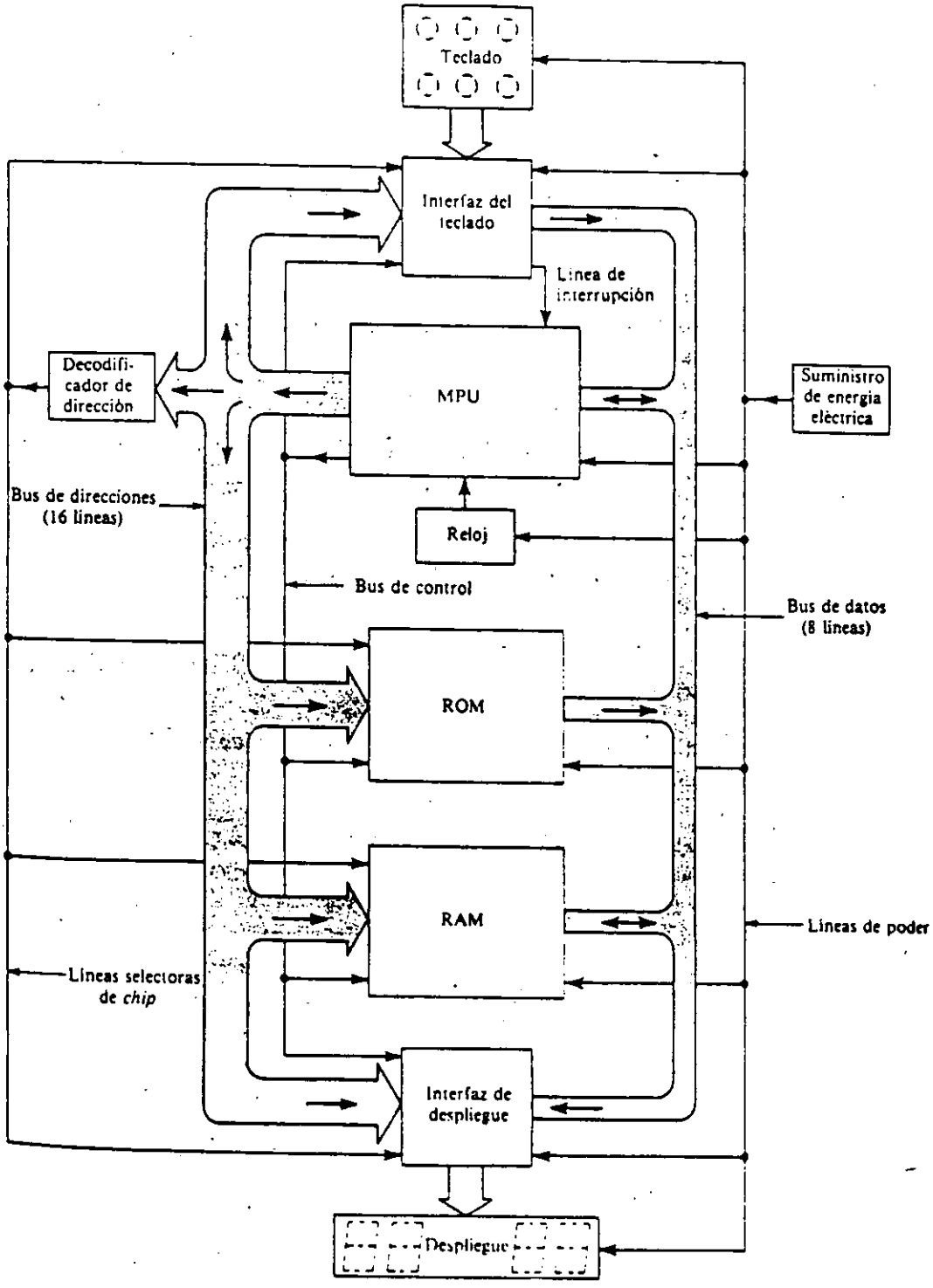
MANIPULACIÓN DE CONOCIMIENTOS, ENTENDIENDO COMO CONOCIMIENTOS A TODOS NODOS DE INFORMACIÓN QUE CONTIENEN UN SIGNIFICADO SEMÁNTICO.

PROCESAMIENTO DE INTELIGENCIA

CORRELACIÓN DE DIFERENTES NODOS (ITEMS) DE CONOCIMIENTOS, A FIN DE QUE EL SISTEMA APRENDER, ADAPTARSE O AUTOCORREGIRSE, DE MANERA SEMEJANTE A LA INTELIGENCIA HUMANA.



Los espacios de datos, información, conocimiento e inteligencia desde el punto de vista del procesamiento por computador.



Arquitectura de una microcomputadora

PROCESAMIENTO DE DATOS

TRATAMIENTO DE DATOS LOS CUALES SON GENERALMENTE CARACTERES DE TIPO ALFANUMÉRICO, LOS CUALES NO NECESARIAMENTE GUARDAN UNA CIERTA ESTRUCTURA SINTÁCTICA.

PROCESAMIENTO DE INFORMACIÓN

TRATAMIENTO DE NODOS O ARTÍCULOS (ITEMS) DE INFORMACIÓN, MISMAS QUE GENERALMENTE SON CONJUNTOS DE DATOS RELACIONADOS POR CIERTA ESTRUCTURA SINTÁCTICA.

PROCESAMIENTO DE CONOCIMIENTOS

MANIPULACIÓN DE CONOCIMIENTOS, ENTENDIENDO COMO CONOCIMIEN A TODOS NODOS DE INFORMACIÓN QUE CONTIENEN UN SIGNIFICADO SEMÁNTICO.

PROCESAMIENTO DE INTELIGENCIA

CORRELACIÓN DE DIFERENTES NODOS (ITEMS) DE CONOCIMIENTOS, A FIN DE QUE EL SISTEMA APRENDER, ADAPTARSE O AUTOCORREGIRSE, DE MANERA SEMEJANTE A LA INTELIGENCIA HUMANA.

CONCEPTOS BASICOS

SISTEMA DE CÓMPUTO

CONJUNTO DE ELEMENTOS ÍNTIMAMENTE RELACIONADOS ENTRE SÍ, CUYO PROPÓSITO ES PROCESAR INFORMACIÓN, DE ACUERDO CON LA SECUENCIA DE PASOS PROPIA DE UN CIERTO ALGORITMO PREFIJADO.

COMPUTADORA DIGITAL

MÁQUINA CAPAZ DE MANIPULAR DÍGITOS BINARIOS (DATOS), SIGUIENDO UNA SECUENCIA ORDENADA DE PASOS (PROGRAMA). CADA PASO DE DICHA SECUENCIA SE DENOMINA INSTRUCCIÓN.

LAS COMPUTADORAS, INDEPENDIENTEMENTE DE SU TAMAÑO, SON MÁQUINAS QUE CUENTAN CON LOS SIGUIENTES ELEMENTOS CONSTITUTIVOS:

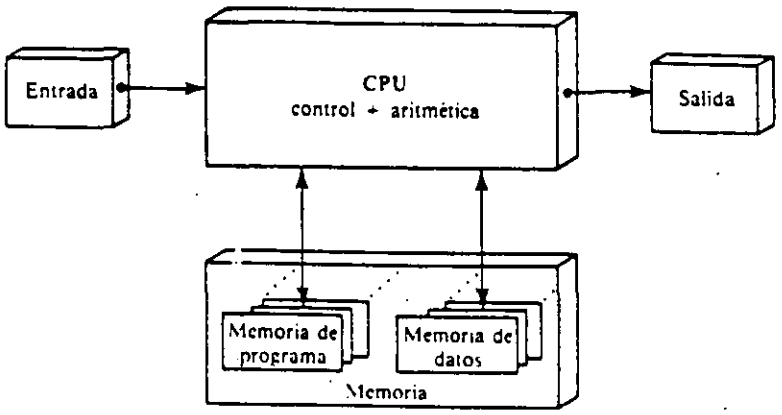
A) UN MEDIO DE ENTRADA POR EL CUAL LAS INSTRUCCIONES Y LOS DATOS PUEDEN INGRESAR AL SISTEMA.

B) UNA MEMORIA, DESDE LA CUAL SE PUEDEN OBTENER TANTO LOS DATOS COMO LAS INSTRUCCIONES, PUDIÉNDOSE ALMACENAR TAMBIÉN LOS RESULTADOS OBTENIDOS.

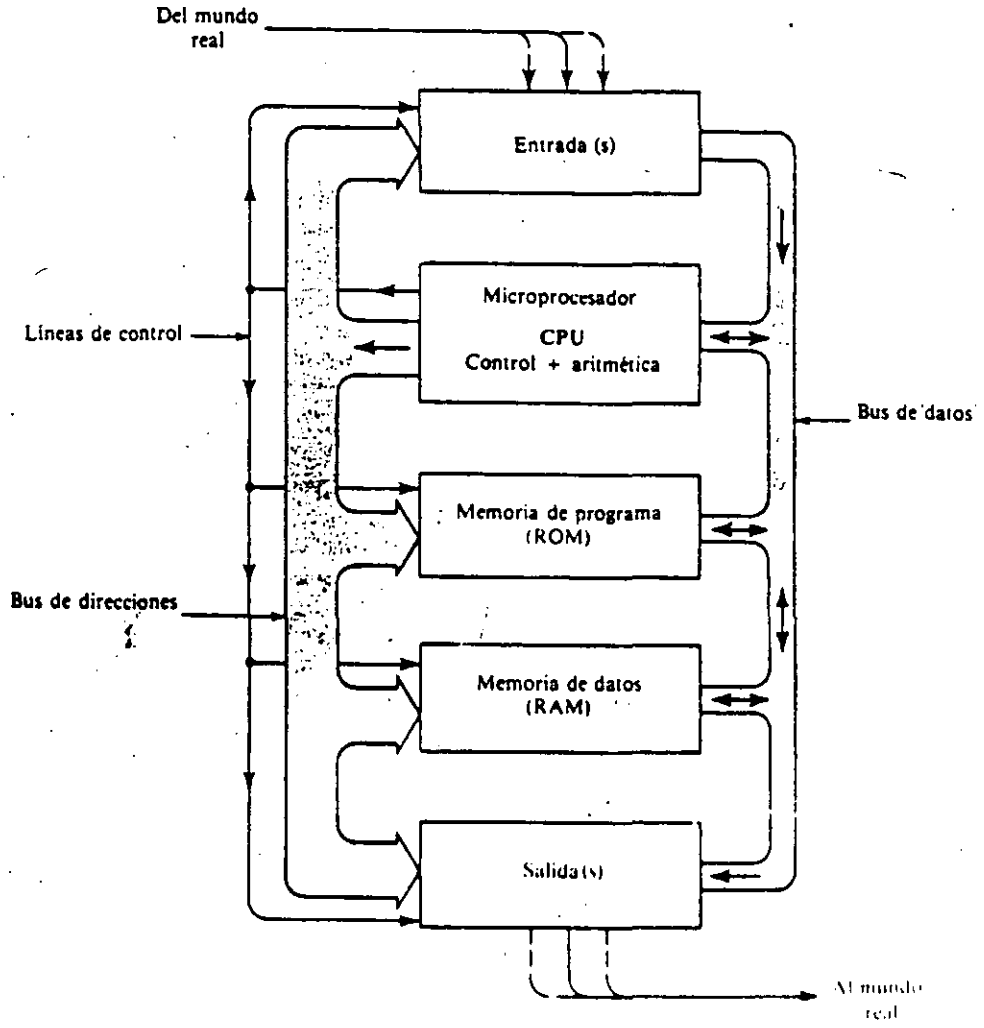
C) UNA SECCIÓN DE CÁLCULO, LA CUAL ES CAPAZ DE REALIZAR OPERACIONES LÓGICAS O ARITMÉTICAS SOBRE CUALQUIER DATO PROVENIENTE DE LA MEMORIA.

D) UNA SECCIÓN DE CONTROL, CAPAZ DE TOMAR LAS DECISIONES REFERENTES A LOS DIFERENTES CURSOS QUE DEBEN TOMAR EL PROCESAMIENTO.

E) UN MEDIO DE SALIDA POR EL CUAL LOS RESULTADOS PUEDEN SER PROPORCIONADOS AL USUARIO.



Organización general de una computadora



Memoria

- Conjunto de elementos que almacenan y retienen temporalmente la información que va a ser procesada por el CPU.

- Las celdas o registros de memoria habitualmente están constituidos por elementos digitales como: transistores, flip-flop's, registros, contadores, etc...

- La memoria se puede clasificar como: Memoria chaché, memoria primaria y memoria secundaria.

- En cuanto a su tipo, las memorias pueden ser:

Lectura y escritura	Permanentes	Acceso directo	Estática
Sólo Lectura	Volátiles	Acceso Secuencial	Dinámicas

- Los principales parámetros a considerar en la selección de memorias son:

  - Tiempo de Acceso
  - Modo de Acceso
  - Capacidad y Organización
  - Modo de Grabación

TIEMPO DE ACCESO:

ES EL TIEMPO TRANSCURRIDO DESDE QUE LA UNIDAD DE LÓGICA-ARITMÉTICA NECESITA ALGUNA INFORMACIÓN (QUE TOTALMENTE DISPONIBLE PARA SU PROCESAMIENTO. TAMBIÉN ES EL TIEMPO TRANSCURRIDO DESDE QUE ALGUNA INFORMACIÓN EN LA UNIDAD DE LÓGICA-ARITMÉTICA ESTÁ LISTA PARA ALMACENARSE HASTA EL MOMENTO QUE SE ALMACENA EN LA UNIDAD DE MEMORIA.

MODO DE ACCESO:

EXISTEN DOS FORMAS EN QUE SE PUEDE TENER ACCESO A UNA DIRECCIÓN DE MEMORIA: SECUENCIAL Y ALEATORIA (O DIRECTA). LA FORMA SELECCIONADA POR LO GENERAL DEPENDE DEL DISPOSITIVO EMPLEADO.

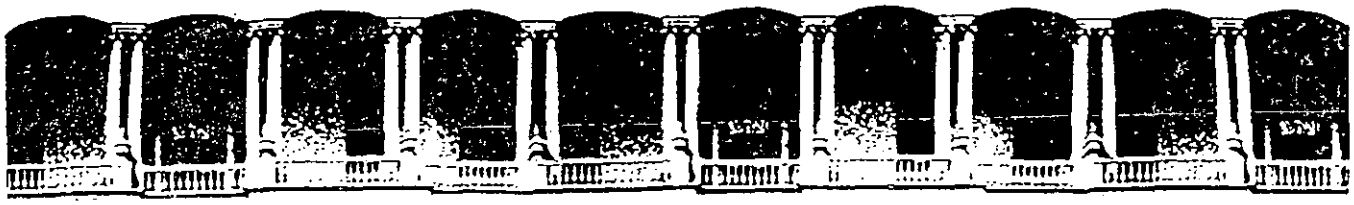
EN EL ACCESO SECUENCIAL PARA LLEGAR A UNA DIRECCIÓN DADA HAY QUE PASAR POR TODAS LAS QUE EXISTEN HASTA ELLA. SI EL ACCESO ES ALEATORIO, ES POSIBLE LLEGAR A LA DIRECCIÓN DADA EN FORMA DIRECTA SIN TENER QUE PASAR POR NINGUNA OTRA; EN ESTE TIPO EL TIEMPO DE ACCESO ES CASI IGUAL PARA CUALQUIER LOCALIDAD DE MEMORIA, NO SIENDO ASÍ PARA MEMORIAS SECUENCIALES QUE TIENEN UN TIEMPO DE ACCESO VARIABLE LO QUE LAS HACE MÁS LENTAS.

CAPACIDAD:

ES LA CANTIDAD DE UNIDADES DE INFORMACIÓN QUE PUEDE EL DISPOSITIVO GENERALMENTE BITS, BYTES, CARACTERES O PALABRAS.

MODO DE GRABACIÓN:

PUEDE GRABARSE EN SERIE, PARALELO, SERIE-PARALELO O PARALELO-SERIE.



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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES
14 DE AGOSTO AL 12 DE SEPTIEMBRE 1992

MICROCONTROLADOR 8031

ING. ALEJANDRO JIMENEZ HERNANDEZ

AGOSTO-SEPTIEMBRE

1992

MEMORIA DE DATOS.

Algunas instrucciones pueden operar directamente sobre 144 bits en RAM interna que forman el procesador booleano. Estos bits pueden ser usados como banderas de software o para almacenar variables del usuario. Las instrucciones que utilizan la bandera carry (C) como un registro de un bit o 'acumulador booleano' para operaciones logicas y transferencia de datos de 1 bit.

PUERTOS DE ENTRADA/SALIDA.

Todas las terminales de entrada/salida (32) pueden ser direccionadas como entradas, salidas o cumpliendo ambas funciones, en cualquier combinaci^n. Cualquier terminal puede ser una salida de control permanente, entrada de estado o puertos de entrada/salida serie implementadas por medio de software.

Existen 33 bits, individualmente direccionables, que reconfiguran, controlan y monitorean el estado del CPU y de todas las funciones perif@ricas del circuito, contadores de tiempo, modos de puerto serie, logica de interrupciones, y son TCON, SCON, PSW, IP e IE.

DIRECCIONAMIENTO DIRECTO DE BIT EN RAM INTERNA.

Los valores entre 0 y 127 (00H y 7FH) de RAM interna contienen un bloque de 32 bytes, entre las direcciones 20H y 2FH (figura 4.a), que son los bits que se pueden direccionar directamente y son numerados consecutivamente desde el bit de menor orden hasta el bit de mayor orden del byte.

Las direcciones entre 128 y 255 (80H y 0FFH) corresponden a registros especiales, utilizados para entrada/salida o para control de perif@ricos. Estas posiciones son numeradas con un esquema diferente que el de la RAM : los 5 bits de direccion de mayor orden equivalen a la direccion propia del registro, mientras los tres bits de menor orden identifican la posicion de bit dentro del registro (figura 4.b).

Los bits con significados especiales en el PSW y otros registros tienen nombres simb^licos correspondientes. Las instrucciones de proposito general (en contradiccion a la definicion del carry) pueden accesar el carry como cualquier otro bit con el uso del mnemonico CY en lugar de C. P0, P1, P2 y P3 son los cuatro puertos de entrada/salida donde el puerto 3 tiene funciones adicionales asignadas a cada una de las ocho terminales de P3 y se muestran en la figura 6.

La figura 7 muestra TCON (control de timer) y SCON (control de puerto serie) controlan y monitorean los perifericos correspondientes, mientras IE (habilitacion de interrupci^n) e IP (prioridad de interrupcion) habilitan y jerarquizan los cinco niveles de interrupci^n por hardware

Byte RAM	(MSB)							(LSB)
7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH 18H	BANCO 3 (8 REGISTROS)							
17H 10H	BANCO 2 (8 REGISTROS)							
0FH 08H	BANCO 1 (8 REGISTROS)							
07H 00	BANCO 0 (8 REGISTROS)							

Figura 4. Mapas de direcci^n de bit.

CONJUNTO DE REGISTROS DIRECCIONABLES.

Hay 20 registros de funciones especiales, pero las ventajas de direccionamiento de bit solo involucran a los 11 descritos abajo. Operaciones logicas del ancho del byte pueden ser usadas para manipular bits en todos los registros y RAM direccionables de bit non.

Direccion Directa de Byte	Direcciones de bit (MSB)							(LSB)	Simbolo Hardware de Registro
0FFH									
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	--	--	--	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	--	--	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0

Figura 4. Mapas de direccion de bit. de los registros de funciones especiales.

(MSB) (LSB)

CY	AC	F0	RS1	RS0	OV	--	P

Simbolo	Posicion	Nombre y Significado
CY	PSW.7	Bandera de carry. Puesto/borrado por hardware o software durante ciertas instrucciones aritmeticas y logicas
AC	PSW.6	Bandera auxiliar de carry. Puesto/borrado por hardware durante instrucciones de suma y resta para indicar carry o pedir salida de bit 3.
F0	PSW.5	Bandera 0. Puesto/borrado/probado por software como una bandera de estado de uso definido.
RS1	PSW.4	Banco de registro elegido para bits de control.
RS0	PSW.3	1 y 0 . Puesto/borrado por software para determinar la operacion del banco de registro (ver nota).
OV	PSW.2	Bandera de overflow. Puesto/borrado por hardware durante instrucciones aritmeticas para indicar condiciones de overflow.
---	PSW.1	(reservado).
P	PSW.0	Bandera de paridad. Puesto/borrado por hardware cada ciclo de instruccion para indicar un numero impar/par de bits "uno" en el acumulador, i.e., paridad par.
	NOTA :	Los contenidos de (RS1,RS0) borran la instruccion de los bancos de registro con la forma : (0,0) - Banco 0 (00H-07H) (0,1) - Banco 1 (08H-0FH) (1,0) - Banco 2 (10H-17H) (1,1) - Banco 3 (18H-1FH)

Figura 5. PSW - Palabra del estado del programa

(MSB)			(LSB)
RD	WR	T1	T0	INT1	INT0	TXD	RXD

Simbolo	Posicion	Nombre y Significado
RD	P3.7	Salida de control de lectura de datos. Activa pulso bajo generado por hardware cuando la memoria de datos externos es leida.
WR	P3.6	Salida de control de escritura de datos. Activa pulso bajo generado por hardware cuando la memoria de datos externos es escrita.
T1	P3.5	Entrada externa para timer/contador 1 o term de prueba.
T0	P3.4	Entrada externa para timer/contador 0 o term de prueba.
INT1	P3.3	term de entrada de interrupcion 1. Nivel bajo o flanco de bajada
INT0	P3.2	term de entrada de interrupcion 0. Nivel bajo o flanco de bajada
TXD	P3.1	term para transmitir datos para puerto serie en modo UART.
RXD	P3.0	term para transmitir datos para puerto serie en modo UART. term de entrada/salida de datos

Figura 6. P3 - Funciones de entrada/salida del puerto 3.

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Simbolo	Posicion	Nombre y Significado
TF1	TCON.7	Bandera de overflow del timer 1.

		Grabado por hardware sobre over-
		flow en timer/contador. Borrado
		cuando la interrupcion es proce-
		sada.
TR1	TCON.6	Bit de control de ejecucion del
		timer 1.
		Grabado/borrado por software pa-
		ra cambiar con on/off el timer/
		contador.
TF0	TCON.5	Bandera de overflow del timer 0.
		Grabado por hardware sobre over-
		flow en timer/contador. Borrado
		cuando la interrupcion es proce-
		sada.
TR0	TCON.4	Bit de control de ejecucion del
		timer 0.
		Grabado/borrado por software pa-
		ra cambiar con on/off el timer/
		contador.
IE1	TCON.3	Bandera de fin de interrupcion 1.
		Grabado por hardware cuando el
		fin de la interrupcion externa
		es detectado. Borrado cuando la
		interrupcion es procesada.
IT1	TCON.2	Bit de control de tipo de la in-
		terrupcion 1.
		Grabado/borrado por software pa-
		ra especificar interrupciones
		externas disparadas en una caida
		de nivel alto/bajo.
IE0	TCON.1	Bandera de fin de interrupcion 0.
		Grabado por hardware cuando el
		fin de la interrupcion externa
		es detectado. Borrado cuando la
		interrupcion es procesada.
IT0	TCON.0	Bit de control de tipo de la in-
		terrupcion 0.
		Grabado/borrado por software pa-
		ra especificar interrupciones
		externas disparadas en una caida
		de nivel alto/bajo.

Figura 7. Registros de configuracion de perifericos.
TCON - Registro de control del temporizador/contador.

(MSB) (LSB)

Simbolo	Posicion	Nombre y Significado
SM0	SCON.7	Bit 0 de control de modo en
		puerto serie.
		Grabado/borrado por software

		(ver nota).
SM1	SCON.6	Bit 1 de control de modo en puerto serie. Grabado/borrado por software (ver nota).
SM2	SCON.5	Bit 2 de control de modo en puerto serie. Grabado por software para deshabititar recepcion de estructuras para las cuales el bit 8 es cero .
REN	SCON.4	Bit de control de habilitacion del receptor. Grabado/borrado por software para habititar/deshabilitar recepcion de datos en serie.
TB8	SCON.3	Bit 8 de transmision. Grabado/borrado por hardware para determinar el estado del noveno bit de datos transmitido en modo UART de 9 bits.
RB8	SCON.2	Bit 8 de recepcion. Grabado/borrado por hardware para indicar el estado del noveno bit de datos recibido.
TI	SCON.1	Bandera de interrupcion de transmision. Grabado por hardware cuando el byte es transmitido. Borrado por software despues del servicio.
RI	SCON.0	Bandera de interrupcion de recepcion. Grabado por hardware cuando el byte es recibido. Borrado por software despues del servicio.

NOTA : El estado de (SM0,SM1) selecciona :
(0,0) Expansion del registro de cambio de ent/sal.
(0,1) UART de 8 bits, declaracion de datos variables
(1,0) UART de 9 bits, declaracion de datos fijos
(1,1) UART de 9 bits, declaracion de datos variables

Figura 7. Registros de configuracion de perifericos.
SCON - Registro de control del puerto serie.

(MSB)							(LSB)
EA	--	--	ES	ET1	EX1	ET0	EX0

Simbolo	Posicion	Nombre y Significado

EA	IE.7	Habilita todo el bit de control. Borrado por software para deshabilitar todas las interrupciones, es independiente del estado de IE.4 - IE.0 .
--	IE.6	(reservado).
--	IE.5	
ES	IE.4	Habilita el bit de control del puerto serie. Grabado/borrado por software para habilitar/deshabilitar las interrupciones de las banderas TI o RI.
ET1	IE.3	Habilita el bit de control del timer 1. Grabado/borrado por software para habilitar/deshabilitar interrupciones del timer/contador 1.
EX1	IE.2	Habilita bit de control de interrupcion externa 1. Grabado/borrado por software para habilitar/deshabilitar interrupciones de INT1.
ET0	IE.1	Habilita el bit de control del timer 0. Grabado/borrado por software para habilitar/deshabilitar interrupciones del timer/contador 0.
EX0	IE.0	Habilita bit de control de interrupcion externa 0. Grabado/borrado por software para habilitar/deshabilitar interrupciones de INT0.

Figura 7. Registros de configuracion de perifericos (continuacion).

c) IE - Registro de habilitacion de interrupcion.

(MSB)							(LSB)
--	--	--	PS	PT1	PX1	PT0	PX0

Simbolo	Posicion	Nombre y Significado

--	IP.7	(reservado) .
--	IP.6	(reservado) .
--	IP.5	(reservado) .
PS	IP.4	Bit de control de prioridad del puerto serie. Grabado/borrado por software para especificar la prioridad alta/baja de las interrupciones del puerto serie.
PT1	IP.3	Bit de control de prioridad del timer 1. Grabado/borrado por software para especificar la prioridad alta/baja de las interrupciones del timer/contador 1.
PX1	IP.2	Bit de control de prioridad de la interrupcion externa 1. Grabado/borrado por software para especificar la prioridad alta/baja de las interrupciones de INT1.
PT0	IP.1	Bit de control de prioridad del timer 0. Grabado/borrado por software para especificar la prioridad alta/baja de las interrupciones del timer/contador 0.
PX0	IP.0	Bit de control de prioridad de la interrupcion externa 0. Grabado/borrado por software para especificar la prioridad alta/baja de las interrupciones de INT0.

Figura 7. Registros de configuraci^n de perifericos
IP - Registro de control de prioridad de interrupci^n.

1. INTRODUCCION

En 1976 INTEL introdujo los microcontroladores con la familia MCS-48, que consistía de los microprocesadores 8048, 8748 y el 8035. Estos son microprocesadores que se caracterizan por integrar en un solo circuito un procesador de 8 bits, memoria de programa, ROM o EPROM, de 1024 palabras por 8 bits, 64 palabras de memoria de datos, puertos de entrada/salida (E/S) y un temporizador/contador de ocho bits.

Estos circuitos se utilizan para diseños dedicados y pueden controlar una diversa gama de dispositivos, desde juguetes, aparatos domesticos, automoviles, aparatos de control industrial hasta equipo de procesamiento de textos y datos en general.

En 1980 aparece la nueva familia MCS-51 con tres nuevos circuitos: 8051, 8751 y 8031. Posteriormente se incluyen el 8032 y el 8054.

A 10 años de utilizarse la familia del 8051 se ha convertido en un standard para diseños de sistemas dedicados o 'embebidos' que requieren un microcontrolador de 8 bits y aunque se puede pensar que este es obsoleto, han aparecido en el mercado dispositivos que se basan en el 8051 y que agregan mas opciones al circuito y hacen que siga vigente.

Estas notas explican la arquitectura y las funciones de la familia MCS-51.

DESCRIPCION DE LA FAMILIA

El diagrama del 8051,8751, 8031 y 8032 se muestra en la Figura 1. Estos circuitos tienen las siguientes funciones :

- Operación con 5V
- 4096 bytes de memoria de programa (no en 8031)
- 128 bytes de memoria de datos (256 para el 8031)
- Cuatro bancos de registros.
- 128 banderas de software definidas por el usuario
- 64 Kbytes de memoria externa direccionable ya sea para programa o datos.
- Ciclos de instruccion de 1μs con cristal a 12MHz.
- 32 lineas de entrada/salida bidireccionales organizadas como 4 puertos de 8 bits (16 lineas en el 8031).
- Puerto serie programable
- 2 Temporizadores/Contadores de 16 bits y modo multiple.
- Stack para almacenamiento de datos de subrutinas
- Direccionamiento directo de bits y bytes.
- Aritmetica binaria o decimal.
- Deteccion de signo, overflow y computo de paridad.
- Multiplicaci^n y divisi^n por hardware

- Multiplicaci^n y divisi^n por hardware
- Procesador Booleano integrado para aplicaciones de control.

Los circuitos integrados estan disponibles en encapsulados DIP de 40 terminales, con la misma distribuci^n de terminales, diagrama de tiempos y caracteristicas electricas. La unica diferencia entre ellos es la cantidad de memoria dispuesta para programa y la interna de trabajo.

En el 8751 hay 4Kbytes de memoria UEPROM para desarrollo de programas, prototipos y produccion limitada. Puede ser programada en un programador de microprocesadores comercial. Si se detectan errores de software o cambian las especificaciones de dise~o, esta puede ser borrada conluz ultravioleta para despues ser reprogramada con nuevos datos. Este ciclo puede ser repetido indefinidamente durante la fase de dise~o y desarrollo.

Si la version final del software debe ser programada en un numero grande de circuitos, el 8051 tiene 4Kbytes de ROM que puede ser programada, en la etapa de ensamblado, por el fabricante con las especificaciones del cliente. Este es considerablemente mas barato, pero la memoria no puede ser borrada o modificada despues de la fabricaci^n.

El 8031 no tiene memoria de programa interna, pero puede usar hasta 64Kbytes de memoria ROM, PROM o EPROM en forma estandar o multiplexada. El 8031 es adecuado para aplicaciones que requieren cantidades mayores, o menores, a los 4Kbytes que tienen los otros microprocesadores de la familia.

(El 8051 y el 8751 accesan automaticamente las localidades en memoria externa para direcciones mayores a las 4096 internas. El External Access : EA [Acceso Externo] es un deshabilitador para toda la memoria de programa interna, el 8051 y el 8751 emulan al 8031 si la terminal 31 esta a V).

En el resto de estas notas se usara el termino 8051 de manera general. A menos que se especifique lo contrario, se aplicara de igual manera a todos los microprocesadores.

El resto de estas notas discuten las diferentes funciones de la familia MCS-51 y como pueden ser usadas. Ejemplos de aplicaciones de software y hardware ilustran la mayoria de los conceptos. Muchas tareas aisladas (en vez del dise~o completo de un sistema) se presentan con la intenci^n de que alguno de ellos se ajuste a las necesidades o experiencias del usuario.

ASM51 es el ensamblador empleado para todos los miembros de la serie 8051.

CONJUNTO DE INSTRUCCIONES

Podemos clasificar las instrucciones en cinco grupos:

* Operaciones aritmeticas.
* Operaciones logicas para variables de un byte.
* Instrucciones de transferencia de datos.
* Manipulaciones de variables booleanas.
* Saltos y control de la m!quina.

Ninguna instruccion requiere mas de tres bytes de la memoria del programa, con el mayor requerimiento de uno o dos bytes. Virtualmente todas las instrucciones se ejecutan en uno o dos ciclos de instruccion -uno o dos microsegundos con un cristal de 12 MHz- con la sola excepcion (multiplicacion y division) que se completan en cuatro ciclos.

Muchas instrucciones, como las funciones aritmeticas y logicas o control de programa proveen una forma corta o larga para la misma operacion, permitiendo al programador optimizar el codigo producido para una aplicacion especifica. El 8051 normalmente tiene dos bytes de instruccion por ciclo, y usando una forma mas corta, puede llevar a una ejecuci^n mas r!pida.

Por ejemplo, cualquier byte de RAM puede ser cargado con una instruccion de tres bytes y dos ciclos, pero los comunmente usados registros de trabajo en RAM pueden ser inicializados en la forma de un ciclo y dos bytes. Cualquier bit, puede ser puesto a uno, limpiado o complementado por una instruccion logica sencilla de tres bytes usando dos ciclos. Pero los bits criticos de control, terminales de E/S y banderas programables pueden ser controladas por instrucciones de dos bytes y un ciclo sencillo. Mientras que los saltos de tres bytes y llamadas pueden ir dondequiera en la memoria del programa, las secciones proximas del codigo, pueden ser alcanzadas por versiones relativas cortas o absolutas.

CY	AC	F0	RS1	RS0	OV	--	P

S#mbolo	Posici^n	Nombre y significado.
CY	PSW.7	Bandera de acarreo. Activada o limpiada por alguna instruccion aritmetica y logica por hardware o software.
AC	PSW.6	Bandera auxiliar de acarreo. Activada o limpiada por hardware durante instrucciones de adicion o sustraccion para indicar acarreo o prestamo mas alladel bit 3.
F0	PSW.5	Bandera 0. Activada o limpiada o probada por programa como bandera de estado definida por el usuario.
RS1	PSW.4	Selecci^n del banco de regis-

RS1	PSW.4	Selecci^n del banco de registros de bits 1 y 0. Activado, limpiado por programa.
RS0	PSW.3	Banco de registros de trabajo. (0,0) -> banco 0 (00H - 07H) (0,1) -> banco 1 (08H - 0FH) (1,0) -> banco 2 (10H - 17H) (1,1) -> banco 3 (18H - 1FH).
OV	PSW.2	Bandera de sobreflujo. Activada, limpiada por hardware durante instrucciones aritmeticas para indicar condiciones de sobreflujo.
--	PSW.1	Reservado.
P	PSW.0	Bandera de paridad. Activada, limpiada por hardware cada ciclo de instruccion para indicar un numero par/impar de un bit en el acumulador, p.ej. paridad par.

----------Fig. 4

PSW. Program Status Word.
Organizaci^n de la Palabra de Estado del Programa.

ACUMULADOR Y PSW.

El 8051, es una arquitectura basada en un acumulador; un registro de 8 bits llamado acumulador (A) sostiene un operando fuente y recibe el resultado de de una instruccion aritmetica (suma, resta, multiplicaci^n y divisi^n). El acumulador puede ser la fuente o el destino para operaciones logicas o un numero de instrucciones de movimiento especial de datos, incluyendo el buscar en tablas y expansion externa en RAM. Muchas funciones se aplican exclusivamente al acumulador; rotaciones, calculo de paridad, prueba de cero, y otras.

Muchas instrucciones afectan implicitamente o explicitamente (o son afectadas por) varias banderas de estado, que son agrupadas en la Palabra de Estado del Programa, (PSW) como se muestra en la figura 4.

El punto en la notaci^n de las entradas es llamado operador punto, e indica una posici^n particular de un bit dentro de un byte de ocho bits. PSW.5 especifica el bit 5 del PSW. Ambos, la documentacion y los ensambladores usan esta notaci^n.

El bit de estado mas importante es la bandera de acarreo (C). Este bit hace posible operaciones aritmeticas de multiple precision incluyendo la suma, resta y rotaci^n. El acarreo tambien sirve como un acumulador booleano para operaciones logicas de un bit e instrucciones de manipulacion de bits. La bandera de sobreflujo (overflow, OV) detecta cuando ocurren sobreflujos aritmeticos en operandos enteros

signados, haciendo posible el complemento a dos aritmetico. La bandera de paridad (P) es actualizada despues de cada ciclo de instrucci^n con la paridad par del contenido del acumulador.

El CPU no controla de manera automatica los bits de seleccion de los dos bancos de registros, RS1 y RS0. Estos son manipulados por programa para habilitar uno de los cuatro bancos de registros utilizando dos banderas del PSW.

Aunque la arquitectura esta basada en acumulador, el acumulador se omite en algunas instrucciones Los datos pueden ser movidos desde cualquier parte del circuito a cualquier registro, direcci^n o direcci^n indirecta (y viceversa), cualquier registro puede ser cargado con una constante, etc., todos sin afectar al acumulador. Las operaciones l^gicas pueden ser ejecutadas contra registros o variables para alterar campos de bits -sin usar o afectar el acumulador. Las variables pueden ser incrementadas, decrementadas, o probadas sin usar el acumulador. Las banderas y bits de control pueden ser manipuladas y probadas sin afectar otras variables.

OTROS REGISTROS DEL CPU.

El registro B es de ocho bits y se utiliza en la ejecuci^n de instrucciones de multiplicaci^n y divisi^n. El registro es usado en conjunto con el acumulador como un segundo operando de resultado que regresa ocho bits del resultado.

La familia de procesadores MCS-51 incluyen un Stack de hardware en RAM interna, usada para subrutinas de ligado, paso de parametros entre rutinas, almacenamiento temporal de variables, o salvando el estado durante rutinas de servicio a interrupciones. El Stack Pointer (SP) es un registro apuntador de ocho bits que indica la direccion del ultimo byte almacenado en la pila. El Stack es automaticamente incrementado o decrementado en todas las instrucciones de push o pop y todos los llamados a subrutinas y retornos. En teoria, el Stack en el 8051 puede contener un total de 128 bytes. En la practica, aun programas simples usan varias localidades de RAM para apuntadores, variables y otras cosas, reduciendo la longitud del Stack en ese n&mero.

El Stack Pointer se posiciona en la localidad 7 de memoria interna durante el RESET, por lo que empieza a aumentar desde la localidad 8. Al alterar el contenido del apuntador el stack puede ser relocalizado en cualquier lugar dentro de la RAM interna.

Finalmente un registro de 16 bits llamado DATA POINTER o apuntador de datos (DPTR) sirve como un registro base para saltos indirectos , una tabla de busqueda para instrucciones y para transferencia de datos externos . La parte alta y baja del DPTR deben ser manipuladas como registros separados (DPH y DPL respectivamente) o bien , como un solo registro con las instrucciones especiales de carga (LOAD) o incremento (INC) ambas de 16 bits . Las tablas de b&squeda pueden

empezar en cualquier parte de la memoria del programa y tener una longitud arbitraria .

ESPACIOS DE MEMORIA .

La memoria del programa es separada y tiene una forma distinta de la memoria de datos . Cada tipo de memoria tiene diferente mecanismo de direccionamiento , diferentes se~ales de control y diferente funci^n .

El arreglo de la memoria de programa (ROM EPROM) es de 64K y nunca pierde informacin aun cuando el voltaje de alimentaci^n se apague . La memoria de programa es usada para informaci^n requerida cada vez que exista voltaje de alimentaci^n : valores iniciales , calibraci^n de constantes , tablas de decodificaci^n de teclados etc. as# como el programa mismo . La memoria de programa tiene un bus de direcciones de 16 bits , y sus elementos son direccionados usando el Program Counter o instrucciones que generan un direccionamiento de 16 bits .

La memoria de datos interna es peque~a y m!s r!pida que la memoria de programa y se maneja de forma aleatoria cuando el voltaje de alimentaci^n es aplicado . El circuto de RAM integrado en el microcontrolador es usado para variables que son determinadas o que deben cambiar mientras el programa se ejecuta.

Una computadora consume la mayor parte de su tiempo manipulando variables , no constantes , y adem!s un n&mero relativamente peque~o de variables ; 8 bits es una longitud de palabra suficiente para la RAM interna (128 x 8) y el registro de direcciones es de un byte de ancho . En contraste con la memoria de programa , el acceso a la memoria de datos necesita un &nico valor de 8 bits - una constante o alguna otra variable - para especificar una &nica localidad , tomando en cuenta que esta es la longitud bsica de la ALU y los diferentes tipos de memorias , estos recursos pueden ser utilizados para el direccionamiento y de esta forma mejorar la eficiencia operativa .La partici^n de la memoria de programa y de datos puede ser extendida con memorias externas de expansi^n . Cada expansi^n es agregada de forma independiente y cada una utiliza el mismo bus de direcciones y de datos pero con diferentes se~ales de control .

La memoria externa de programa es habilitada y conectada al bus de datos por la pata 29 de control de salida ,@sta se habilita en bajo y se denomina PSEN (program store enable).

La memoria externa de datos (RAM) es le#da y conectada al bus de datos por la habilitaci^n de la pata de salida 17 , habilitada en bajo y denominada RD , y para la lectura se utiliza la pata 16 de salida habilitada en bajo y denominada WR . Ambos tipos de memoria pueden ser expandidos hasta 64 K y opcionalmente se pueden expander en p!ginas de 256 bytes para preservar el uso del puerto 2 como de entrada-salida(entrada/salida) .

Los programas que controlan a este chip son &nicos y no son modificables , no tienen una arquitectura de Von Newman como el com&n de los sistemas computarizados . El procesador de datos de memoria (ya sea interna o externa) MCS-51 no puede ser usado para c^digos de

programa , no hay forma de que el CPU ejecute instrucciones de RAM . En contraparte esta arquitectura permite unas aplicaciones eficientes de control : con un programa localizado en ROM y variables en RAM y con diferentes metodos de direccionamiento para cada una .

Las arquitecturas de Von Newman son de gran utilidad para el desarrollo de software . Un sistema 8051 puede ser modificado para tener un solo chip de memoria externa utilizando como entradas de una compuerta AND las patas de control PSEN y RD , habilitadas en bajo .

De esta forma el CPU puede escribir datos en el arreglo com&n de memoria externa utilizando la pata de WR , habilitada en bajo , y los datos grabados pueden mandar instrucciones o manipular otros datos .

Adem!s de los arreglos de memoria , hay otro espacio f#sico direccionable . Conectado al bus interno de datos hay una zona de prop^sito especial con registros de 8 bits . Algunos de estos son B , SP , PSW , DPH Y DPL . Otros registros para los puertos de entrada-salida y para los perifericos seran vistos posteriormente . En conjunto estos registros son designados como registros de funciones especiales , incluyendo al acumulador para mayor flexibilidad y uniformidad .

La arquitectura del MCS-51 puede direccionar varios espacios de memoria , funcionalmente separados por hardware y con diferentes tipos de direccionamientos y se~ales de lectura y escritura o ambas :

- .Memoria de programa interna
- .Memoria de datos interna
- .Memoria de programa externa
- .Memoria de datos externa
- .Registros internos de funciones especiales

Lo que el programador ve son espacios logicos direccionables . Por ejemplo : solo hay un tipo de memoria para programa , de 64 K de longitud , el hecho de ser interna o externa es invisible , el CPU realiza la busqueda autom!tica .

Los modos l^gicos de direccionamiento son descritos en el cap#tulo del conjunto de instrucciones en t@rminos de espacios f#sicos direccionables . Entender su diferencia ayuda para entender y utilizar este dispositivo

PUERTOS DE ENTRADA-SALIDA

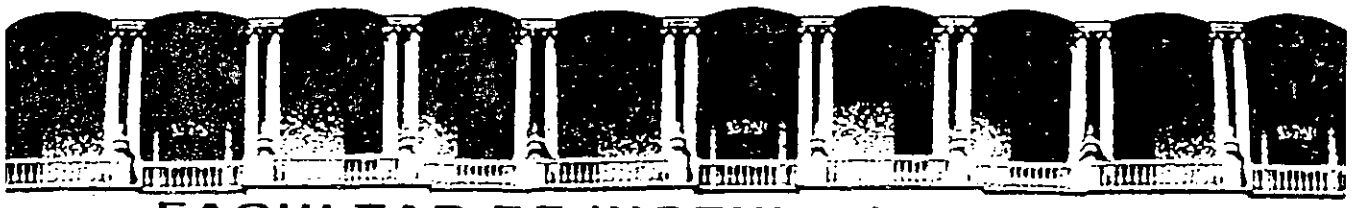
Tanto en el 8051 como el 8751 tienen 32 terminales de entrada/salida configuradas en 4 puertos de 8 bits en paralelo (P0 , P1 , P2 y P3) . Cada terminal lee y/o escribe datos dependiendo del control por software y cada una puede ser referida especificamente dentro de la longitud del puerto para operaciones de bytes o de bits .

En varios modos de expansi^n , algunas de estas terminales de entrada/salida pueden ser utilizadas para funciones especiales de entrada o salida . Las instrucciones que accesan a la memoria externa utilizan el puerto 0 multiplexado como bus de datos y direcciones .

Para el ciclo de memoria externa primero salen los 8 bits de direcciones , posteriormente el dato es transferido a las mismas 8 patas . Las instrucciones de 16 bits para transferencia de datos externos y cualquier instruccin de acceso a memoria externa tienen los 8 bits ms significativos en el puerto 2 durante el ciclo de acceso . El 8031 siempre utiliza el puerto 0 y el 2 para direccionamiento externo y el puerto 1 y 3 son para entrada/salida comunes .

Cada uno de las 8 terminales del puerto 3 tiene una funci^n especial .Dos interrupciones externas , dos contadores de entrada , dos lineas para puerto de comunicaciones serie y dos patas de control para control de temporizadores llamadas strobes . ver figura 6 . Cuando estas terminales no se usan para funciones especiales se pueden utilizar para entrada/salida .

Aun con un solo puerto las funciones de entrada/salida pueden ser combinadas en muchas formas . Entrada y salida pueden ser ejecutadas al mismo tiempo usando diferentes terminales o usando las mismas pero alternando salida y luego entrada (multiplexar en tiempo) , unas veces en paralelo otras veces en serie o como funciones especiales adicionales .



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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

14 DE AGOSTO AL 12 DE SEPTIEMBRE 1992

AN 8031 IN - CIRCUIT EMULATOR

EXPOSITOR:
ING. ALEJANDRO JIMENEZ HERNANDEZ

AGOSTO-SEPTIEMBRE
1992

AN 8031 IN-CIRCUIT EMULATOR

BY GEORGE DINWIDDIE

Build this emulator and avoid debugging headaches

DEBUGGING A microprocessor-based program in assembly language can be difficult for a home engineer. The equipment to aid such a task—logic analyzers and emulators—is usually incredibly expensive. Even a large company might not be able to afford the thousands of dollars needed for an emulator because of the processor-specific nature of the tool. Such was the case when I was developing firmware for an embedded processor controlling a commercial product. The processor in use was an Intel 8031, and at that time, only Intel had an emulator for it. Obviously, Intel wanted a lot of money for the emulator. Worse, it needed a new microprocessor development system to host it.

My alternative was to edit, assemble, burn EPROMs, trace the execution using a logic analyzer, and deduce where the code was going wrong. Once I found a bug, I usually had to start over at the edit step before I could hunt for the next one. This was a slow process.

A BETTER WAY

It didn't take me long to become frustrated with this method. There had to be a better way. At the suggestion of a microprocessor guru, I decided to build a monitor. Because the product hardware was mostly developed, I designed the monitor board so that it could function as an emulator—plugging into the processor socket of the application hardware.

The primary benefit of my monitor is that it gives you the ability to patch errors without repeating the edit, assemble, burn EPROM cycle. Sometimes a patch alters only a byte or two, but more often than not it involves a jump to an unused section of RAM where a hand-assembled routine is inserted.

Another handy debugging tool is the monitor's breakpoint capability. While the breakpoint in my monitor is extremely limited and simple, it does provide a way to verify which path of a conditional jump was taken without having to resort to the logic analyzer. Furthermore, you can check the actual values in variables and internal registers instead of deducing them from external behavior.

Another technique I found helpful was to insert temporary test patches that displayed the value of some variable each time through a loop. This was accomplished by calling monitor subroutines via a convenient jump table provided at the start of the monitor. This technique was at the expense of real-time operation, but it provided insights into the dynamics of the software that were unobtainable by any other means.

I also included some basic but necessary functions that let you initialize memory, copy a block of memory, compare two blocks of memory, and perform a hexadecimal dump on a block of memory. Of course, I had to provide commands to edit the contents of memory and to execute instructions starting at any arbitrary location. Finally, I added some convenience commands to display the command syntax and the locations in the jump table and to perform hexadecimal arithmetic.

THE PROCESSOR

The Intel 8031 single-chip microcomputer is the ROM-less version of the

(continued)

George Dinwiddie (13808 Wayside Dr, Clarksville, MD 21029) is a software engineer for Mitron Systems Corp. He is currently working on his M.S. degree in computer science.

Listing 1: *The code for performing* ASCII-*hexadecimal to binary conversion.*

```
06EB 9430              SUBB    A,#'0'
06ED 4017              JC      BAD             ; ACC < '0'
06EF 940A              SUBB    A,#10
06F1 5005              JNC     $+7             ; ACC > '9'
06F3 240A              ADD     A,#10   ;RESTORE
06F5 020704            LJMP    GOOD            ; '0' <= ACC <= '9'

06F8 2403              ADD     A,#('0'+10-'A'+10)      ;CORRECT FOR (-'0'-10)
                                                        & MAP 'A' INTO 10
06FA 20E709            JB      ACC.7,BAD       ; '9' < ACC < 'A'
06FD C3                CLR     C
06FE 9410              SUBB    A,#16
0700 5004              JNC     BAD             ; ACC > 'F'
0702 2410              ADD     A,#16
0704 C3          GOOD: CLR     C
0705 22                RET
0706 D3          BAD:  SETB    C
0707 22                RET
```

8051, the high end of Intel's 8-bit microcontrollers. It has five interrupts—two external inputs, two hardware timers, and one hardware asynchronous serial port (UART). It sports a 16-bit external address bus and 16 bidirectional I/O lines. The processor has three separate memory spaces—a 64K-byte program memory space, a 64K-byte external data memory space, and 128 bytes of internal RAM. The 8031 even features a hardware multiply and divide.

While this processor has a lot of speed and power, it has its share of problems, too. The architecture is accumulator-based. This means a lot of processing time and bytes of code are used to load and store the accumulator. The instruction set is highly irregular. For example, the only compare instruction is a compare-and-jump-if-not-equal. To do a compare-and-jump-if-equal requires an exclusive-or, a subtract, or a jump-around-a-jump. A jump-less-than or a jump-greater-than also involves extra work. You can see some of the problems this causes in the ASCII-hexadecimal to binary conversion in listing 1. The instruction set also has few instructions that are capable of accessing the two large external memory spaces.

THE EMULATOR BOARD HARDWARE

The hardware design of the emulator board is straightforward. Figures 1a and 1b show the schematic diagram. It features 4K bytes of monitor EPROM, 8K bytes of application EPROM space, and 8K bytes of emulation RAM. With the current prices for larger memories it would make more sense to use 8K by 8 monitor and application EPROM

(*continued*)

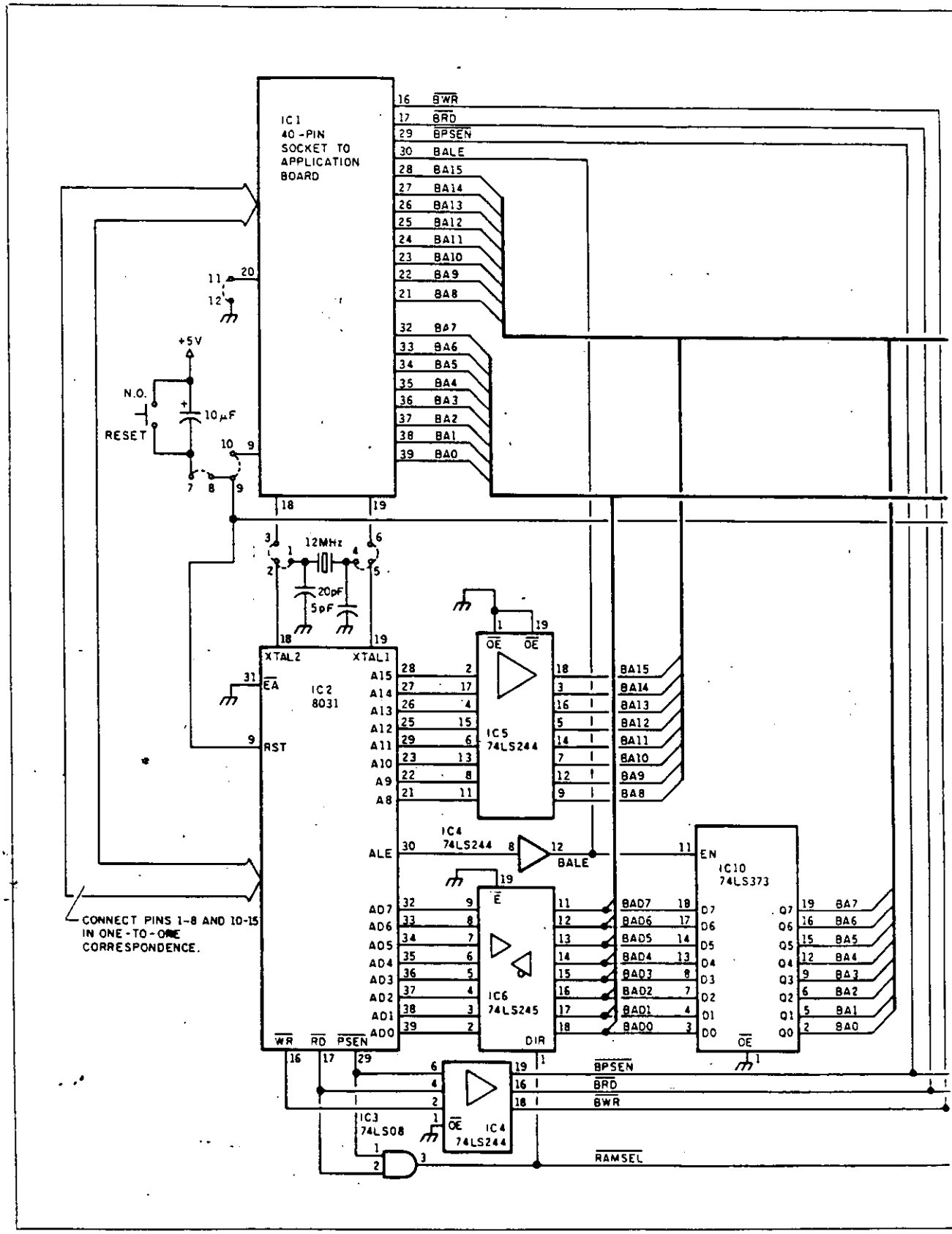
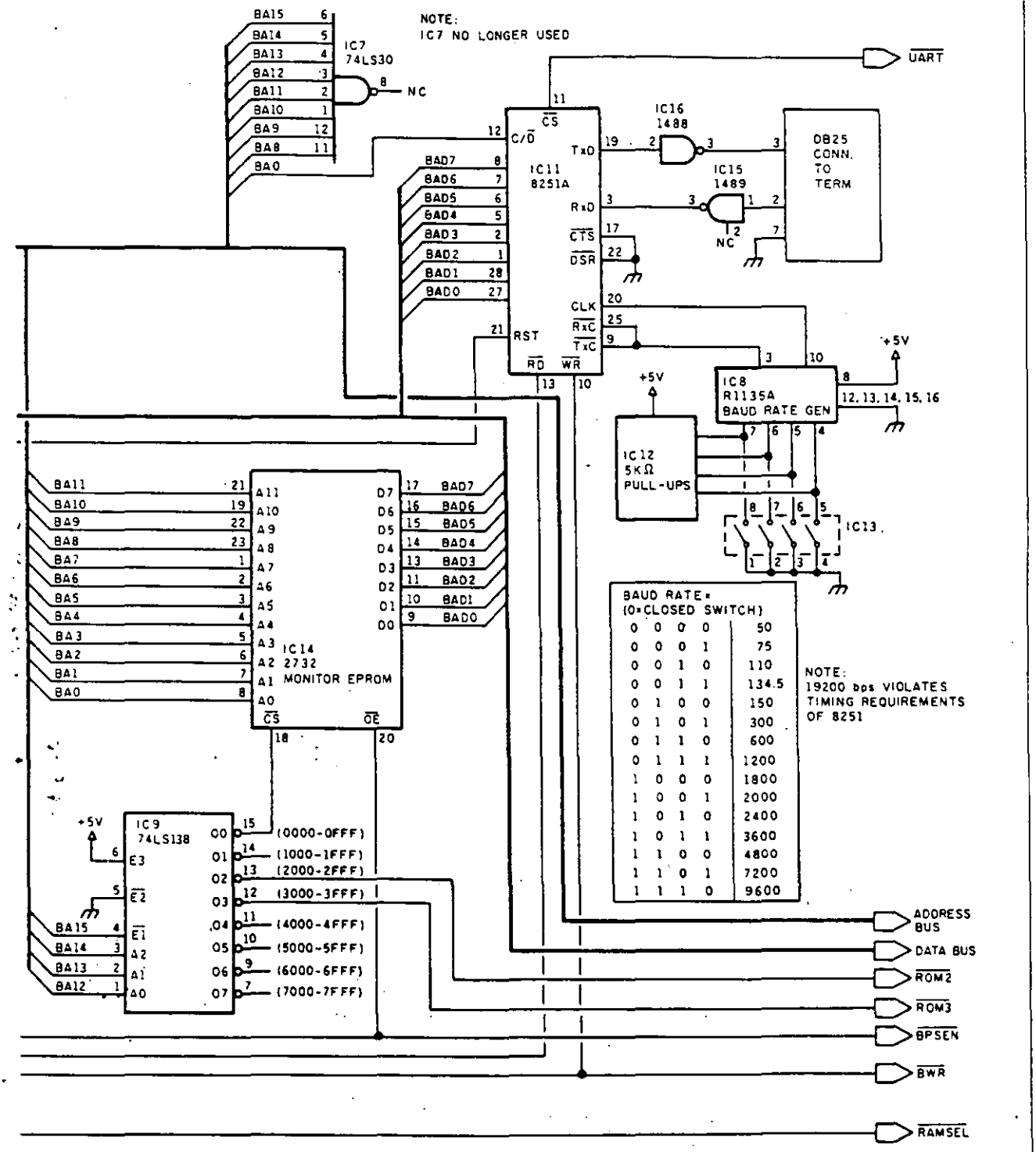


Figure 1a: *Schematic of the 8031 emulator board, showing the central processor, UART, and address-decoding circuitry.*


NOTE:
IC7 NO LONGER USED
IC7
74LS30
NC
BA15
BA14
BA13
BA12
BA11
BA10
BA9
BA8
BA0
UART
IC11
8251A
C/D
CS
TxD
RxD
CTS
DSR
CLK
RxC
TxC
RST
RD
WR
BAD7
BAD6
BAD5
BAD4
BAD3
BAD2
BAD1
BAD0
IC16
1488
IC15
1489
DB25
CONN.
TO
TERM
+5V
IC8
R1135A
BAUD RATE GEN
12, 13, 14, 15, 16
IC12
5KΩ
PULL-UPS
IC13
IC14
2732
MONITOR EPROM
IC9
74LS138
E3
E2
E1
A2
A1
A0
O0 (0000-0FFF)
O1 (1000-1FFF)
O2 (2000-2FFF)
O3 (3000-3FFF)
O4 (4000-4FFF)
O5 (5000-5FFF)
O6 (6000-6FFF)
O7 (7000-7FFF)
BAUD RATE=
(0=CLOSED SWITCH)
0 0 0 0 50
0 0 0 1 75
0 0 1 0 110
0 0 1 1 134.5
0 1 0 0 150
0 1 0 1 300
0 1 1 0 600
0 1 1 1 1200
1 0 0 0 1800
1 0 0 1 2000
1 0 1 0 2400
1 0 1 1 3600
1 1 0 0 4800
1 1 0 1 7200
1 1 1 0 9600
NOTE:
19200 bps VIOLATES
TIMING REQUIREMENTS
OF 8251
ADDRESS BUS
DATA BUS
ROM2
ROM3
BPSEN
BWR
RAMSEL

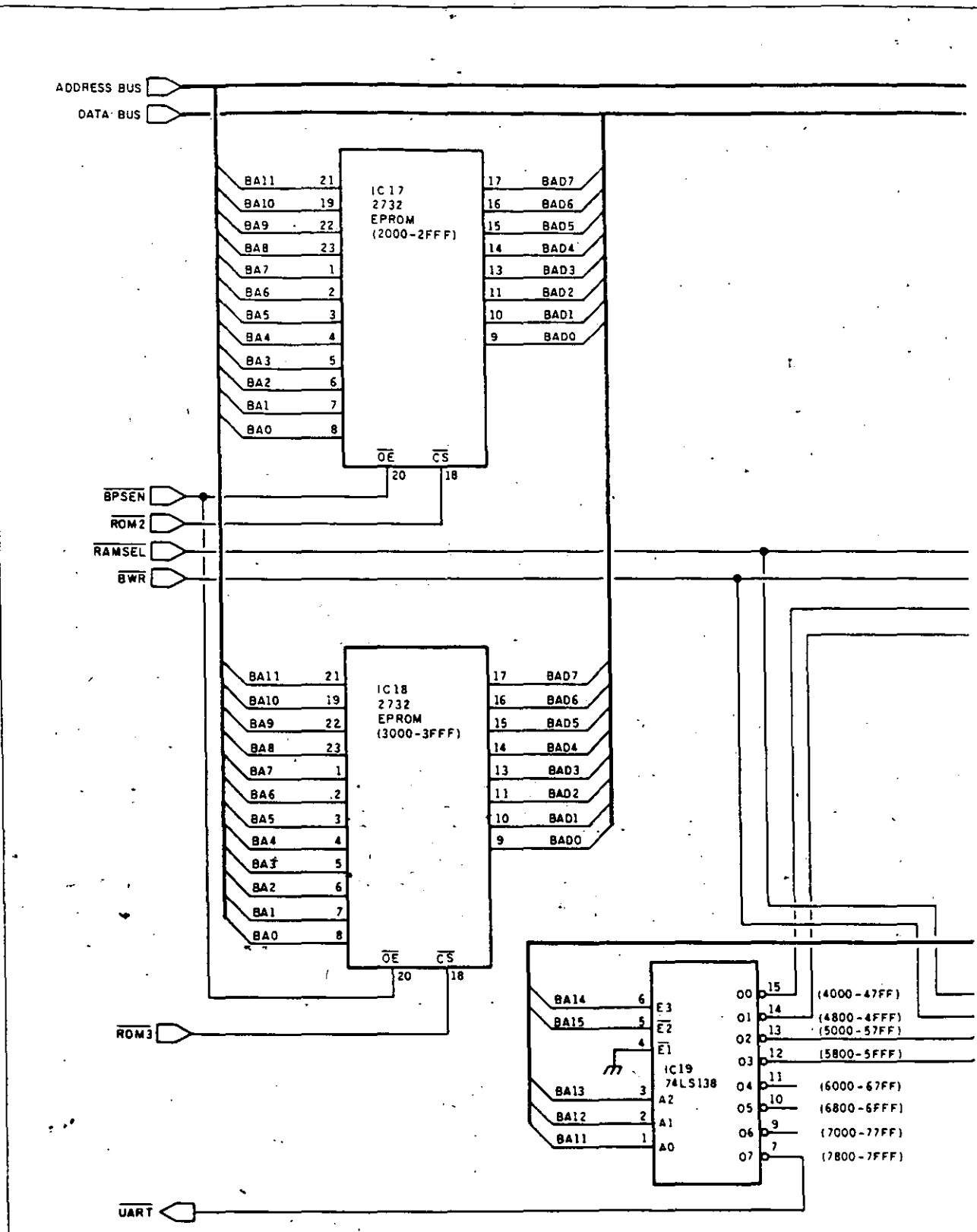
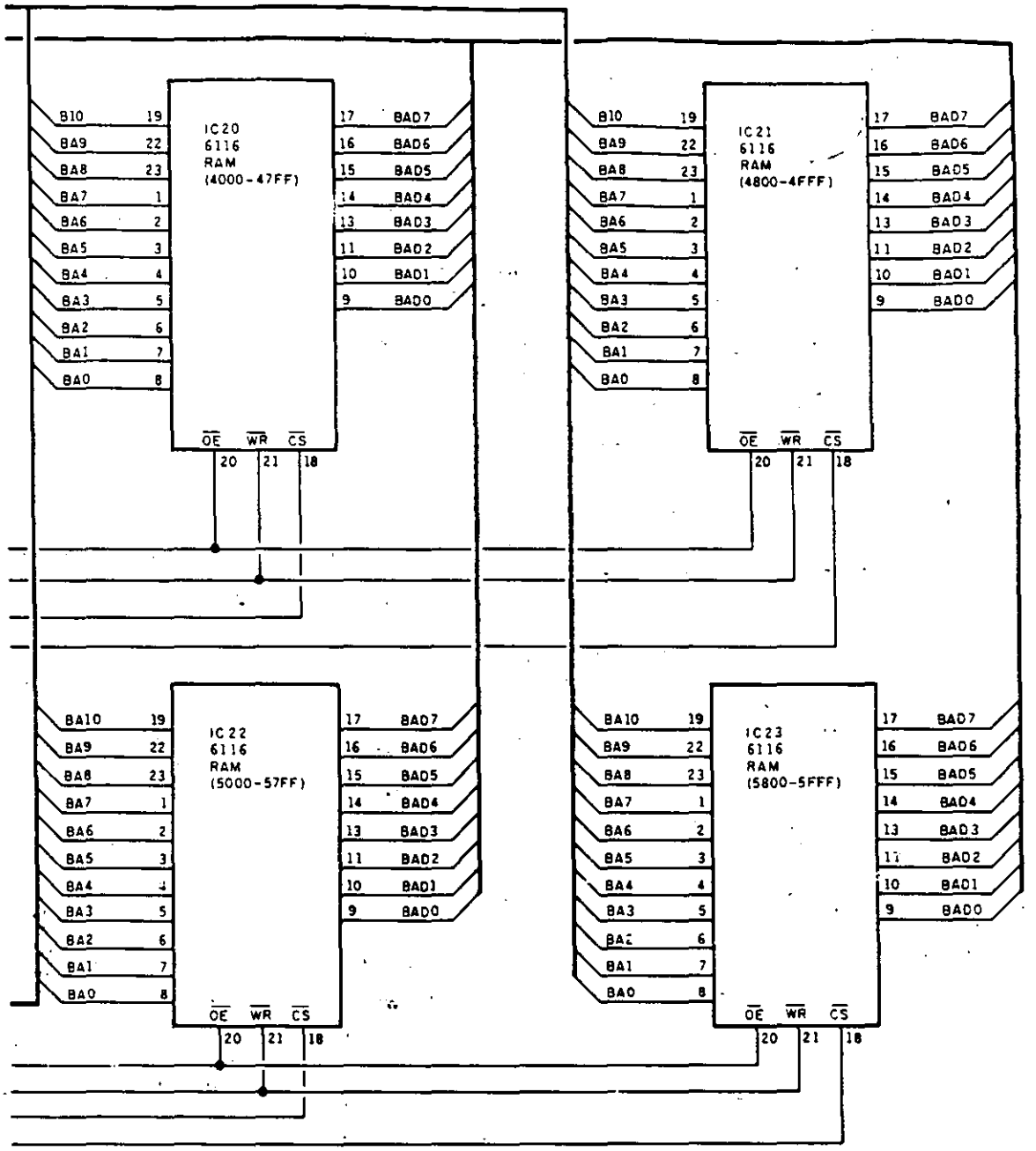


Figure 1b: *Schematic of the* 8031's ROM *and* RAM *circuitry.*


IC20
6116
RAM
(4000-47FF)
IC21
6116
RAM
(4800-4FFF)
IC22
6116
RAM
(5000-57FF)
IC23
6116
RAM
(5800-5FFF)
BA10
BA9
BA8
BA7
BA6
BA5
BA4
BA3
BA2
BA1
BA0
BAD7
BAD6
BAD5
BAD4
BAD3
BAD2
BAD1
BAD0
OE
WR
CS

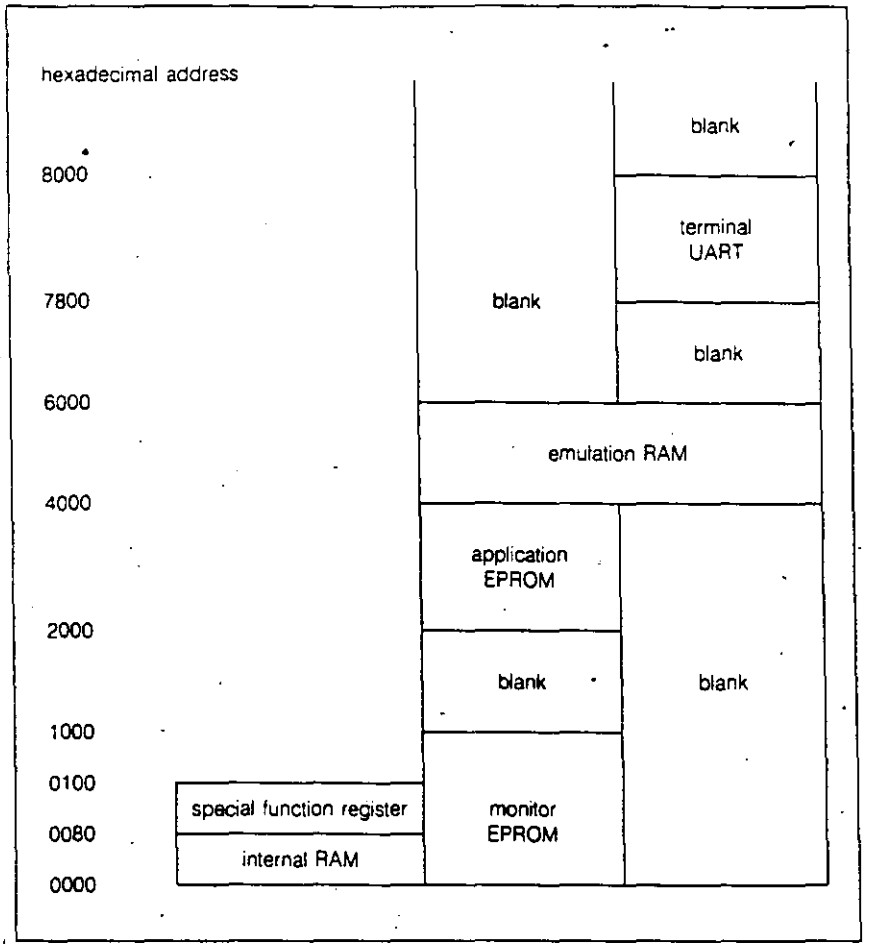


Figure 2: *The emulator board's memory map.*

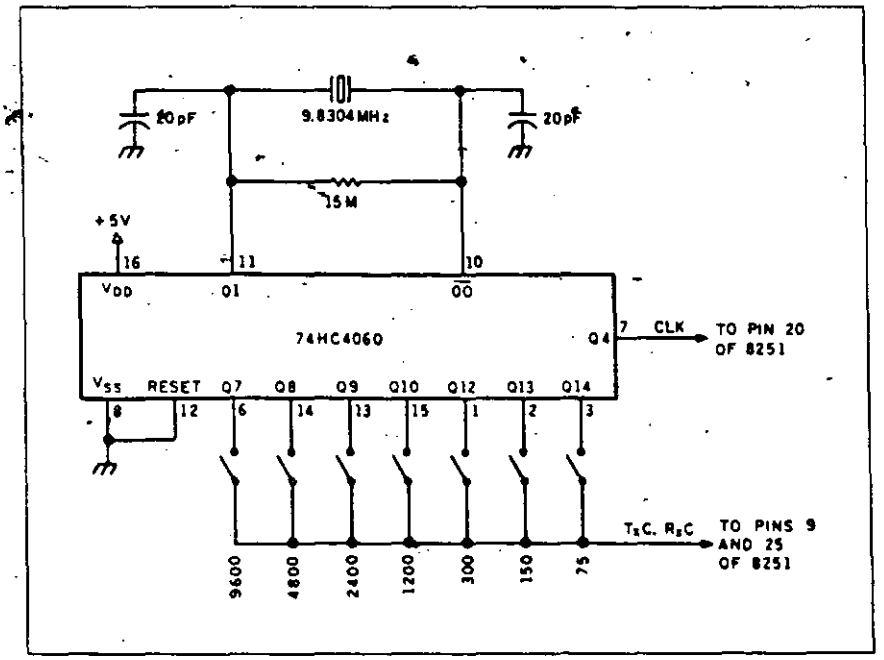


Figure 3: *Alternate circuit for generating baud-rate clock signal.*

chips and an 8K by 8 emulation RAM chip. When this circuit was first designed, however, an 8K by 8 static RAM chip cost approximately $50. Using the larger memory chip could save one 74LS138 address-decoder chip, not to mention the additional sockets and wiring.

The 8031's separate external program and data memories pose a problem in the design of the emulator. The monitor and application EPROMS are in the program memory—that's easy enough. The RAM, however, must be in the data memory because the 8031 has no instructions for writing to the program memory. On the other hand, the purpose of the emulation RAM is to allow you to alter the executed code. Therefore, the RAM must also be part of the program memory so that instruction fetches may access it. The solution, of course, is to map the RAM into both memory spaces. The signal $\overline{\text{RAMSEL}}$ is true if $\overline{\text{PSEN}}$ is true or if $\overline{\text{RD}}$ is true. Figure 2 shows a memory map of the emulator board.

I tried to tie up as few of the resources of the processor as possible. The exception to this is the large amount of external memory space used. The project that prompted this monitor used very little external memory, but it did use the internal UART. Therefore, I included an 8251 USART for communicating with the terminal rather than the internal UART of the 8031. Clocks for the 8251 are provided by a Motorola K1135A dual baud-rate generator. The K1135A contains both a crystal oscillator and two divider chains in one 18-pin dual in-line package. If you are unable to find a K1135A, you can substitute the alternate circuit, shown in figure 3, that uses a 74HC4060 oscillator/divider chip.

The address, data, and control signals are buffered to allow for the extra loads placed on them by circuitry on the monitor board. If these signals are also buffered on the target board, it may cause excessive propagation delays. If necessary, replace one set of buffers with jumpers. Jumpers are also provided to choose either the on-board crystal or an external clock

(continued)

from whatever application board you are using the emulator with. Similarly, there is a jumper selection enabling the on-board reset button, the application board reset, or both. Another jumper allows connecting or isolating the grounds between the target and emulator boards.

OPERATION OF THE SOFTWARE

First, edit and assemble your source code. Program the object code in EPROMs and plug them into the application code space. Note that the code will be executed out of the emulator board's RAM. Therefore, the origin statement in the source code should specify 4000 hexadecimal, the start of RAM, rather than 2000 hexadecimal, the start of the application EPROM. The interrupt vectors in the monitor EPROM all jump to the equivalent offset in the RAM space, allowing use of all the interrupts with only a slight additional latency. The reset vector, however, jumps to the monitor initialization code. This scheme allows the application EPROMs, burned for development purposes, to be used in the final project in some cases. In the products I developed with the aid of this monitor, the memory decoding was incomplete. The same EPROM was addressed at 4000 hexadecimal and 000 hexadecimal. [*Editor's note: The source code for the author's monitor program, UGHBUG.ASM, is available in a variety of formats. See pages 459–461 for details.*]

THE COMMANDS

The commands are invoked by the first character of the command name. There are a few exceptions to this rule. The internal varieties of the DUMP and ALTER commands (which access the 8031's internal memory) have an "I" appended to distinguish them from their external equivalents. The HEXMATH command is invoked with a number sign (#). All numeric values are expressed in hexadecimal. If you mistype an address or a data value, just keep typing. The monitor accepts the last four digits entered for address values and the last two for data values. You need not type leading zeros unless you're covering up a mistake. To cancel a command that you've started to type, just type any illegal character. To abort a command that outputs to the screen, merely type any character. These rules are consistent for all the commands.

The first command needed in any debugging session is the COPY command. Copy the application code from the EPROM to the RAM. Be sure not to overwrite the last 9 bytes of RAM; these are used by the breakpoint routine.

Next, use the VERIFY command to make sure the transfer was successful. VERIFY indicates agreement between two blocks of memory by doing nothing. Any differences are displayed. (I never got around to adding a memory test. With only four RAM chips it

didn't seem worth the effort. Usually, corrupted memory was the result of stack overflow or some other errant code.) Always VERIFY the damages after your code goes into the weeds.

Run your code using the GO command. If you do not specify an address, execution will restart at the breakpoint. (I'll talk more about breakpoints later.) Although the most likely starting place is 4000 hexadecimal—the reset vector of the application code—you can GO to any address in the program memory.

I had to resort to some tricks in my monitor's code. The 8031 does have an indexed jump instruction, JMP @A + DPTR. Unfortunately, the data pointer is only easily loaded by a constant—a variable must be loaded a byte at a time—and the accumulator is only 8 bits. Besides that, I wanted to be able to restore the data pointer and the accumulator when resuming after a breakpoint. The simple solution, shown in listing 2, is to push the address onto the stack and execute a return-from-subroutine instruction.

Before running your code you may want to set a breakpoint. The BREAK routine requires a little explanation. Many processors have a single-byte software-interrupt instruction that can be used for breaking back to the monitor. This single byte may be substituted for the first byte of any instruction. When the software interrupt is executed, it transfers control to a breakpoint routine. The 8031 lacks such an instruction; you have to use a long jump instruction, which is 3 bytes long. The break address must be aligned with the first byte of an instruction so that it will be executed and not treated as data.

First you want to save the original instructions. Because an 8031 instruction may be 1, 2, or 3 bytes long, inserting the 3-byte jump instruction at a given point in the code may clobber a sequence of 3, 4, or 5 bytes of code. The number of bytes affected is the optional final parameter of the break command; 3 is the most convenient value and is the default. This parameter is stored at the location BYTENUM, and the bytes of code are stored in the following 5 bytes (padded with NOP instructions if necessary). Following this, the next 3 bytes of RAM are filled with a jump instruction to the location following the code that was copied, that is, BYTENUM bytes after the break address. After all this is done, a jump-to-the-breakpoint-routine instruction is written at the break address. When execution reaches the break address, control passes to the breakpoint routine. The breakpoint routine saves the registers that are treated as volatile memory and displays the values that they contained.

A GO command without a starting address resumes from the break address. First it restores the saved registers and executes the saved code.

(continued)

and then it jumps back to the application code following the break address. If you execute the breakpoint routine, you should reset the board before attempting to initiate a GO command to a starting address. This will recover the 5 bytes of stack space used to store the volatile registers.

The code saved at location BYTENUM+1 is restored to its original location when a new break address is entered or if the BREAK command is invoked without a new address. You should do this before a reset because the initialization code clears BYTENUM and the following locations.

The DUMP command comes in two flavors, internal and external. The external version does a memory dump of the program memory in hexadecimal, showing the ASCII translation to the right (see figure 4). If no ending address is specified, 0FFFFH is assumed. A DUMP may be interrupted, as can any command that writes to the screen, by typing any key.

The internal DUMP command is similar but dumps the internal RAM and the special function registers. No ASCII translation is shown since it is unlikely to find ASCII strings in internal memory. The special function registers are indicated symbolically in addition to their addresses. This is shown in figure 5.

The ALTER command also comes in two flavors. The external version displays the current byte from program memory followed by a dash. If you enter a hexadecimal value, that value will be inserted at that location in external data memory. A space or a carriage return leaves the location unchanged and displays the next byte (carriage return puts you on a new line, space leaves you on the current line). A period or a backspace backs up the displayed byte by one location. Any other nonhexadecimal character cancels the command. Remember that the RAM on the monitor board is mapped to both the program memory and the data memory. The ALTER command writes to data memory because there is no way to write to program memory. It is frequently convenient to use the ALTER command to check code, hitting carriage return after each instruction for readability.

The internal form of the ALTER command accesses the 8031's internal RAM. As in the DUMP command, when you reach the special function registers (locations above 7F hexadecimal) the name of the register is displayed.

(continued)

Listing 2: *The author's solution to the 8031's lack of an indexed jump instruction.*

```
060F            GXXXX:
060F C049               PUSH    LOBYTE
0611 C048               PUSH    HIBYTE
0613 22                 RET
```

```
UGH:d 23 00b1
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0020                  02 40 23 02 00  F5 02 00 F8 02 06 B3 02      .@#.. ........
0030  06 C6 02 06 F0 02 07 20  02 07 48 02 07 C9 02 06     ....... ..H.....
0040  D7 02 06 CD 02 07 6C 02  07 69 02 07 A7 02 07 A1     ......l. .i......
0050  02 08 16 02 07 70 02 06  AD 02 07 BD 02 07 AF 02     .....p.. ........
0060  07 EA 02 05 91 0D 0A 55  67 68 62 75 67 20 4D 43     .......U ghbug MC
0070  53 2D 35 31 20 6D 6F 6E  69 74 6F 72 2C 20 76 65     S-51 mon itor, ve
0080  72 73 69 6F 6E 20 31 2E  30 30 0D 0A 63 6F 70 79     rsion 1. 00..copy
0090  72 69 67 68 74 20 31 39  38 36 20 62 79 20 47 65     right 19 86 by Ge
00A0  6F 72 67 65 20 44 69 6E  77 69 64 64 69 65 2E 0A     orge Din widdie..
0080  04 75                                                .u
UGH:
```

Figure 4: *Sample output of the external version of the* DUMP *command.*

```
UGH:di 69

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
60. .'                        9E AC C9 1C CE CC E4
70   88 90 80 30 02 20 88 A2  DF 9B EC E4 B8 46 E6 E4

80=P0  :55  81=SP  :52  82=DPL :69  83=DPH :08  87=PCON:7F  88=TCON:00
89=TMOD:00  8A=TL0 :00  8B=TL1 :00  8C=TH0 :00  8D=TH1 :00  90=P1  :FF
98=SCON:00  99=SBUF:00  A0=P2  :08  A8=IE  :60  B0=P3  :FF  B8=IP  :E0
D0=PSW :29  E0=ACC :01  F0=B   :0D
UGH:
```

Figure 5: *Sample output of the internal version of the* DUMP *command.*

Both the DUMP and ALTER commands use the SFR table (special function registers table—it can be found near the end of the monitor's source code listing) to access these registers. The special function registers cannot be accessed indirectly, and the memory space they occupy is sparsely populated. The SFR table provides a solution to both of these problems. Each table entry is 11 bytes long. The first 5 bytes give the symbolic name of the register. At an offset of five is a subroutine to read the register. Within this subroutine, at an offset of six, is the hexadecimal address of the register. Eight bytes from the start of each entry is a subroutine to alter the contents of the register. Some registers, such as the stack pointer, would crash the system if they were altered, so these entries return an error flag instead. Some registers, such as the accumulator, are treated as volatile by the monitor. You can alter them, but the monitor makes no attempt to preserve the contents of these registers.

The MODIFY command allows you to enter information into external data memory as characters instead of hexadecimal values. There are only two special characters to remember within this command. A backspace backs up the address pointer to allow the correction of mistakes. An EOT (Control-D) terminates the command. There is no internal version of the MODIFY command since it seems unlikely that the limited internal RAM would be used for storing strings.

The INSERT command fills a selected block of external RAM with a single hexadecimal byte. I used INSERT rather than FILL because I intended to add a FIND command.

The final three commands, HELP, JUMPTABLE, and HEXMATH, are conveniences. HELP displays the syntax of the commands and JUMPTABLE displays the entry points to the utility subroutines. HEXMATH, invoked by #, performs both addition and subtraction in hexadecimal. This is convenient for calculating relative jumps.

Some people might consider these last commands to be insignificant, but I disagree. It takes a short time to forget the command syntax. The HELP command also makes this tool easily available to others. The JUMPTABLE command allowed me to create test patches easily. Before I added this command I was constantly searching for the monitor program listing to look up the jump-table addresses. The HEXMATH is a cheap convenience—convenient to include when compared with the trouble caused by missing a jump target by 1 byte. Even simple software such as this should attempt to be as helpful as possible—there are better ways to spend time.

CONCLUSION

In this article, I have focused on one particular implementation of a debug monitor. If you are using a different processor it is well worth the effort to create your own. In addition to the debugging help, writing a monitor is a good exercise. It helps you to become familiar with a new instruction set.

Build your monitor one function at a time. First start with displaying a sign-on message. Then accept input and echo it back to the screen. Once you have the terminal interface working, add the more basic commands such as DUMP and ALTER. At this point you will have tools to aid you in developing the rest of the code.

It took me two weeks to develop the hardware and enough of the software to start using my new tool to develop application code. A year later I was still adding features and refining functions. Every minute I spent working on the monitor was quickly repaid in time saved. In the first week I used it, I accomplished six weeks of debugging measured by my previous standards.

What features would I like to add to this or any other monitor? A LOAD command to download a hexfile directly to RAM would be first choice. A FIND or SEARCH command to pick out variable-length byte sequences would be nice. A disassembler and single-line assembler would be a great help. These two are big jobs, though. I got to be pretty good at patching code with hand assembly, but I was doing it every day. The list of features could go on and on. A monitor is never finished until you quit using it. ■

Special thanks to Jim Gaudreault.

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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

14 DE AGOSTO AL 12 DE SEPTIEMBRE 1992

SISTEMA DE INSTRUMENTACION DIGITAL

ING. ANTONIO SALVA CALLEJA
ING. AGUSTIN SOTO URRIETA
ING. GLORIA CORREA PALACIOS

AGOSTO-SEPTIEMBRE
1992

SISTEMA MINIMO DE MICROPROCESADOR CONTROLADO POR UNA MICROCOMPUTADORA

AUTORES: ING. ANTONIO SALVA CALLEJA.

ING. VICTOR MANUEL SANCHEZ ESQUIVEL .

FACULTAD DE INGENIERIA .

DIVISION DE INGENIERIA MECANICA Y ELECTRICA .

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO .

CIUDAD UNIVERSITARIA, C. P. 04510 .

R E S U M E N

Este trabajo presenta un sistema mínimo de microprocesador constituido por los siguientes elementos:

1.- CPU .

2.- Memoria RAM .

3.- Memoria EPROM .

4.- Tres puertos paralelos (Dos de salida y uno de entrada) .

5.- Un puerto serie .

6.- Ocho puertos de entrada analógicos .

7.- Dos puertos de salida analógicos .

El sistema es manejado por una microcomputadora a través del puerto serie - de la misma, pudiéndose editar, guardar en disco o leer de disco programas- en lenguaje de máquina del CPU empleado en el sistema mediante la microcomputadora, de esta manera se puede tener un medio de almacenamiento permanente y versátil para dichos programas, ya sea en disco flexible o duro, dependiendo de la microcomputadora empleada, facilitándose así el desarrollo del software para el sistema que fuere necesario en una aplicación específica- del mismo .

INTRODUCCION

En la figura 1 se muestra un diagrama de bloques simplificado del sistema mínimo de microprocesador que de aquí en adelante se denominará SIMMP-1; puede apreciarse que el CPU empleado es el Z-80 de ZILOG que si bien es un microprocesador que apareció hace varios años en el mercado sigue siendo a la fecha sumamente versátil para las aplicaciones hacia las cuales esta orientado el SIMMP-1, además de que su costo es actualmente el más bajo en el mercado, siendo además los periféricos comunmente asociados con este microprocesador fácilmente acequibles en el país .

Para establecer la comunicación entre la microcomputadora y el SIMMP-1 se emplea el chip 8251 que es un trasmisor-receptor universal síncrono y asíncrono (USART) fabricado por Intel. Al inicializarse el sistema el puerto serie, queda habilitado para transmitir y recibir información a 300 bauds, pudiéndose mediante un comando enviado por la microcomputadora cambiarlo a 1200 bauds.

En la EPROM 2716 existe un programa que toma una cadena de bytes procedentes de la microcomputadora localizándolos en una zona especifíca de la memoria RAM 2016 autoejecutándose el programa correspondiente en el SIMMP-1 .

A continuación se describen algunos aspectos relevantes del software y del hardware relacionados con el SIMMP-1 y la microcomputadora empleada para comandarlo; así como también un ejemplo de aplicación del sistema .

PAGINACION DE MEMORIA DEL SIMMP-1

En la figura 2 se muestra el mapa de memoria del SIMMP-1 .

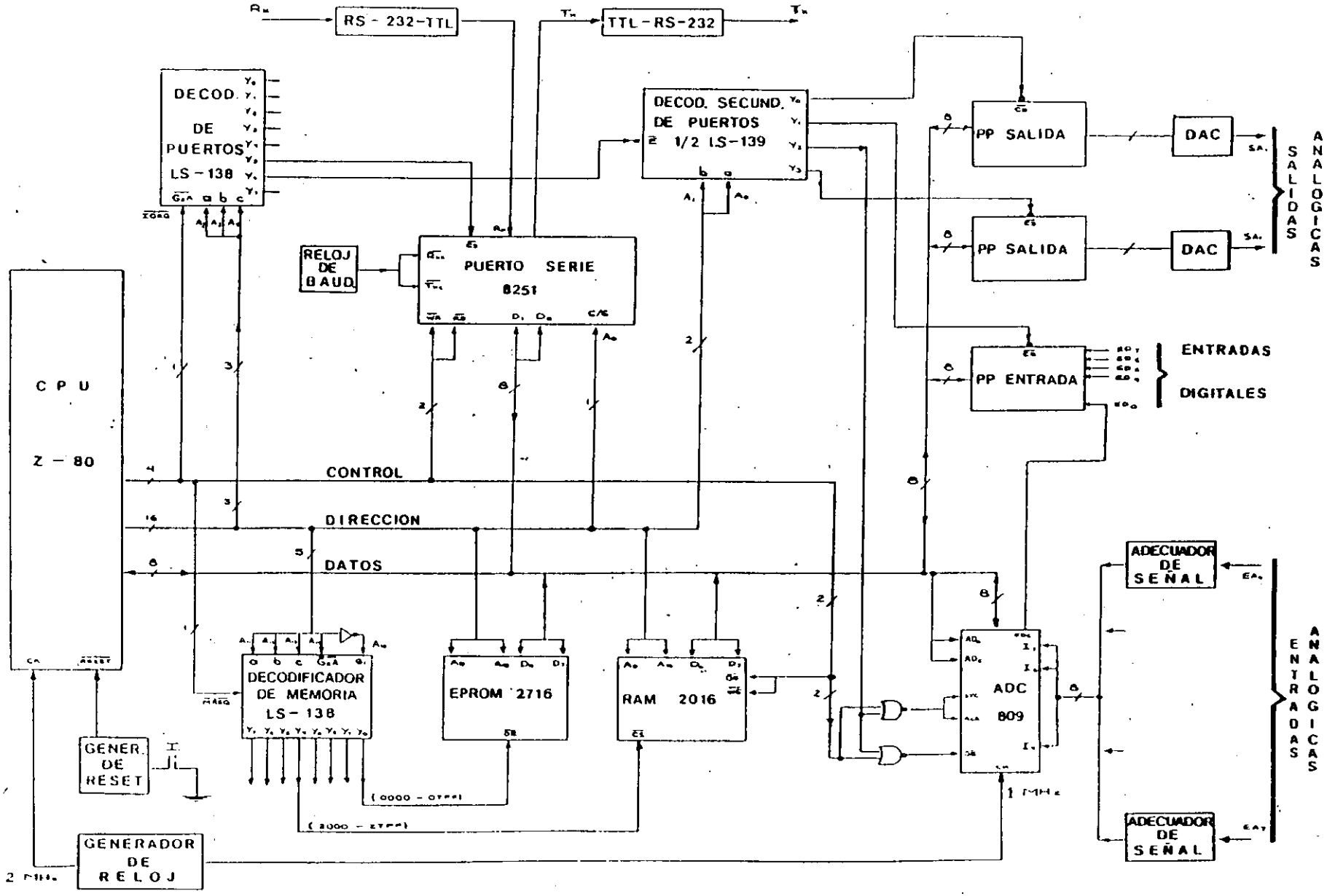


FIGURA 1: DIAGRAMA DE BLOQUES SIMPLIFICADO DEL SISTEMA MINIMO DE MICROPROCESADOR SIMMP - I

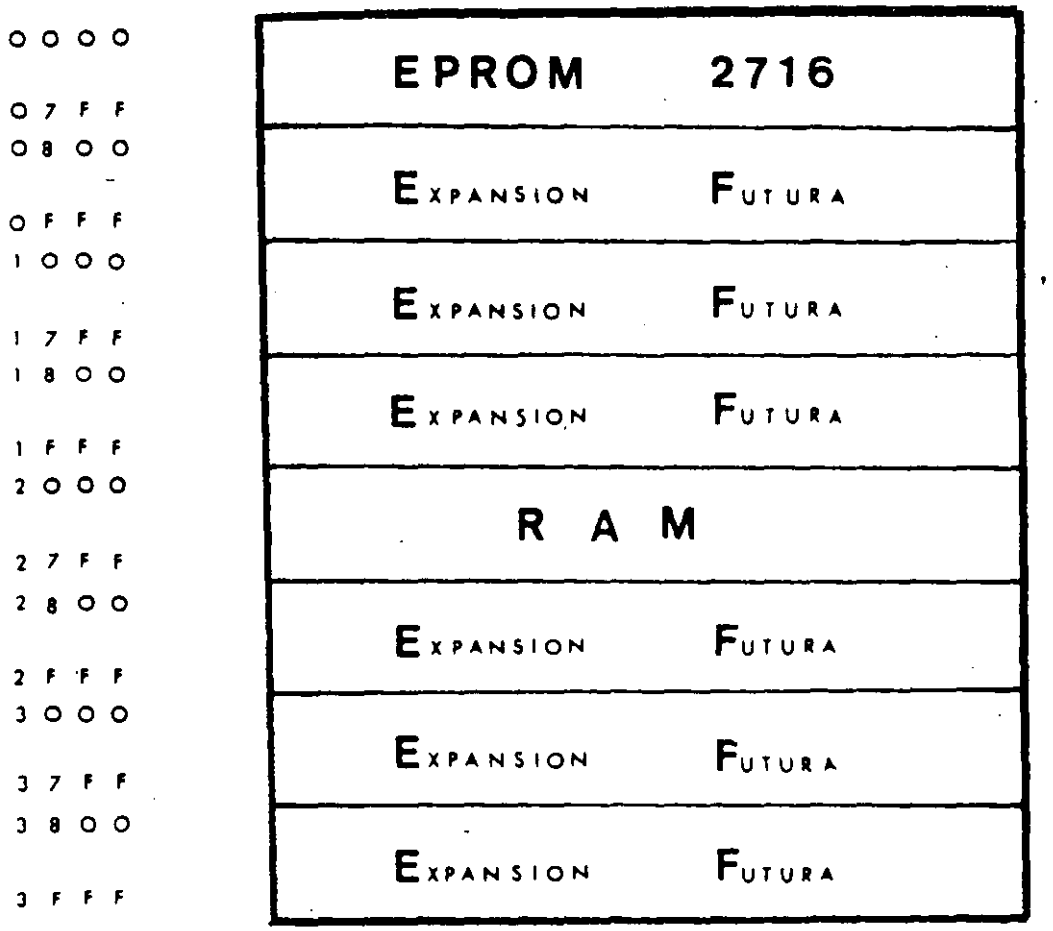


FIGURA 2 MAPA DE MEMORIA DEL **SIMMP-I**

Puede apreciarse que la paginación está hecha en bloques de 2K bytes pudiéndose decodificar hasta 16K bytes, esto es de la dirección 0000 a la dirección 3FFF. En la versión inicial del sistema únicamente se ocupan 4K bytes de memoria; 2K bytes de RAM situados de la dirección 2000 a la dirección 27FF y 2K bytes de EPROM situados de la dirección 0000 a la dirección 07FF; quedando disponibles 12K bytes de memoria para expansión futura .

Físicamente la paginación de memoria se realiza mediante el circuito integrado 74LS138 como puede apreciarse en la Fig. 1.

PAGINACION DE PUERTOS

En la Fig. 3 se muestra el mapa de puertos del SIMMP-1 .

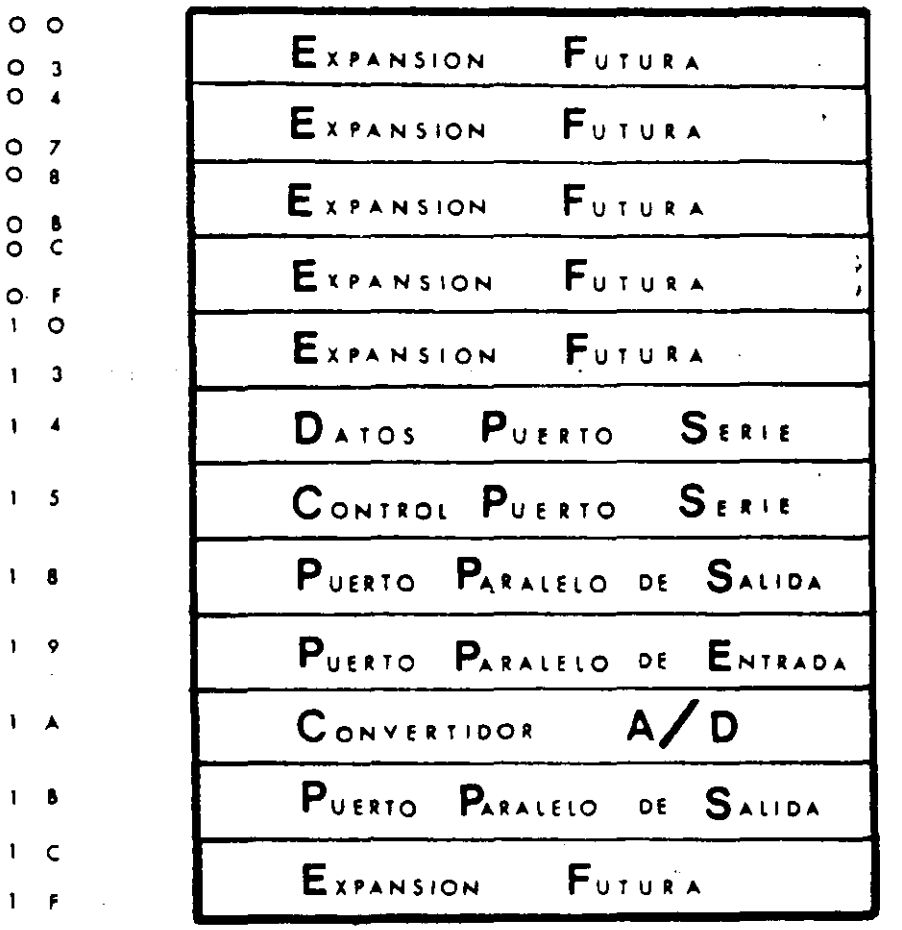


FIGURA 3 MAPA DE PUERTOS DEL SIMMP-I

Se observa que se dispone de un máximo de 32 direcciones asociadas con puertos de las cuales, la 14 y 15 están vinculadas con las direcciones de control y datos del puerto serie. Otras cuatro direcciones de la 18 a la 1B se usan para los 3 puertos paralelos y el convertidor analógico digital. Cabe señalar que los dos puertos paralelos de salida están vinculados con sendos converti

tidores digital analógico; las 26 direcciones restantes pueden usarse en expansiones futuras del SIMMP-1 .

La decodificación de puertos se lleva a cabo mediante dos circuitos integrados un 74LS138 y un 74LS139, como se observa en la Fig. 1, la porción no empleada del 74LS139 puede ser usada para decodificar otras cuatro direcciones de puertos; de esta manera sin necesidad de añadir más chips - decodificadores el sistema mínimo puede contar hasta con ocho puertos' .

R E L O J

El SIMMP-1 trabaja con un reloj de 2MHz, el circuito empleado para generar la señal correspondiente se muestra en la Fig. 4.

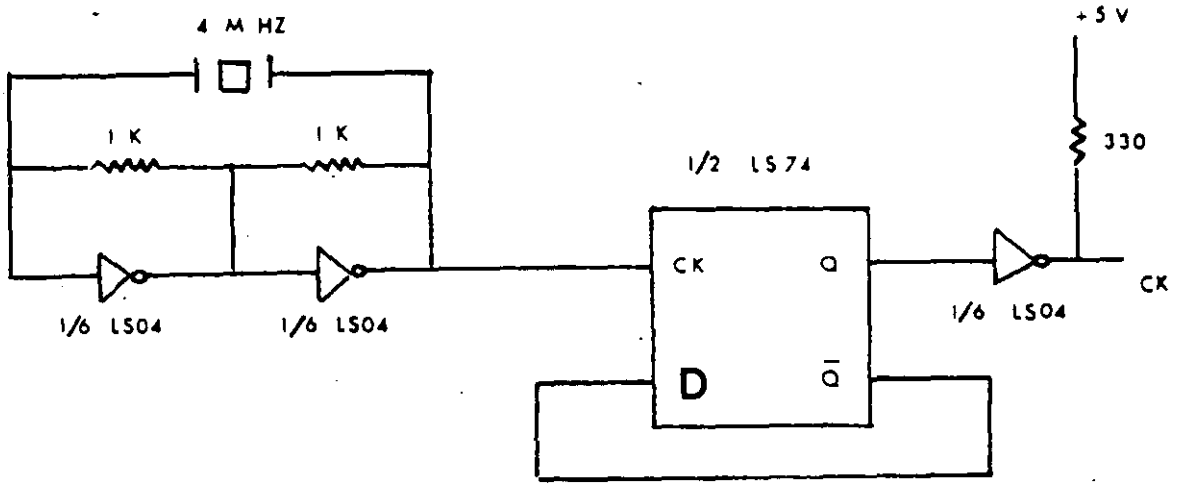


FIGURA 4 CIRCUITO EMPLEADO PARA GENERAR LA SEÑAL DE RELOJ DEL SIMMP-I

Para generar la señal correspondiente al reloj de baudaje se emplearon circuitos digitales convencionales que dividen la señal de reloj original de 2MHz entre 104 generandose así una señal de 19230 Hz, la cual hace posible

que el puerto serie pueda ser programado por software para poder trabajar ya sea a 300 bauds ó a 1200 bauds .

CIRCUITO DE RESTABLECIMIENTO (RESET)

El circuito para generar la señal de restablecimiento (Reset) en el --- SIMMP-1 se ilustra en la Fig. 5.

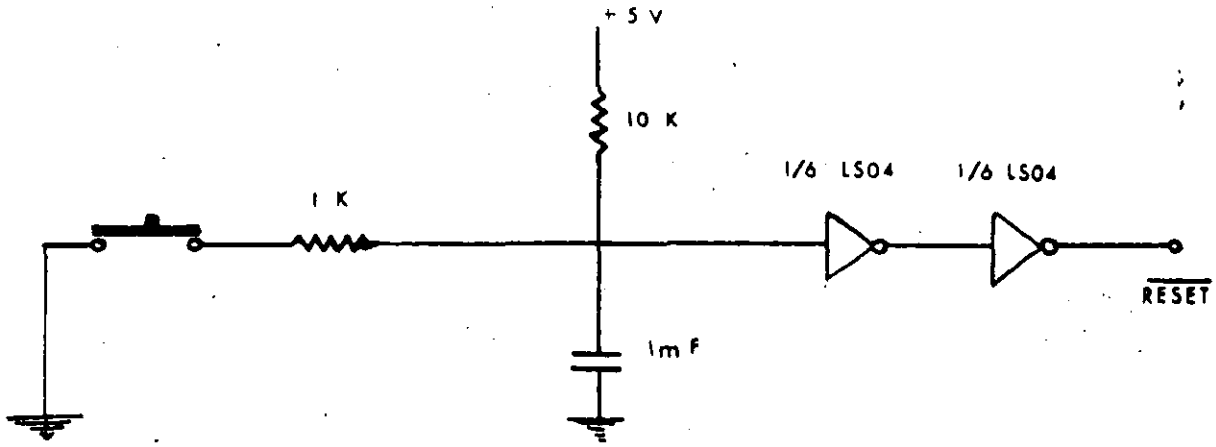


FIGURA 5 - CIRCUITO DE RESTABLECIMIENTO

PUERTOS PARALELO DE ENTRADA Y SALIDA

Debido a que los puerto paralelo del SIMMP-1 están ya prefijados como -- puertos de salida o entrada, para su realización se empleó el circuito - integrado 74LS373 resultando así el costo de los puertos más económico - comparado con el que se tendría si se hubiera empleado para el mismo fin un circuito integrado de alta escala de integración como el 8255 fabricado por Intel .

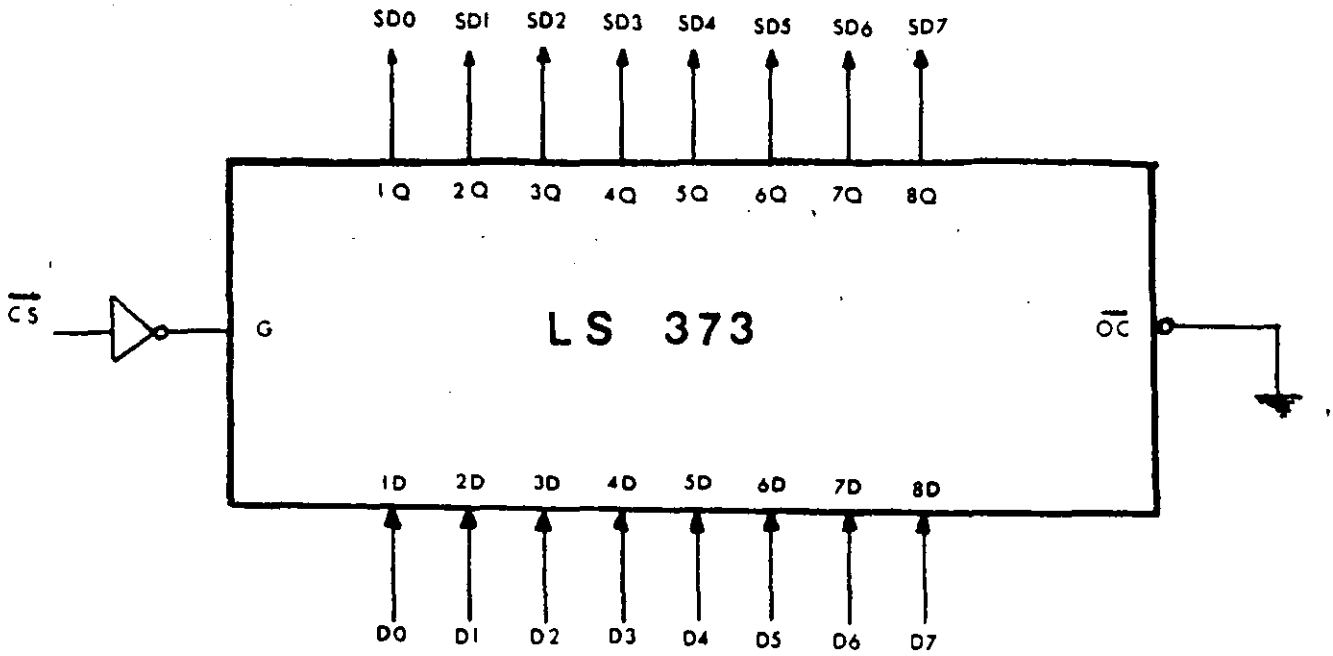


FIGURA 6 PUERTO PARALELO DE SALIDA

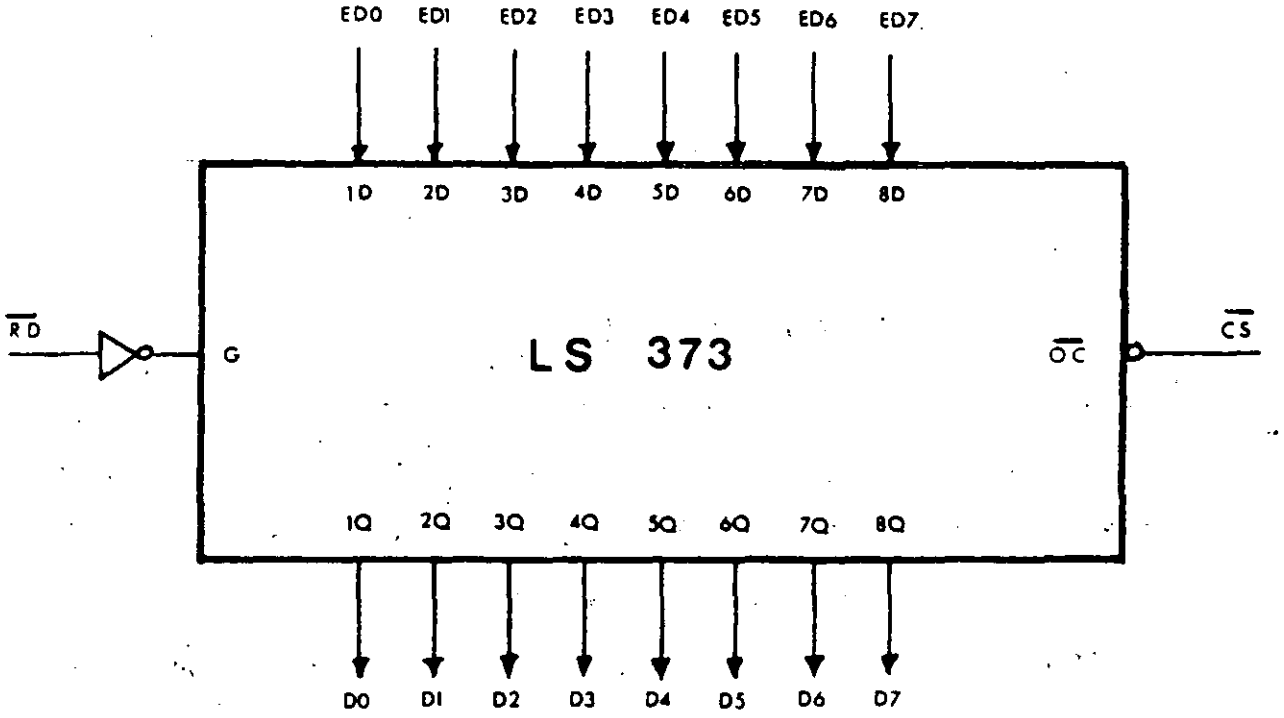


FIGURA 7 PUERTO PARALELO DE ENTRADA

En la Fig. 6 se ilustra el circuito correspondiente para realizar un puerto paralelo de salida; en la Fig. 7 el correspondiente a un puerto de entrada.

CIRCUITO ADECUADOR DE SEÑAL

Debido a que el rango de la señal de entrada para el convertidor analógico digital va de 0 a 5 volts, para poder registrar señales bipolares hay que hacer un acondicionamiento de las mismas. En el SIMMP-1 esto se hizo mediante un circuito analógico que transforma un rango que va de- 10 volts a + 10 volts en un rango que va de 0 volts a + 5 volts. En la Fig.8 se muestra el circuito correspondiente. En caso de que se deseara que el rango de las señales de entrada sea diferente se pueden emplear circuitos de atenuación o amplificación antes del adecuador de señal correspondiente .

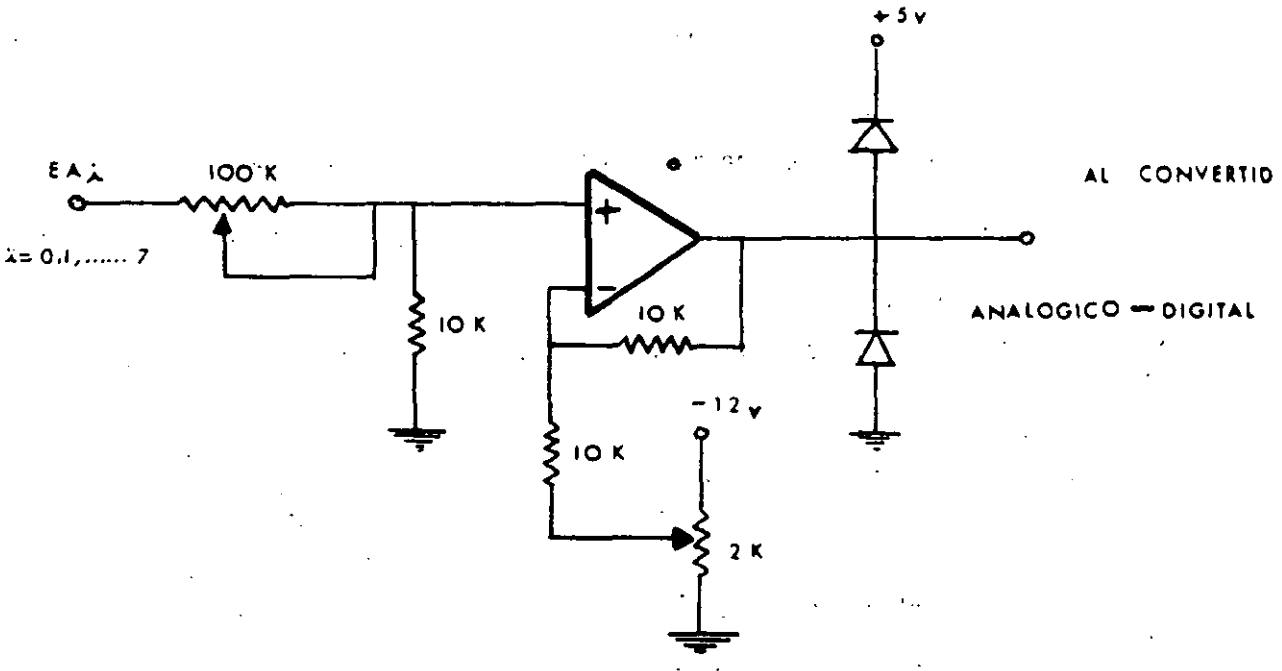


FIGURA 8 CIRCUITO ADECUADOR DE SEÑAL DE ENTRADA ANALOGICA

CONVERSION ANALOGICA DIGITAL

El convertidor analógico digital que se utilizó en el SIMMP-1 es el circuito integrado ADC0809 fabricado por National. Trabajando con un reloj de 1MHz, que se deriva del reloj del sistema, el tiempo de conversión es del orden de 70 microsegundos. En la Fig.1 se puede apreciar la conexión correspondiente .

SOFTWARE BASICO EN EL SIMMP-1

Como se menciona en la introducción la esencia del SIMMP-1 se encuentra en un programa grabado en la EPROM, dicho programa recibe de la microcomputadora una cadena de bytes que integran un programa en lenguaje de máquina del Z-80, autoejecutandose el mismo una vez que se ha terminado de bajar. En la Fig.9 se muestra un diagrama de flujo de dicho programa, que se ejecuta automaticamente al restablecer el sistema o bien al energizarse el mismo.

Al bajar al SIMMP-1 un programa, la secuencia que envía la microcomputadora es la siguiente:

1.- Envía un caracter de identificación que indica al SIMMP-1 que los caracteres que siguen representan a un programa para el microprocesador Z-80.

2.- Trasmite 4 bytes con información referente a la dirección inicial y la dirección final del programa a bajar .

3.- Trasmite la cadena de bytes correspondiente al programa que se este bajando .

Cabe señalar que en esta primera versión del SIMMP-1 no se manejan interrupciones, sin embargo el programa básico en la EPROM esta localizado en una zona de memoria que deja libre las localidades correspondientes que pudieran requerirse programar a futuro al trabajar con interrupciones .

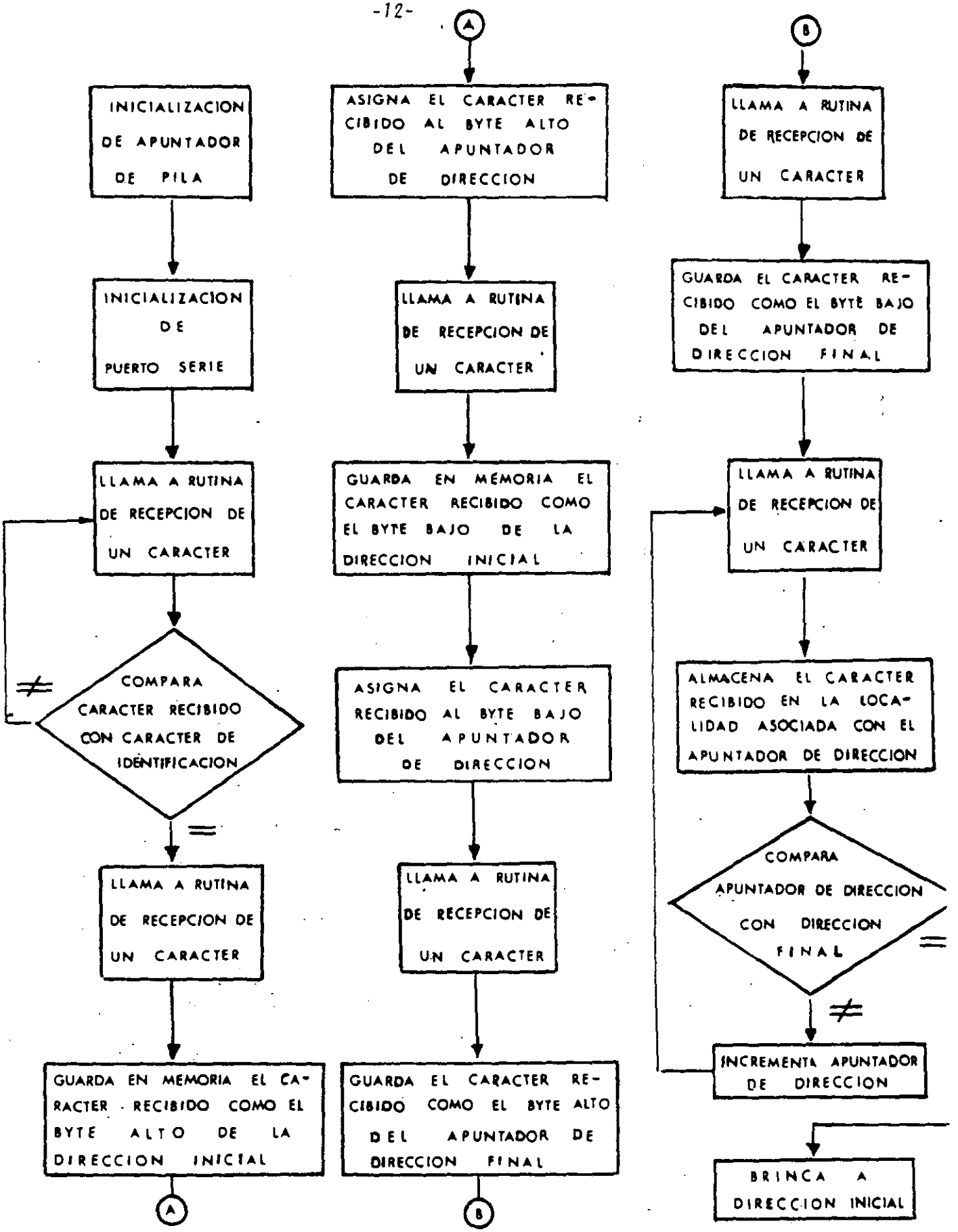


FIGURA 9 DIAGRAMA DE FLUJO DEL PROGRAMA DE VACIADO Y AUTOEJECUCION DE PROGRAMAS A TRAVES DE UNA MICROCOMPUTADORA EN EL SIMMP-I

SOFTWARE EN LA MICROCOMPUTADORA

En lo que toca al software del lado de la microcomputadora el lenguaje - utilizado fue el BASIC ya que para los requerimientos de velocidad de las aplicaciones hacia las cuales esta orientado el SIMMP-1, dicho lenguaje - es adecuado. En el programa básico se manejan dos menús uno de comando- y otro de edición. El menú de comando es el siguiente:

1.- Cargar un programa en lenguaje de máquina Z-80.

2.- Cargar de disco un programa para Z-80.

3.- Cargar en disco un programa para Z-80.

4.- Bajar a SIMMP-1 programa que se autoejecute.

5.- Editar programa.

6.- Definir baudaje .

7.- Terminar la sesión .

A continuación se describe brevemente lo que acontece al seleccionar cada una de las opciones del menú de comandos .

OPCION UNO

Al hacer esta selección el programa pide las direcciones inicial y final - así como también el nombre del programa que se va ha cargar. Una vez efectuado lo anterior el programa despliega en la pantalla de la microcomputadora las direcciones en forma sucesiva debiendo el usuario introducir cada vez el valor del byte correspondiente en notación hexadecimal. El programa asigna a cada dirección una variable de tipo cadena (String) que represen ta lo que ha de almacenarse en las locaciones de memoria correspondientes- del SIMMP-1, de esta manera la cadena de bytes que representa al programa- queda contenida en un arreglo cuyo tamaño es igual al número de bytes que- integran el programa.

OPCION DOS

Cuando se escoge esta opción la microcomputadora requiere del usuario el - nombre del programa a tomar de disco y la unidad correspondiente. Una vez

que se ha ejecutado esta opción la cadena de bytes correspondiente al programa que se ha tomado de disco queda en un arreglo similar al resultante al de la opción uno.

OPCION TRES

Mediante esta opción el arreglo correspondiente a un programa determinado se guarda en un archivo de disco junto con las direcciones inicial y final correspondientes .

OPCION CUATRO

Esta opción permite bajar al SIMMP-1 las direcciones inicial y final de un programa para Z-80, así como también la cadena que lo constituyen. - Una vez que el programa es bajado se autoejecuta .

OPCION CINCO

Al seleccionar esta opción se pasa al menú de edición que se describe -- más adelante.

OPCION SEIS

Si se desea cambiar el valor de baudaje, esto se puede hacer mediante esta opción; los baudajes posibles en esta primera versión del SIMMP-1 son- 300 y 1200 .

OPCION SIETE

Escogiendo esta opción el usuario regresa al sistema operativo de la mi-- crocomputadora .

El menú de edición consta de cinco alternativas a saber:

1.- Insertar bytes.

2.- Borrar bytes .

3.- Listar .

4.- Cambiar bytes .

5.- Continuar .

A continuación se describe brevemente el accionamiento de cada alternativa.

ALTERNATIVA UNO

Esta alternativa, permite insertar instrucciones en lenguaje de máquina en un programa en el cual este trabajando el usuario .

ALTERNATIVA DOS

Mediante esta alternativa el usuario puede borrar instrucciones del programa en lenguaje de máquina del Z-80 que se este depurando .

ALTERNATIVA TRES

Para poder listar todo un programa en lenguaje de máquina para Z-80 ó una parte del mismo se emplea esta alternativa.

ALTERNATIVA CUATRO

Esta alternativa permite cambiar una o varias instrucciones de un programa para Z-80 con el cual se este trabajando .

ALTERNATIVA CINCO

Al seleccionar esta alternativa el sistema pasa a un submenú que comprende las siguientes acciones:

1.- Editar .

2.- Retornar a menú principal .

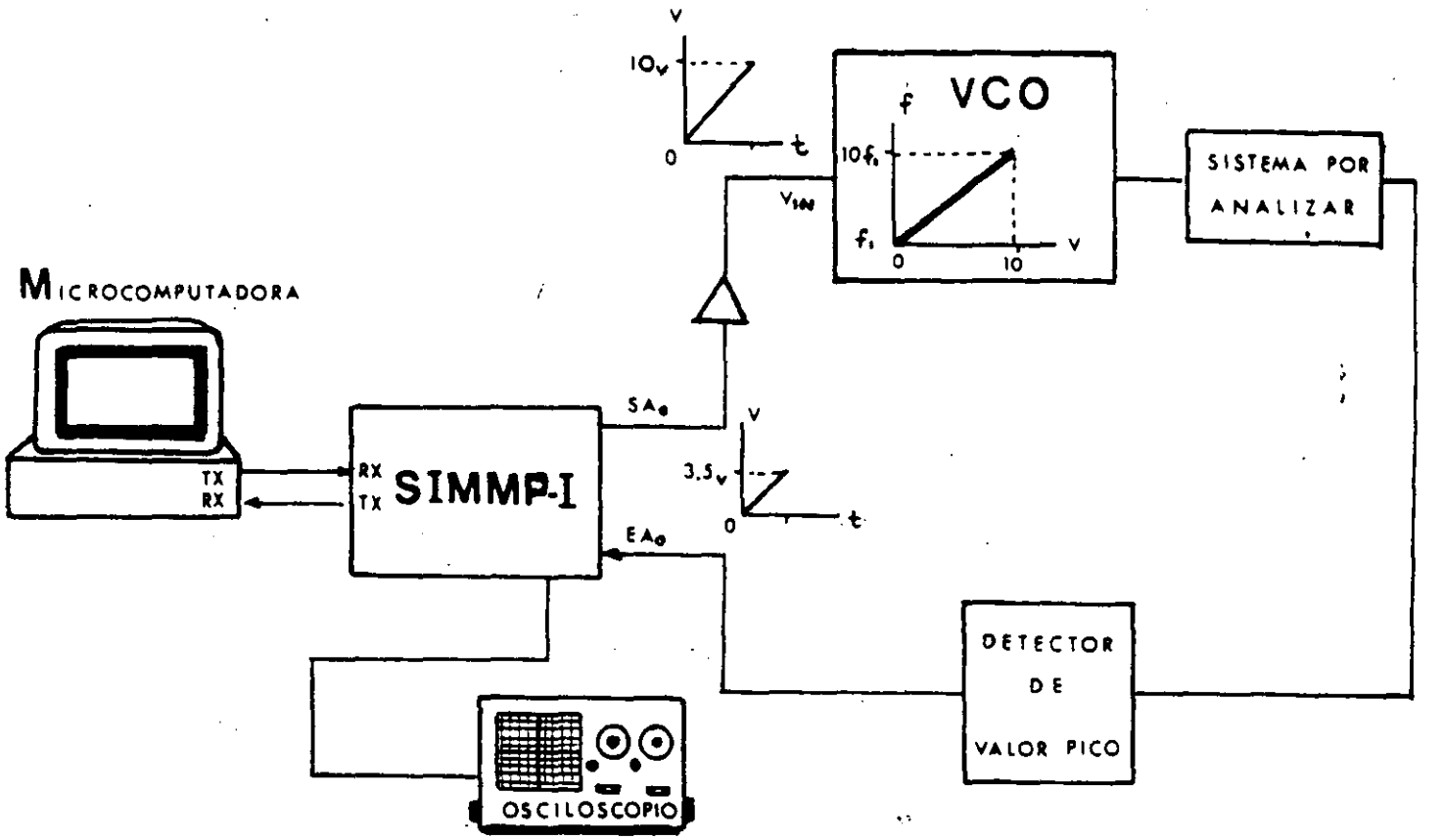
Si se toma la acción 1 el usuario retorna al menú de edición, si se toma la acción 2 el usuario retorna al menú principal ó de comandos. Cabe señalar- que una vez que se ha ejecutado cualquiera de las opciones 1 a 6 del menú - de comandos, el submenú descrito en este párrafo aparece nuevamente .

Para cada uno de los comandos del menú principal existe una subrutina que - ejecuta la opción correspondiente, de esta manera si el SIMMP-1 se emplea - como hardware auxiliar de un programa en la microcomputadora, para poder co mandarlo desde la misma basta que dentro del programa en la microcomputado- ra se genere el arreglo que contiene la cadena de bytes correspondiente a -

un programa en lenguaje de máquina para Z-80 que realize en el SIMMP-1 el accionamiento requerido por el programa en la microcomputadora, para lo cual se requerirá que el programa en la microcomputadora cuente con la subrutina correspondiente a la opción cuatro del menú de comandos .

EJEMPLO DE APLICACION

Los posibles ejemplos de aplicación del SIMMP-1 son muchos y muy variados, para este trabajo se eligió un caso en el cual el SIMMP-1 es operado como hardware auxiliar de un programa en la microcomputadora . Mediante dicho programa y el SIMMP-1 se puede obtener la respuesta en frecuencia en magnitud de un sistema en el rango de una década. La respuesta en frecuencia correspondiente es desplegada en la pantalla de la microcomputadora y en la pantalla de un osciloscopio auxiliar. En la Fig. 10 se muestra un diagrama de bloques del sistema mencionado que de aquí en adelante se denominará Bodimetro de una década (BUD) . Como se aprecia en la figura, el VCO empleado barre un rango de frecuencia de una década cuando el voltaje de control al mismo varia de 0 a 10 volts. Dado que la salida analógica del SIMMP-1 varia en el rango de 0 a 3.5 volts, se requiere un amplificador de ganancia 2.857 para poder lograr el rango de barrido requerido por el VCO. Es conveniente señalar que tanto el VCO como el amplificador de rango requerido se encuentran en un solo aparato comercial fabricado por la compañía BWD ELECTRONICS denominado MINI-LAB; el instrumento mencionado permite al usuario seleccionar la frecuencia inicial de la década de barrido que se desee.



SISTEMA BUD

FIGURA 10 DIAGRAMA DE BLOQUES PARA OBTENER LA RESPUESTA EN FRECUENCIA DE UN SISTEMA, CON REGISTRO EN EL OSCILOSCOPIO Y PANTALLA DE MICROCOMP. EMPLEANDO EL SIMMP-I

El programa correspondiente en el SIMMP-1 que se ha llamado CV01 efectua las siguientes acciones:

a) Carga en registros internos del CPU las direcciones inicial y final de una tabla de 256 bytes donde han de almacenarse los valores de -

la magnitud de la respuesta en frecuencia del sistema que se esté - analizando:

b) El programa pasa a un lazo del cual no sale a menos que el bit 1 -- del puerto paralelo de entrada del SIMMP-1 cambie de 0 a 1. De esta manera puede inciarse el barrido de frecuencia en el momento en- que el usuario efectue físicamente el cambio mencionado .

c) Si se sale del lazo mencionado en el paso b), se inicia el barrido- de frecuencia, para lo cual el SIMMP-1 pone sucesivamente en un puer to paralelo de salida una cuenta que va de 00 a FF, de esta manerá- en el puerto analógico de salida correspondiente (SA_0) aparecerá- la rampa de barrido requerida por el amplificador de rango. Para - cada paso en la cuenta el SIMMP-1 lee el valor pico de la respuesta en frecuencia y lo coloca en una locación de la tabla asignada para tal fin, una vez hecho lo anterior el programa va a una subrutina - de retraso de 20 milisegundos e incrementa la cuenta, de esta manera una vez que se ha colocado la cuenta FF se ha concluido con el - llenado de la tabla que contiene 256 puntos de la magnitud de la -- respuesta en frecuencia del sistema analizado.

d) Concluida la acción anterior el SIMMP-1 sube a la microcomputadora- la tabla que contiene los datos de la respuesta en frecuencia requerida. Una vez terminada la transferencia, la microcomputadora despliega una gráfica ilustrando dicha respuesta .

e) La tabla de respuesta en frecuencia es reciclada en el puerto analógico de salida SA_1, con lo cual es posible desplegar tal respuesta en un osciloscopio.

Del lado de la microcomputadora se ejecuta un programa que como primera acción del mismo baja al SIMMP-1 el programa CVC01 descrito a grandes - razgos anteriormente; el programa en la microcomputadora contiene ade-- más las rutinas necesarias para interaccionar con el programa CVC01 ---

que se ejecuta con el SIMMP-1 .

En la Fig. 11 se muestra la gráfica desplegada en la computadora correspondiente a la respuesta en frecuencia del circuito mostrado en la Fig. 12 .

Dicho circuito puede emplearse como una de las bandas elementales de un ecualizador gráfico de audio.

Mediante análisis convencional de redes puede demostrarse que dicho circuito puede presentar una atenuación o amplificación de 12.5 decibeles a una frecuencia de 5.486KHz. La máxima amplificación se logra cuando K=0 .

Para obtener mediante el sistema de la Fig. 10 la magnitud de la respuesta en frecuencia del circuito mostrado en la Fig.12 se procedió de la siguiente manera:

a) Se calibró el VCO de modo que la frecuencia inicial de barrido fuera 1000 Hz .
b) Se ajustó la amplitud de la salida del VCO a 1 volt de pico .
c) Se pusieron en operación el sistema SIMMP-1 y la microcomputadora .
d) Se ejecuta el programa en la microcomputadora que usa el sistema --- SIMMP-1 como auxiliar en la obtención de la respuesta en frecuencia .
e) Una vez ejecutado el programa correspondiente en la microcomputadora, se obtuvo la gráfica mostrada en la Fig. 11.

Como puede apreciarse, los valores experimentales obtenidos para la ganancia máxima y la frecuencia a la cual esta ocurre difieren de los valores teóricos mencionados. Esto puede deberse a que el circuito analizado fue construido con elementos que presentan una cierta tolerancia en sus valores nominales.

En la Fig. 13 se muestra la curva de respuesta en frecuencia teórica del circuito de la Fig. 12

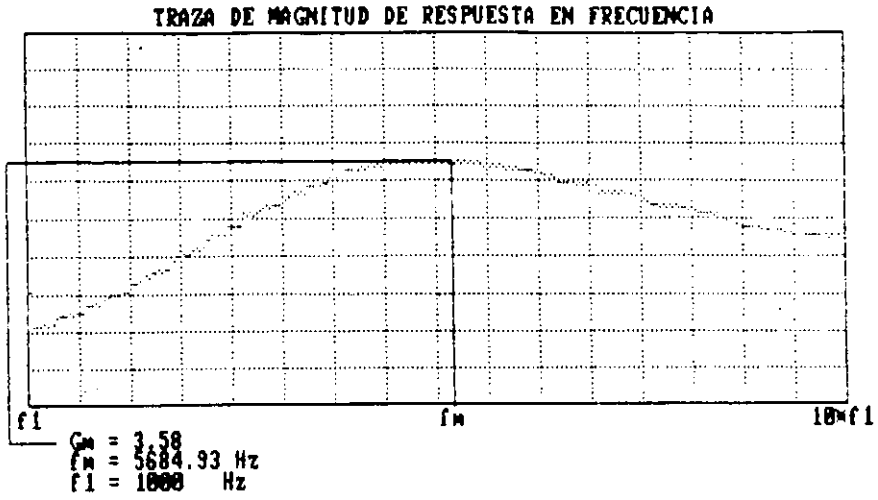


Figura 11: Respuesta en frecuencia del circuito de la figura 12 obtenida mediante el sistema de la figura 10

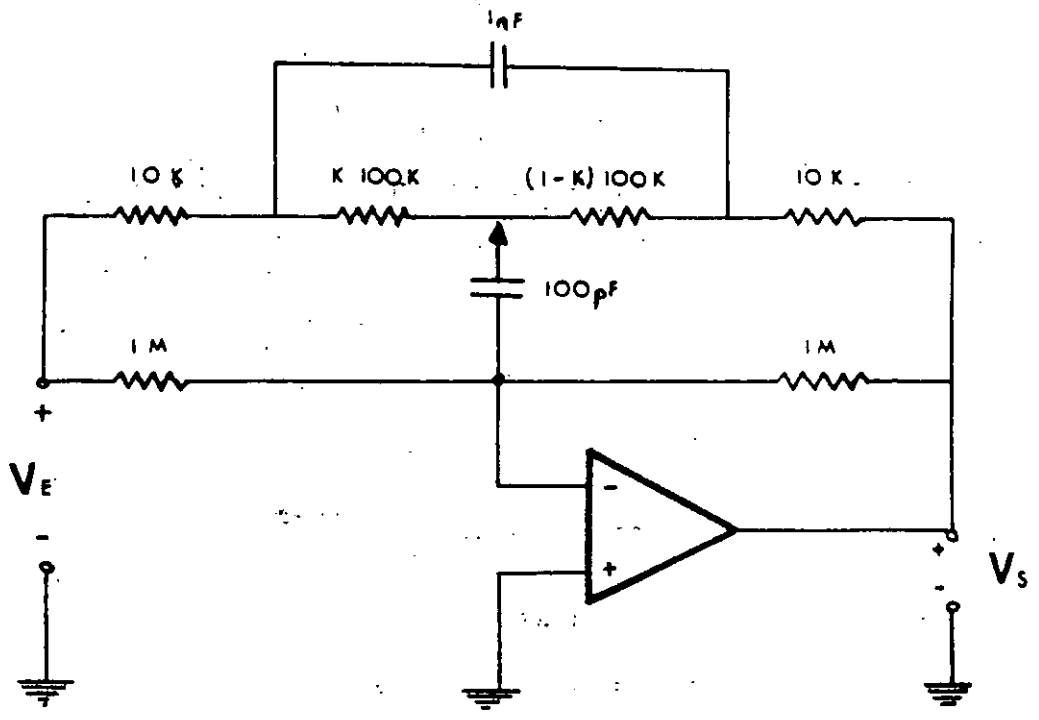


FIGURA 12 CIRCUITO ANALIZADO EN FRECUENCIA POR EL BUD

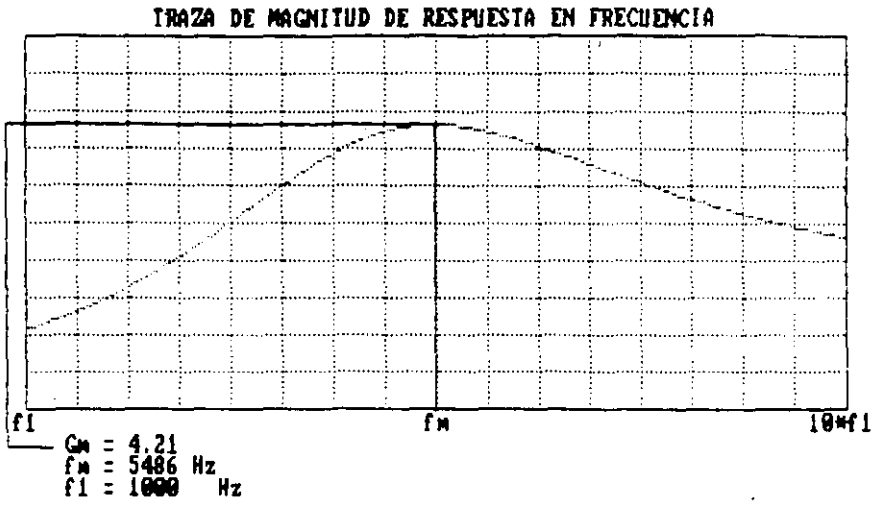


Figura 13:Respuesta en frecuencia teorica del circuito de la figura 12

Cabe señalar que el sistema BUD descrito anteriormente se alambró exprofeso para mostrar un ejemplo de aplicación del SIMMP-1, que pudiera construirse en un tiempo breve, por esa razón se utilizo un VCO que se controla analogicamente con los naturales problemas que esto implica .

Actualmente los autores trabajan en el desarrollo de un sistema que analice respuesta en frecuencia ampleando un generador de funciones monolítico al cual se le pueda agregar la circuiteria analógica y digital necesaria para que el SIMMP-1 pueda controlarlo digitalmente .

CONCLUSIONES

Como se ha apreciado en el presente trabajo las posibles aplicaciones del SIMMP-1 son muy diversas entre otras estas podrían ser :

a) Adquisidor de datos de baja velocidad, que pudieran ser procesados fuera de linea en una microcomputadora empleando un lenguaje de alto nivel.

b) Procesamiento en tiempo real de señales de muy baja frecuencia .

c) Control lógico de secuencias .

d) Enseñanza de la teoría y práctica de los microprocesadores .

e) Instrumentación electrónica empleando microcomputadoras y microprocesadores .

Si bien se trabajó en este desarrollo alrededor de un microprocesador de 8 bits es claro que la misma idea básica puede realizarse empleando microprocesadores más evolucionados tales como el 8086 de Intel y el 68000 de Motorola. Los autores planean a futuro desarrollar una nueva versión del SIMMP-1 basada en alguno de los microprocesadores mencionados anteriormente .

REFERENCIAS

The Engineering Staff of Texas Instuments Incorporated Semiconductor Group .

" The TTL Data Book for Design Engineers " .

Texas Instruments Incorporated, 1976 .

Garland H.

" Introduction to microprocessor system design " .

McGraw Hill, 1979 .

Hayes J.P.

" Diseño de sistemas digitales y microprocesadores "

McGraw Hill, 1986.

National Semiconductor

" Linear databook "

National Semiconductor Corporation, 1986 .

Hall D.V.

"Microprocessors and interfacing.Programming and hardware"

McGraw Hill, 1986 .

Hall D.V.

"Microprocessors and digital systems"

McGraw Hill, 1983 .

SISTEMAS PARA DESARROLLO CON MICROPROCESADORES AUXILIADOS POR UNA COMPUTADORA DE TIPO PC

En temas anteriores se explicó la estructura básica de una microcomputadora digital basada en un microprocesador y circuitos integrados complementarios, una característica notoria de un sistema mínimo como el descrito en forma genérica en la figura 3.10 del tema anterior, es la ausencia en el mismo de lo que en la literatura de computación se denomina bajo el nombre de memoria secundaria (v.g.cinta magnética,unidades de disco duro o flexible, etc) esto es común en sistemas basados en microprocesador que estén dedicados a realizar labores de instrumentación o control, ya que en tales casos la microcomputadora podrá estar destinada únicamente a atender un instrumento o servir de adquisidor de datos y enlace con otra computadora de mayor capacidad, por lo tanto, el programa que se debe ejecutar usualmente se encontrará almacenado en memorias de tipo ROM o EPROM. Es claro que el firmware contenido en memorias de sólo lectura debe haber sido probado previamente, para llevar a cabo esto se puede utilizar cualquiera de las siguientes herramientas:

1) Sistemas de desarrollo autocontenidos (SDA) con firmware de soporte que permiten al usuario probar programas introduciendolos a memoria RAM mediante un teclado. Tales sistemas por lo regular cuentan con las siguientes facilidades:

a) Desplegado hexadecimal de seis dígitos; cuatro de ellos están dedicados a mostrar direcciones de memoria, los otros dos se emplean para indicar el contenido de la dirección de memoria

desplegada en los otros cuatro. Como la unidad de despliegue está ligada con un puerto del sistema el usuario puede emplearla para mostrar cualquier otra información relacionada con la aplicación que el mismo le este dando al sistema, desde luego que para hacer esto el programador debera contar con información adecuada acerca de la arquitectra del sistema. Usualmente los desplegados de estos sistemas son de siete segmentos.

b) Un teclado mediante el cual el usuario puede introducir a memoria RAM un programa en lenguaje de máquina empleando para ello notación hexadecimal. El teclado cuenta con varias teclas de proposito especifico dedicadas a comandos tales como: ejecución de un programa a partir de la dirección mostrada en el deplegado, modificación del contenido de registros internos de la CPU previamente a la ejecución de un programa, lectura de el contenido de locaciones de memoria lectura o escritura de puertos, etc. . Conociendo la arquitectura del sistema el usuario podrá emplear el teclado para propósitos especificos relacionados con la aplicación que en un momento dado el mismo este programando.

c) Algunos sistemas de este tipo cuentan con programadores de memorias de tipo EPROM mediante los cuales el usuario puede pasar a un medio de almacenamiento permanente un programa que ya haya sido depurado.

d) Permiten colocar *puntos de ruptura* que facilitan la depuración de programas. Un punto de ruptura es una facilidad que detiene

la ejecución de un programa en una dirección previamente definida por el usuario, desplegandose entonces el estado de diversas variables lógicas tales como el valor contenido en los registros internos de la CPU.

Al usar una herramienta como ésta se presenta el inconveniente de que la memoria RAM es volátil por lo que el desarrollador deberá continuamente programar y borrar memorias de tipo EPROM, lo que complica mucho la fase de desarrollo del sistema basado en microprocesador que se esté diseñando en un momento dado.

2) Emuladores en circuito (In-Circuit Emulators), este tipo de herramientas permiten emular diversos microprocesadores y por lo regular operan en combinación con otra computadora anfitriona (host) la que auxiliada por hardware dedicado permite hacer depuración al diseñar el firmware de soporte requerido para la aplicación que se esté desarrollando. El proceso a seguir a utilizar una herramienta como ésta es a grandes rasgos el siguiente:

a) A la tarjeta basada en el microprocesador diseñada exprofeso para las necesidades requeridas por la aplicación de instrumentación o control se le conecta en lugar de una CPU normal una CPU con facilidades de monitoreo de señales propias de ella.

b) Mediante hardware intermediario entre la tarjeta que se esté desarrollando y la computadora anfitriona se puede hacer un seguimiento de diversas señales de la propia tarjeta. El despliegue de estas señales es hecho en la pantalla de la

computadora anfitriona.

c) Cuentan con facilidades que permiten colocar puntos de ruptura que auxilian en el desarrollo del firmware requerido por la aplicación.

3) Sistemas de desarrollo comandados por una computadora anfitriona *(host-target systems)*, aquí los designaremos bajo el nombre se sistemas HT, las características de este tipo de sistemas son las siguientes:

a) Constan de dos partes principales a saber:

i) Computadora anfitriona con software de soporte que arbitra el funcionamiento de una microcomputadora de una sola tarjeta basada en el microprocesador con el que se desee instrumentar una determinada aplicación.

ii) Microcomputadora en una sola tarjeta (target) que contiene circuitería de apoyo como puertos, memoria de sólo lectura (que contiene el firmware elemental que permita la comunicación con la computadora anfitriona), memoria RAM (a la cual la computadora anfitriona puede bajar programas para su ejecución) y circuitería lógica auxiliar que en conjunto con lo mencionado anteriormente constituyen un sistema con la arquitectura del que se muestra en la figura 3.10.

b) La comunicación entre los dos subsistemas que integran a un sistema HT se efectúa a través de puertos serie o paralelos.

c) La computadora anfitriona cuenta con software que permite entre otras cosas efectuar las siguientes acciones en la computadora destino: Leer el contenido de localidades de memoria o puertos

desplegando el valor en cuestión en pantalla, cargar memoria o puertos, bajar a la RAM un programa para su ejecución, colocar puntos de ruptura que faciliten la depuración de programas a ejecutarse, precargar registros internos de la CPU antes de ejecutar un cierto programa, ejecutar paso a paso un programa desplegando en la pantalla de la computadora anfitriona el estado de la CPU después de la ejecución de cada intrucción, cambiar la velocidad de transferencia de información *(baudaje)* entre las dos comutadoras cuando la comunicación es por puerto serie.

d) En la memoria secundaria de la computadora anfitriona se pueden guardar programas en código de máquina de la computadora destino para su posterior edición o ejecución, esto evita el tener que estar programando y borrando memorias EPROM al estar desarrollando una aplicación.

e) Algunos sistemas cuentan con facilidades para programar memorias EPROM que son parte de la arquitectura de la computadora destino. Lo anterior permite, una vez que ya se ha depurado y probado un programa cargarlo en memoria permanente a modo de que la computadora destino trabaje en forma autónoma.

f) Otra característica importante de los sistemas HT es el contar con un *ensamblador cruzado* (cross assembler) mediante el cual se puede obtener el código de máquina requerido por la computadora destino a partir de un programa en lenguaje ensamblador que se ensambla en la computadora anfitriona.

A continuación se describe un sistema HT basado en el

microprocesador Z80, que puede usar como computadora anfitriona a una PC XT/AT o PS/2 que cuente al menos con un puerto serie. Varias de las aplicaciones de microprocesadores que se expondrán en este curso se desarrollaron empleando este sistema, el cual se denomina con el nombre de *SIMMP-1* que son las siglas de Sistema *I*nterconectado de *M*icrocomputadora y *M*icroprocesador *P*aralelo, el dígito 1 indica que el microprocesador de la computadora destino es el Z80.

Es un sistema que se comanda por una microcomputadora a través de su puerto serie, consta de 2k de memoria RAM expandible a 4k, 2k de memoria EPROM expandible a 4k, lo que le da una capacidad total de memoria en la tarjeta de 8k, pudiendo expandirse con tarjetas adicinales hasta 64k; tiene además la opción de programación de la EPROM de expansión contenida en la tarjeta mediante el hardware que contiene el sistema.

En lo que a puertos se refiere el sistema cuenta con:

a) Dos puertos paralelos de 8 bits con capacidad para que el usuario pueda programarlos como entrada o salida.

b) Un puerto de comunicaciones en formato serie asíncrono implantado por medio de un USART comercial; mediante software el usuario puede modificar el formato de transmisión (vg. número de bits de datos, número de stop-bits, ausencia o presencia de bit de paridad). A este puerto se le denomina PS2 (puerto serie dos). Las velocidades de transmisión y recepción posibles del puerto PS2 son: 300, 600, 1200 y 2400 bps.

c) Un puerto de comunicación serie asíncrono implantado mediante

software que emplea un puerto de entrada de 1 bit y un puerto de salida de 1 bit. A este puerto se le denomina PS1 (puerto s e uno). Las velocidades de transmisión y recepción son idénticas a las del PS2, su formato de transmisión es fijo, 8 bits de datos, no paridad y un bit de paro.

El software de control del sistema reside tanto en él como en la microcomputadora (PC XT, AT, PS2) por lo que se proporciona al usuario un diskette que contiene la parte de software de control que debe ejecutarse en la microcomputadora.

El usuario con ayuda del software que se proporciona en disco puede examinar y/o cargar la memoria del sistema, ejecutar un programa a partir de una cierta localidad de memoria, bajar y autoejecutar un programa que previamente se introdujo en la microcomputadora en formato hexadecimal (lenguaje de máquina) contando con auxiliares para editar, guardar en disco o tomar d disco programas en formato hexadecimal; acomodar en el SIMMP-1 en un solo bloque un conjunto de programas en formato hexadecimal en una dirección de colocación inicial especificada por el usuario, ejecutar paso a paso un programa a partir de una dirección que también especifica el usuario con despliegue en la pantalla de la microcomputadora del contenido de los registros internos de la CPU y de locaciones de memoria de interés para el usuario; insertar puntos de ruptura en un programa que se ejecuta en el SIMMP-1 con despliegue del estado de la CPU al llegar a cada punto de ruptura, manejar los tres modos de interrupción enmascarable con que cuenta la CPU que se utiliza en el sistema, mover bloques de datos en la

memoria del sistema SIMMP-1, programar la EPROM de expansión a partir de: bloque de datos en formato hexadecimal contenido en disco, bloque de datos contenido en la memoria del SIMMP-1, bytes aislados tecleados junto con su especificación de dirección en la microcomputadora; mediante la modificación del estado de dos puentes eléctricos hacer que el sistema salte a la EPROM de expansión en el momento de restablecer o arrancar el sistema, esto último permite al usuario diseñar con base en el sistema SIMMP-1 dispositivos que funcionen de manera autómoma (vg. sin comandos de la microcomputadora) empleando el software de desarrollo del SIMMP-1 cuyas facilidades de programación se describieron brevemente en parrafos anteriores.

El SIMMP-1 cuenta con un ensamblador cruzado para el microprocesador del sistema escrito especialmente para él. Así el usuario puede editar, guardar o tomar de disco programas fuente en lenguaje ensamblador y una vez ensamblados guardarlos en disco en formato hexadecimal para su ejecución posterior o inmediata en el sistema SIMMP-1.

Aunque existen ensambladores cruzados en el mercado se decidió que el del SIMMP-1 fuera escrito en forma original por las siguientes razones:

a) El ensamblador cruzado forma parte del sistema.
b) Al escribir el ensamblador se crea una estructura básica de software a partir de la cual se hace fácil escribir ensambladores cruzados para otros microprocesadores o microcontroladores que existen en el mercado.

c) Tomar experiencia en el desarrollo de este tipo de herramientas de software.

Cabe mencionar que por medio de pequeñas modificaciones al hardware se pueden tener versiones con diferente capacidad de memoria y puertos o aún el empleo de otro microprocesador como CPU.

La aplicación original del producto es un sistema de desarrollo educativo que puede ser empleado también por profesionales de la ingeniería electrónica en el diseño de dispositivos profesionales de instrumentación digital. En tales casos el SIMMP-1 se usaría como circuitería básica funcionando en forma autónoma o bien como periférico de una microcomputadora, de tal forma que para el usuario final del sistema será irrelevante conocer el funcionamiento interno del dispositivo.

A sistemas con las características del SIMMP-1 se les denomin. HOST-TARGET; donde la parte que se llama "HOST" es la microcomputadora y la que se denomina "TARGET" es el sistema, en este caso el SIMMP-1. También se les llama "EMBEDDED SYSTEMS" a sistemas de microprocesador empleados como bloques básicos en la construcción de dispositivos de instrumentación y/o control para la industria y/o el hogar.

El rango de aplicaciones cerradas es amplio, sólo que en cada caso hay que desarrollar el sistema de puertos que se requiera, así como también la tarjeta de circuito impreso correspondiente, usando el sistema SIMMP-1 en estos casos como "cerebro" de la aplicación de que se trate en un momento dado.

En la figura 3.11 se muestra un diagrama de bloques del sistema SIMMP-1, como puede apreciarse la CPU empleada es la Z80 de Zilog, que si bien apareció hace varios años en el mercado, conserva su versatilidad para aplicaciones hacia las cuales está orientado el SIMMP-1. Además su costo es el más bajo del mercado y sus periféricos comúnmente asociados son de fácil adquisición.

Para establecer la comunicación entre la microcomputadora y el SIMMP-1 se emplea el chip 8251, transmisor-receptor universal síncrono y asíncrono (USART), fabricado por Intel. Al inicial el funcionamiento del sistema, el puerto serie queda habilitado para transmitir y recibir información a 300 bauds.

En la EPROM 2716 existe el firmware de soporte que permite a la microcomputadora bajar programas a RAM del SIMMP-1 y ejecutarlos para su prueba.

A continuación se describen algunos aspectos relevantes del software y hardware relacionados con el SIMMP-1 y la microcomputadora empleada para comandarlo.

Paginación de memoria del SIMMP-1

En la figura 3.12 se presenta el mapa de memoria del SIMMP-1. La paginación está hecha en bloques de 2k bytes pudiendose decodificar hasta 16k bytes, esto es, de la dirección 0000 a la 3FFF. En la versión inicial del sistema únicamente se ocupan 8k bytes de memoria, 4k bytes de RAM situados de la dirección 1800 a la 27FF y 4k bytes de EPROM situados de la dirección 0000 a la dirección 0FFF. Físicamente, la paginación de memoria se realiza mediante el CI 74LS138 (Fig. 3.11).

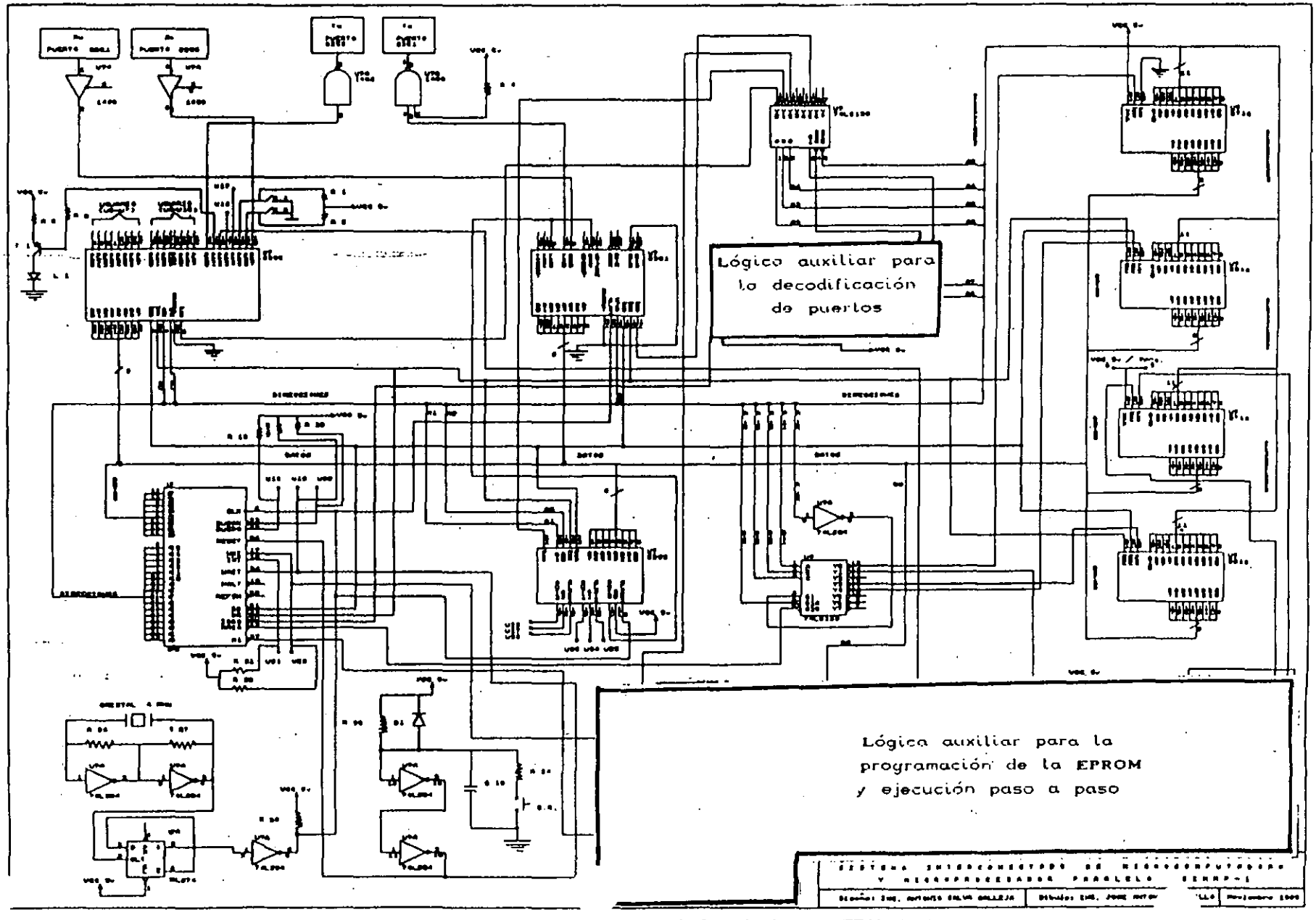


Figura 3.11 Diagrama de bloques del sistema SIMMP-1

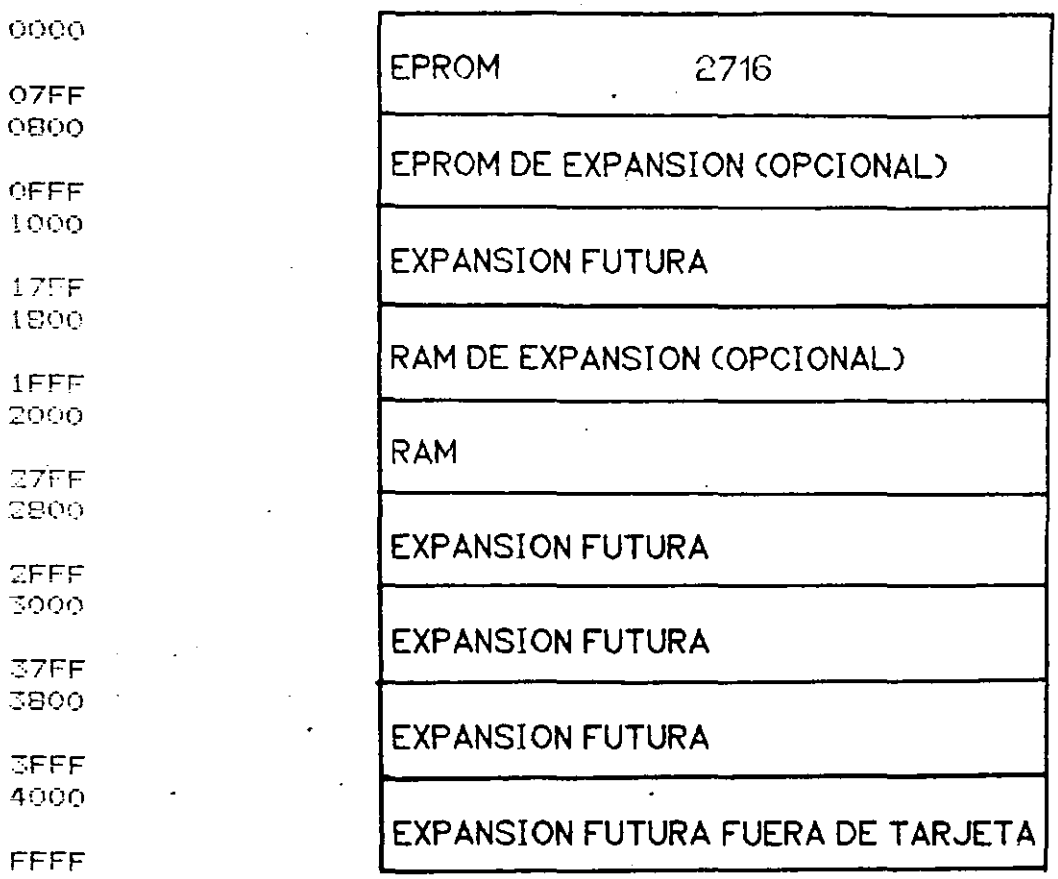


Figura 3.12 Mapa de memoria del SIMMP-1

Como se aprecia en la figura 3.12 la zona de memoria RAM empleada por la pila (stack), va de las direcciones 2780 a 27FF, de la dirección 2700 a la 2780 se define una zona de memoria para uso de núcleo básico de entrada salida del sistema, esto es, este conjunto de locaciones de memoria es usado tanto por el firmware del sistema como por el software de la microcomputadora que lo comanda, a este intervalo de direcciones se le denomina zona de servicio y se recomienda al usuario no modificar el contenido de cualquier localidad en ese rango. Las direcciones de memoria comprendidas de la 2600 a la 26FF son utilizadas para localizar

rutinas de servicio tanto de interrupciones no enmascarables como enmascarables de modo 1 (se recomienda consultar hoja de datos del microprocesador Z80). Como en el caso de la zona de servicio se recomienda al usuario no modificar el contenido de las localidades de la zona de servicio de interrupciones, aunque esto no es tan crítico como en la zona de servicio ya que no todas las aplicaciones usarán interrupciones.

Paginación de puertos del SIMMP-1

En la figura 3.13 se muestra el mapa de puertos del SIMMP-1. Con el hardware contenido en la tarjeta se pueden decodificar hasta 32 direcciones asociadas con puertos de las cuales el usuario puede usar 16, las otras 16 son usadas por el sistema en los puertos que el mismo contiene, a continuación se describe esto:

a) Las direcciones 14 y 15 están vinculadas con las de control datos del puerto serie 8251. (Se recomienda consultar hoja de datos del CI 8251).

b) Las direcciones 00 a 03 están vinculadas con el multipuerto paralelo 8255, por lo tanto para los puertos A, B y C de ese chip las direcciones respectivas son 00, 01 y 02. Siendo la dirección 03 la correspondiente al registro de control el chip. (Se recomienda consultar hoja de datos del CI 8255).

c) Las direcciones 04 a 07 están vinculadas con el temporizador programable de tres canales 8253. La dirección 07 es la correspondiente al registro de control del chip, siendo las direcciones 04 a 06 las asociadas con cada uno de los tres

canales de temporización del 8253. (Se recomienda consultar hoja de datos del CI 8253).

d) La dirección 08 es empleada por un puerto de salida de 1 bit que es parte de la lógica de ejecución de programas paso a paso.

Las otras cuatro señales de habilitación de puerto provenientes del decodificador de puertos están disponibles para el usuario a modo de que éste pueda colocar más puertos al sistema.

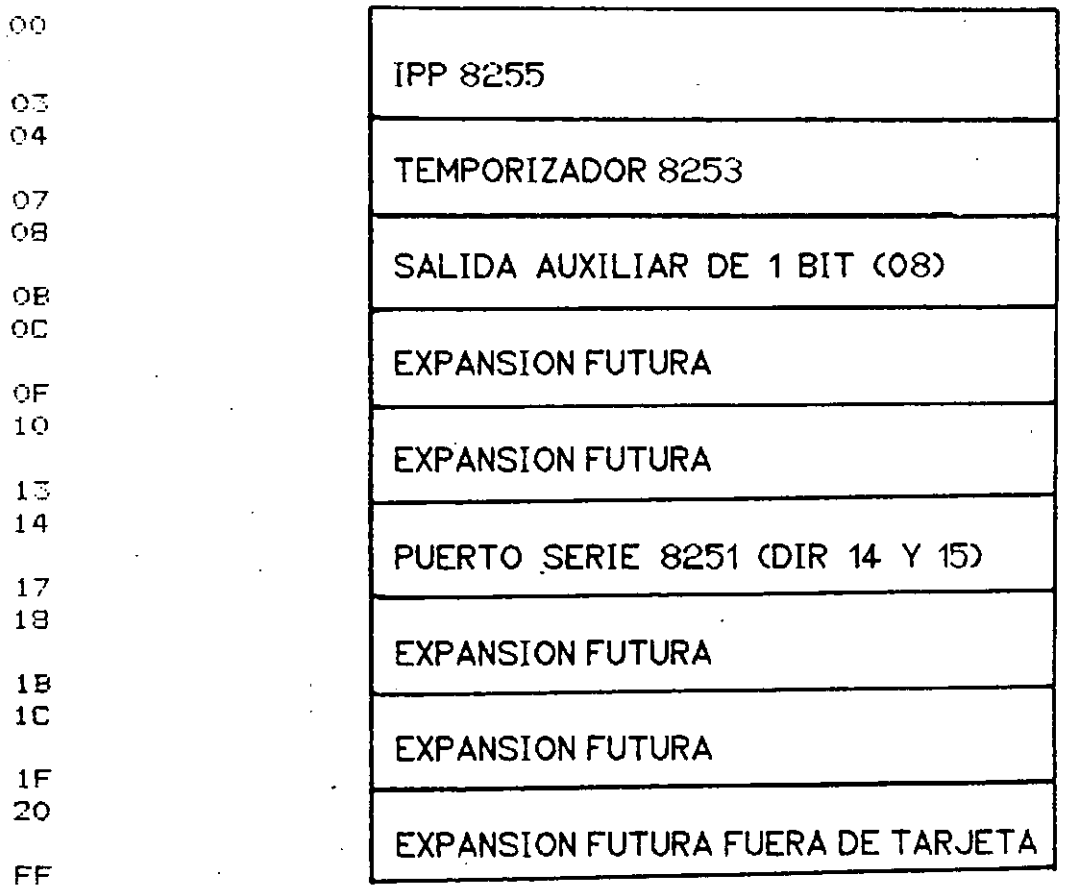


Figura 3.13 Mapa de puertos del SIMMP-1

Descripción de puertos del SIMMP-1

A continuación se describe brevemente la forma en que el sistema SIMMP-1 emplea varios de los puertos contenidos en su arquitectura.

a) Puerto serie uno (PS1). Este es implantado programando como salida la parte baja y como entrada la parte alta del puerto C del 8255, empleando como salida y entrada de transmisión las correspondientes con PC_0 y PC_7 respectivamente, siendo el funcionamiento del puerto PS1 arbitrado por firmware contenido en la EPROM del sistema. El formato de serialización de este puerto es el siguiente: un bit de inicio, un bit de paro, no paridad y ocho bits de datos. El baudaje puede variarse de 300 a 2400 bauds.

b) Puerto serie dos (PS2). Este puerto es implantado mediante el CI 8251, pudiendo el usuario modificar el formato de serialización (vg. número de bits de paro, paridad) escribiendo el software requerido. El baudaje de este puerto puede variar de 300 a 2400 bauds empleandose el canal cero del temporizador 8253 para generar la señal de cadencia requerida para cada baudaje. (Se recomienda consultar hoja de datos del CI 8251).

c) Puerto de indicación de espera de comandos. Este es implantado empleando la salida PC_1 del 8255 que está conectado con un diodo emisor de luz que ha de brillar siempre que el sistema SIMMP-1 esté en espera de un comando o dato proveniente de la microcomputadora.

d) Puerto de habilitación de programación de la EPROM. Este es

implantado empleando la salida PC_2 del 8255. Estando PC_2 en cero la EPROM de expansión contenida en la tarjeta está habilitada para leer, si PC_2 es uno la habilitación será para programar.

e) Puerto de servicio de salida de un bit. Implantado con la línea de salida PC_3 del 8255, puede ser empleado por el usuario para los fines que a él convengan teniendo cuidado de preservar el estado de los bits PC_0 PC_1 y PC_2.

f) Puertos de entrada PC_5 y PC_6. Estas dos entradas de un bit están ligadas con los puentes eléctricos (jumpers) que caracterizan el modo de funcionamiento del sistema al restablecerse. Esto se describe en la siguiente tabla:

PC_6	PC_5	Acción de restablecimiento
0	0	Salta a la dirección de memoria 0800
0	1	Salta a la dirección de memoria 0400
1	0	Salta a recibir comandos a través de PS1
1	1	Salta a recibir comandos a través de PS2

g) Puerto de entrada de un bit PC_4. Puede ser usado por el usuario para los fines que a él convengan.

h) Puerto A de entrada o salida. Puerto de ocho bits que puede ser programado por el usuario como de entrda o de salida, se encuentra en la dirección 00. (se recomienda consultar las hojas de datos del CI 8255).

i) Puerto B de entrada o salida. Puerto de ocho bits que puede ser programado por el usuario como de entrda o de salida, se encuentra en la dirección 01. (Se recomienda consultar las

hojas datos del CI 8255).

Es importante puntualizar que de acuerdo con lo explicado antes la parte baja del puerto C del CI 8255 siempre ha de estar programada como salida y la parte alta como entrada, en caso de que el usuario modifique esta programación, alterará el funcionamiento correcto del sistema.

Reloj

El SIMMP-1 funciona con un reloj de 2 MHz; el circuito que genera la señal correspondiente se observa en la figura 3.14; la señal correspondiente al reloj de baudaje se genera mediante el canal cero del temporizador programable 8253 a partir de la señal original de reloj de 2 MHz, lo que hace posible que el puerto serie RS2 pueda programarse a modo de que opere a los siguientes baudajes: 300, 600, 1200 y 2400 bauds.

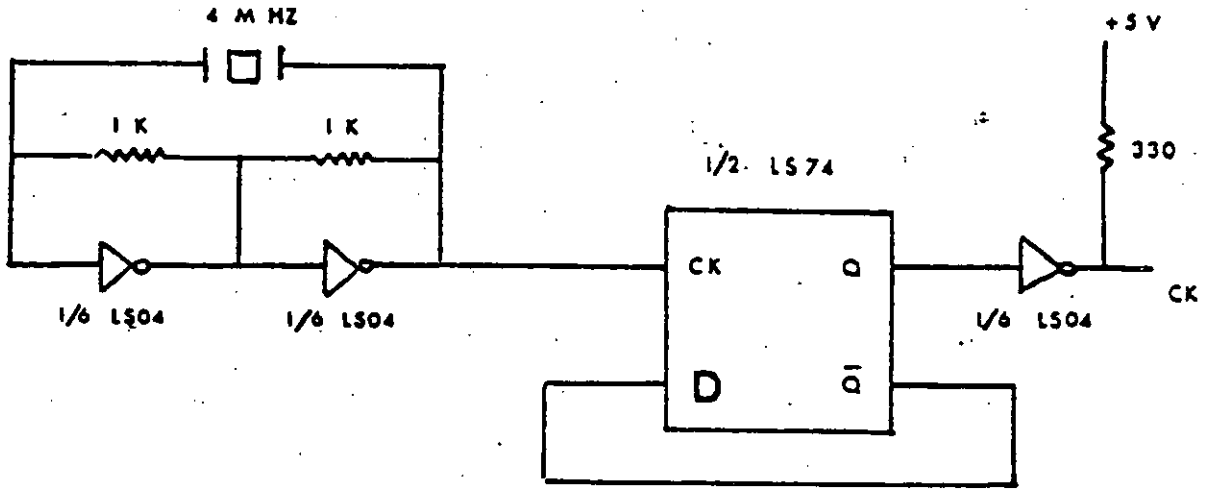


Figura 3.14 Circuito empleado para generar la señal de reloj del SIMMP-1

Circuito de restablecimiento (Reset)

El circuito para generar la señal de. restablcimiento (Reset) en el SIMMP-1 se ilustra en la figura 3.15.

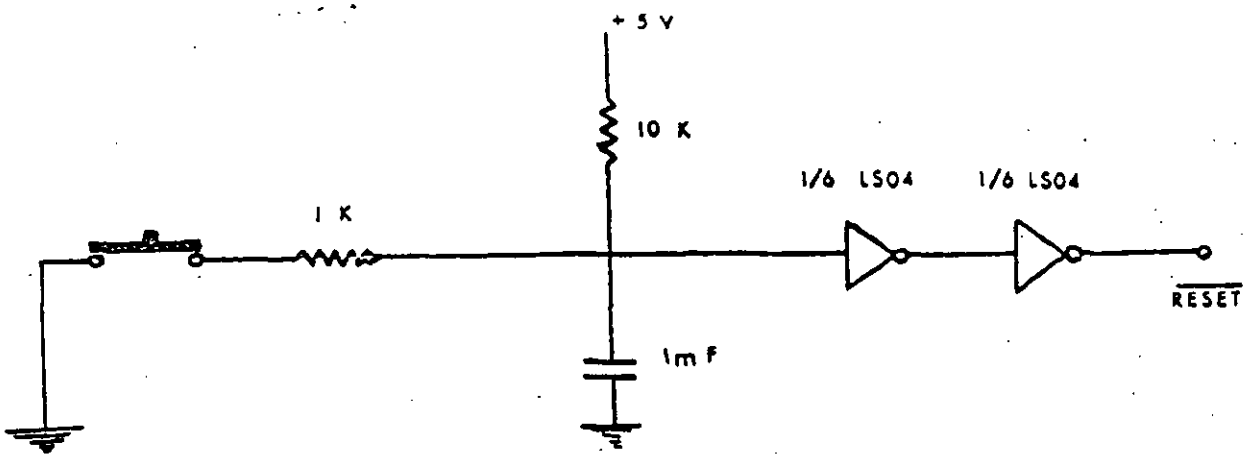


Figura 3.16 Circuito de restablecimiento

Software básico en el SIMMP-1

La esencia del SIMMP-1 se encuentra en un programa grabado en su EPROM, que recibe de la microcomputadora una cadena be bytes que integran un programa en lenguaje de máquina de Z80, autoejecutandose una vez que ha terminado de bajar. Al bajar a SIMMP-1 un programa la secuencia que envia la microcomputadora es la siguiente:

-- Envía un caracter de identificación que indica al SIMMP-1 que los que siguen representan un programa para el microprocesador Z80.

-- Transmite 4 bytes con información referente a la dirección inicial y final del programa a bajar.

-- Transmite la cadena de bytes correspondiente al programa que se esté bajando.

Software en la microcomputadora

El software está escrito en un leguaje de alto nivel. El programa se denomina *CCA** y permite al usuario operar el sistema SIMMP-1 desde la microcomputadora, manejandose la información en notación hexadecimal. El programa se base en menús, el principal se denomina de comando generandose por cada opción de este último ramificaciones de menús para cada caso. El de comandos consta de nueve opciones:

1. Cargar un programa en lenguaje de máquina Z80
2. Cargar de disco un programa para Z80
3. Cargar en disco un programa para Z80
4. Bajar a SIMMP-1 un programa que se autoejecute
5. Editar un programa
6. Manipulación de memoria
7. Definir baudaje
8. Manejo de disco
9. Retornar al sistema operativo

A continuación se describe brevemente lo que acontese al seleccionar cada una de las opciones del menú de comandos.

Opción uno

Al hacer esta selección el programa pide las direcciones inicial y final así como el nombre del programa que se va a cargar. Una vez efectuado lo anterior, se despliegan en la pantalla de la microcomputadora las direcciones en forma sucesiva, debiendo el usuario introducir cada vez el valor del byte correspondiente en notación hexadecimal. El programa asigna a cada

dirección una variable de tipo *cadena* (string) que representa lo que ha de almacenarse en locaciones de memoria correspondientes del SIMMP-1. De esta manera, la cadena de bytes que representa el programa queda contenida en un arreglo cuyo tamaño es igual al número de bytes que lo integra.

Opción dos

Cuando se escoge ésta, la microcomputadora requiere del usuario el nombre del programa a tomar de disco y la unidad correspondiente. Una vez que se ha ejecutado esta opción, la cadena de bytes del programa que se ha tomado de disco queda en un arreglo similar al resultante de la opción uno.

Opción tres

En ésta, el arreglo correspondiente a un programa determinado se guarda en un archivo de disco junto con las direcciones inicial y final correspondientes.

Opción cuatro

Permite bajar al SIMMP-1 las direcciones inicial y final de un programa para Z80, así como la cadena de bytes que lo constituyen. Una vez que el programa es bajado se autoejecuta.

Opción cinco

Mediante esta opción se pasa al menú de edición que permite al usuario editar el programa en lenguaje de máquina con el que se trabaje en un momento dado. Las opciones que contempla el menú de edición son las siguientes:

a) Listar el programa. Permite listar todo un programa en lenguaje de máquina o parte del mismo.

b) Borrar bytes. Mediante esta opción el usuario puede borrar bytes del programa en lenguaje de máquina que se esté depurando.

c) Insertar bytes. Permite insertar instrucciones en lenguaje de máquina al programa que se esté trabajando.

d) Cambiar bytes. Permite cambiar uno o más bytes de un programa en lenguaje de máquina.

e) Continuar. Con esta selección, el sistema pasa a un submenú que comprende las siguientes acciones:

1) Editar

2) Retornar a menú de comando

Si se toma la acción 1, el usuario retorna al menú de edición; si se toma la 2, regresa al menú principal o de comando. Cabe señalar que una vez que se ha ejecutado cualquiera de las opciones 1 a 8 del menú de comandos, el submenú descrito en este párraf. aparece nuevamente.

Opción seis

Mediante ésta se pasa a un menú que comprende los siguientes puntos:

a) Examinar memoria de SIMMP-1. Permite examinar contenido de localidades de memoria del sistema.

b) Cargar memoria de SIMMP-1. Permite cargar bytes en memoria RAM del sistema.

c) Ejecutar un programa cargado previamente en memoria RAM del sistema. El usuario puede escoger entre ejecución paso a paso o velocidad plena.

d) Transferir bloques en memoria de SIMMP-1. Aquí el usuario puede mover bloques contenidos en memoria ROM o RAM a memoria RAM.

e) Acomodar bloques provenientes de disco en memoria RAM de SIMMP-1. Mediante esta opción el usuario puede bajar a memoria RAM una cadena de bloques contenidos originalmente en disco.

f) Programar la EPROM de expansión del SIMMP-1. Con esta opción el usuario podrá programar la EPROM auxiliar del sistema con fuente de datos en: disco, memoria de SIMMP-1 o datos teclados en microcomputadora aisladamente.

g) Retornar a menú de comandos. El usuario podrá retornar al menú de comando con esta opción.

Opción siete

Mediante ésta el usuario puede programar el baudaje de cualesquiera de los dos puertos serie del sistema, los baudajes permisibles son: 300, 600, 1200, y 2400 bauds.

Opción ocho

Manejo de disco. Al seleccionar ésta el usuario pasa al menú de manipulación de disco que comprende las siguientes opciones:

a) Examinar directorio activo en alguna unidad de disco.

b) Cambiar directorio activo en alguna unidad de disco.

c) Borrar programa en código para Z80 en alguna unidad de disco.

d) Retornar a menú de comando.

Opción nueve

Mediante ésta el usuario puede retornar al sistema operativo de la microcomputadora.

Ensamblador cruzado

El software del lado de la microcomputadora asociado con el sistema SIMMP-1 cuenta con un programa ensamblador escrito exprofeso para el sistema. El ensamblador es de dos pasadas, admitiendo cualesquiera mnemónico dentro de la lexicografía asociada con el Z80. En la primera pasada se genera código de máquina para todas las instrucciones a excepción de las que involucren saltos o llamadas a etiquetas, colocando al ensamblador en estos casos banderas reconocibles, en la segunda pasada se coloca el código correcto asociado con las instrucciones de salto o llamada. El manejo del ensamblador por parte del usuario es sumamente sencillo, se maneja al igual que el programa CCA, en base a menús. El principal o de comando consta de las siguientes opciones:

1. Cargar un programa en lenguaje ensamblador para Z80.
2. Tomar de disco un programa fuente en ensamblador.
3. Cargar en disco un programa fuente en ensamblador.
4. Editar un programa fuente.
5. Guardar en disco el código objeto correspondiente a un programa fuente recién ensamblado.
6. Ensamblar un programa fuente especificando el usuario la dirección inicial a partir de la cual se desea cargar el programa objeto en memoria del sistema SIMMP-1. Una vez que un programa está ensamblado el usuario podrá bajarlo al SIMMP-1 para su ejecución a modo de hacer una prueba del programa sin necesidad de almacenarlo en disco, si es necesario hacer algún cambio el usuario puede retornar al editor para hacer las

modificaciones necesarias a fin de ensamblarlo y probarlo de nuevo, una vez que el programa funciona correctamente se puede cargar su código objeto en disco siendo el archivo correspondiente compatible con el programa CCA de manejo hexadecimal del SIMMP-1.

REFERENCIAS

1. Fundamentos de los microprocesadores.

   Roger L. Tokheim.

   Serie Schaum, Mc Graw Hill.

2. Contruya una microcomputadora basada en el Z80.

   Steve Ciarcia.

   BYTE BOOKS / Mc Graw Hill.

3. Microprocesadores, programación e interconexión.

   José Ma. Uruñuela M.

   Mc Graw Hill.

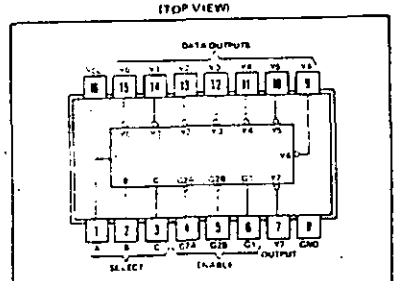
TYPES SN54LS138, SN54LS139, SN54S138, SN54S139, 'SN74LS138, SN74LS139, SN74S138, SN74S139 DECODERS/DEMULTIPLEXERS

BULLETIN NO. DL-S 7611804, DECEMBER 1972—REVISED OCTOBER 1976

- Designed Specifically for High-Speed:
  Memory Decoders
  Data Transmission Systems
- 'S138 and 'LS138 3-to-8-Line Decoders Incorporate 3 Enable Inputs to Simplify Cascading and/or Data Reception
- 'S139 and 'LS139 Contain Two Fully Independent 2-to-4-Line Decoders/Demultiplexers
- Schottky Clamped for High Performance

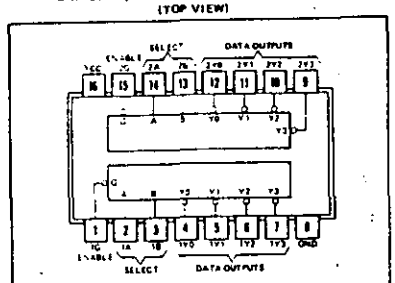
TYPE	TYPICAL PROPAGATION DELAY (3 LEVELS OF LOGIC)	TYPICAL POWER DISSIPATION
'LS138	22 ns	32 mW
'S138	8 ns	245 mW
'LS139	22 ns	34 mW
'S139	7.5 ns	300 mW

SN54LS138, SN54S138 . . . J OR W PACKAGE
SN74LS138, SN74S138 . . . J OR N PACKAGE
(TOP VIEW)



positive logic: see function table

SN54LS139, SN54S139 . . . J OR W PACKAGE
SN74LS139, SN74S139 . . . J OR N PACKAGE
(TOP VIEW)



positive logic: see function table

description

These Schottky-clamped TTL MSI circuits are designed to be used in high-performance memory-decoding or data-routing applications requiring very short propagation delay times. In high-performance memory systems these decoders can be used to minimize the effects of system decoding. When employed with high-speed memories utilizing a fast-enable circuit the delay times of these decoders and the enable time of the memory are usually less than the typical access time of the memory. This means that the effective system delay introduced by the Schottky-clamped system decoder is negligible.

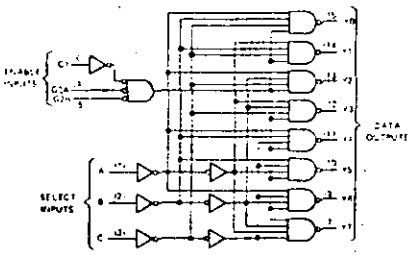
The 'LS138 and 'S138 decode one-of-eight lines dependent on the conditions at the three binary select inputs and the three enable inputs. Two active-low and one active-high enable inputs reduce the need for external gates or inverters when expanding. A 24-line decoder can be implemented without external inverters and a 32-line decoder requires only one inverter. An enable input can be used as a data input for demultiplexing applications.

The 'LS139 and 'S139 comprise two individual two-line-to-four-line decoders in a single package. The active-low enable input can be used as a data line in demultiplexing applications.

All of these decoders/demultiplexers feature fully buffered inputs each of which represents only one normalized Series 54LS/74LS load ('LS138, 'LS139) or one normalized Series 54S/74S load ('S138, 'S139) to its driving circuit. All inputs are clamped with high-performance Schottky diodes to suppress line-ringing and simplify system design. Series 54LS and 54S devices are characterized for operation over the full military temperature range of −55°C to 125°C. Series 74LS and 74S devices are characterized for 0°C to 70°C industrial systems.

functional block diagrams and logic

'LS138, 'S138



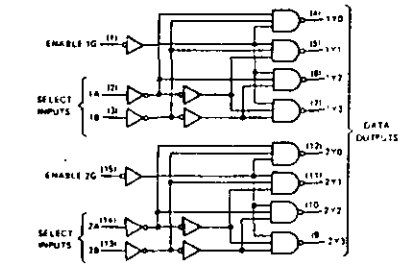
'LS138, 'S138
FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*G2 = G2A + G2B

H = high level, L = low level, X = irrelevant

'LS139, 'S139

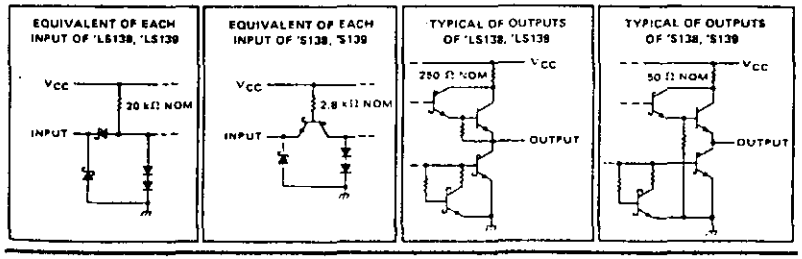


'LS139, 'S139
(EACH DECODER/DEMULTIPLEXER)
FUNCTION TABLE

INPUTS			OUTPUTS			
ENABLE	SELECT					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

H = high level, L = low level, X = irrelevant

schematics of inputs and outputs



REVISED OCTOBER 1976

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 7 V
Operating free-air temperature range: SN54LS138, SN54LS139 Circuits . . . . . −55°C to 125°C
SN74LS138, SN74LS139 Circuits . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

	SN54LS138 SN54LS139			SN74LS138 SN74LS139			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−400			−400	μA
Low-level output current, I_{OL}			4			8	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†		SN54LS138 SN54LS139			SN74LS138 SN74LS139			UNIT
			MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH} High-level input voltage			2			2			V
V_{IL} Low-level input voltage					0.7			0.8	V
V_{IK} Input clamp voltage	V_{CC} = MIN, I_I = −18 mA				−1.5			−1.5	V
V_{OH} High-level output voltage	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = $V_{IL\ max}$, I_{OH} = −400 μA		2.5	3.4		2.7	3.4		V
V_{OL} Low-level output voltage	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = $V_{IL\ max}$	I_{OL} = 4 mA		0.25	0.4		0.25	0.4	V
		I_{OL} = 8 mA					0.35	0.5	
I_I Input current at maximum input voltage	V_{CC} = MAX, V_I = 7 V				0.1			0.1	mA
I_{IH} High-level input current	V_{CC} = MAX, V_I = 2.7 V				20			20	μA
I_{IL} Low-level input current	V_{CC} = MAX, V_I = 0.4 V				−0.4			−0.4	mA
I_{OS} Short-circuit output current§	V_{CC} = MAX		−6		−40	−5		−42	mA
I_{CC} Supply current	V_{CC} = MAX, Outputs enabled and open	LS138		6.3	10		6.3	10	mA
		LS139		6.8	11		6.8	11	

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions for the applicable device type.
‡All typical values are at V_{CC} = 5 V, T_A = 25°C.
§Not more than one output should be shorted at a time.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	LEVELS OF DELAY	TEST CONDITIONS	SN54LS138 SN74LS138			SN54LS139 SN74LS139			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
t_{PLH}	Binary Select	Any	2	C_L = 15 pF, R_L = 2 kΩ, See Note 2	13	20		13	20		ns
t_{PHL}					27	41		22	33		ns
t_{PLH}			3		18	27		18	29		ns
t_{PHL}					26	39		25	38		ns
t_{PLH}	Enable	Any	2		12	18		16	24		ns
t_{PHL}					21	32		21	32		ns
t_{PLH}			3		17	26					ns
t_{PHL}					25	38					ns

¶t_{PLH} = propagation delay time, low-to-high-level output; t_{PHL} = propagation delay time, high-to-low-level output.
NOTE 2: Load circuits and waveforms are shown on page 3-11.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 5.5 V
Operating free-air temperature range: SN54S138, SN54S139 Circuits . . . . . −55°C to 125°C
SN74S138, SN74S139 Circuits . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

	SN54S138 SN74S139			SN74S138 SN74S139			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−1			−1	mA
Low-level output current, I_{OL}			20			20	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†		SN54S138 SN74S138			SN54S139 SN74S139			UNIT
			MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH} High-level input voltage			2			2			V
V_{IL} Low-level input voltage					0.8			0.8	V
V_{IK} Input clamp voltage	V_{CC} = MIN, I_I = −18 mA				−1.2			−1.2	V
V_{OH} High-level output voltage	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = 0.8 V, I_{OH} = −1 mA	SN54S	2.5	3.4		2.5	3.4		V
		SN74S	2.7	3.4		2.7	3.4		
V_{OL} Low-level output voltage	V_{CC} = MIN, V_{IH} = 2 V, V_{IL} = 0.8 V, I_{OL} = 20 mA				0.5			0.5	V
I_I Input current at maximum input voltage	V_{CC} = MAX, V_I = 5.5 V				1			1	mA
I_{IH} High-level input current	V_{CC} = MAX, V_I = 2.7 V				50			50	μA
I_{IL} Low-level input current	V_{CC} = MAX, V_I = 0.5 V				−2			−2	mA
I_{OS} Short-circuit output current§	V_{CC} = MAX		−40		−100	−40		−100	mA
I_{CC} Supply current	V_{CC} = MAX, Outputs enabled and open			49	74		60	90	mA

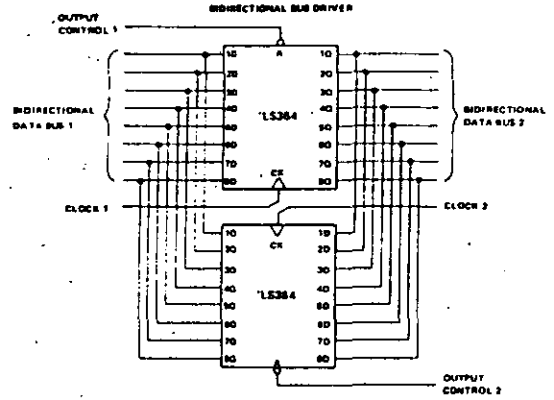
†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions for the applicable device type.
‡All typical values are at V_{CC} = 5 V, T_A = 25°C.
§Not more than one output should be shorted at a time, and duration of the short-circuit test should not exceed one second.

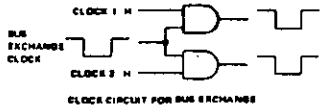
switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER¶	FROM (INPUT)	TO (OUTPUT)	LEVELS OF DELAY	TEST CONDITIONS	SN54S138, SN74S138			SN54S139 SN74S139			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
t_{PLH}	Binary select	Any	2	C_L = 15 pF, R_L = 280 Ω, See Note 3	4.5	7		5	7.5		ns
t_{PHL}					7	10.5		6.5	10		
t_{PLH}			3		7.5	12		7	12		ns
t_{PHL}					8	12		8	12		
t_{PLH}	Enable	Any	2		5	8		5	8		ns
t_{PHL}					7	11		6.5	10		
t_{PLH}			3		7	11					ns
t_{PHL}					7	11					

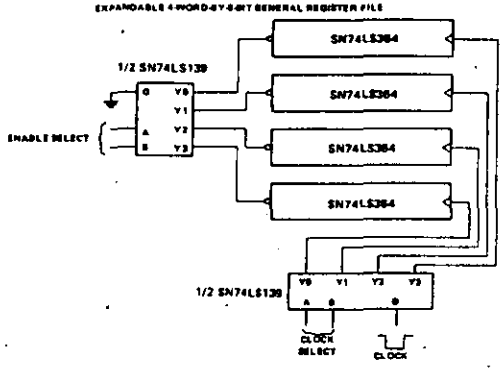
¶t_{PLH} = propagation delay time, low-to-high-level output
t_{PHL} = propagation delay time, high-to-low-level output
NOTE 3: Load circuits and waveforms are shown on page 3-10.

TYPICAL APPLICATION DATA





CLOCK CIRCUIT FOR BUS EXCHANGE

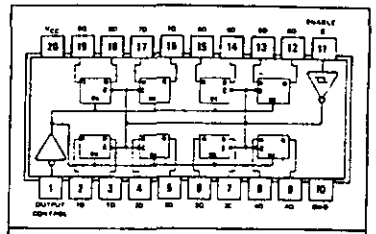


TYPES SN54LS373, SN54LS374, SN54S373, SN54S374, SN74LS373, SN74LS374, SN74S373, SN74S374 OCTAL D-TYPE TRANSPARENT LATCHES AND EDGE-TRIGGERED FLIP-FLOPS

BULLETIN NO. DL-S 7712350, OCTOBER 1976—REVISED AUGUST 1977

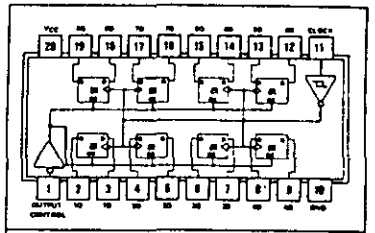
- Choice of 8 Latches or 8 D-Type Flip-Flops In a Single Package
- 3-State Bus-Driving Outputs
- Full Parallel-Access for Loading
- Buffered Control Inputs
- Clock/Enable Input Has Hysteresis to Improve Noise Rejection
- P-N-P Inputs Reduce D-C Loading on Data Lines ('S373 and 'S374)
- SN54LS363 and SN74LS364 Are Similar But Have Higher V_{OH} For MOS Interface

SN54LS373, SN54S373 . . . J PACKAGE
SN74LS373, SN74S373 . . . J OR N PACKAGE
(TOP VIEW)



logic: see function table

SN54LS374, SN54S374 . . . J PACKAGE
SN74LS374, SN74S374 . . . J OR N PACKAGE
(TOP VIEW)



logic: see function table

'LS373, 'S373
FUNCTION TABLE

OUTPUT CONTROL	ENABLE G	D	OUTPUT
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

'LS374, 'S374
FUNCTION TABLE

OUTPUT CONTROL	CLOCK	D	OUTPUT
L	↑	H	H
L	↑	L	L
L	L	X	Q_0
H	X	X	Z

See explanation of function tables on page 3-8.

description

These 8-bit registers feature totem-pole three-state outputs designed specifically for driving highly-capacitive or relatively low-impedance loads. The high-impedance third state and increased high-logic-level drive provide these registers with the capability of being connected directly to and driving the bus lines in a bus-organized system without need for interface or pull-up components. They are particularly attractive for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers.

The eight latches of the 'LS373 and 'S373 are transparent D-type latches meaning that while the enable (G) is high the Q outputs will follow the data (D) inputs. When the enable is taken low the output will be latched at the level of the data that was setup.

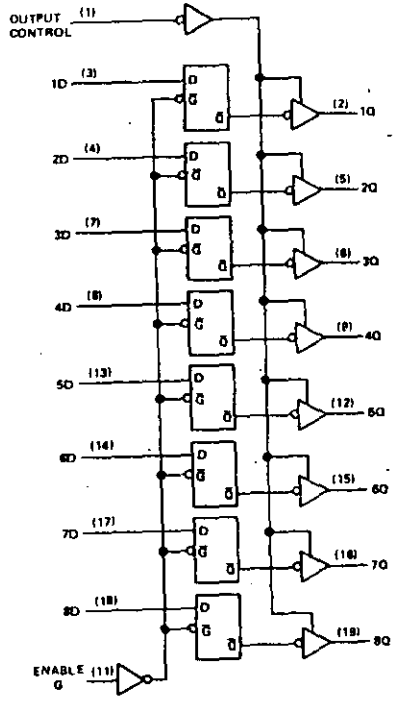
description (continued)

The eight flip-flops of the 'LS374 and 'S374 are edge-triggered D-type flip-flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were setup at the D inputs.

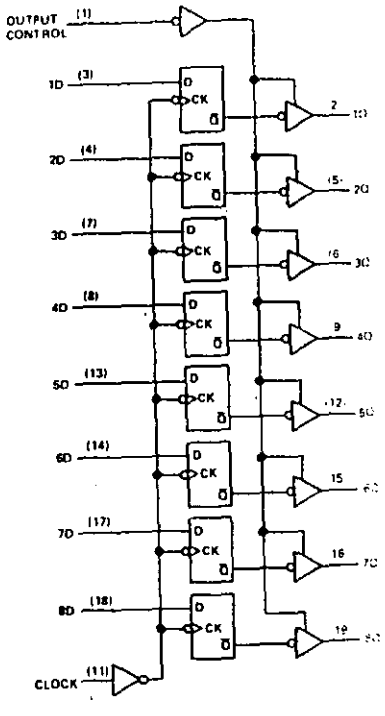
Schmitt-trigger buffered inputs at the enable/clock lines simplify system design as ac and dc noise rejection is improved by typically 400 mV due to the input hysteresis. A buffered output control input can be used to place the eight outputs in either a normal logic state (high or low logic levels) or a high-impedance state. In the high-impedance state the outputs neither load nor drive the bus lines significantly.

The output control does not affect the internal operation of the latches or flip-flops. That is, the old data can be retained or new data can be entered even while the outputs are off.

'LS373, 'S373
TRANSPARENT LATCHES

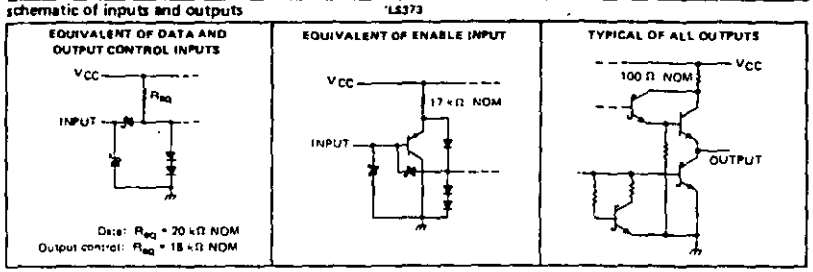


'LS374, 'S374
POSITIVE-EDGE-TRIGGERED FLIP-FLOPS

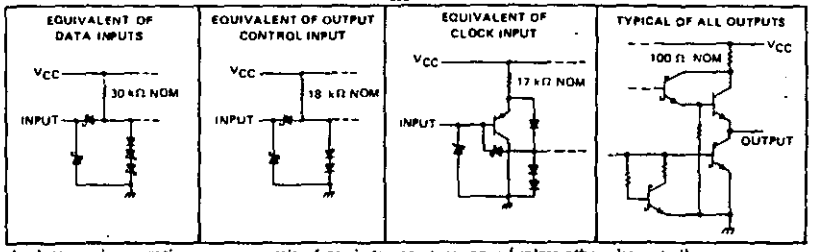


schematic of inputs and outputs

'LS373



'LS374



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 7 V
Off-state output voltage . . . . . 7 V
Operating free-air temperature range: SN54LS' . . . . . −55°C to 125°C
SN74LS' . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

		SN54LS'			SN74LS'			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}		4.5	5	5.5	4.75	5	5.25	V
High-level output voltage, V_{OH}				5.5			5.5	V
High-level output current, I_{OH}				−1			−2.6	mA
Width of clock/enable pulse, t_w	High	15			15			ns
	Low	15			15			
Data setup time, t_{su}	'LS373	0↓			0↓			ns
	'LS374	20↑			20↑			
Data hold time, t_h	'LS373	10↓			10↓			ns
	'LS374	0↑			0↑			
Operating free-air temperature, T_A		−55		125	0		70	°C

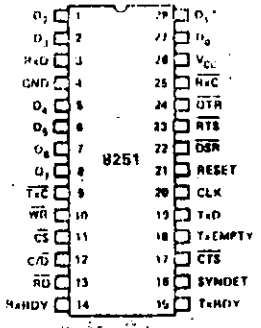
↑↓ The arrow indicates the transition of the clock/enable input used for reference: ↑ for the low-to-high transition, ↓ for the high-to-low transition.

PROGRAMMABLE COMMUNICATION INTERFACE

- Synchronous and Asynchronous Operation
  - Synchronous:
    - 5-8 Bit Characters
    - Internal or External Character Synchronization
    - Automatic Sync Insertion
  - Asynchronous:
    - 5-8 Bit Characters
    - Clock Rate — 1, 16 or 64 Times Baud Rate
    - Break Character Generation
    - 1, 1½, or 2 Stop Bits
    - False Start Bit Detection
- Baud Rate — DC to 56k Baud (Sync Mode)
  DC to 9.6k Baud (Async Mode)
- Full Duplex, Double Buffered, Transmitter and Receiver
- Error Detection — Parity, Overrun, and Framing
- Fully Compatible with 8080 CPU
- 28-Pin DIP Package
- All Inputs and Outputs Are TTL Compatible
- Single 5 Volt Supply
- Single TTL Clock

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Chip designed for data communications in microcomputer systems. The USART is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM Bi-Sync). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPT. The chip is constructed using N-channel silicon gate technology.

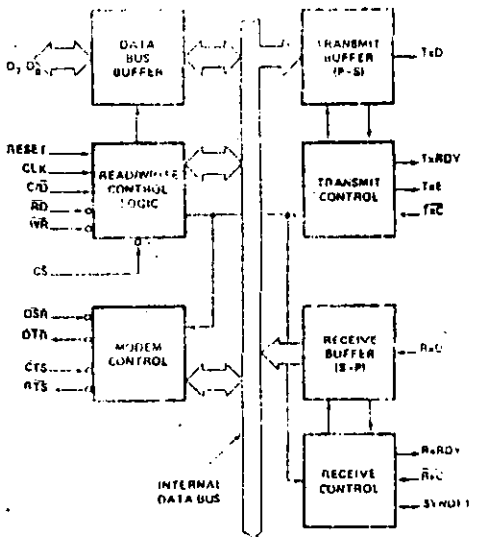
PIN CONFIGURATION



Pin Name	Pin Function
D_7-D_0	Data Bus (8 bits)
C/D	Control or Data is to be Written or Read
$\overline{RD}$	Read Data Command
$\overline{WR}$	Write Data or Control Command
$\overline{CS}$	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for char. from 8080)

Pin Name	Pin Function
DSR	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET	Sync Detect
$\overline{RTS}$	Request to Send Data
$\overline{CTS}$	Clear to Send Data
TxE	Transmitter Empty
V_{CC}	+5 Volt Supply
GND	Ground

BLOCK DIAGRAM



8251 BASIC FUNCTIONAL DESCRIPTION

General

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter designed specifically for the 8080 Microcomputer System. Like other I/O devices in the 8080 Microcomputer System its functional configuration is programmed by the systems software for maximum flexibility. The 8251 can support virtually any serial data technique currently in use (including IBM "bi-sync").

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8251 to the 8080 system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the 8080 CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer.

Read/Write Control Logic

This functional block accepts inputs from the 8080 Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for device functional definition.

RESET (Reset)

A "high" on this input forces the 8251 into an "idle" mode. The device will remain at "idle" until a new set of control words is written into the 8251 to program its functional definition.

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the 8224 Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter clock inputs for synchronous mode (4.5 times for asynchronous mode).

$\overline{WR}$ (Write)

A "low" on this input informs the 8251 that the CPU is outputting data or control words, in essence, the CPU is writing out to the 8251.

$\overline{RD}$ (Read)

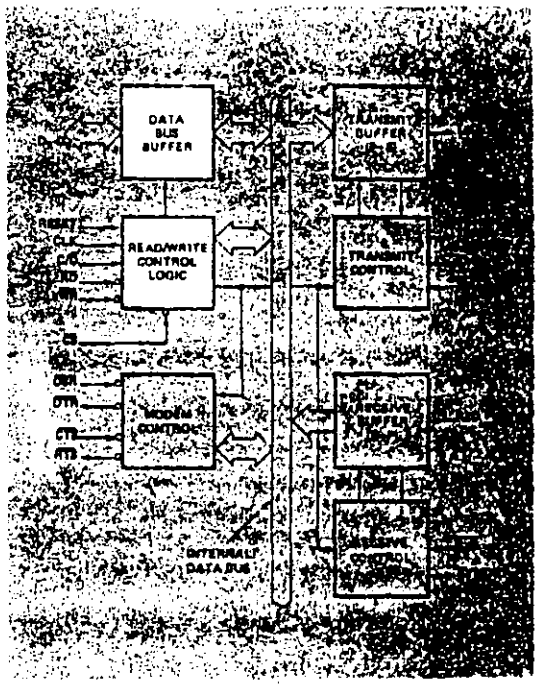
A "low" on this input informs the 8251 that the CPU is inputting data or status information, in essence, the CPU is reading from the 8251.

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs informs the 8251 that the word on the Data Bus is either a data character, control word or status information.
1 = CONTROL, 0 = DATA

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8251. No reading or writing will occur unless the device is selected.



C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251 → DATA BUS
0	1	0	0	DATA BUS → 8251
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	X	X	1	DATA BUS → 3-STATE

Modem Control

The 8251 has a set of control inputs and outputs that can be used to simplify the interface to almost any Modem. The modem control signals are general purpose in nature and can be used for functions other than Modem control, if necessary.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input signal is general purpose in nature. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test Modem conditions such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready)

The $\overline{DTR}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{DTR}$ output signal is normally used for Modem control such as Data Terminal Ready or Rate Select.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{RTS}$ output signal is normally used for Modem control such as Request to Send.

$\overline{CTS}$ (Clear to Send)

A "low" on this input enables the 8251 to transmit data (serial) if the Tx EN bit in the Command byte is set to a "one."

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

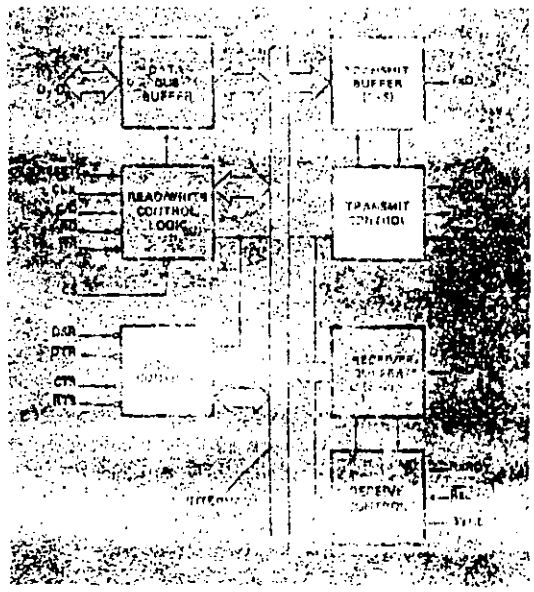
TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. It can be used as an interrupt to the system or for the Polled operation the CPU can check TxRDY using a status read operation. TxRDY is automatically reset when a character is loaded from the CPU.

TxE (Transmitter Empty)

When the 8251 [illegible] TxE output will go "high" [illegible] character from [illegible] end of a transmission [illegible] to "turn the line around" [illegible] mode.

In SYNChronous [illegible] that a character has not been [illegible] ter or characters are inserted [illegible] as "fillers."



$\overline{TxC}$ (Transmitter Clock)

The Transmitter [illegible] controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the frequency of $\overline{TxC}$ is equal to the actual Baud Rate (1X). In Asynchronous transmission mode, the frequency of $\overline{TxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1X, 16X or 64X the Baud Rate.

For Example:

If Baud Rate [illegible]
$\overline{TxC}$ [illegible]
$\overline{TxC}$ equals [illegible]
$\overline{TxC}$ equal [illegible]
If Baud [illegible]
$\overline{TxC}$ equals [illegible]

The falling [illegible] on output of the 8251.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to the RxD pin.

Receiver Control

This functional block manages all receiver-related activities.

RxRDY (Receiver Ready)

This output indicates that the 8251 contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or for Polled operation the CPU can check the condition of RxRDY using a status read operation. RxRDY is automatically reset when the character is read by the CPU.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the frequency of $\overline{RxC}$ is equal to the actual Baud Rate (1x). In Asynchronous Mode, the frequency of $\overline{RxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1x, 16x or 64x the Baud Rate.

For Example: If Baud Rate equals 300 Baud,
$\overline{RxC}$ equals 300 Hz (1x)
$\overline{RxC}$ equals 4800 Hz (16x)
$\overline{RxC}$ equals 19.2 kHz (64x).
If Baud Rate equals 2400 Baud,
$\overline{RxC}$ equals 2400 Hz (1x)
$\overline{RxC}$ equals 38.4 kHz (16x)
$\overline{RxC}$ equals 153.6 kHz (64x).

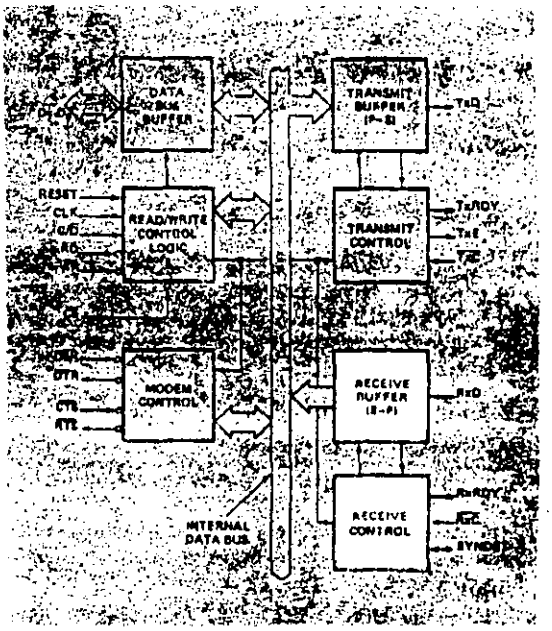
Data is sampled into the 8251 on the rising edge of $\overline{RxC}$.

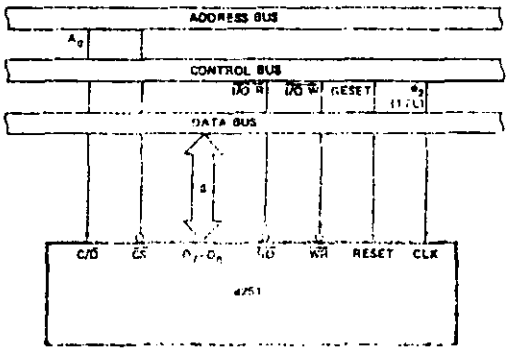
NOTE: In most communications systems, the 8251 will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

SYNDET (SYNC Detect)

This pin is used in SYNChronous Mode only. It is used as either input or output, programmable through the Control Word. It is reset to "low" upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251 has located the SYNC character in the Receive mode. If the 8251 is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input, (external SYNC detect mode), a positive going signal will cause the 8251 to start assembling data characters on the falling edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. The duration of the high signal should be at least equal to the period of $\overline{RxC}$.





8251 Interface to 8080 Standard System Bus

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize the 8251 to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD PARITY etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251 is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251 is ready to receive a character. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251. On the other hand, the 8251 receives serial data from the MODEM or I/O device, upon receiving an entire character the RxRDY output is raised "high" to signal the CPU that the 8251 has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU read operation.

The 8251 cannot begin transmission until the TxEN(Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send (CTS) input. The TxD output will be held in the marking state upon Reset.

Programming the 8251

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

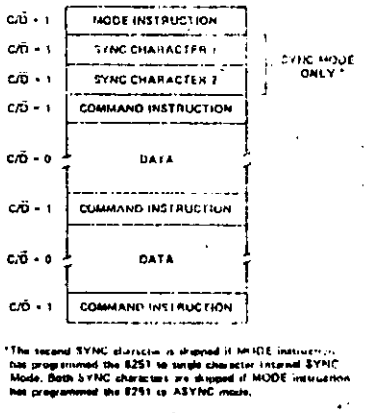
This format defines the general operational characteristics of the 8251. It must follow a Reset operation (internal or external). Once the Mode instruction has been written into the 8251 by the CPU, SYNC characters or Command Instructions may be inserted.

Command Instruction

This format defines a status word that is used to control the actual operation of the 8251.

Both the Mode and Command instructions must conform to a specified sequence for proper device operation. The Mode Instruction must be inserted immediately following a Reset operation, prior to using the 8251 for data communication.

All control words written into the 8251 after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251 at any time in the data block during the operation of the 8251. To return to the Mode Instruction format a bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251 back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.

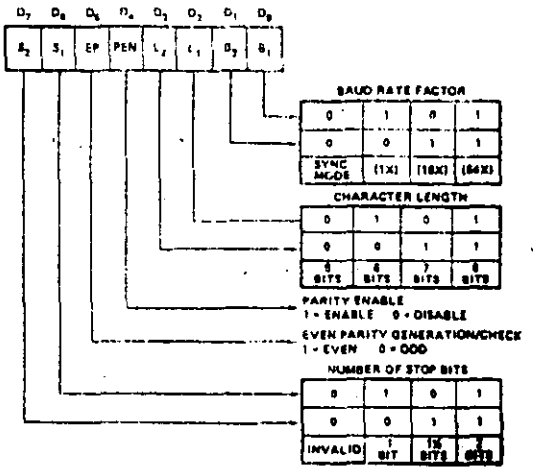


*The second SYNC character is skipped if MODE instruction has programmed the 8251 to single character internal SYNC Mode. Both SYNC characters are skipped if MODE instruction has programmed the 8251 to ASYNC mode.

Typical Data Block

Mode Instruction Definition

The 8251 can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251 the designer can best view the device as two separate components sharing the same package. One Asynchronous the other Synchronous. The format definition can be changed "on the fly" but for explanation purposes the two formats will be isolated.



Mode Instruction Format, Asynchronous Mode

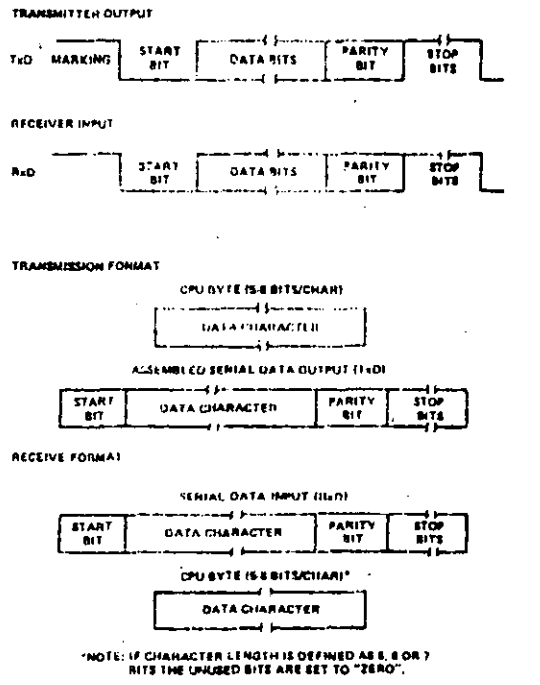
Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251 automatically adds a Start bit (low level) and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, 1/16, or 1/64 that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

When no data characters have loaded into the 8251 the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center. If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of $\overline{RxC}$. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. This character is then loaded into the parallel I/O buffer of the 8251. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN flag is raised (thus the previous character is lost). All of the error flags can be reset by a command instruction. The occurrence of any of these errors will not stop the operation of the 8251.



Asynchronous Mode

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251 which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251 with a character before the 8251 becomes empty, the SYNC characters (or character if in single SYNC word mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251 is empty and SYNC characters are being sent out. The TxEMPTY pin is internally reset by the next character being written into the 8251.

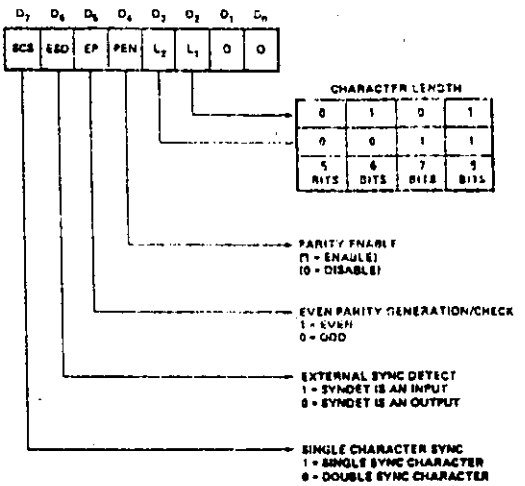
Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the internal SYNC mode has been programmed, the receiver starts in a HUNT mode. Data on the RxD pin is then sampled in on the rising edge of $\overline{RxC}$. The content of the Rx buffer is continuously compared with the first SYNC character until a match occurs. If the 8251 has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ.

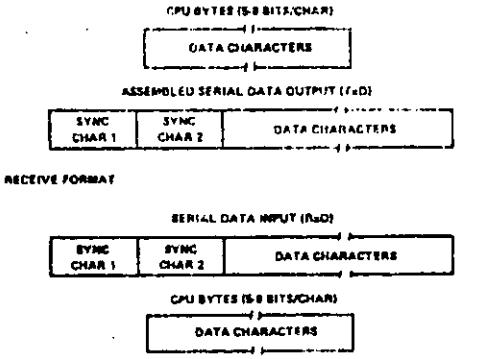
In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin. The high level can be removed after one $\overline{RxC}$ cycle.

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost.



Mode Instruction Format, Synchronous Mode

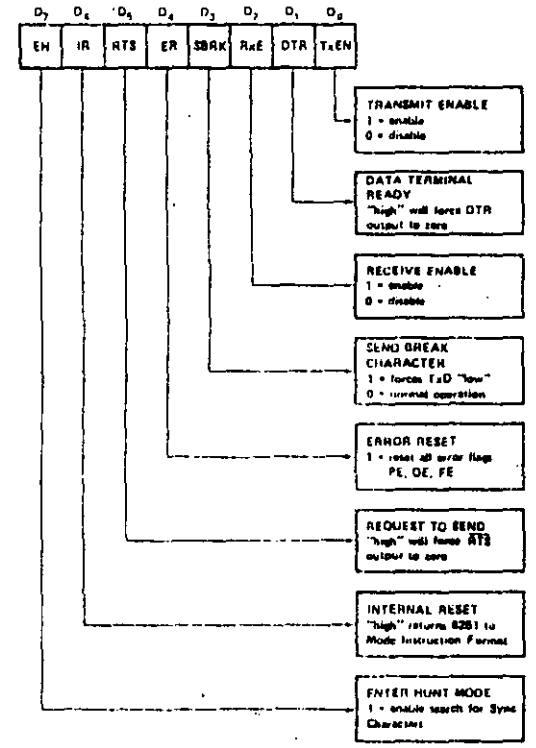


Synchronous Mode, Transmission Format

COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251 has been programmed by the Mode Instruction and the Sync Characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

Once the Mode Instruction has been written into the 8251 and Sync characters inserted, if necessary, then all further "control writes" ($C/\overline{D}$ = 1) will load the Command Instruction. A Reset operation (internal or external) will return the 8251 to the Mode Instruction Format.



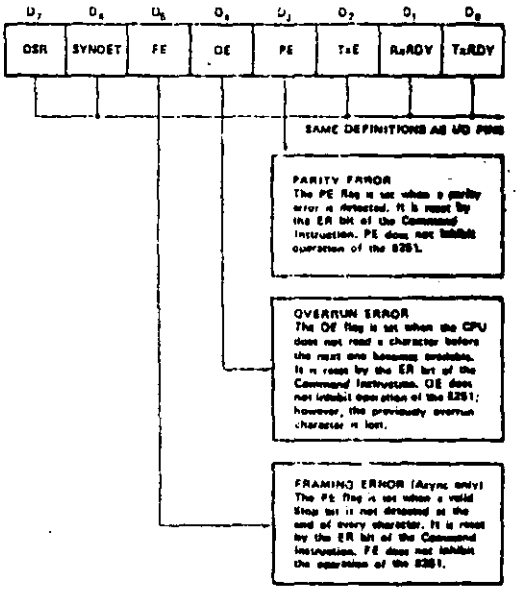
Command Instruction Format

STATUS READ DEFINITION

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251 has facilities that allow the programmer to "read" the status of the device at any time during the functional operation.

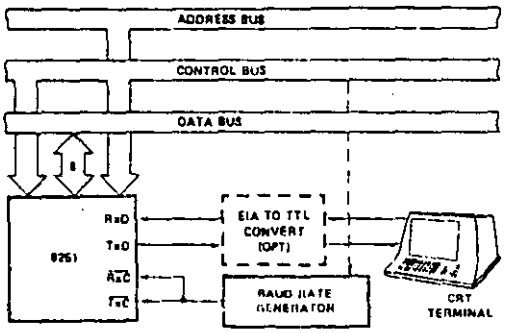
A normal "read" command is issued by the CPU with the C/D input at one to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251 can be used in a completely Polled environment or in an interrupt driven environment.

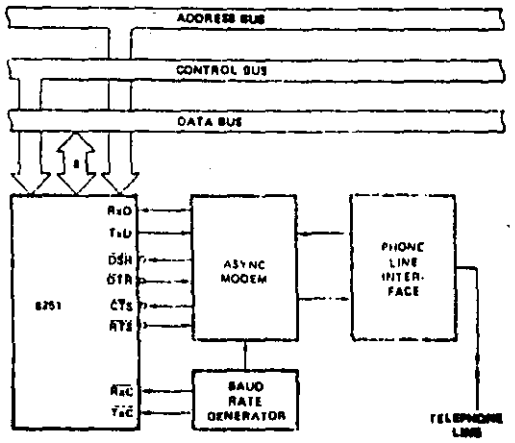


Status Read Format

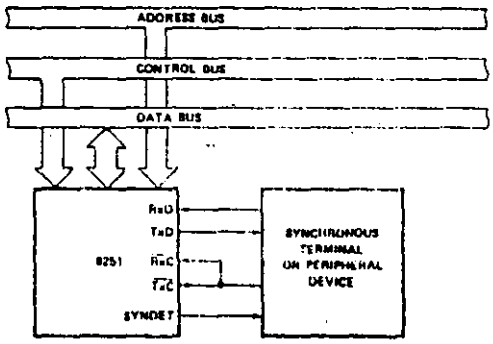
APPLICATIONS OF THE 8251



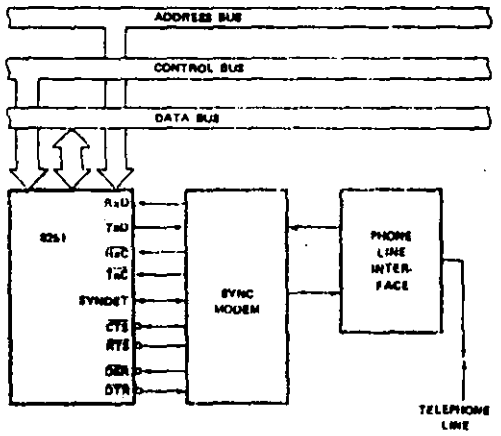
Asynchronous Serial Interface to CRT Terminal, DC-9600 Baud



Asynchronous Interface to Telephone Lines



Synchronous Interface to Terminal or Peripheral Device



Synchronous Interface to Telephone Lines

D.C. Characteristics:

T_A = 0°C to 70°C; V_{CC} = 5.0V ± 5%; V_{SS} = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	V_{SS}-.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage	2.2			V	I_{OH} = -100μA (DB_{0-7}) I_{OH} = -100μA (Others)
I_{DL}	Data Bus Leakage			50	μA	V_{OUT} = 4.5V
I_{LI}	Input Load Current			10	μA	@ 5.5V
I_{CC}	Power Supply Current		45	80		

Capacitance

T_A = 25°C; V_{CC} = V_{SS} = 0V

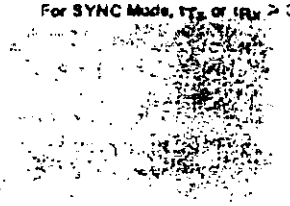
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}.

A.C. Characteristics:

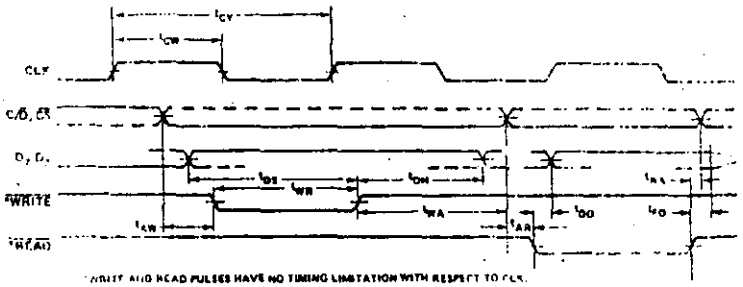
T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; V_{SS} = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
t_{CY}	Clock Period	.420		1.35	µs	
$t_{\phi W}$	Clock Pulse Width	220		300	ns	
t_R, t_F	Clock Rise and Fall Time	0		50	ns	
t_{WR}	$\overline{WRITE}$ Pulse Width	430			ns	
t_{DS}	Data Set-Up Time for $\overline{WRITE}$	0			ns	
t_{DH}	Data Hold Time for $\overline{WRITE}$	65			ns	
t_{AW}	Address Stable before $\overline{WRITE}$	20			ns	
t_{WA}	Address Hold Time for $\overline{WRITE}$	35			ns	
t_{RD}	$\overline{READ}$ Pulse Width	430			ns	
t_{DD}	Data Delay from $\overline{READ}$	350			ns	C_L=100pF
t_{DF}	$\overline{READ}$ to Data Floating	150			ns	C_L=100pF
t_{AR1}	Address Stable before $\overline{READ}$, CE (C/D)	50			ns	
t_{RA1}	Address Hold Time for $\overline{READ}$, CE	5			ns	
t_{RA2}	Address Hold Time for $\overline{READ}$, C/D	370			ns	
t_{DTx}	TxD Delay from Falling Edge of TxC	1			µs	C_L=100pF
t_{SRx}	Rx Data Set-Up Time to Sampling Pulse	2			µs	C_L=100pF
t_{HRx}	Rx Data Hold Time to Sampling Pulse	2			µs	C_L=100pF
f_{Tx}	Transmitter Input Clock Frequency 1X Baud Rate 16X and 64X Baud Rate	 DC DC		 56 615	 KHz KHz	
f_{Rx}	Receiver Input Clock Frequency 1X Baud Rate 16X and 64X Baud Rate	 DC DC		 56 615	 KHz KHz	
t_{Tx}	TxRDY Delay from Center of Data Bit			16	CLK Period	C_L=50pF
t_{Rx}	RxRDY Delay from Center of Data Bit	15		20	CLK Period	
t_{IS}	Internal Syndet Delay from Center of Data Bit	20		25	CLK Period	
t_{ES}	External Syndet Set-Up Time before Falling Edge of RxC			15	CLK Period	

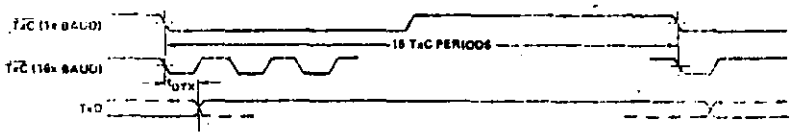
Note: The TxC and RxC frequencies have the following limitation with respect to CLK.
For ASYNC Mode, t_{Tx} or $t_{Rx} > 4.5\ t_{CY}$
For SYNC Mode, t_{Tx} or $t_{Rx} > 30\ t_{CY}$



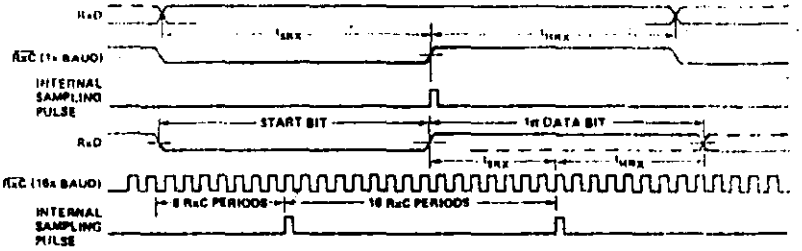
READ AND WRITE TIMING



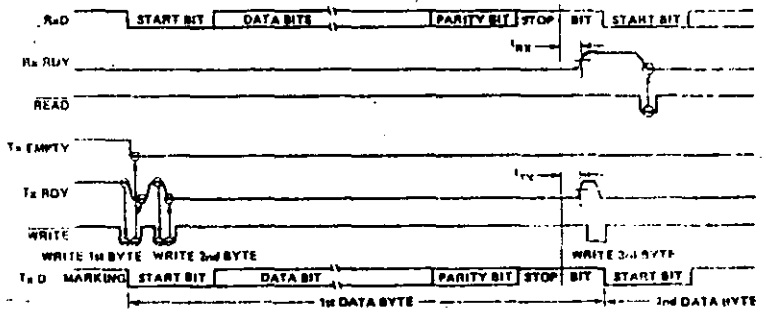
TRANSMITTER CLOCK AND DATA



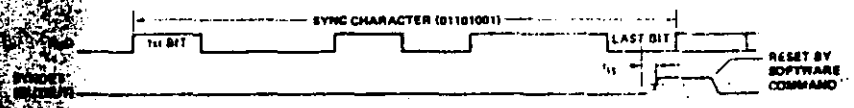
RECEIVER CLOCK AND DATA



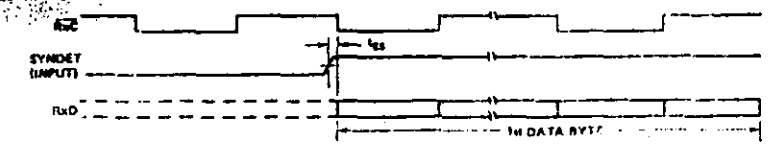
Tx RDY AND Rx RDY TIMING (ASYNC MODE)



INTERNAL SYNC DETECT



EXTERNAL SYNC DETECT



MOSTEK

Z80 MICROCOMPUTER DEVICES

Technical Manual

MK3880 CENTRAL PROCESSING UNIT

A block diagram of the internal architecture of the Z80-CPU is shown in Figure 2.0-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

Z80-CPU BLOCK DIAGRAM

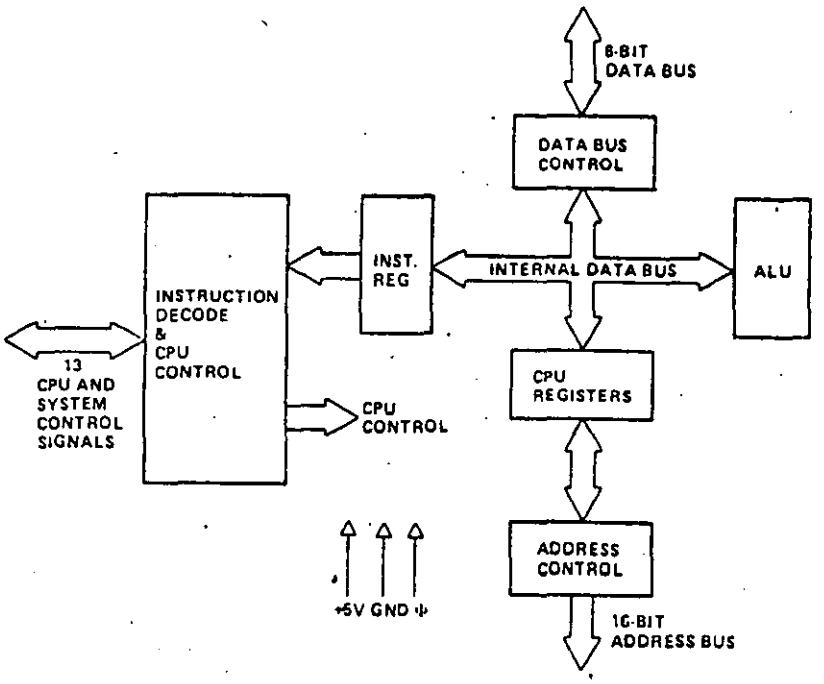


FIGURE 2.0-1

2.1 CPU REGISTERS

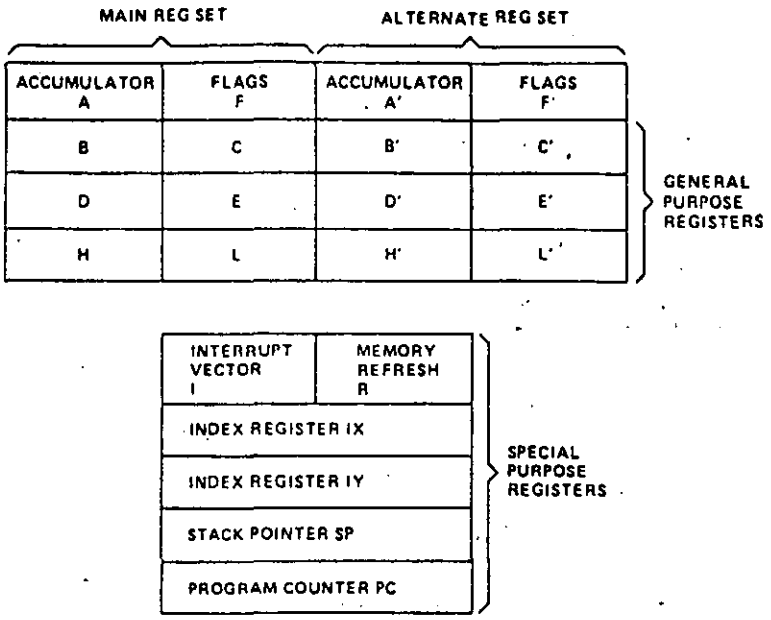
The Z80-CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2.0-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

1. Program Counter (PC). The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.

2. Stack Pointer (SP). The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

PU REGISTER CONFIGURATION



JRE 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 Z80-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 Z80-CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with with a single exchange instruction so that he may easily work with either pair.

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE, and HL while the complementary set is called BD', DE' and HL'. At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

2.2 ARITHMETIC & LOGIC UNIT (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally, the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

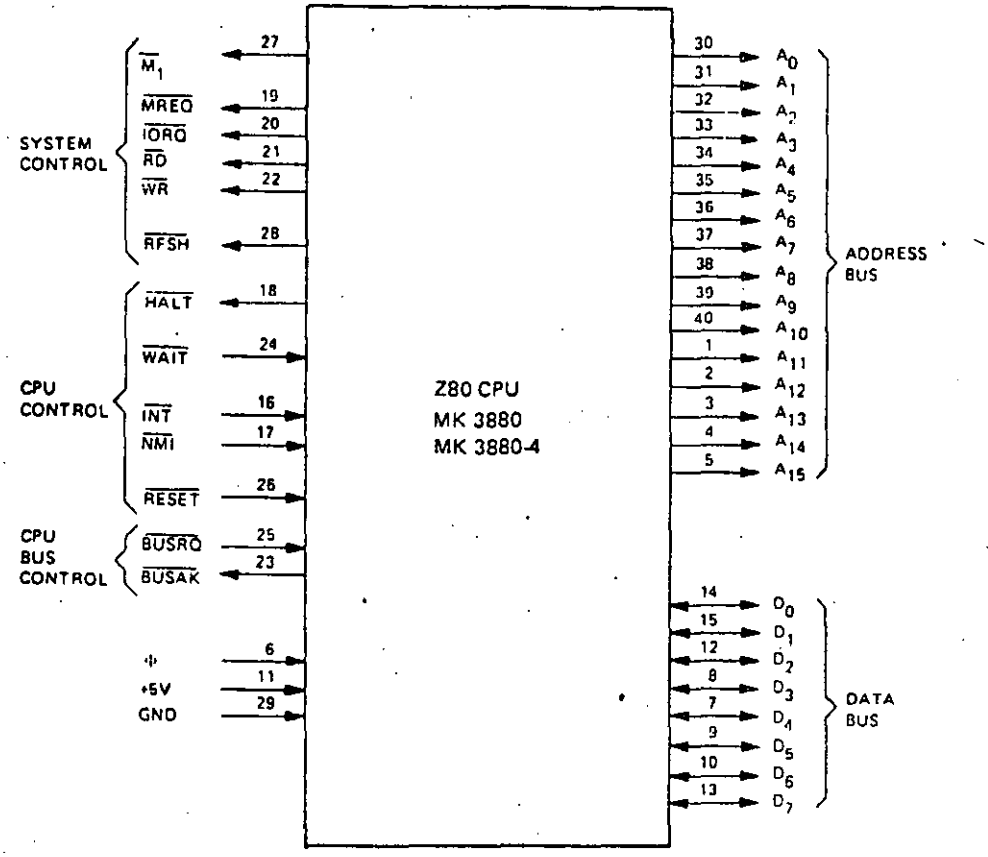
Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test bit

2.3 INSTRUCTION REGISTER AND CPU CONTROL

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control section performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, controls the ALU and provides all required external control signals.

The Z80-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.

PIN CONFIGURATION



URE 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, $\overline{M_1}$ is generated as each op code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH, or FDH. $\overline{M_1}$ 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.

$\overline{IORQ}$ (Input/Output Request)	Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{IORQ}$ signal is also generated with an $\overline{M_1}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.
$\overline{RD}$ (Memory Read)	Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.
$\overline{WR}$ (Memory Write)	Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.
$\overline{RFSH}$ (Refresh)	Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories. A_7 is a logic zero and the upper 8 bits of the Address Bus contains the I Register.
$\overline{HALT}$ (Halt state)	Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.
$\overline{WAIT}$* (Wait)	Input, active low. $\overline{WAIT}$ indicates to the Z80-CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.
$\overline{INT}$ (Interrupt Request)	Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{BUSRQ}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{IORQ}$ during M_1 time) is sent out at the beginning of the next instruction cycle. The CPU can respond to an interrupt in three different modes that are described in detail in section 8.
$\overline{NMI}$	Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z80-CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that $\overline{BUSRQ}$ will override a $\overline{NMI}$.

$\overline{\text{RESET}}$	Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes: 1) Disable the interrupt enable flip-flop 2) Set Register I = 00_H 3) Set Register R = 00_H 4) Set Interrupt Mode 0 During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state. No refresh occurs.
$\overline{\text{BUSRQ}}$ (Bus Request)	Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.
$\overline{\text{BUSAK}}$* (Bus Acknowledge)	Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.
Φ	Single phase system clock.

While the Z80-CPU is in either a $\overline{\text{WAIT}}$ state or a Bus Acknowledge condition, Dynamic Memory Refresh will not occur.

The Z80-CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write

I/O device read or write

Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T states and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T states long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 7, the exact timing for each instruction is specified.

BASIC CPU TIMING EXAMPLE

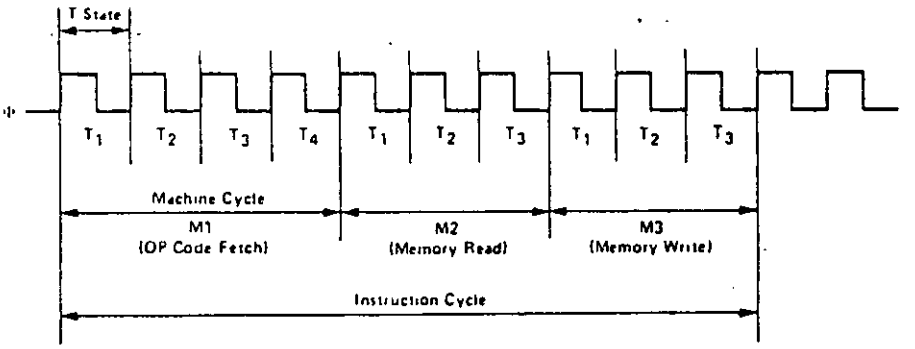


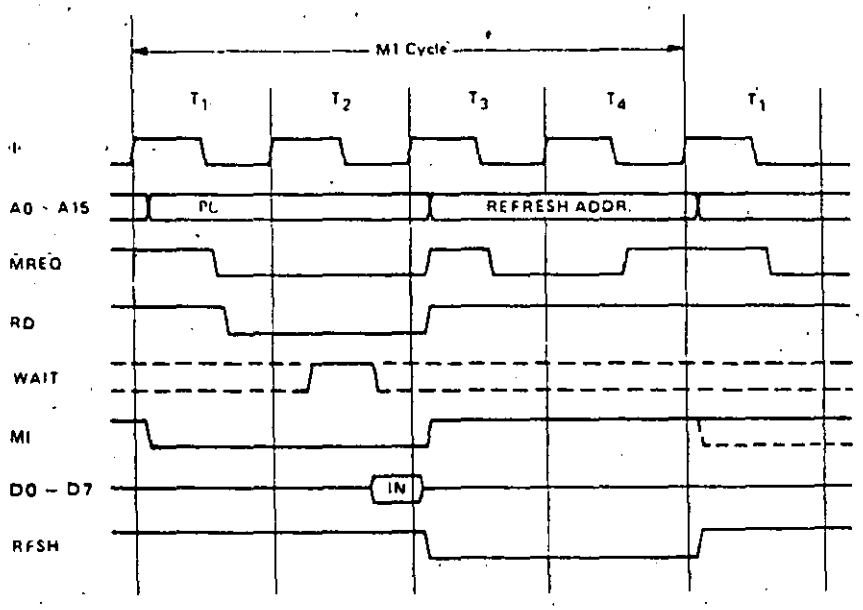
FIGURE 4.0-0

All CPU timing can be broken down into a few very simple timing diagrams as shown in Figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)
4.0-2. Memory data read or write cycles
4.0-3. I/O read or write cycles
4.0-4. Bus Request/Acknowledge Cycle
4.0-5. Interrupt Request/Acknowledge Cycle
4.0-6. Non maskable Interrupt Request/Acknowledge Cycle
4.0-7. Exit from a HALT instruction

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 $\overline{MREQ}$ 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.

NSTRUCTION OP CODE FETCH



IGURE 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.

INSTRUCTION OP CODE FETCH WITH WAIT STATES

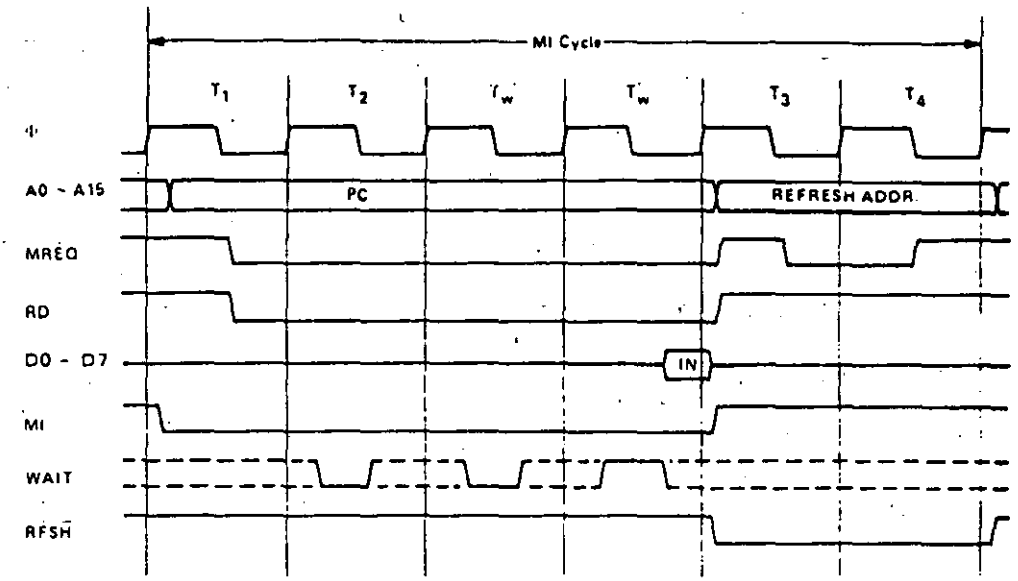


FIGURE 4.0-1A

MEMORY READ OR WRITE

Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the $\overline{WAIT}$ signal. The $\overline{MREQ}$ signal and the $\overline{RD}$ signal are used the same as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the $\overline{WR}$ signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.

MEMORY READ OR WRITE CYCLES

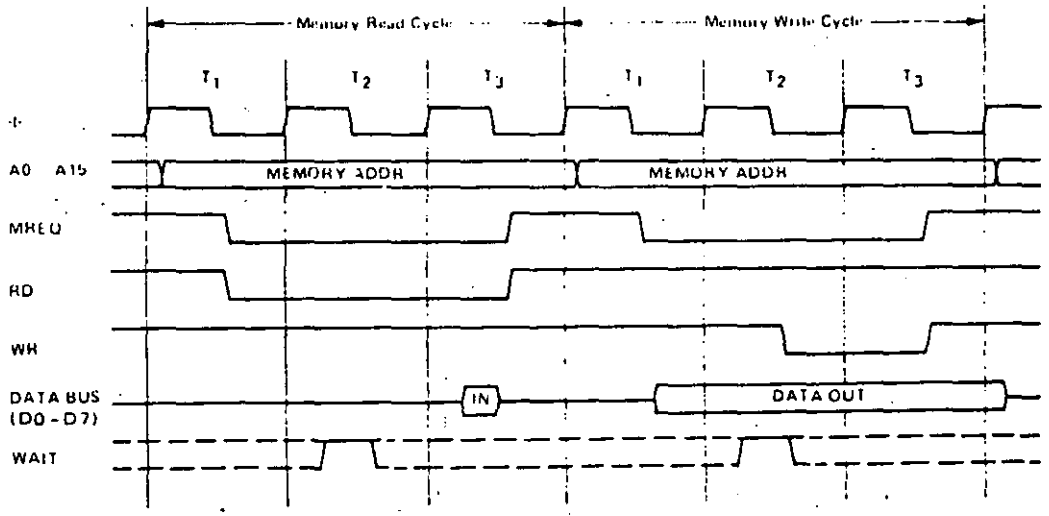
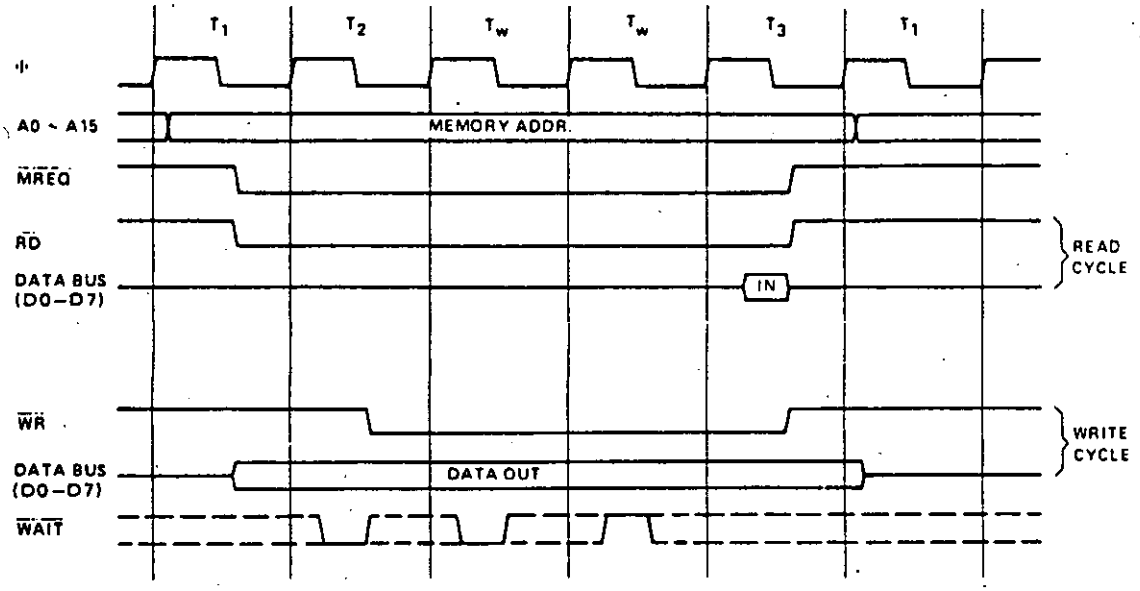


FIGURE 4.0-2

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.

MORY READ OR WRITE CYCLES WITH WAIT STATES



URE 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.

INPUT OR OUTPUT CYCLES

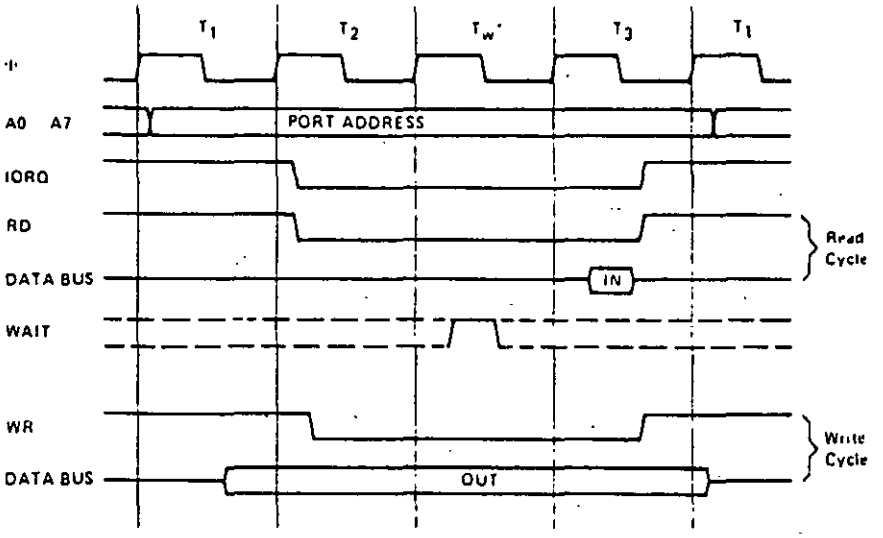


FIGURE 4.0-3

INPUT OR OUTPUT CYCLES WITH WAIT STATES

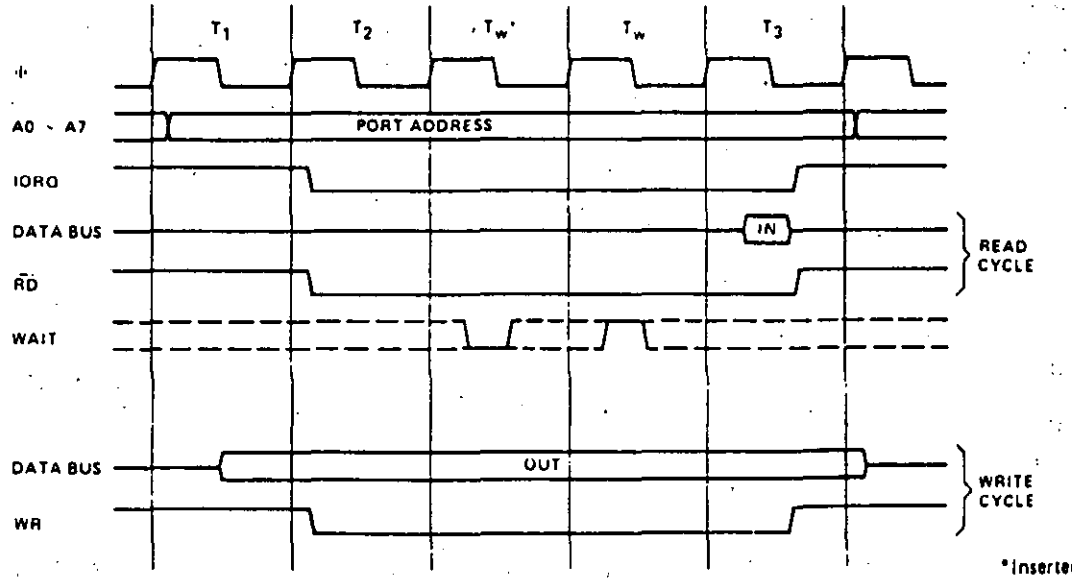


FIGURE 4.0-3A

BUS REQUEST/ACKNOWLEDGE CYCLE

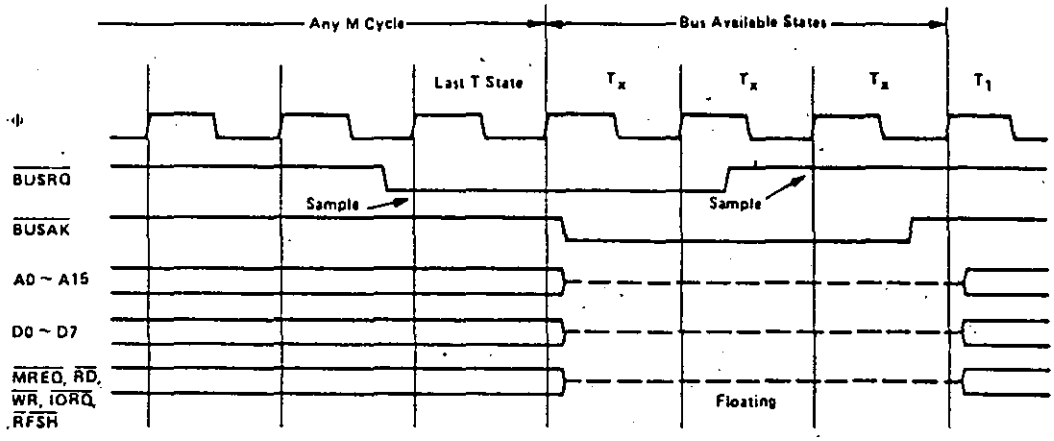


FIGURE 4.0-4

INTERRUPT REQUEST/ ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSRQ}$ signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the $\overline{IORQ}$ signal becomes active (instead of the normal $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stablilize 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

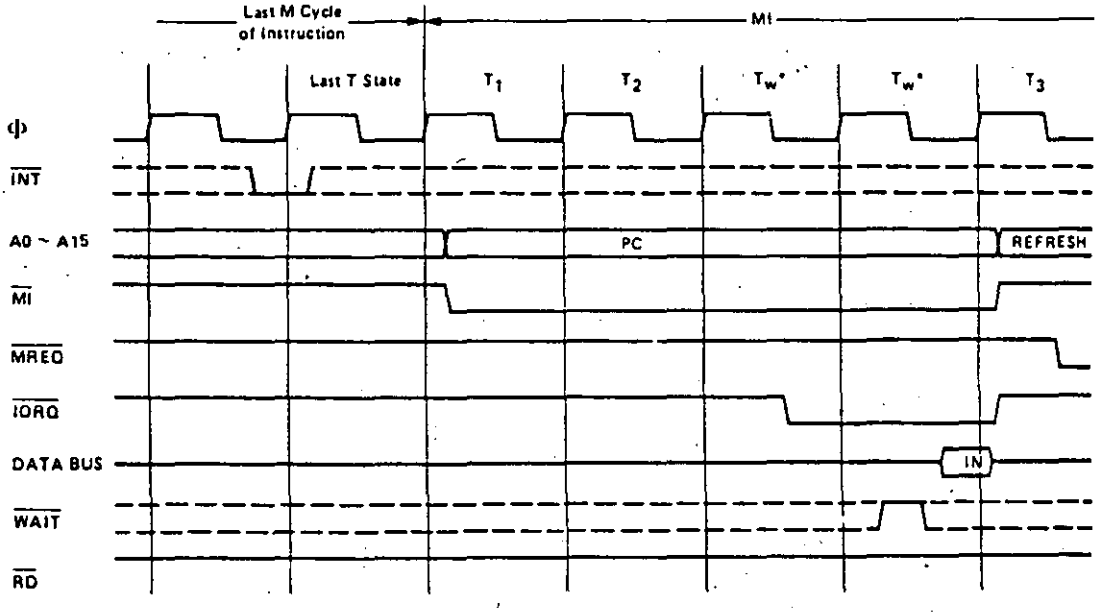


FIGURE 4.0-5

6.0 FLAGS

Each of the two Z80-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 Z80 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 that can be represented two's complement notation. For example consider adding:

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

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 oonsider 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.

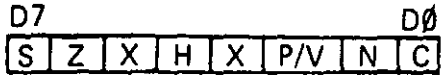
For logical operations (AND, OR, XOR) this flag is set if the parity of the result is even and it is reset if it is odd.

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) Add/Subtract Flag (N) – Since the agorithim 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:



X means flag is indeterminate.

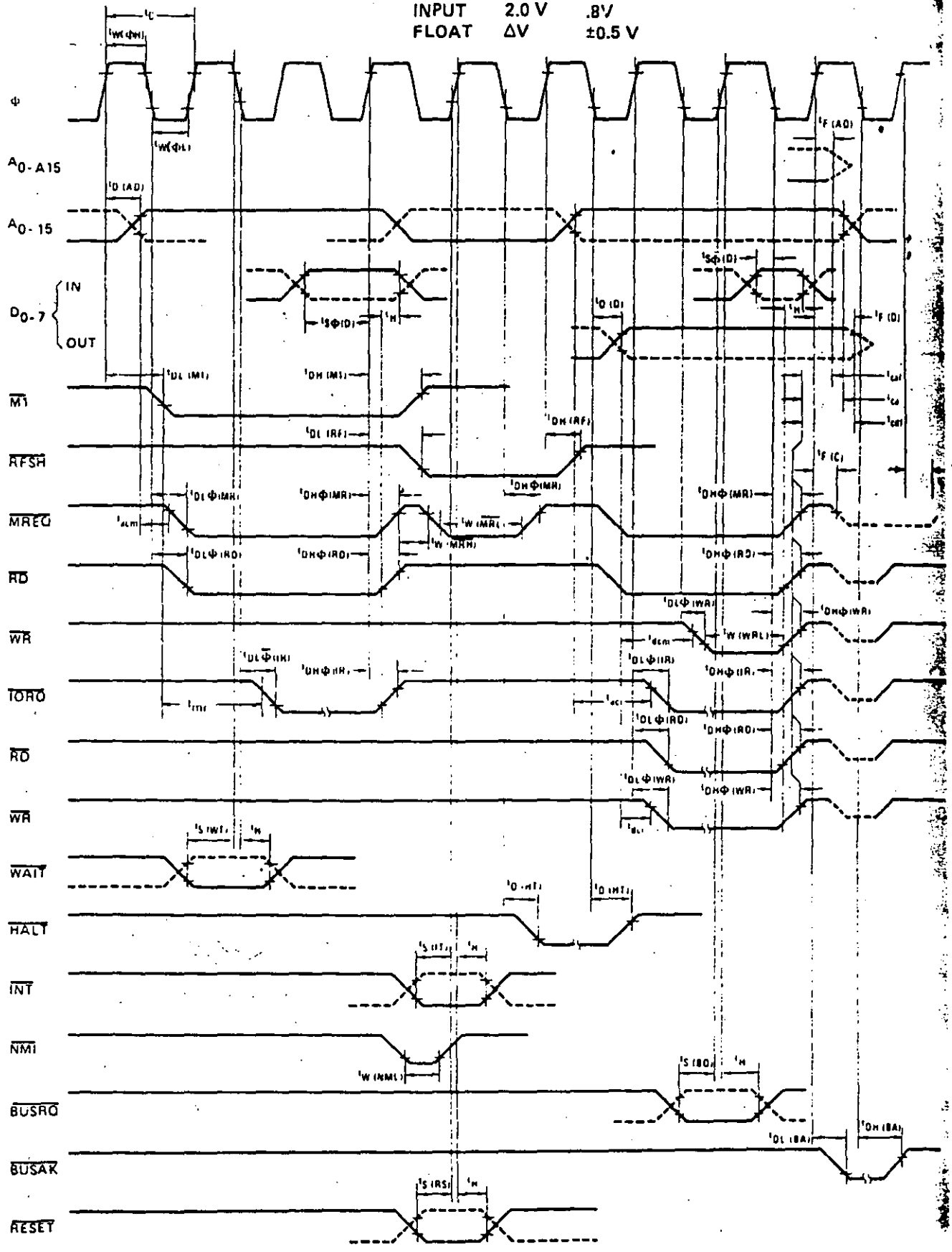
Table 6.0-1 lists how each flag bit is affected by various CPU instructions. In this table a '•' indicates that the instruction does not change the flag, an 'X' means that the flag goes to an indeterminate state, an '0' means that it is reset, a '1' means that it is set and the symbol ↕ indicates that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

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

A.C. TIMING DIAGRAM

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

	"1"	"0"
CLOCK	V_{CC}−.6	.8V
OUTPUT	2.0 V	.8V
INPUT	2.0 V	.8V
FLOAT	ΔV	±0.5 V



8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000 B
										← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011 E
										01 r 110					100 H
										← d →					101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111 A
										01 r 110					
										← d →					
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	
										01 110 r					
										← d →					
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	
										01 110 r					
										← d →					
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10	
										← n →					
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13	
										← n →					
										← n →					
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13	
										← n →					
										← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 010 111	57				
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 011 111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 000 111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 001 111	4F				

Notes: r, s 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.

16-BIT LOAD GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 001	FD E1	2	4	14	

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

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

Table 7.0-2

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE—HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF—AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC—BC' DE—DE' HL—HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H —(SP+1) L —(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H —(SP+1) IX_L —(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H —(SP+1) IY_L —(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)—(HL) DE — DE+1 HL — HL+1 BC — BC-1	•	•	X	0	X	↕ (1)	0	•	11 101 101 10 100 000	ED A0	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	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)—(HL) DE — DE-1 HL — HL-1 BC — BC-1	•	•	X	0	X	↕ (1)	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)—(HL) DE — DE-1 HL — HL-1 BC — BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL — HL+1 BC — BC-1	↕	↕ (2)	X	↕	X	↕ (1)	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL — HL+1 BC — BC-1 Repeat until A = (HL) or BC = 0	↕	↕ (2)	X	↕	X	↕ (1)	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL — HL-1 BC — BC-1	↕	↕ (2)	X	↕	X	↕ (1)	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL — HL-1 BC — BC-1 Repeat until A = (HL) or BC = 0	↕	↕ (2)	X	↕	X	↕ (1)	1	•	11 101 101 10 111 001	ED B9	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.

Table 7-5

8-BIT ARITHMETIC AND LOGICAL GROUP

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

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

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

Table 7.0-4

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	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	↕	↕	X	↕	X	P	•	↕	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	X	1	X	•	1	•	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \leftarrow \overline{A} + 1$	↕	↕	X	↕	X	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	•	•	X	X	X	•	0	↕	00 111 111	3F	1	1	4	Complement carry flag
SCF	CY ← 1	•	•	X	0	X	•	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110	76	1	1	4	
DI*	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011	F3	1	1	4	
EI*	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 01 011 110	ED 5E	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.

*Interrupts are not sampled at the end of EI or DI

16-Bi. ARITHMETIC GROUP

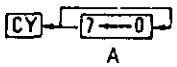
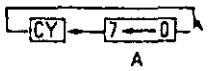
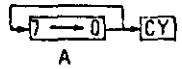
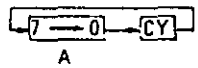
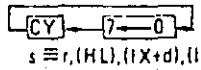
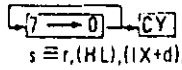
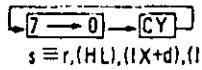
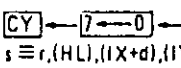
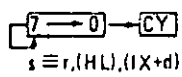
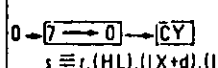
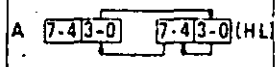
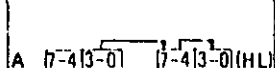
Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	↕	00 ss1 001		1	3	11	ss
															00
ADC HL, ss	HL ← HL+ss+CY	↕	↕	X	X	X	V	0	↕	11 101 101	ED	2	4	15	01
										01 ss1 010					10
															11
SBC HL, ss	HL ← HL-ss-CY	↕	↕	X	X	X	V	1	↕	11 101 101	ED	2	4	15	
										01 ss0 010					
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101	DD	2	4	15	pp
										00 pp1 001					00
															01
															10
															11
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101	FD	2	4	15	rr
										00 rr1 001					00 BC
															01 DE
															10 IY
															11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10	
										00 100 011	23				
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10	
										00 100 011	23				
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6	
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10	
										00 101 011	28				
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10	
										00 101 011	28				

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

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

Table 7.0-6

ROTATE AND SHIFT GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		•	•	X	0	X	•	0	↕	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA		•	•	X	0	X	•	0	↕	00 010 111	17	1	1	4	Rotate left accumulator
RRCA		•	•	X	0	X	•	0	↕	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA		•	•	X	0	X	•	0	↕	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r	 r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	11 001 011 00 000 r	CB	2	2	8	Rotate left circular register r
RLC (HL)		↕	↕	X	0	X	P	0	↕	11 001 011 00 000 110	CB	2	4	15	r Reg. 000 B 001 C
RLC (IX+d)		↕	↕	X	0	X	P	0	↕	11 011 101 11 001 011 – d – 00 000 110	DD CB	4	6	23	010 D 011 E 100 H 101 L 111 A
RLC (IY+d)		↕	↕	X	0	X	P	0	↕	11 111 101 11 001 011 – d – 00 000 110	FD CB	4	6	23	
RL s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	001					
RR s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	011					
SLA s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	100					
SRA s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	101					
SRL s	 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	111					
RLD		↕	↕	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	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		↕	↕	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	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.

Table 7.0-7

BIT SET, RESET AND TEST GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z = \overline{r_b}$	X	↕	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	r 000
BIT b, (HL)	$Z = \overline{(HL)_b}$	X	↕	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	001 010
BIT b, $(IX+d)_b$	$Z = \overline{(IX+d)_b}$	X	↕	X	1	X	X	0	•	11 011 101 11 001 011 - d - 01 b 110	DD CB	4	5	20	011 100 101 111
BIT b, $(IY+d)_b$	$Z = \overline{(IY+d)_b}$	X	↕	X	1	X	X	0	•	11 111 101 11 001 011 - d - 01 b 110	FD CB	4	5	20	b 000 001 010 011 100 101 110 111
SET b, r	$r_b = 1$	•	•	X	•	X	•	•	•	11 001 011 11 b r	CB	2	2	8	
SET b, (HL)	$(HL)_b = 1$	•	•	X	•	X	•	•	•	11 001 011 11 b 110	CB	2	4	15	
SET b, (IX+d)	$(IX+d)_b = 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 - d - 11 b 110	DD CB	4	6	23	
SET b, (IY+d)	$(IY+d)_b = 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 - d - 11 b 110	FD CB	4	6	23	
RES b, s	$s_b = 0$ s ≡ r, (HL), (IX+d),	•	•	X	•	X	•	•	•	10					To form Code replace of SET [10] Flags states for RES instruction

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

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

Table 7.0-8

JUMP GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•		X	•	X	•	•	•	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	•	•	X	•	X	•	•	•	00 011 000 ← e-2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e-2 →	38	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	•	•	X	•	X	•	•	•	00 110 000 ← e-2 →	30	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	•	•	X	•	X	•	•	•	00 101 000 ← e-2 →	28	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	•	•	X	•	X	•	•	•	00 100 000 ← e-2 →	20	2	2	7	If condition not met
	If Z = 0, PC ← PC+e											2	3	12	If condition is met
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 ← e-2 →	10	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.

Table 7.0-9

CALL AND RETURN GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC \leftarrow nn$	•	•	X	•	X	•	•	•	11 ← ←	001 n n	101 → →	CD	3	5	17	
CALL cc, nn	If condition cc is false	•	•	X	•	X	•	•	•	11 ← ←	cc n n	100 → →		3	3	10	If cc is false
	continue, otherwise same as CALL nn													3	5	17	If cc is true
RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	•	•	X	•	X	•	•	•	11	001	001	C9	1	3	10	
RET cc	If condition cc is false	•	•	X	•	X	•	•	•	11	cc	000		1	1	5	If cc is false
	continue, otherwise same as RET													1	3	11	If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 01	101 001	101 101	ED 4D	2	4	14	
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 01	101 000	101 101	ED 45	2	4	14	
RST p	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 0$ $PC_L \leftarrow p$	•	•	X	•	X	•	•	•	11	t	111		1	3	11	

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

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

[1] RETN loads $IFF_2 \rightarrow IFF_1$

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.

Table 7.0-10

INPUT AND OUTPUT GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 ← n →	DB	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	↕ (1)	X	X	X	X	1	•	11 101 101 10 100 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 110 010	ED B2	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	↕ (1)	X	X	X	X	1	•	11 101 101 10 101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 111 010	ED BA	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011	D3	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	↕ (1)	X	X	X	X	1	•	11 101 101 10 100 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 110 011	ED B3	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	↕ (1)	X	X	X	X	1	•	11 101 101 10 101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	•	11 101 101 10 111 011	ED BB	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

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.

Table 7.0-11

8.0 INTERRUPT RESPONSE

The prupose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

INTERRUPT ENABLE – DISABLE

The Z80-CPU has two interrupt inputs, a software maskable interrupt and a non-maskable interrupt. The non-maskable interrupt ($\overline{NMI}$) can not be disabled by the programmer and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as an impending power failure. The maskable interrupt ($\overline{INT}$) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80-CPU there is an enable flip flop (called IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt can not be accepted by the CPU.

Actually, for purposes that will be subsequently explained, there are two enable flip flops, called IFF_1 and IFF_2.

IFF_1	IFF_2
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is used to actually inhibit interrupts while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instructions sets both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

The purpose of IFF_2 is to save the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted maskable interrupts are disabled but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF_1 is thru the execution of a Return From Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

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

INTERRUPT ENABLE/DISABLE 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	•	$IFF_2 \rightarrow IFF_1$
Accept INT	0	0	
RETI	•	•	

"•" indicates no change

FIGURE 8.0-1

CPU RESPONSE

Non-Maskable

A non-maskable 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 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 4.0 illustrates the detailed timing for an interrupt response. After the application of RESET the CPU will automatically enter interrupt Mode 0.

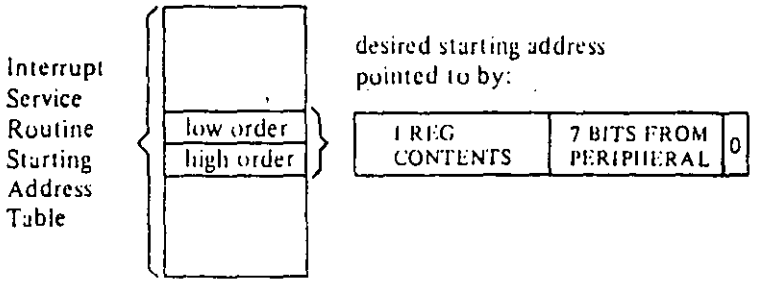
Mode 1

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.

Mode 2

This mode is the most powerful interrupt response mode. With a single 8-bit byte from user an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16 bit starting addresses for ev terrupt service routine. This table may be located anywhere in memory. When an inte is accepted, a 16 bit pointer must be formed to obtain the desired interrupt service rou starting address from the table. The upper 8 bits of this pointer is formed from the con of the I register. The I register must have been previously loaded with the desired valu the programmer, i.e. LD I, A. Note that a CPU reset clears the I register so that it is tialized to zero. The lower eight bits of the pointer must be supplied by the interrupt device. Actually, only 7 bits are required from the interrupting device as the lo bit must be a zero. This is required since the pointer is used to get two adjacent byte from a complete 16 bit service routine starting address and the addresses must always st in even locations.



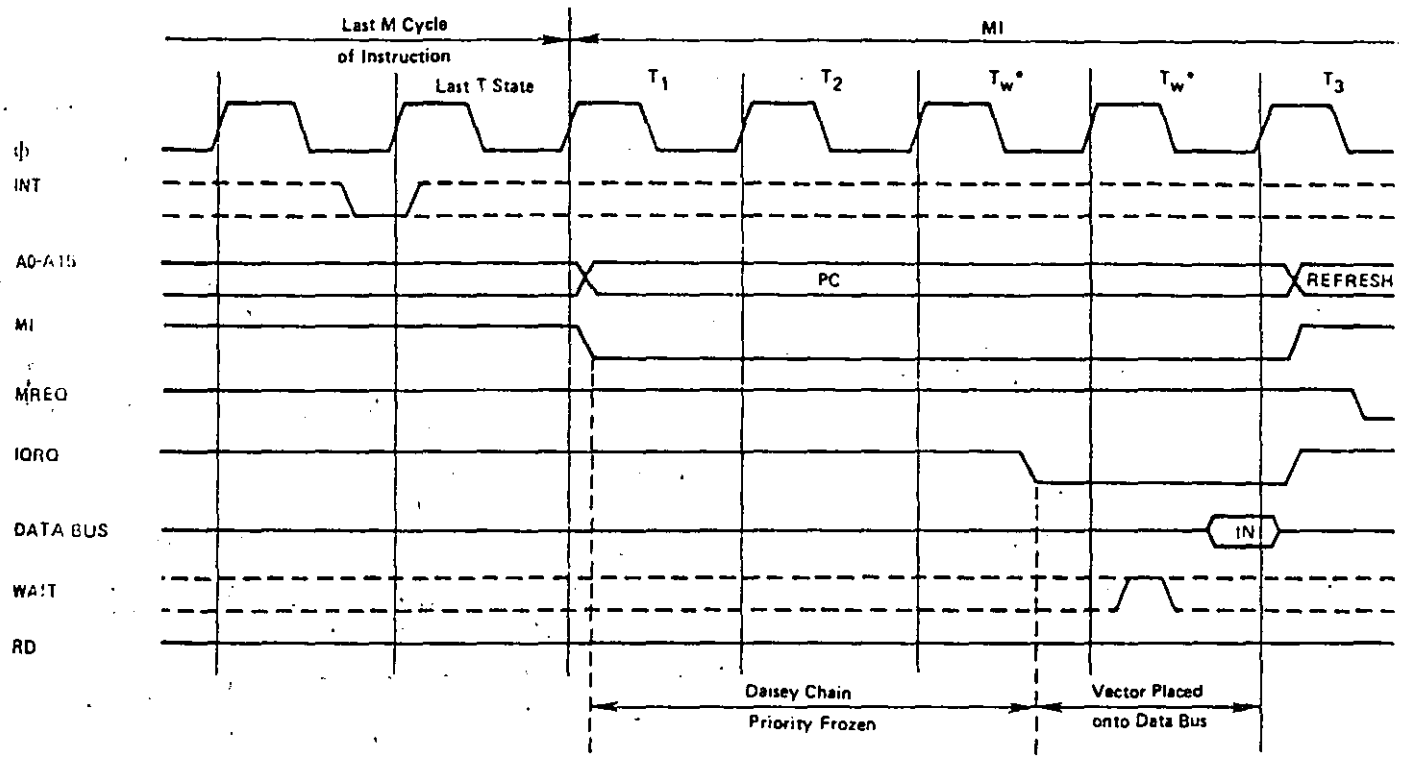
The first byte in the table is the least significant (low order) portion of the address. programmer must obviously fill this table in with the desired addresses before any interrupt are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/ Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting device supplies the lower portion of the pointer, the CPU automat cally pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80-PIO, Z80-SIO and Z80-CTC manuals for details.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE



Z80 INTERRUPT ACKNOWLEDGE SUMMARY

1) PERIPHERAL DEVICE REQUESTS INTERRUPT. Any device requesting and interrupt can pull the wired-or line $\overline{INT}$ low.

2) CPU ACKNOWLEDGES INTERRUPT. Priority status is frozen when $\overline{M1}$ goes low during the Interrupt Acknowledge sequence. Propagation delays down the IEI/IEO daisy chain must be settled out when $\overline{IORQ}$ goes low. If IEI is HIGH, an active Peripheral Device will place its Interrupt Vector on the Data Bus when $\overline{IORQ}$ goes low. That Peripheral then releases its hold on $\overline{INT}$ allowing interrupts from a higher priority device. Lower priority devices are inhibited from placing their Vector on the Data Bus or Interrupting because IEO is low on the active device.

3) INTERRUPT IS CLEARED. An active Peripheral device (IEI=1, IEO=0) monitors OP Code fetches for an RETI (ED 4D) instruction which tells the peripheral that its Interrupt Service Routine is over. The peripheral device then re-activates its internal Interrupt structure as well as raising its IEO line to enable lower priority devices.

11.0 ELECTRICAL SPECIFICATIONS
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

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		0.8	V	
V_{IHC}	Clock Input High Voltage	Vcc-.6		Vcc+.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} = −250 μA
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	$0 \leq V_{IN} \leq V_{CC}$

*200mA for -4, -10 or -20 devices

CAPACITANCE
T_A = 25°C, f = 1MHz 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

*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.

A C CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ± 5%, Unless Otherwise Noted

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITION
Φ	t_c	Clock Period	.4	[12]	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	180	(D)	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	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	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	C_L = 50pF
	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	Except T3-M1
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230	nsec	
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge at Clock During M2 to M5	60		nsec	C_L = 50pF
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From $\overline{WR}$	[7]		nsec	
	t_H	Input Hold Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		100	nsec	
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		100	nsec	C_L = 50 pF
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		90	nsec	
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		110	nsec	C_L = 50 pF
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		110	nsec	
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		100	nsec	
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		130	nsec	C_L = 50pF
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		100	nsec	
	$t_{DH\overline{\Phi}(BD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		110	nsec	
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		80	nsec	
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock $\overline{WR}$ Low		90	nsec	C_L = 50pF
	$t_{DH\Phi(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		100	nsec	
	$t_{w(\overline{WRL})}$	Pulse Width, $\overline{WR}$ Low	[10]		nsec	

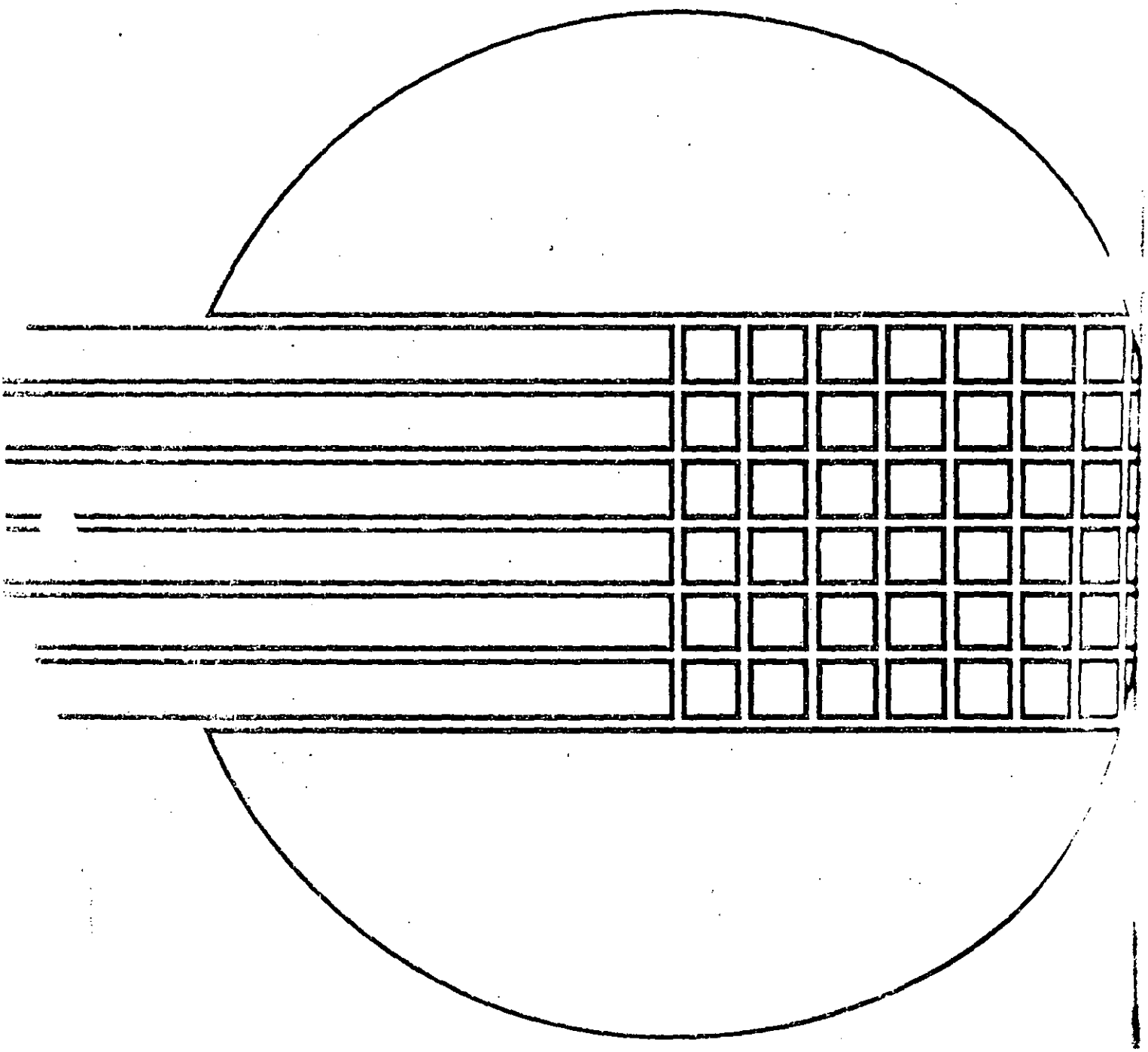
NOTES:

A. ... should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data ... $\overline{M1}$ ... $\overline{IORQ}$ are both active.

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

Continued ... page 79

S GNET CS 16,384-BIT ERASABLE AND REPROGRAMMABLE MOS ROM (2048X8) 2716



16,384-BIT ERASABLE AND REPROGRAMMABLE MOS ROM (2048X8) 2716

PRELIMINARY SPECIFICATION 2716-I

DESCRIPTION

The Signetics 2716 is a 16,384 bit erasable programmable read only memory (EPROM). The 2716 is organized as 2048 words of 8 bits each and features fast single address location programming. Erasure is accomplished by exposure to ultraviolet light and programming is performed electrically. Once a program is finalized the 2716 can convert to Signetics pin-for-pin compatible 16K ROM, the 2616.

The 2716 operates from a single 5 volt power supply which makes it ideal for use with the newer high performance 5 volt microprocessors. A power down mode reduces power during standby to 25% that of operating power.

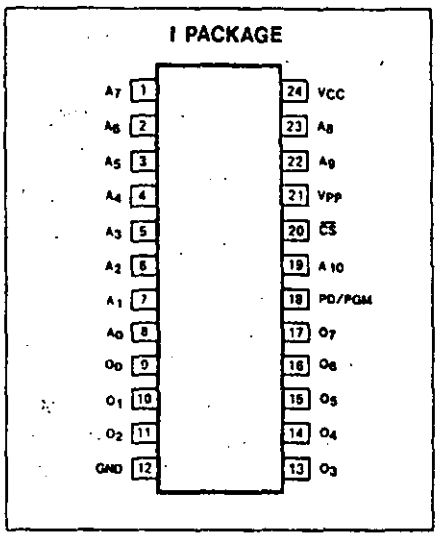
Single pulse TTL level programming makes the 2716 simple and fast to program. All control signals are TTL level allowing on board programming. Words can be selected individually, sequentially or randomly. Total programming time for all 16,384 bits is 100 seconds.

FEATURES

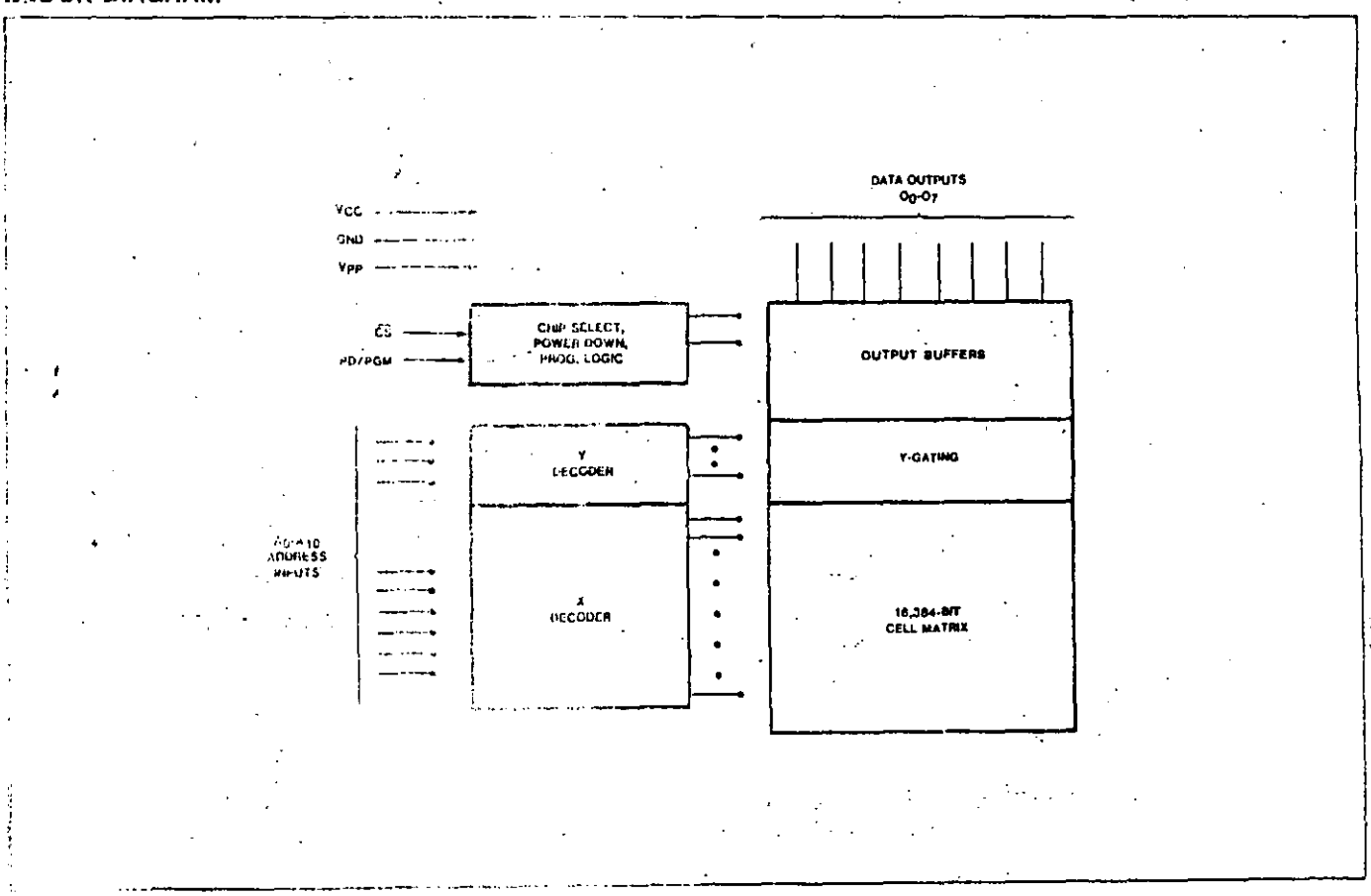
- Single +5V power supply
- Simple programming:
  - Single location programming
  - Single 50ms pulse
  - TTL level signals
- Fast programming—100sec
- Fast access time—450ns max
- Low Power dissipation:
  - 525mW max, active power
  - 132mW max, standby power
- Pin compatible to Signetics 2616 ROM

PIN CONFIGURATION

I PACKAGE



BLOCK DIAGRAM



PRELIMINARY SPECIFICATION

ABSOLUTE MAXIMUM RATINGS[1]

	PARAMETER	RATING	UNIT
	Temperature range		°C
T_A	Operating	−10 to 80	
T_{STG}	Storage	−65 to 125	
	All input or output voltages with respect to ground	−0.3 to 6	V
	V_{PP} supply voltage with respect to ground	−0.3 to 28	V

NOTE

1. Stresses above these listed as "Absolute Maximum Ratings" may damage the device. These ratings are meant for short term stress only, prolonged exposure at these ratings may affect device reliability.

PIN DESIGNATION

PIN NO.	SYMBOL	FUNCTION
1-8, 22-23, 19	A_0-A_{10}	Address inputs
18	PD/PGM	Power down/Program
20	$\overline{CS}$	Chip select
9-11, 13-17	O_0-O_7	Outputs
24	V_{CC}	Power (+5V)
21	V_{PP}	Program voltage (+25V)
12	GND	Ground

DC ELECTRICAL CHARACTERISTICS T_A = 0°C to 70°C, V_{CC}[2] = +5V, ±5%, V_{PP}[3] = V_{CC} ±0.6V[4], unless otherwise specified

	PARAMETER	TEST CONDITIONS	LIMITS			UNITS
			Min	Typ[6]	Max	
	Input voltage					V
V_{IL}	Low		−0.1		0.8	
V_{IH}	High		2.2		V_{CC}+1	
	Output voltage					
V_{OL}	Low	I_{OL} = 2.1mA			0.45	
V_{OH}	High	I_{OH} = −400μA	2.4			
I_{LI}	Input load current	V_{IN} = 5.25V			10	μA
I_{LO}	Output leakage current	V_{OUT} = 5.25V			10	μA
I_{PP1}[3]	V_{PP} current	V_{PP} = 5.85V			5	mA
I_{CC1}[3]	V_{CC} current (standby)	PD/PGM = V_{IH}, $\overline{CS}$ = V_{IL}		10	25	mA
I_{CC}[5]	V_{CC} current (active)	$\overline{CS}$ = PD/PGM = V_{IL}		57	100	mA
	Capacitance[6]	T_A = 25°C, f = 1MHz				pF
C_{IN}	Input	V_{IN} = 0V		4	6	
C_{OUT}	Output	V_{OUT} = 0V		8	12	

NOTES

1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
3. V_{PP} may be connected directly to V_{CC} except during programming. The V_{CC} supply current would then be the sum of I_{CC} and I_{PP1}.
4. The tolerance of 0.6V allows the use of a driver circuit for switching the V_{PP} supply pin from V_{CC} in read to 25V for programming.
6. Typical values are for T_A = 25°C and nominal supply voltages.
5. This parameter is only sampled and is not 100% tested.
2. V_{CC} is referenced to PD/PGM or the addresses, whichever occure last.

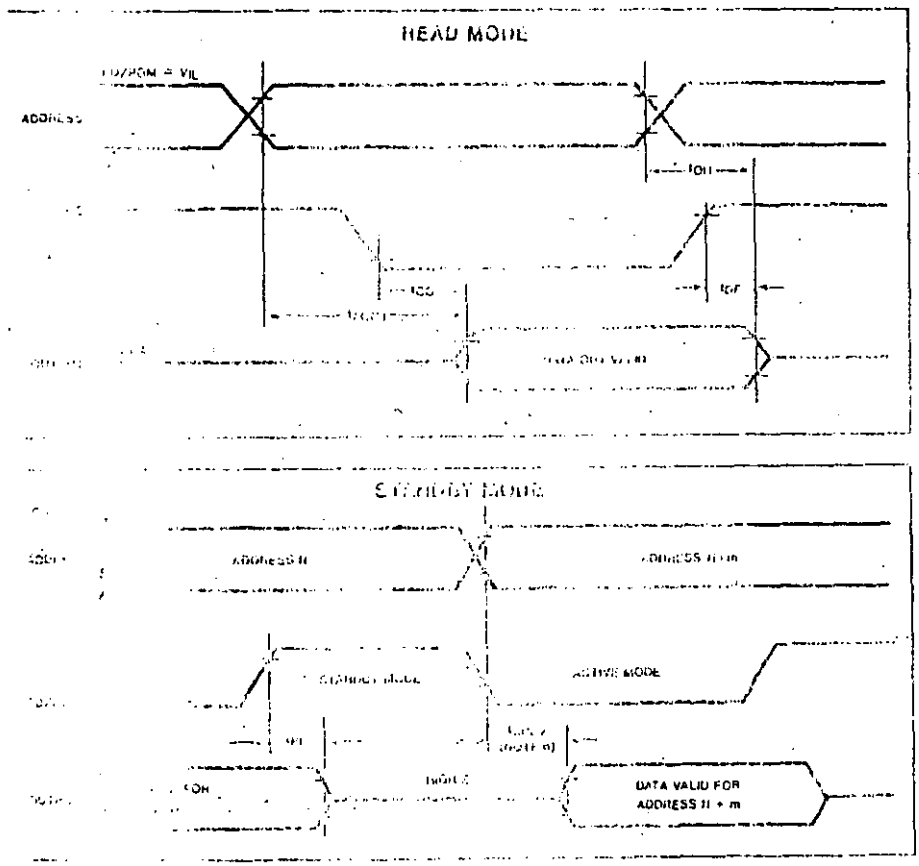
16,384-BIT ERASABLE AND REPROGRAMMABLE MOS ROM (2048X8) 2716

PRELIMINARY SPECIFICATION 2716-I

AC ELECTRICAL CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5V, ±5%, V_{PP} = V_{CC} ±0.6V, unless otherwise specified.

PARAMETER	TO	FROM	TEST CONDITIONS	LIMITS			UNIT
				Min	Typ[5]	Max	
Delay time							ns
t_{ACC1}	Output	Address	PD/PGM = $\overline{CS}$ = V_{IL}		250	450	
t_{ACC2}[7]	Output	PD/PGM			280	450	
t_{CO}	Output	Chip select	PD/PGM = V_{IL}			150	
Float time							ns
t_{PF}	Output	PD/PGM	$\overline{CS}$ = V_{IL}	0		100	
t_{DF}	Output	Chip deselect	PD/PGM = V_{IL}	0		100	
Hold time							ns
t_{OH}	Output	Address	PD/PGM = $\overline{CS}$ = V_{IL}	0			

VOLTAGE WAVEFORMS



DC PROGRAMMING CHARACTERISTICS[8] T_A = 25°C ±5°C, V_{CC}[9] = 5V ±5%, V_{PP}[9,10] = 25V ±1V, unless otherwise specified

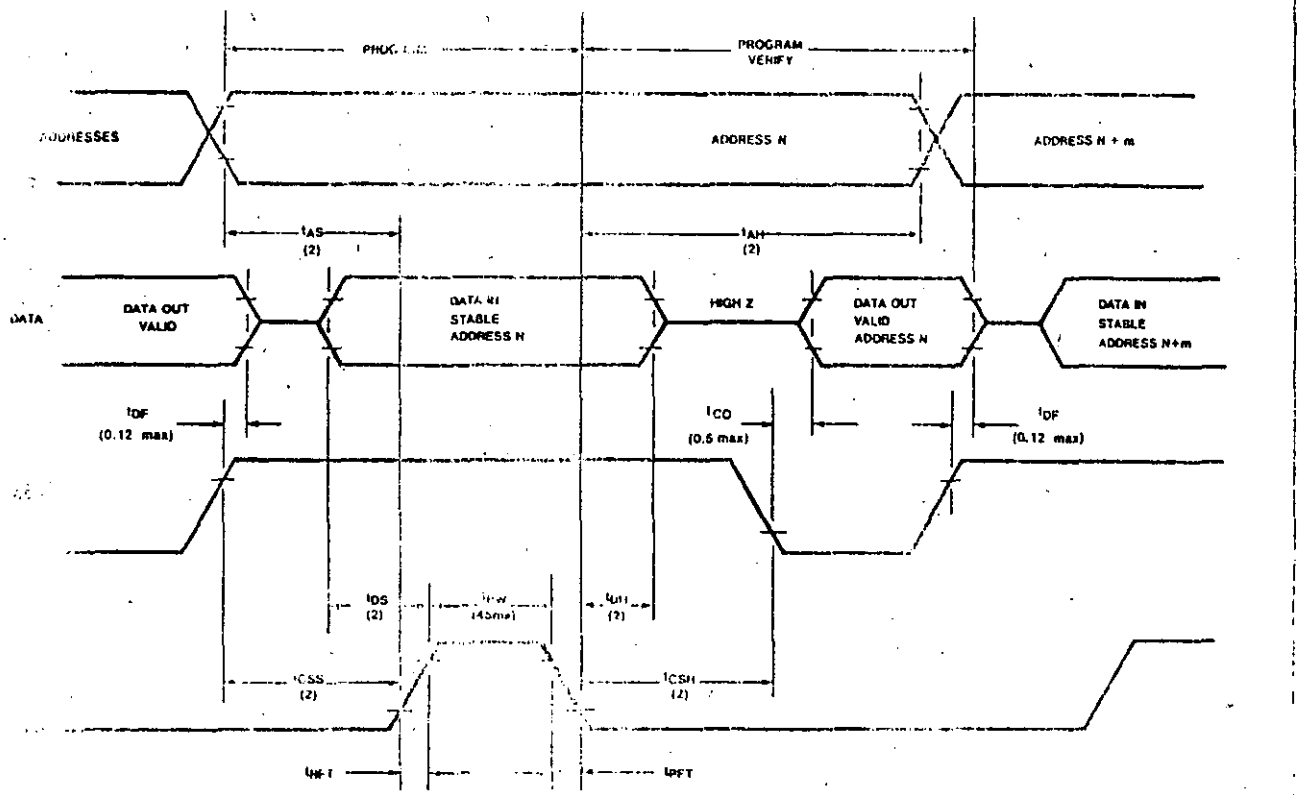
PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
I_{LI}	Input current (for any input)	V_{IN} = 5.25V/0.45			10	µA
I_{PP1}	V_{PP} supply current	PD/PGM = V_{IL}			5	mA
I_{PP2}	V_{PP} supply current during programming pulse	PD/PGM = V_{IH}			30	mA
I_{CC}	V_{CC} supply current				100	mA
V_{IL}	Input low level		−0.1		0.8	V
V_{IH}	Input high level		2.2		V_{CC}+1	V

AC PROGRAMMING CHARACTERISTICS[8] T_A = 25°C ±5°C, V_{CC}[9] = 5V ±5%, V_{PP}[9,10] = 25V ±1V, unless otherwise specified

PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
t_{AS}	Address setup time		2			µs
t_{CSS}	CS setup time		2			µs
t_{DS}	Data setup time		2			µs
t_{AH}	Address hold time		2			µs
t_{CSH}	CS hold time		2			µs
t_{DH}	Data hold time		2			µs
t_{DF}	Chip deselect to output float delay		0		120	ns
t_{CO}	Chip select to output delay				150	ns
t_{PW}	Program pulse width		45	50	55	ms
t_{PRT}	Program pulse rise time		5			ns
t_{PFT}	Program pulse fall time		5			ns

8. Signetics standard product warranty applies only to devices programmed to specifications described herein.
9. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}. The 2716 must not be inserted into or removed from a board with V_{PP} at 25 ±1V to prevent damage to the device.
10. The maximum allowable voltage which may be applied to the V_{PP} pin during programming is +26V. Care must be taken when switching the V_{PP} supply to prevent overshoot exceeding this 26V maximum specification.

PROGRAMMING WAVEFORM



All parentheses are minimum times and are µs unless otherwise noted.

...be erased by using the follow-
...
- ...ultraviolet lamp with a wave length ... angstroms (Å)
- ...tube within 1 inch of device
- ...filter from lamp tube
- ...ce for a total dose (intensity x ... time) of 15W-sec/cm²

...places all 16,384 bits in the ... state ready for selective ... programming. Light sources with ... shorter than 4000Å can cause ... measure. Direct sunlight could ... 2716 in 1 week while room ... could do the same in ap... years. Signetics has opaque ... if the 2716 is to be exposed ... light levels for extended ... labels could also be used ... of parts on the board. The ... be placed on devices with ... programs.

...following list of ultraviolet lamps are ...available from Ultra-violet Products Inc., 5114 Walnut Grove Ave., San Gabriel, CA:

MODEL	POWER RATING	TIME FOR 15W-sec/cm²
S-68	12000 µW/cm²	21 minutes
S-52	12000 µW/cm²	21 minutes
UVS-54	5700 µW/cm²	44 minutes
R-52	13000 µW/cm²	20 minutes
UVS-11	5500 µW/cm²	46 minutes

It should be noted that ultraviolet lamps have a tendency to degrade with constant On-Off operation.

PROGRAM

After erasure, all bits are in the "1" state. Programming to a "0" is accomplished using the following procedure:

- PD/PGM (initially low (V_{IL}))
- V_{CC} to +5V, V_{PP} to +25V
- $\overline{CS}$ HIGH (V_{IH})
- Select desired word by applying addresses
- Pull outputs to "1" level low (V_{IL})
- Apply single, active HIGH TTL program pulse to PD/PGM
- Repeat for all words

Multiple 2716's can be loaded with the same data by paralleling all pins and following the program procedure.

Multiple 2716's can be loaded with different data by paralleling all pins except PD/PGM and following the program procedure. The program pulse should be applied to the selected chip while all deselected chips have their PD/PGM inputs held LOW.

The program can be verified without reducing Vpp by holding both the PD/PGM and $\overline{CS}$ pins LOW.

READ

The Vpp pin should be at +5 ±5% for all except program operations. A LOW on both the PD/PGM and $\overline{CS}$ pins provides the data of the selected word to the output. A HIGH on either PD/PGM or $\overline{CS}$ deselects the chip for reading of parallel devices. A HIGH PD/PGM reduces the active power dissipation by 75%.

OPERATING MODES

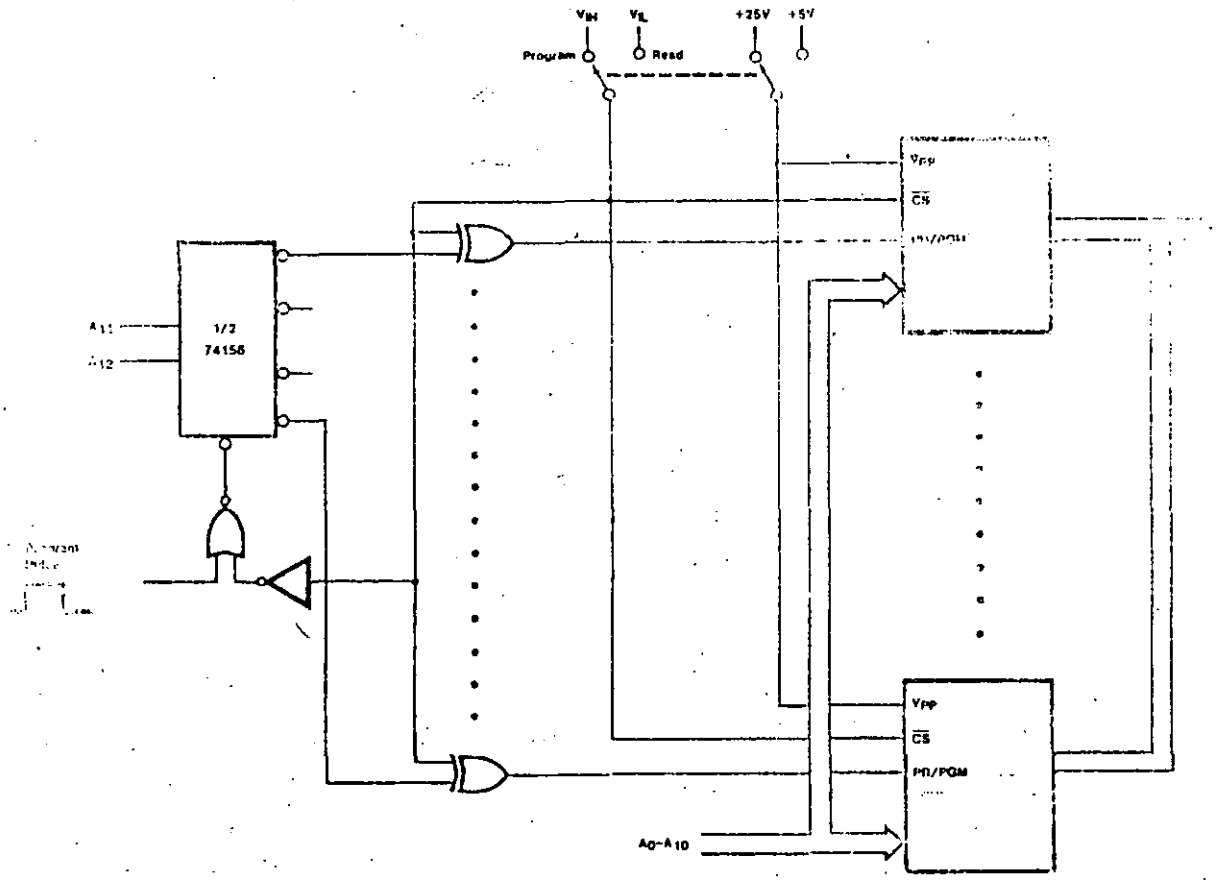
MODE \ PINS	PD/PGM (18)	CS (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9–11, 13–17)
PROGRAM					
Write	⎍	H	+25	+5	D_{IN}
Inhibit	L	H	+25	+5	High Z
Verify	L	L	+25	+5	D_{OUT}
READ					
Read	L	L	+5	+5	D_{OUT}
Power down	H	Don't care	+5	+5	High Z
Deselect	Don't care	H	+5	+5	High Z

APPLICATIONS

On-board programming with the 2716 is an easy task. The circuit shown is for an 8K x 8 EPROM system. This technique uses all of the 6 different modes of operation. During programming the selected device is in the write mode while the unselected devices are in the inhibit mode. During read the selected device is in the read mode while unselected devices are in the power down mode.

APPLICATION DIAGRAM

8Kx8 ON-BOARD PROGRAMMING SYSTEM



Signetics
a subsidiary of U.S. Philips Corporation

Signetics Corporation
P.O. Box 409
811 East Arques Avenue
Sunnyvale, California 94086
Telephone (408) 739-7700

Printed in U.S.A.

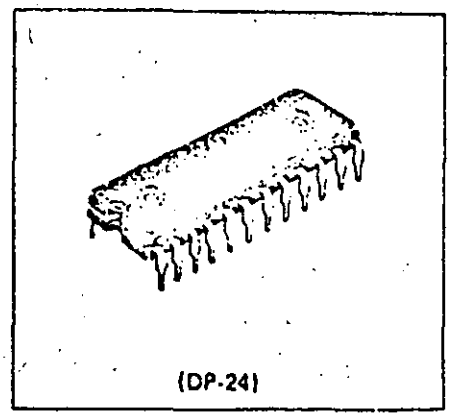
HM6116P-2, HM6116P-3, HM6116P-4

(0) HITACHI

2048-word × 8-bit High Speed Static CMOS RAM

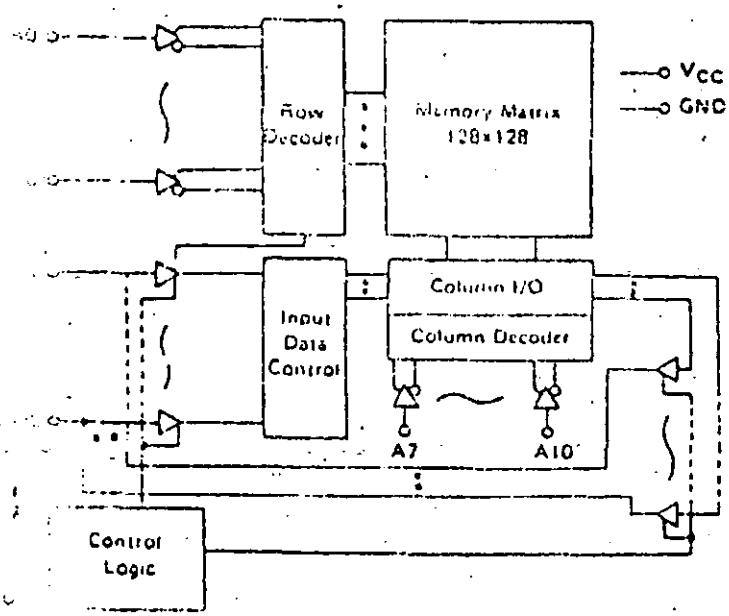
FEATURES

- Single 5V Supply and High Density 24 pin Package
- High Speed: Fast Access Time 120ns/150ns/200ns (max.)
- Low Power Standby and Low Power Operation; Standby: 100μW (typ.)
  Operation: 180mW (typ.)
- Completely Static RAM: No clock or Timing Strobe Required
- Directly TTL Compatible: All Input and Output
- Pin Out Compatible with Standard 16K EPROM/MASK ROM
- Equal Access and Cycle Time

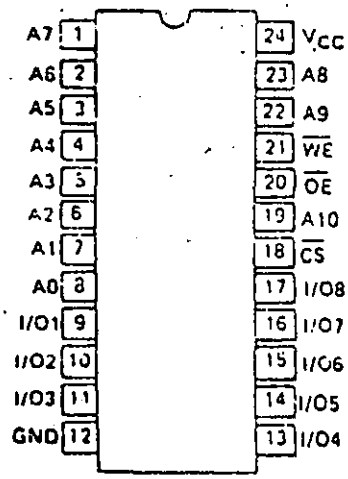


(DP-24)

FUNCTIONAL BLOCK DIAGRAM



PIN ARRANGEMENT



(Top View)

ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Rating	Unit
Voltage on Any Pin Relative to GND	V_{IN}	−0.5 to +7.0	V
Operating Temperature	T_{opr}	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Temperature Under Bias	T_{bias}	−10 to +85	°C
Power Dissipation	P_T	1.0	W

TRUTH TABLE

OE	WE	Mode	I_{CC} Current	I/O Pin	Ref. Cycle
X	X	Not Selected	I_{SB}, I_{SB1}	High Z	
L	H	Read	I_{CC}	D_{out}	Read Cycle (1) ~ (3)
H	L	Write	I_{CC}	D_{in}	Write Cycle (1)
L	L	Write	I_{CC}	D_{in}	Write Cycle (2)

■ RECOMMENDED DC OPERATING CONDITIONS (T_a = 0 to +70°C)

Item	Symbol	min.	typ.	max.	Unit
Supply Voltage	V_{CC}	4.5	5.0	5.5	V
	GND	0	0	0	V
Input Voltage	V_{IH}	2.2	3.5	6.0	V
	V_{IL}	−1.0*	–	0.8	V

* Pulse Width: 50 ns, DC: V_{IL} min. = −0.3V.

■ DC AND OPERATING CHARACTERISTICS (V_{CC} = 5V ±10%, GND = 0V, T_a = 0 to +70°C)

Item	Symbol	Test Conditions	HM6116P-2			HM6116P-3/-4			Unit
			min.	typ.*	max.	min.	typ.*	max.	
Input Leakage Current	$\|I_{LI}\|$	V_{CC} = 5.5V, V_{in} = GND to V_{CC}	–	–	10	–	–	10	μA
Output Leakage Current	$\|I_{LO}\|$	$\overline{CS}$ = V_{IH} or $\overline{OE}$ = V_{IH}, $V_{I/O}$ = GND to V_{CC}	–	–	10	–	–	10	μA
Operating Power Supply Current	I_{CC}	$\overline{CS}$ = V_{IL}, $I_{I/O}$ = 0mA	–	40	80	–	35	70	mA
	I_{CC1}**	V_{IH} = 3.5V, V_{IL} = 0.6V, $I_{I/O}$ = 0mA	–	35	–	–	30	–	mA
Average Operating Current	I_{CC2}	Min. cycle, duty = 100%	–	40	80	–	35	70	mA
Standby Power Supply Current	I_{SB}	$\overline{CS}$ = V_{IH}	–	5	15	–	5	15	mA
	I_{SB1}	$\overline{CS}$ ≥ V_{CC} −0.2V, V_{in} ≥ V_{CC} −0.2V or V_{in} ≤ 0.2V	–	0.02	2	–	0.02	2	mA
Output Voltage	V_{OL}	I_{OL} = 4mA	–	–	0.4	–	–	–	V
		I_{OL} = 2.1mA	–	–	–	–	–	0.4	V
	V_{OH}	I_{OH} = −1.0mA	2.4	–	–	2.4*	–	–	V

*: V_{CC} = 5V, T_a = 25°C
**: Reference Only

■ AC CHARACTERISTICS (V_{CC} = 5V ±10%, T_a = 0 to +70°C)

• AC TEST CONDITIONS

Input Pulse Levels: 0.8 to 2.4V
Input Rise and Fall Times: 10 ns
Input and Output Timing Reference Levels: 1.5V
Output Load: 1TTL Gate and C_L = 100pF (including scope and Jig)

• READ CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min.	max.	min.	max.	min.	max.	
Read Cycle Time	t_{RC}	120	–	150	–	200	–	ns
Address Access Time	t_{AA}	–	120	–	150	–	200	ns
Chip Select Access Time	t_{ACS}	–	120	–	150	–	200	ns
Chip Selection to Output in Low Z	t_{CLZ}	10	–	15	–	15	–	ns
Output Enable to Output Valid	t_{OE}	–	80	–	100	–	120	ns
Output Enable to Output in Low-Z	t_{OLZ}	10	–	15	–	15	–	ns
Chip deselection to Output in High Z	t_{CHZ}	0	40	0	50	0	60	ns
Chip Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Output Hold from Address Change	t_{OH}	10	–	15	–	15	–	ns

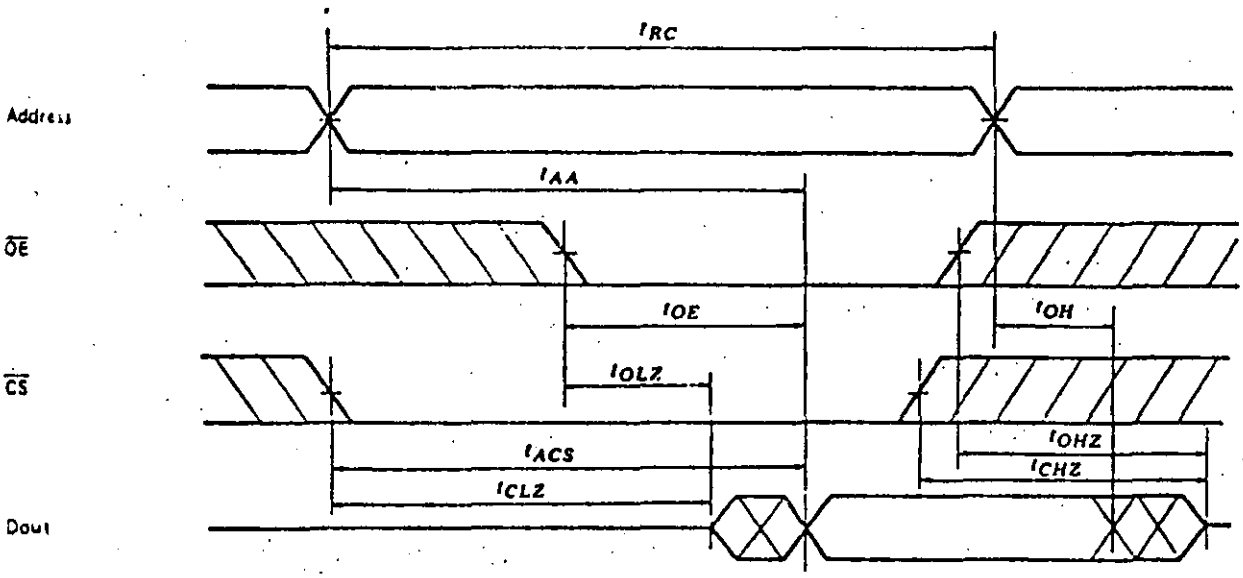
• WRITE CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min.	typ.	min.	max.	min.	max.	
Write Cycle Time	t_{WC}	120	–	150	–	200	–	ns
Chip Selection to End of Write	t_{CW}	70	–	90	–	120	–	ns
Address Valid to End of Write	t_{AW}	105	–	120	–	140	–	ns
Address Set Up Time	t_{AS}	20	–	20	–	20	–	ns
Write Pulse Width	t_{WP}	70	–	90	–	120	–	ns
Write Recovery Time	t_{WR}	5	–	10	–	10	–	ns
Output Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Write to Output in High Z	t_{WHZ}	0	50	0	60	0	60	ns
Data to Write Time Overlap	t_{DW}	35	–	40	–	50	–	ns
Data Hold from Write Time	t_{DH}	5	–	10	–	10	–	ns
Output Active from End of Write	t_{OW}	5	–	10	–	10	–	ns

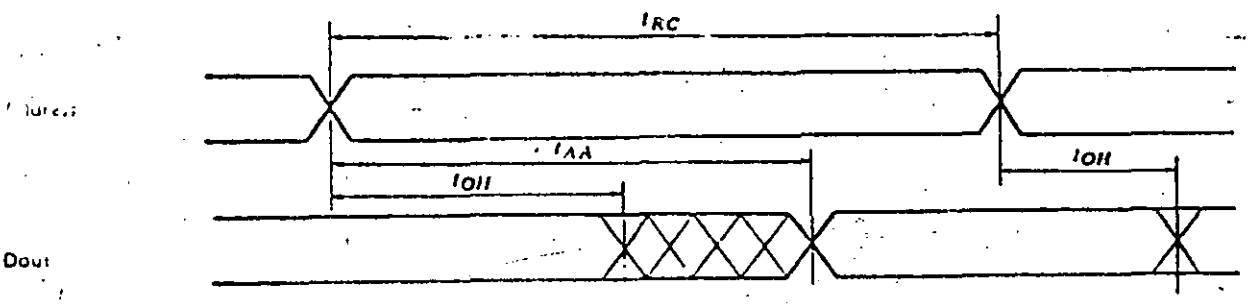
● CAPACITANCE (f = 1MHz, T_a = 25°C)

Item	Symbol	Test Conditions	typ.	max.	Unit
Input Capacitance	C_{in}	V_{in} = 0V	3	5	pF
Input/Output Capacitance	$C_{I/O}$	$V_{I/O}$ = 0V	5	7	pF

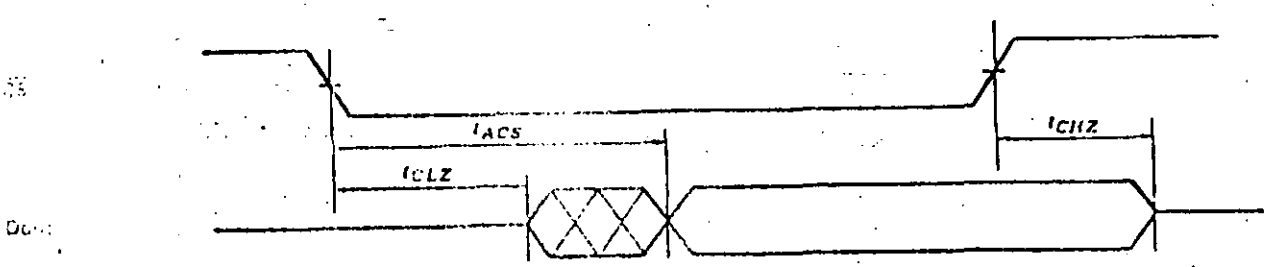
● Read Cycle (1) Notes) 1, 5



● Read Cycle (2) Notes) 1, 2, 4, 5



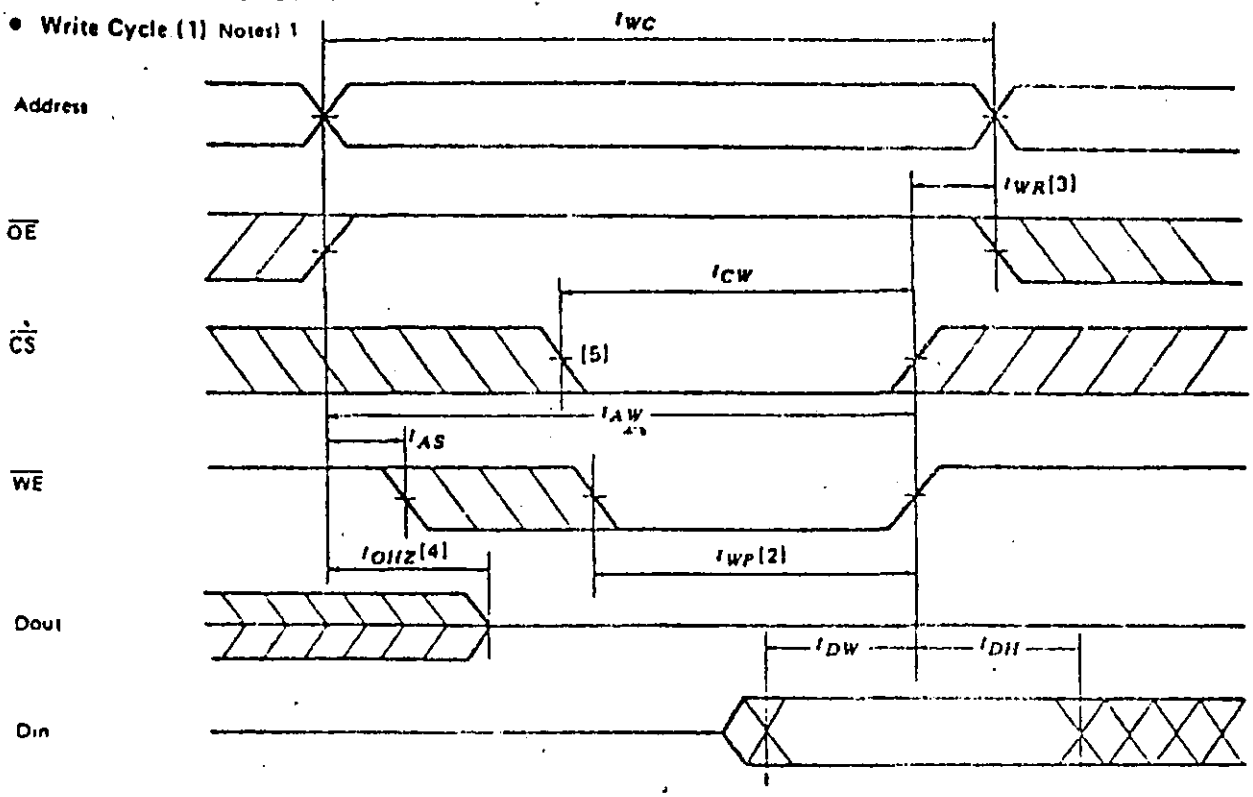
● Read Cycle (3) Notes) 1, 3, 4, 5



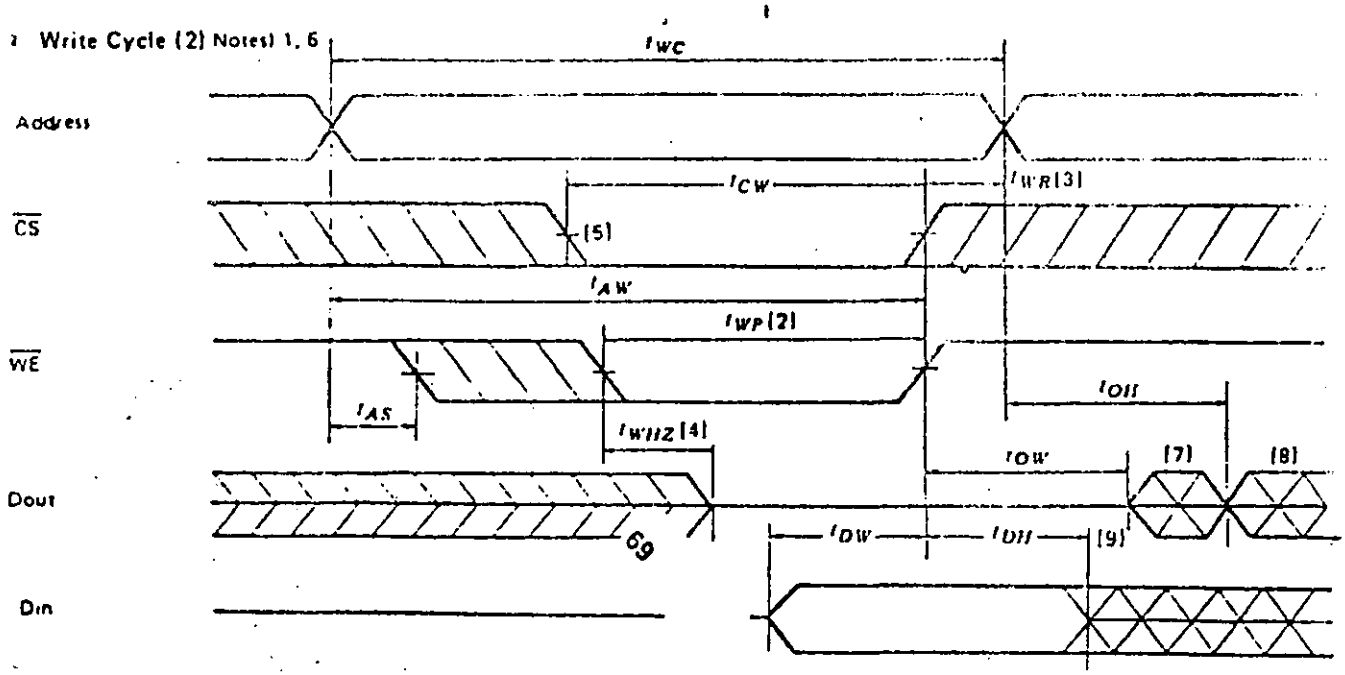
NOTES: 1. $\overline{WE}$ is High for Read Cycle.
2. Device is continuously selected, $\overline{CS}$ = V_{IL}.
3. Address Valid prior to or coincident with $\overline{CS}$ transition Low.
4. $\overline{OE}$ = V_{IL}.
5. When $\overline{CS}$ is Low, the address input must not be in the high impedance state.

■ TIMING WAVEFORM

• Write Cycle (1) Notes) 1

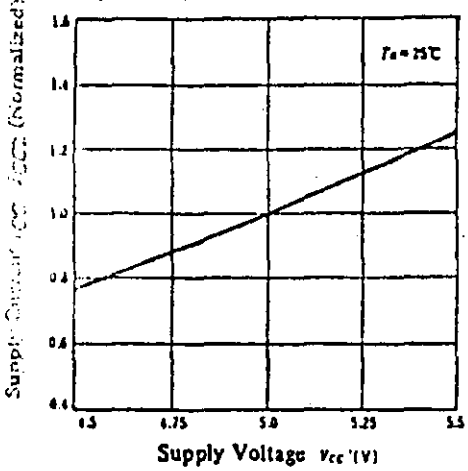


• Write Cycle (2) Notes) 1, 6

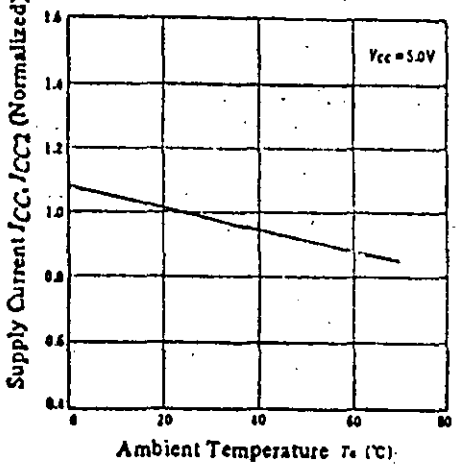


NOTES:
1. $\overline{WE}$ must be high during all address transitions.
2. A write occurs during the overlap (t_{WP}) of a low $\overline{CS}$ and a low $\overline{WE}$.
3. t_{WR} is measured from the earlier of $\overline{CS}$ or $\overline{WE}$ going high to the end of write cycle
4. During this period, I/O pins are in the output state so that the input signals of opposite phase to the outputs must not be applied.
5. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transitions or after the $\overline{WE}$ transition, output remain in a high impedance state.
6. $\overline{OE}$ is continuously low. ($\overline{OE}$ = V_{IL})
7. D_{out} is the same phase of write data of this write cycle.
8. D_{out} is the read data of next address.
9. If $\overline{CS}$ is Low during this period, I/O pins are in the output state. Then the data input signals of opposite phase to the outputs must not be applied to them.

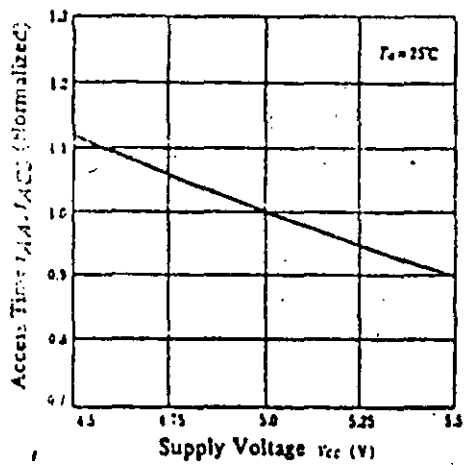
SUPPLY CURRENT
vs. SUPPLY VOLTAGE



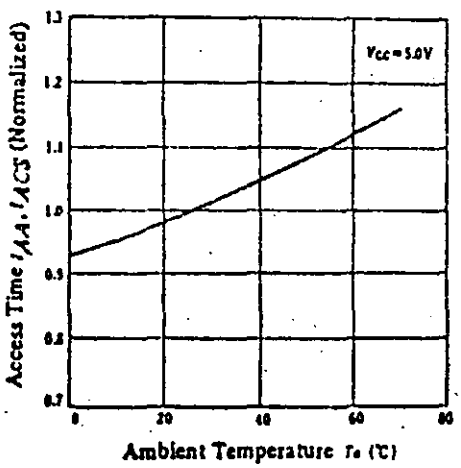
SUPPLY CURRENT
vs. AMBIENT TEMPERATURE



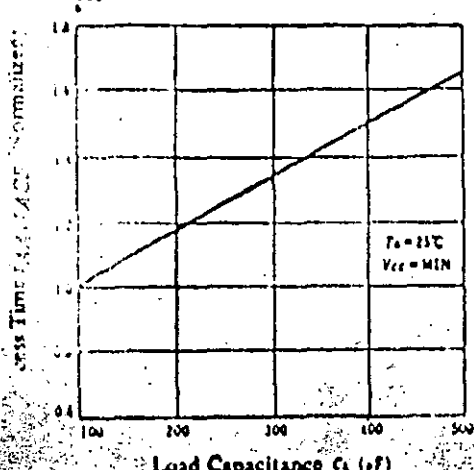
ACCESS TIME
vs. SUPPLY VOLTAGE



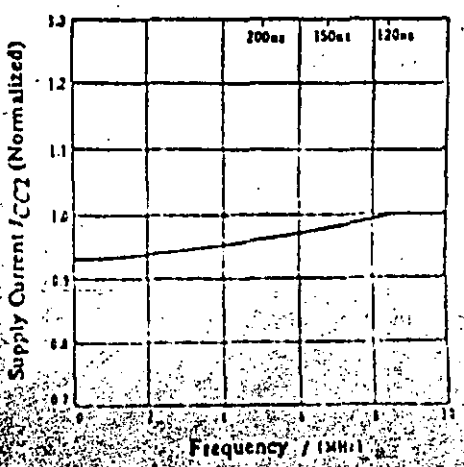
ACCESS TIME
vs. AMBIENT TEMPERATURE

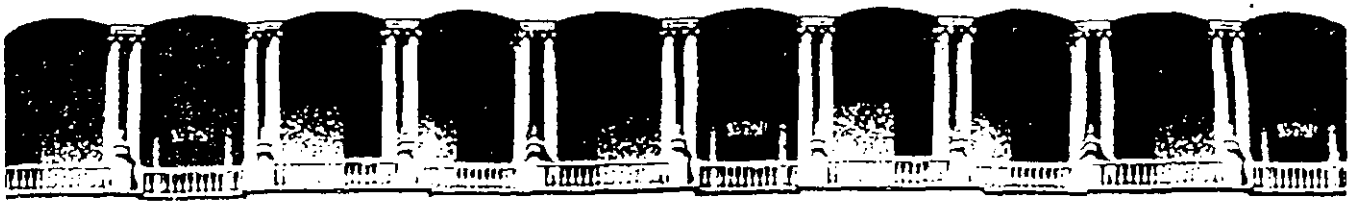


ACCESS TIME
vs. LOAD CAPACITANCE



SUPPLY CURRENT
vs. FREQUENCY





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

CURSOS ABIERTOS

LOS MICROPROCESADORES Y SUS APLICACIONES

Del 14 de agosto al 12 de septiembre de 1992

PROCESAMIENTO DIGITAL DE SEÑALES

APLICACIONES DIVERSAS

PROF: JOSE ANTONIO ARREDONDO GARZA

AGOSTO - 1992

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

Order this document
by BR781/D

MC68HC11F1

Technical Summary
8-Bit Microcontroller

Introduction

The MC68HC11F1 is a high-performance member of the M68HC11 Family of microcontroller units (MCUs). High-speed expanded systems required the development of this chip with its extra input/output (I/O) ports, an increase in static RAM (1K), internal chip-select functions, and a non-multiplexed bus which reduces the need for external interface logic. The timer, serial I/O, and analog-to-digital (A/D) converter enable functions similar to those found in the MC68HC11E9.

For more detailed information about this product, refer to the *M68HC11 Reference Manual*, document number M68HC11 RM/AD.

Features

- MC68HC11 CPU
- 512 Bytes of On-Chip Electrically Erasable Programmable ROM (EEPROM) with Block Protect
- 1024 Bytes of On-Chip RAM (All Saved During Standby)
- Enhanced 16-Bit Timer System
  - — 3 Input Capture (IC) Functions
  - — 4 Output Compare (OC) Functions
  - — 4th IC or 5th OC (Software Selectable)
- On-Board Chip-Selects with Clock Stretching
- Real-Time Interrupt Circuit
- 8-Bit Pulse Accumulator
- Synchronous Serial Peripheral Interface (SPI)
- Asynchronous Nonreturn to Zero (NRZ) Serial Communications Interface (SCI)
- Power Saving STOP and WAIT Modes
- Eight-Channel 8-Bit A/D Converter
- Computer Operating Properly (COP) Watchdog System and Clock Monitor
- 68-Pin PLCC Package

Ordering Information

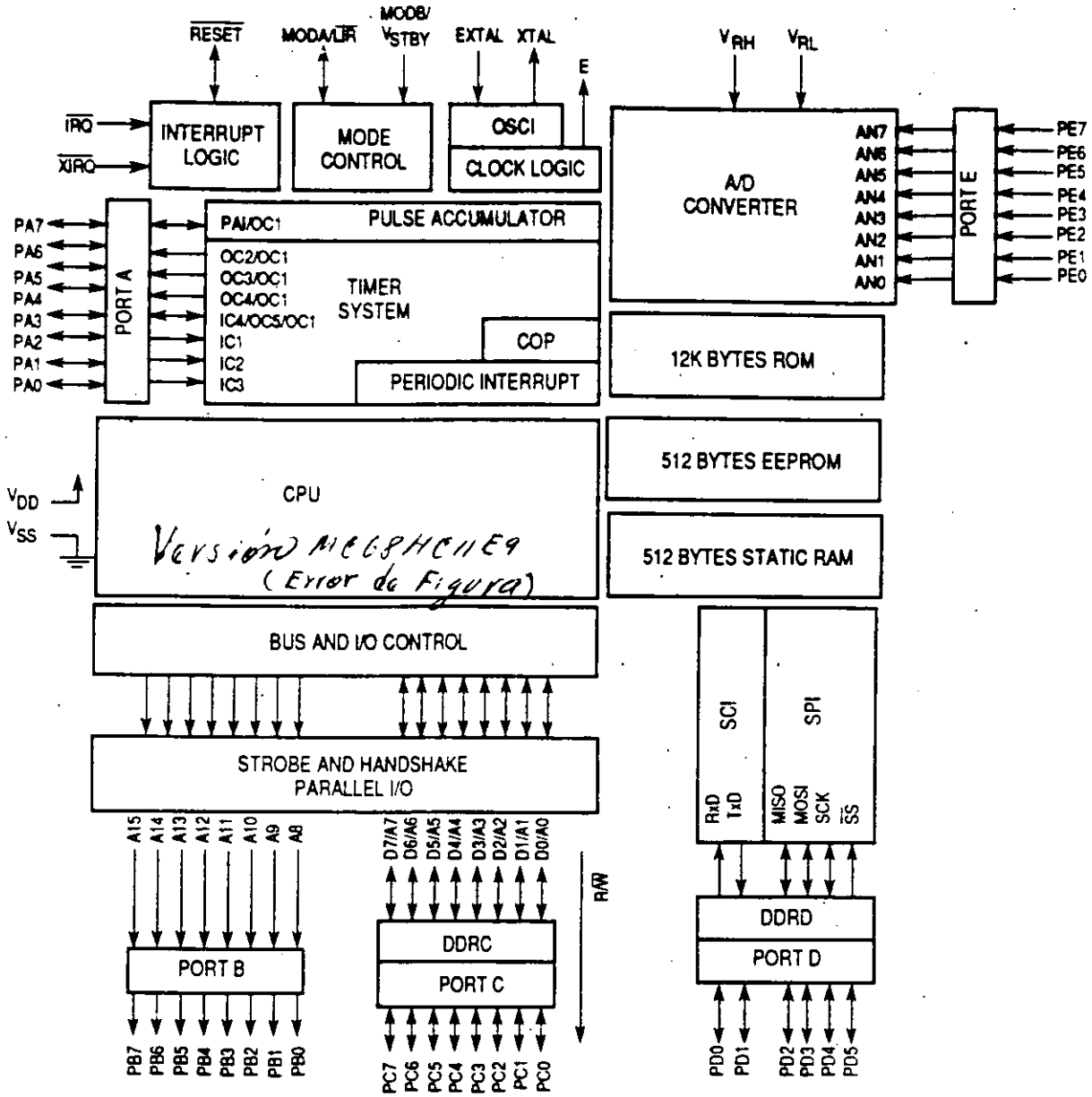
Package	Temperature	MC Order Number
PLCC (FN Suffix)	– 40° to + 85° C	MC68HC11F1FN

This document contains new product information. Specifications and information herein are subject to change without notice.

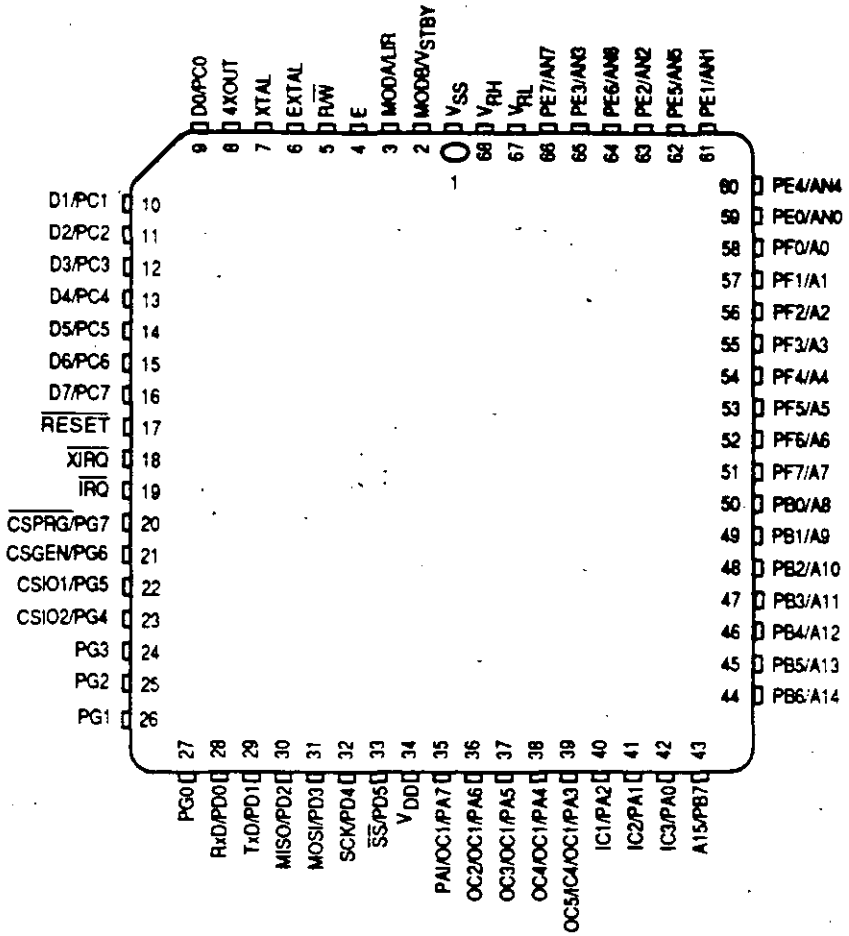
MOTOROLA

Table of Contents

Register Index



Block Diagram



Pin Assignments

Register and Control Bit Assignments (1 of 3)

(The register block can be remapped to any 4K boundary after reset.)

	Bit 7	6	5	4	3	2	1	Bit 0	
$1000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PORTA
$1001	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
$1002	PG7	PG6	PG5	PG4	PG3	PG2	PG1	PG0	PORTG
$1003	DDG7	DDG6	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0	DDRG
$1004	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	PORTB
$1005	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0	PORTF
$1006	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	PORTC
$1007	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
$1008	0	0	PD5	PD4	PD3	PD2	PD1	PD0	PORTD
$1009	0	0	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	PORTE
$100B	FOC1	FOC2	FOC3	FOC4	FOC5	0	0	0	CFORC
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0	OC1M
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0	OC1D
$100E	Bit 15	14	13	12	11	10	9	Bit 8	TCNT (High)
$100F	Bit 7	6	5	4	3	2	1	Bit 0	TCNT (Low)
$1010	Bit 15	14	13	12	11	10	9	Bit 8	TIC1 (High)
$1011	Bit 7	6	5	4	3	2	1	Bit 0	TIC1 (Low)
$1012	Bit 15	14	13	12	11	10	9	Bit 8	TIC2 (High)
$1013	Bit 7	6	5	4	3	2	1	Bit 0	TIC2 (Low)
$1014	Bit 15	14	13	12	11	10	9	Bit 8	TIC3 (High)
$1015	Bit 7	6	5	4	3	2	1	Bit 0	TIC3 (Low)
$1016	Bit 15	14	13	12	11	10	9	Bit 8	TOC1 (High)
$1017	Bit 7	6	5	4	3	2	1	Bit 0	TOC1 (Low)
$1018	Bit 15	14	13	12	11	10	9	Bit 8	TOC2 (High)
$1019	Bit 7	6	5	4	3	2	1	Bit 0	TOC2 (Low)
$101A	Bit 15	14	13	12	11	10	9	Bit 8	TOC3 (High)
$101B	Bit 7	6	5	4	3	2	1	Bit 0	TOC3 (Low)
$101C	Bit 15	14	13	12	11	10	9	Bit 8	TOC4 (High)
$101D	Bit 7	6	5	4	3	2	1	Bit 0	TOC4 (Low)
$101E	Bit 15	14	13	12	11	10	9	Bit 8	TI4O5 (High)
$101F	Bit 7	6	5	4	3	2	1	Bit 0	TI4O5 (Low)

Memory M p and Modes of Oper tion

Because the MC68HC11F1 is intended to operate principally in expanded mode, there is no internal ROM and the bus is nonmultiplexed. Memory consists of 64K bytes of memory-addressing capability. On-chip, there are 1K bytes of static RAM, 512 bytes of EEPROM, and 96 bytes of status and control registers. The RAM, the EEPROM, and the on-chip register block can be independently remapped to any 4K boundary in memory. In the special bootstrap mode, an additional 256 bytes of bootloader ROM are present.

If the control and status registers are relocated so they conflict with the functions of RAM, the register block has priority and the RAM at those locations becomes inaccessible (no harmful conflicts result — lower priority resources simply become inaccessible). Bootstrap ROM has priority over EEPROM.

Reset locates the RAM from $0000–$03FF, and register space from $1000–$105F, where 1 represents the decoded value of the four low-order bits of the INIT register. RAM and control register locations are defined by the INIT register, which can be written to only once within the first 64 E-clock cycles after a reset in normal modes. It becomes a read-only register thereafter.

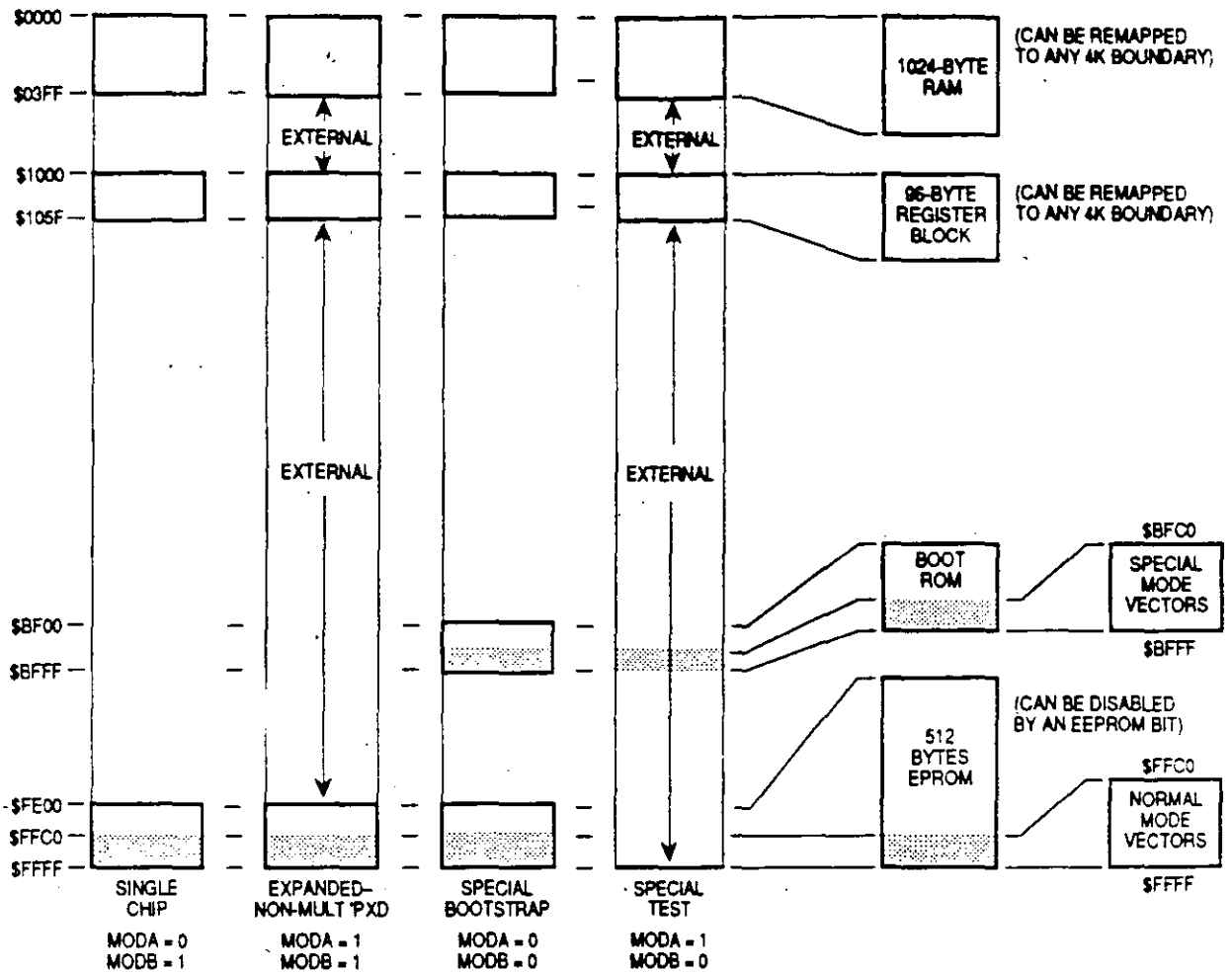
The logic level on the pin (MODB, MODA) determines the mode of operation. Refer to the hardware mode select summary.

In expanded nonmultiplexed and special test modes, EEPROM is located from $xE00–$xFFF, where x represents the value of the four high-order bits of the CONFIG register. EEPROM is enabled by the EEON bit of the CONFIG register. In single-chip and bootstrap modes, the EEPROM is located from $FE00–$FFFF.

Bootstrap ROM is mapped to location $BF00–$BFFF upon reset in bootstrap mode.

Hardware Mode Select Summary

Inputs		Mode Description	Control Bits in HPRIO (Latched at Reset)		
MODB	MODA		RBOOT	SMOD	MDA
1	0	Normal Single Chip	0	0	0
1	1	Normal Expanded	0	0	1
0	0	Special Bootstrap	1	1	0
0	1	Special Test	0	1	1



NOTE: Software can change some aspects of the memory map after reset.

Memory Map

Register and Control Bit Assignments (2 of 3)

	Bit 7	6	5	4	3	2	1	Bit 0	
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1
$1021	EDG4B	EDG4A	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2
$1022	OC1I	OC2I	OC3I	OC4I	I4O5I	IC1I	IC2I	IC3I	TMSK1
$1023	OC1F	OC2F	OC3F	OC4F	I4O5F	IC1F	IC2F	IC3F	TFLG1
$1024	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	TMSK2
$1025	TOF	RTIF	PAOVF	PAIF	0	0	0	0	TFLG2
$1026	0	PAEN	PAMOD	PEDGE	0	I4/05	RTR1	RTR0	PACTL
$1027	Bit 7	6	5	4	3	2	1	Bit 0	PACNT
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
$1029	SPIF	WCOL	0	MODF	0	0	0	0	SPSR
$102A	Bit 7	6	5	4	3	2	1	Bit 0	SPDR
$102B	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	BAUD
$102C	R8	T8	0	M	WAKE	0	0	0	SCCR1
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	0	SCSR
$102F	Bit 7	6	5	4	3	2	1	Bit 0	SCDR
$1030	CCF	0	SCAN	MULT	CD	CC	CB	CA	ADCTL
$1031	Bit 7	6	5	4	3	2	1	Bit 0	ADR1
$1032	Bit 7	6	5	4	3	2	1	Bit 0	ADR2
$1033	Bit 7	6	5	4	3	2	1	Bit 0	ADR3
$1034	Bit 7	6	5	4	3	2	1	Bit 0	ADR4
$1035	0	0	0	PTCON	BPRT3	BPRT2	BPRT1	BPRT0	BPROT
$1036									Reserved
$1037									Reserved
$1038	GWOM	CWOM	CLK4X						OPT2
$1039	ADPU	CSEL	IRQE	DLY	CME	FCME	CR1	CR0	OPTION
$103A	Bit 7	6	5	4	3	2	1	Bit 0	COPRST
$103B	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM	PPROG
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO

Register and Control Bit Assignments (3 of 3)

$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT
$103E	TILOP	0	OCCR	CBYP	DISR	FCM	FCOP	—	TEST1
$103F	EE3	EE2	EE1	EE0	1	NOCOP	1	EEON	CONFIG
$1040									Reserved
$105B									Reserved
$105C	I01SA	I01SB	I02SA	I02SB	GSTHA	GSTGB	PSTHA	PSTHB	CSSTRH
$105D	I01EN	I01PL	I02EN	I02PL	GCSPR	PCSEN	PSIZA	PSIZB	CSCTL
$105E	GA15	GA14	GA13	GA12	GA11	GA10	—	—	CSGADR
$105F	I01AV	I02AV	—	GNPOL	GAVLD	GSIZA	GSIZB	GSIZC	CSGSIZ

HPRIO — Highest Priority I-Bit Interrupt and Miscellaneous **$103C**

	Bit 7	6	5	4	3	2	1	Bit 0	
	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	
RESETS:									
	0	0	0	0	0	1	0	1	Single-Chip Mode
	0	0	1	0	0	1	0	1	Exp'd-Mux'd
	1	1	0	1	0	1	0	1	Bootstrap
	0	1	1	1	0	1	0	1	Special Test

RBOOT, SMOD, MDA, and IRVNE resets depend on mode selected at power-up.

RBOOT — Read Bootstrap ROM

Valid only when SMOD is set to one (special bootstrap or special test mode). Can only be written in special modes.

0 = Boot loader ROM disabled and not in map

1 = Boot loader ROM enabled and in map at $BF00–$BFFF

SMOD and MDA — Special Mode Select and Mode Select A

These two bits can be read at any time. SMOD can only be written in special modes; MDA can be written at any time in special modes, but only once in normal modes.

Inputs		Mode	Latched at Reset	
MODB	MODA		SMOD	MDA
1	0	Single-Chip	0	0
1	1	Expanded Nonmultiplexed	0	1
0	0	Special Bootstrap	1	0
0	1	Special Test	1	1

IRV — Internal Read Visibility

This bit can be read at any time. It can be written at any time in special modes, but only once in normal modes.

0 = Boot loader ROM disabled and not in map

1 = Boot loader ROM enabled and in map at $BF00–$BFFF

Bits 3–0 — Refer to **Resets and Interrupts**.

INIT — RAM and I/O Mapping **$103D**

	Bit 7	6	5	4	3	2	1	Bit 0
	RAM3	RAM2	RAM1	RAM0	REG3	REG4	REG1	REG0
RESET:	0	0	0	0	0	0	0	1

RAM3–RAM0 — Internal RAM Map Position

REG3–REG0 — 128-Byte Register Block Map Position

NOTE

Can be written only once in first 64 cycles out of reset in normal modes, or at any time in special modes. Refer to **Memory Maps and Modes of Operation** for additional information.

TEST1 — Factory Test **$103E**

	Bit 7	6	5	4	3	2	1	Bit 0
	TILOP	0	OCCR	CBYP	DISR	FCM	FCOP	0
RESET:	0	0	0	0	—	0	0	0

These bits writable in test and bootstrap modes only.

TILOP — Test Illegal Opcode

OCCR — Output Condition Code Register to Timer Port

CBYP — Timer Divider Chain Bypass

DISR — Disable Resets from COP and Clock Monitor

In test and bootstrap modes, this bit is reset to 1 to inhibit clock monitor and COP resets. In normal modes, DISR is reset to 0.

FCM — Force Clock Monitor Failure

FCOP — Force COP Watchdog Failure

CONFIG — EEPROM Mapping, COP, EEPROM Enables **$103F**

	Bit 7	6	5	4	3	2	1	Bit 0
	EE3	EE2	EE1	EE0	1	NOCOP	1	EEON
RESET:	U	U	U	U	1	U	1	U

EE3–EE0 — EEPROM Map Position

EEPROM is located at $xE00–xFFF, where x is the value represented by these four bits. In single-chip and bootstrap modes, EEPROM is forced to $FE00–$FFFF, regardless of the state of these bits.

Bits 3 and 1 — Not implemented; always read one

NOCOP — Refer to **Resets and Interrupts**.

EEON — EPROM Enable

This bit is forced to one in single-chip and bootstrap modes. In the test mode, EEON is forced to zero out of reset. In expanded mode, the EEPROM obeys the state of this bit.

0 = EEPROM is removed from the memory map

1 = EEPROM is present in the memory map

Resets and Interrupts

The MC68HC11F1 has 3 reset vectors and 18 interrupt vectors. The reset vectors are as follows:

- $\overline{\text{RESET}}$, or Power-On
- COP Clock Monitor Fail
- COP Failure

The 18 interrupt vectors service 22 interrupt sources (3 non-maskable, 19 maskable). The 3 non-maskable interrupt vectors are as follows:

- Illegal Opcode Trap
- Software Interrupt
- $\overline{\text{XIRQ}}$ Pin (Pseudo Non-Maskable Interrupt)

On-chip peripheral systems generate maskable interrupts that are recognized only if the global interrupt mask bit (I) in the condition code register (CCR) is clear. Maskable interrupts are prioritized according to a default arrangement. However, any one source can be elevated to the highest maskable priority position by a software-accessible control register (HPRIO). The HPRIO register can be written at any time, provided the I bit in the CCR is set.

Nineteen interrupt sources in the MC68HC11F1 are subject to masking by a global interrupt mask bit (I bit in the CCR). In addition to the global I bit, all of these sources, except the external interrupt ($\overline{\text{IRQ}}$ pin), are subject to local enable bits in control registers. Most interrupt sources in the M68HC11 have separate interrupt vectors and there is usually no need for software to poll control registers to determine the cause of an interrupt. The maskable interrupt sources respond to a fixed-priority relationship, except that any one source can be dynamically elevated to the highest priority position of any maskable source. Refer to the following table for a list of interrupt and reset vector assignments.

Vector Address	Interrupt Source	CC Register Mask	Local Mask
FFC0, C1 — FFD4, D5	Reserved	—	—
FFD6, D7	SCI Serial System	I Bit	
	• SCI Transmit Complete		TCIE
	• SCI Transmit Data Register Empty		TIE
	• SCI Idle Line Detect		ILIE
	• SCI Receiver Overrun		RIE
	• SCI Receive Data Register Full		RIE
FFD8, D9	SPI Serial Transfer Complete	I Bit	SPIE
FFDA, DB	Pulse Accumulator Input Edge	I Bit	PAII
FFDC, DD	Pulse Accumulator Overflow	I Bit	PAOVI
FFDE, DF	Timer Overflow	I Bit	TOI
FFE0, E1	Timer Input Capture 4/Output Compare 5	I Bit	I4O5I
FFE3, E2	Timer Output Compare 4	I Bit	OC4I
FFE4, E5	Timer Output Compare 3	I Bit	OC3I
FFE6, E7	Timer Output Compare 2	I Bit	OC2I
FFE8, E9	Timer Output Compare 1	I Bit	OC1I
FFEA, EB	Timer Input Capture 3	I Bit	IC3
FFEC, ED	Timer Input Capture 2	I Bit	IC2I
FFEE, EF	Timer Input Capture 1	I Bit	IC1I
FFF0, F1	Real-Time Interrupt	I Bit	RTII
FFF2, F3	$\overline{\text{IRQ}}$ (External Pin)	I Bit	None
FFF4, F5	$\overline{\text{XIRQ}}$ Pin	X Bit	None
FFF6, F7	Software Interrupt	None	None
FFF8, F9	Illegal Opcode Trap	None	None
FFFA, FB	COP Failure	None	NOCOP
FFFC, FD	COP Clock Monitor Fail	None	CME
FFFE, FF	$\overline{\text{RESET}}$	None	None

For some interrupt sources, such as the SCI interrupts, the flags are automatically cleared during the normal course of responding to the interrupt requests. For example, the RDRF flag in the SCI system is cleared by the automatic clearing mechanism, consisting of a read of the SCI status register while RDRF is set, followed by a read of the SCI data register. The normal response to an RDRF interrupt request is to read the SCI status register to check for receive errors, then to read the received data from the SCI data register. These two steps satisfy the automatic clearing mechanism without requiring any special instructions.

OPT2 — System Configuration Options Register 2 **$1038**

	Bit 7	6	5	4	3	2	1	Bit 0
	GWOM	CWOM	CLK4X	—	—	—	—	—
RESET:	0	0	1	0	0	0	0	0

GWOM — Port G Wired-OR Mode Option
This bit affects all port G pins together.
0 = Port G outputs are normal CMOS outputs
1 = Port G outputs act as open-drain outputs

CWOM — Port C Wired-OR Mode Option
This bit affects all port C pins together.
0 = Port C outputs are normal CMOS outputs
1 = Port C outputs act as open-drain outputs

CLK4X — 4X Clock Output Enable
This bit can only be written once after reset in all modes.
0 = Output of 4XOUT clock is disabled
1 = Output of 4XOUT clock is enabled

Bits 4–0 — Not implemented

OPTION — System Configuration Options **$1039**

	Bit 7	6	5	4	3	2	1	Bit 0
	ADPU	CSEL	IRQE*	DLY*	CME	FCME*	CR1*	CR0*
RESET:	0	0	0	1	0	0	0	0

*Can be written only once in first 64 cycles out of reset in normal modes, or at any time in special modes.

ADPU — A/D Power Up
0 = A/D powered down
1 = A/D powered up

CSEL — Clock Select
0 = A/D and EEPROM use system E-Clock
1 = A/D and EEPROM use internal RC clock

IRQE — $\overline{\text{IRQ}}$ Select Edge Sensitive Only
0 = Low level recognition
1 = Falling edge recognition

DLY — Enable Oscillator Start-Up Delay on Exit from STOP
0 = No stabilization delay on exit from STOP
1 = Stabilization delay of 4064 E-clock cycles is enabled on exit from STOP

CME — Clock Monitor Enable

0 = Clock monitor disabled; slow clocks can be used
1 = Slow or stopped clocks cause clock failure reset

FCME — Force Clock Monitor Enable

0 = Clock monitor circuit follows the state of the CME bit
1 = Cock monitor circuit is enabled until the next reset

CR1, CR0 — COP Timer Rate Select

COP Timer Rate Select

CR [1:0]	Divide E/2^{15} By	XTAL = 8.0 MHz Timeout –0/+16.4 ms	XTAL = 12.0 MHz Timeout –0/+10.9 ms	XTAL = 16.0 MHz Timeout –0/+ 8.2 ms
0 0	1	16.384 ms	10.923 ms	8.192 ms
0 1	4	65.536 ms	43.691 ms	32.768 ms
1 0	16	262.14 ms	174.76 ms	131.07 ms
1 1	64	1.049 sec	699.05 ms	524.29 ms
	E =	2.0 MHz	3.0 MHz	4.0 MHz

COPRST — Arm/Reset COP Timer Circuitry **$103A**

	Bit 7	6	5	4	3	2	1	Bit 0
	7	6	5	4	3	2	1	0
RESET:	0	0	0	0	0	0	0	0

Write $55 to COPRST to arm COP watchdog clearing mechanism. Write $AA to COPRST to reset COP watchdog.

HPRIO — Highest Priority I-Bit Interrupt and Miscellaneous **$103C**

	Bit 7	6	5	4	3	2	1	Bit 0
	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0
RESET:					0	1	0	1

Bits 7–4 — Refer to **Memory Maps and Modes of Operation**.

PSEL3–PSEL0 — Priority Select Bits 3–0

Writable only while the I bit in the CCR is set (interrupts disabled). These bits select one interrupt source to have priority over other I-bit related sources.

PSEL [3:0]	Interrupt Source Promoted
0000	Timer Overflow
0001	Pulse Accumulator Overflow
0010	Pulse Accumulator Input Edge
0011	SPI Serial Transfer Complete
0100	SCI Serial System
0101	Reserved (Default to $\overline{IRQ}$)
0110	$\overline{IRQ}$ (External Pin)
0111	Real-Time Interrupt
1000	Timer Input Capture 1
1001	Timer Input Capture 2
1010	Timer Input Capture 3
1011	Timer Output Compare 1
1100	Timer Output Compare 2
1101	Timer Output Compare 3
1110	Timer Output Compare 4
1111	Timer Output Compare 5/Input Capture 4

Electrically Erasable Programmable ROM (EEPROM)

The MC68HC11F1 has 512 bytes of EEPROM. A nonvolatile, EEPROM-based configuration register (CONFIG) controls whether the EEPROM is present or absent, and its position in the memory map. In single-chip and bootstrap modes the EEPROM is positioned at $FE00–$FFFF. In expanded and special test modes, the EEPROM can be repositioned to any 4K boundary ($xE00–$xFFF).

The EEON bit in CONFIG controls whether the EEPROM is present in the memory map. When EEON = 1, the EEPROM is enabled. When EEON = 0, the EEPROM is disabled and out of the memory map. EEON is forced to 1 out of reset in single-chip and special bootstrap modes to enable EEPROM. EEON is forced to 0 out of reset in special test mode to force EEPROM out of the memory map, although test software can turn it back on. In normal expanded mode, EEON is reset to the value last programmed into CONFIG.

An on-chip charge pump develops the high voltage required for programming and erasing. When E-clock is 1 MHz or above, the charge pump is driven by the E-clock. When the E-clock is less than 1 MHz, select an internal clock to drive the EEPROM charge pump by writing 1 to the CSEL bit in the OPTION register. Refer to the OPTION register.

Programming and erasing the EEPROM is controlled by the PPROG register, subject to the block protect (BPROT) register value. To erase the EEPROM, ensure that the proper bits of the BPROT register are cleared, then complete the following steps:

1. Write to PPROG with the ERASE, EELAT, and appropriate BYTE and ROW bits set.
2. Write to the appropriate EEPROM address with any data. Row erase ($xE00–$xE0F, $xE10–$xE1F, . . . $xFF0–$xFFF) requires a single write to any location in the row. Perform bulk erase by writing to any location in the array.
3. Write to PPROG with ERASE, EELAT, EEPGM, and the appropriate BYTE and ROW bits set.
4. Delay for 10 ms or more, depending on the type of memory.
5. Clear the EEPGM bit in PPROG to turn off the high voltage.
6. Clear the PPROG register to reconfigure EEPROM address and data buses for normal operations.

To program the EEPROM, ensure that the proper bits of the BPROT register are cleared, then complete the following steps:

1. Write to PPROG with the EELAT bit set.
2. Write data to the desired address.
3. Write to PPROG with the EELAT and EEPGM bits set.
4. Delay for 10 ms or more, depending on the type of memory.
5. Clear the EEPGM bit in PPROG to turn off the high voltage.
6. Clear the PPROG register to reconfigure EEPROM address and data buses for normal operations.

BPROT — Block Protect **$1035**

	Bit 7	6	5	4	3	2	1	Bit 0
	0	0	0	PTCON	BPRT3	BPRT2	BPRT1	BPRT0
RESET:	0	0	0	1	1	1	1	1

Bits 7–5 — Not implemented; always read zero

PTCON — Protect for CONFIG

0 = CONFIG register can be programmed or erased normally
1 = CONFIG register cannot be programmed or erased

BPRT3–BPRT0 — Block Protect Bits for EEPROM

0 = Protection disabled
1 = Protection enabled

Bit Name	Block Protected	Block Size
BPRT3	$xEE0–xFFF	288 Bytes
BPRT2	$xE60–xEDF	128 Bytes
PBRT1	$xE20–xE5F	64 Bytes
BPRT0	$xE00–xE1F	32 Bytes

NOTE

Block protect register bits can be written to zero (protection disabled) only once within 64 cycles of a reset in normal modes, or at any time in special modes. Block protect register bits can be written to 1 (protection enabled) at any time.

OPTION — System Configuration Options **$1039**

	Bit 7	6	5	4	3	2	1	Bit 0
	ADPU	CSEL	IRQE*	DLY*	CME	FCME	CR1*	CR0*
RESET:	0	0	0	1	0	0	0	0

*Can be written only once in first 64 cycles out of reset in normal modes, or at any time in special modes.

ADPU — A/D Power Up
- 0 = A/D powered down
- 1 = A/D powered up

CSEL — Clock Select
- 0 = A/D and EEPROM use system E-clock
- 1 = A/D and EEPROM use internal RC clock

IRQE — $\overline{\text{IRQ}}$ Select Edge Sensitive Only
- 0 = Low level recognition
- 1 = Falling edge recognition

DLY — Enable Oscillator Start-Up Delay on Exit from STOP
- 0 = No stabilization delay on exit from STOP
- 1 = Stabilization delay of 4064 E-clock cycles is enabled on exit from STOP

CME — Clock Monitor Enable
- 0 = Clock monitor disabled; slow clocks can be used
- 1 = Slow or stopped clocks cause clock failure reset

FCME — Force Clock Monitor Enable
- 0 = Clock monitor circuit follows the state of the CME bit
- 1 = Cock monitor circuit is enabled until the next reset

CR1, CR0 — COP Timer Rate Select

COP Timer Rate Select

CR [1:0]	Divide E/2^{15} By	XTAL = 4.0 MHz Timeout -0/+32.8 ms	XTAL = 8.0 MHz Timeout -0/+16.4 ms	XTAL = 12.0 MHz Timeout -0/+10.9 ms
0 0	1	32.768 ms	16.384 ms	10.923 ms
0 1	4	131.07 ms	65.536 ms	43.691 ms
1 0	16	524.29 ms	262.14 ms	174.76 ms
1 1	64	2.097 sec	1.049 sec	699.05 ms
	E =	1.0 MHz	2.0 MHz	3.0 MHz

PPROG — EEPROM Programming Control **$103B**

	Bit 7	6	5	4	3	2	1	Bit 0
	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM
RESET:	0	0	0	0	0	0	0	0

ODD — Program Odd Rows (TEST)

EVEN — Program Even Rows (TEST)

ROW and BYTE — Row Erase Select Bit and Byte Erase Select

The value of these bits determines the manner in which EEPROM is erased. Bits encode as follows:

BYTE	ROW	Action
0	0	Bulk Erase (All 512 Bytes)
0	1	Row Erase (16 Bytes)
1	0	Byte Erase
1	1	Byte Erase

ERASE — Erase/Normal Control for EEPROM

0 = Normal read or program mode

1 = Erase mode

EELAT — EEPROM Latch Control

0 = EEPROM address and data bus configured for normal reads

1 = EEPROM address and data bus configured for programming or erasing

EEPGM — EEPROM Program Command

0 = Program or erase voltage to EEPROM array switched off

1 = Program or erase voltage to EEPROM array switched on

NOTE

To program EEPROM, set EELAT and write data to the desired address, then set EEPGM for the required programming time. To erase EEPROM, select ROW = 0 or 1, select BYTE = 0 or 1; set ERASE = EELAT = 1; write to an EEPROM address to be erased; then set EEPGM for the required erase time.

CONFIG — EEPROM Mapping, COP, EEPROM Enables **$103F**

	Bit 7	6	5	4	3	2	1	Bit 0
	EE3	EE2	EE1	EE0	1	NOCOP	1	EEON
RESET:	U	U	U	U	1	U	1	U

Bits 7–3 and 1–0 — Refer to **Electrically Erasable Programmable ROM (EEPROM).**

NOCOP — COP System Disable

0 = COP enabled (forces reset on timeout)

1 = COP disabled (does not force reset on timeout)

Parallel Input/Output

The MC68HC11F1 has six 8-bit I/O ports (A, B, C, E, F, and G), and one 6-bit I/O port (D). I/O functions on some ports (B, C, F, and G) are affected by the mode of operation selected. In the single-chip and bootstrap modes, they are configured as parallel I/O data ports. In expanded nonmultiplexed and test modes, ports B, C, F, G, and pin R/$\overline{W}$ are configured as a memory expansion bus, with ports B and F as the address bus, port C as the data bus, the R/$\overline{W}$ signal as data bus direction control, and the upper four bits of port G as optional external chip-selects. The remaining ports are unaffected by mode changes.

Ports A, D, and G can be used as general-purpose I/O ports, though each has an alternate function. Port A bits control the timer functions, Port D handles the SPI and SCI functions, and Port G controls chip-select functions. Port E can be used for general-purpose static inputs and/or A/D converter channel inputs.

Port A is a general-purpose I/O port with both a data register (PORTA) and a data direction register (DDRA). PORTA pins are available for shared use among main timer, pulse accumulator, and general I/O functions, regardless of mode. Four pins can be used for timer output compare functions (OC), three for input capture (IC), and one as either a fourth IC or a fifth OC.

Port B (PORTB) is a general-purpose output port in single-chip modes; in expanded modes, PORTB is used for the high-order address lines (A15–A8) of the address bus, and accesses to PORTB are treated as external.

Port C is a general-purpose I/O port with a data register (PORTC) and a data direction register (DDRC). In the single-chip modes, PORTC is a general-purpose I/O (PC7–PC0). PORTC can be configured for wired-OR operation in single-chip modes by setting the CWOM bit of the OPT2 register. In the expanded nonmultiplexed modes, PORTC is the data bus (D7–D0), and accesses to PORTC are treated as external.

Port D is a 6-bit, general-purpose I/O port with a data register (PORTD) and a data direction register (DDRD). In all modes, the six port D bits (D5–D0) can be used for general-purpose I/O, or for the SCI and SPI subsystems. Port D can also be configured for wired-OR operation.

Port E (PORTE) is used for general-purpose static inputs (PE7–PE0) and analog-to-digital (A/D) channel inputs. If Port E is read as digital input while an A/D conversion is in progress, small errors can occur in the A/D conversion results.

Port F (PORTF) is a general-purpose output port in single-chip modes, and a low-order address (A7–A0) of the address bus in expanded nonmultiplexed modes.

Port G is an 8-bit, general-purpose I/O port with both a data register (PORTG) and a data direction register (DDRG). The upper four bits (PG7–PG4) can be used as chip-select outputs in expanded modes. When any of these pins are not being used for chip-selects, they can be used as general-purpose I/O.

PORTA — Port A Data **$1000**

	Bit 7	6	5	4	3	2	1	Bit 0
	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
RESET:	Reset configures pins as HiZ inputs							
Alt. Pin Funct.:	PAI	OC2	OC3	OC4	OC5/IC4	IC1	IC2	IC3
And/or:	OC1	OC1	OC1	OC1	OC1	—	—	—

DDRA — Data Direction Register for Port A **$1001**

	Bit 7	6	5	4	3	2	1	Bit 0
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
RESET:	0	0	0	0	0	0	0	0

For DDRx bits, 0 = input and 1 = output.

PORTG — Port G Data **$1002**

	Bit 7	6	5	4	3	2	1	Bit 0
	PG7	PG6	PG5	PG4	PG3	PG2	PG1	PG0
RESET:	Reset does not affect the state of Port G latches							
Alt. Pin Funct.:	$\overline{\text{CSPRG}}$	CSGEN	CSIO1	CSIO2				

DDRG — Data Direction Register for Port G **$1003**

	Bit 7	6	5	4	3	2	1	Bit 0
	DDG7	DDG6	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0
RESET:	0	0	0	0	0	0	0	0

For DDRx bits, 0 = input and 1 = output.

PORTB — Port B Data **$1004**

	Bit 7	6	5	4	3	2	1	Bit 0
	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
RESET:	0	0	0	0	0	0	0	0
Alt. Pin Funct.:	A15	A14	A13	A12	A11	A10	A9	A8

PORTF— Port F Data **$1005**

	Bit 7	6	5	4	3	2	1	Bit 0
	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0
RESET:	0	0	0	0	0	0	0	0
Alt. Pin Funct.:	A7	A6	A5	A4	A3	A2	A1	A0

PORTC — Port C Data **$1006**

	Bit 7	6	5	4	3	2	1	Bit 0
	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
RESET:	Reset does not affect the state of Port C latches							
Alt. Pin Funct.:	D7	D6	D5	D4	D3	D2	D1	D0

DDRC — Data Direction Register for Port C **$1007**

	Bit 7	6	5	4	3	2	1	Bit 0
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
RESET:	0	0	0	0	0	0	0	0

For DDRx bits, 0 = input and 1 = output.

PORTD — Port D Data **$1008**

	Bit 7	6	5	4	3	2	1	Bit 0
	0	0	PD5	PD4	PD3	PD2	PD1	PD0
RESET:	Reset does not affect the state of Port D latches							
Alt. Pin Funct.:	—	—	$\overline{SS}$	SCK	MOSI	MISO	TxD	RxD

DDRD — Data Direction Register for Port D **$1009**

	Bit 7	6	5	4	3	2	1	Bit 0
	0	0	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
RESET:	0	0	0	0	0	0	0	0

For DDRx bits, 0 = input and 1 = output.

PORTE — Port E Data **$100A**

	Bit 7	6	5	4	3	2	1	Bit 0
	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
RESET:	Inputs only. Reset does not affect these pins.							
Alt. Pin Funct.:	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0

OPT2 — System Configuration Options Register 2 **$1038**

	Bit 7	6	5	4	3	2	1	Bit 0
	GWOM	CWOM	CLK4X	—	—	—	—	—
RESET:	0	0	1	0	0	0	0	0

GWOM — Port G Wired-OR Mode Option

This bit affects all port G pins together.

0 = Port G outputs are normal CMOS outputs

1 = Port G outputs act as open-drain outputs

CWOM — Port C Wired-OR Mode Option

This bit affects all port C pins together.

0 = Port C outputs are normal CMOS outputs

1 = Port C outputs act as open-drain outputs

Bit 5 — Refer to **Resets and Interrupts.**

Bits 4–0 — Not implemented; always read zero

Chip-Selects

Chip-selects eliminate the need for additional external components to interface with peripherals in expanded nonmultiplexed modes. Chip-select registers control polarity, address block size, and clock stretching.

There are four programmable chip-selects on the MC68HC11F1: two for external I/O (CSIO1 and CSIO2), one for external program space ($\overline{\text{CSPRG}}$), and one general-purpose chip-select (CSGEN).

The external program chip-select ($\overline{\text{CSPRG}}$) is active low and becomes active at address valid time. $\overline{\text{CSPRG}}$ is enabled by the PCSEN bit of the chip-select control register (CSCTL), and its address block size is selected by the PSIZA and PSIZB bits of CSCTL.

Use the I/O chip-selects (CSIO1 and CSIO2) for external I/O devices. These chip-select addresses are found in the memory map 4K block that contains the status and control registers. CSIO1 is mapped from $1060–$17FF, and CSIO2 is mapped from $1800–$1FFF, where 1 is a character representing the value of the high-order nibble of the register block address. Polarity and enable/disable selections are controlled by CSCTL register bits IO1EN, IO1PL, IO2EN, and IO2PL. The IO1AV and IO2AV bits of the CSGSIZ register determine whether the chip-selects are valid during address or E-clock valid times.

The general-purpose chip-select is the most flexible of the four chip-selects. Polarity, address as opposed to E-clock valid, and address block size are determined by the GNPOL, GAVLD, GSIZA, GSIZB, and GSIZC bits of the CSGSIZ register. The starting address is selected with the CSGADR register.

Each of the four chip-selects is associated with two bits in the chip-select clock stretch register (CSSTRH). These bits allow clock stretching from zero to three cycles (full E-clock periods) to permit slow device interfaces. Any of the chip-selects can be programmed to cause a clock stretch to occur only during access to addresses that fall within that particular chip-select's address range.

During the stretch period, the E-clock is held high and the bus remains in the state that it is normally in at the end of E high time. Internally, the clocks continue to run, which maintains the integrity of the timers and baud-rate generators.

The assignment of priority levels prevents the four chip-selects from conflicting with each other or with internal memory and registers. There are two sets of priorities controlled by the value of the general-purpose chip-select priority bit (GCSPR) of the CSCTL register. Refer to the chip-select priorities table for the highest-to-lowest priority of on-chip memory, on-chip registers, and chip-selects.

CSSTRH — Clock Stretching **$105C**

	Bit 7	6	5	4	3	2	1	Bit 0
	IO1SA	IO1SB	IO2SA	IO2SB	GSTHA	GSTHB	PSTHA	PSTHB
RESET:	0	0	0	0	0	0	0	0

Refer to the following table for the amount of clock stretching for each of the chip-selects in CSSTRH.

Bit A–B [1:0]	Clock Stretch
0 0	0 Cycles
0 1	1 Cycle
1 0	2 Cycles
1 1	3 Cycles

IO1SA and IO1SB — I/O Chip-Select 1 Clock Delay

IO2SA and IO2SB — I/O Chip-Select 2 Clock Delay

GSTHA and GSTHB — General-Purpose Chip-Select Clock Delay

PSTHA and PSTHB — Program Chip-Select Clock Delay

CSCTL — Chip-Select Control **$105D**

	Bit 7	6	5	4	3	2	1	Bit 0
	IO1EN	IO1PL	IO2EN	IO2PL	GCSPR	PCSEN	PSIZA	PSIZB
RESET:	0	0	0	0	0	—	0	0

IO1EN — I/O Chip-Select 1 Enable
0 = CSIO1 disabled
1 = CSIO1 enabled

IO1PL — I/O Chip-Select 1 Polarity
0 = CSIO1 active low
1 = CSIO1 active high

IO1EN — I/O Chip-Select 2 Enable
0 = CSIO2 disabled
1 = CSIO2 enabled

IO2PL — I/O Chip-Select 2 Polarity
0 = CSIO2 active low
1 = CSIO2 active high

GCSPR — General-Purpose Chip-Select Priority
0 = Program chip-select has priority over general-purpose chip-select
1 = General-purpose chip-select has priority over program chip-select

Chip-Select Priorities

GCSPR = 0	GCSPR = 1
On-Chip Registers	On-Chip Registers
On-Chip RAM	On-Chip RAM
Bootloader ROM	Bootloader ROM
On-Chip EEPROM	On-Chip EEPROM
I/O Chip-selects	I/O Chip-Selects
Program Chip-Select	General-Purpose Chip-Select
General-Purpose Chip-Select	Program Chip-Select

PCSEN — Program Chip-Select Enable

Reset clears PCSEN in single-chip mode, sets PCSEN in expanded nonmultiplexed mode.

0 = CSPROG disabled

1 = CSPROG enabled

PSIZA, PSIZB — Select Size of Program Chip-Select

PSIZA	PSIZB	Size (Bytes)	Address Range
0	0	64K	$0000–$FFFF
0	1	32K	$8000–$FFFF
1	0	16K	$C000–$FFFF
1	1	8K	$E000–$FFFF

CSGADR— General-Purpose Chip-Select Address Register **$105E**

	Bit 7	6	5	4	3	2	1	Bit 0
	GA15	GA14	GA13	GA12	GA11	GA10	—	—
RESET:	0	0	0	0	0	0	0	0

GA15–GA10 — General-Purpose Chip-Select Starting Address

These bits determine the starting address of the CSGEN valid address space and correspond to the high-order address bits A15–A10. The following table illustrates how the address size selected determines which of this register's bits are valid:

Address Size Selected	CSGADR Bits Valid
0K Bytes	None
1K Bytes	GA15 — GA10
2K Bytes	GA15 — GA11
4K Bytes	GA15 — GA12
8K Bytes	GA15 — GA13
16K Bytes	GA15 — GA14
32K Bytes	GA15
64K Bytes	None

Bits 1 and 0 — Not implemented

CSGSIZ — General-Purpose Chip-Select Size Register **$105F**

	Bit 7	6	5	4	3	2	1	Bit 0
	IO1AV	IO2AV	—	GNPOL	GAVLD	GSIZA	GSIZB	GSIZC
RESET:	0	0	0	0	0	0	0	0

IO1AV — I/O Chip-Select 1 Address Valid
- 0 = CSIO1 is valid during E-clock valid time (E-clock high)
- 1 = CSIO1 is valid during address valid time

IO2AV — I/O Chip-Select 2 Address Valid
- 0 = CSIO2 is valid during E-clock valid time (E-clock high)
- 1 = CSIO2 is valid during address valid time

Bit 5 — Not implemented; always reads zero

GNPOL — General-Purpose Chip-Select Polarity
- 0 = CSGEN is active low
- 1 = CSGEN is active high

GAVLD — General-Purpose Chip-Select Address Valid
- 0 = CSGEN is valid during E-clock valid time (E-clock high)
- 1 = CSGEN is valid during address valid time

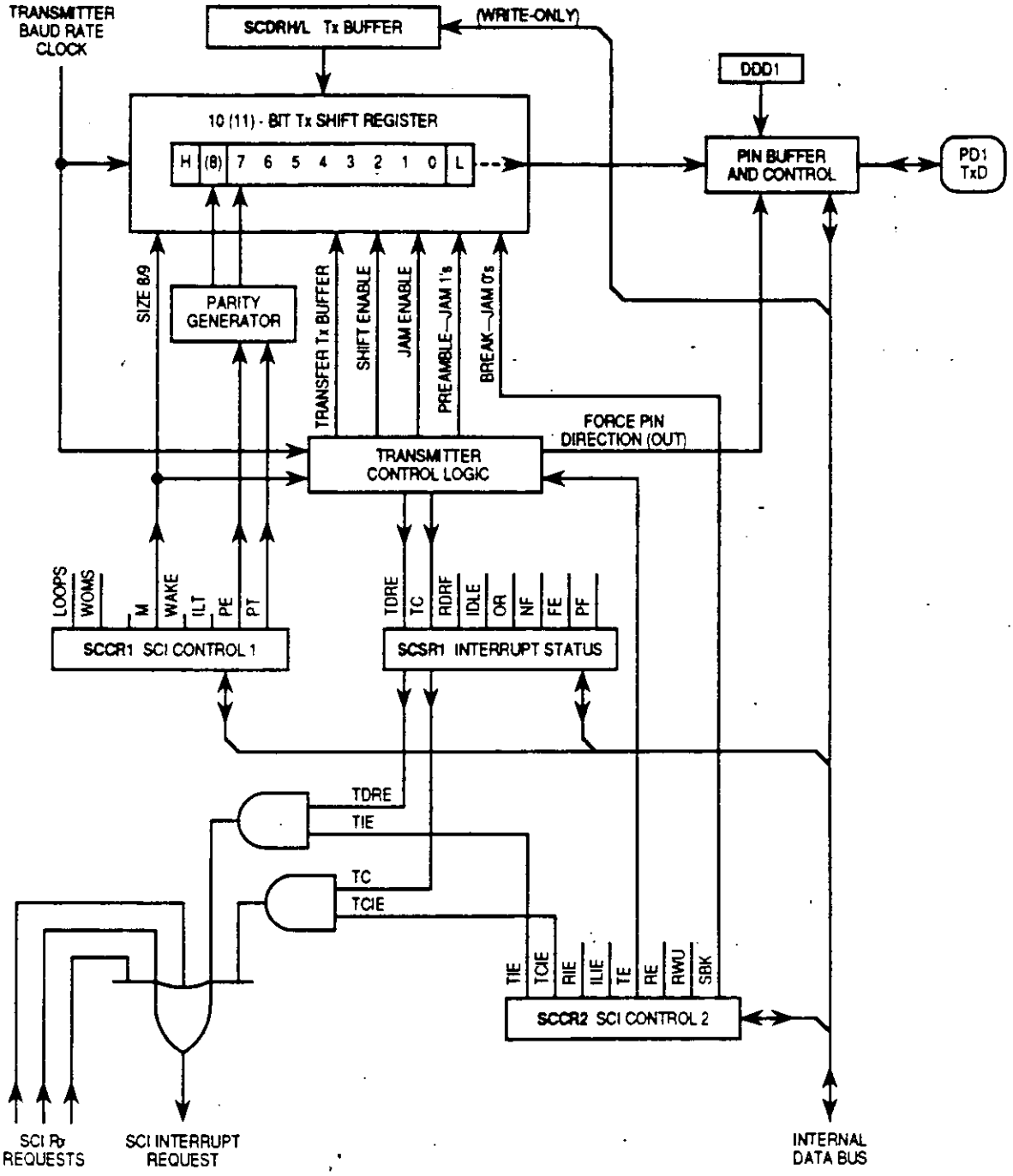
GSIZA, GSIZB, and GSIZC — Address Size for CSGEN

Refer to the following table for bit values:

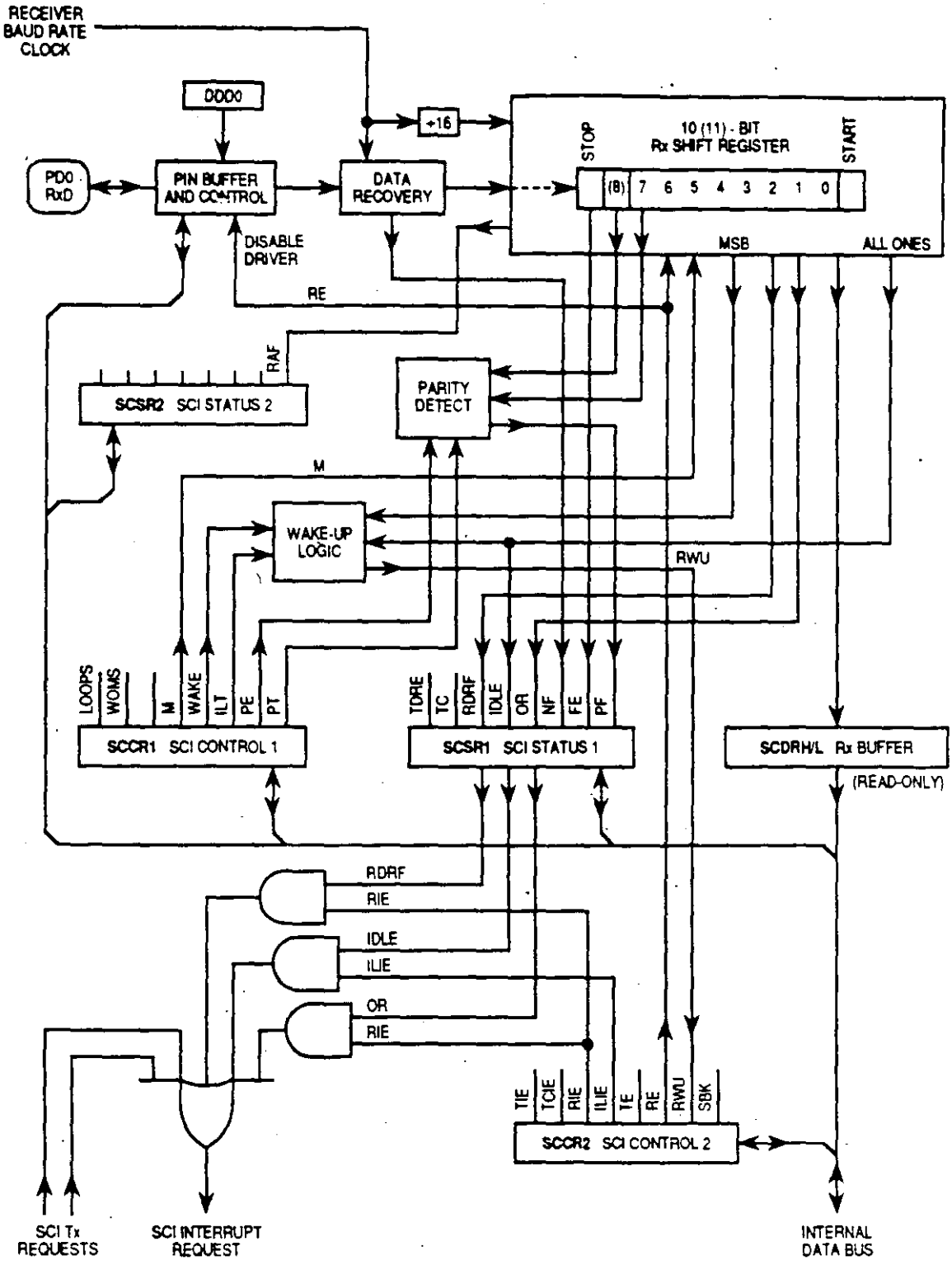
GSIZA–C [2:0]	Address Size
000	64K Bytes
001	32K Bytes
010	16K Bytes
011	8K Bytes
100	4K Bytes
101	2K Bytes
110	1K Bytes
111	0K Bytes

Serial Communications Interface (SCI)

The SCI, a universal asynchronous receiver transmitter (UART) serial communications interface, is one of two independent serial I/O subsystems in the MC68HC11F1. The SCI has a standard non-return to zero (NRZ) format (one start, eight or nine data, and one stop bit) and several selectable baud rates. The transmitter and receiver are independent, but use the same data format and bit rate.



SCI Transmitter Block Diagram



SCI Receiver Block Diagram

BAUD — Baud Rate **$102B**

	Bit 7	6	5	4	3	2	1	Bit 0
	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
RESET:	0	0	0	0	0	U	U	U

TCLR — Clear Baud Rate Counters (TEST)

Bit 6 – Not implemented; always reads zero

SCP1, SCP0 — SCI Baud Rate Prescaler Selects
Refer to the following tables and figures.

RCKB – SCI Baud-Rate Clock Check (Test)

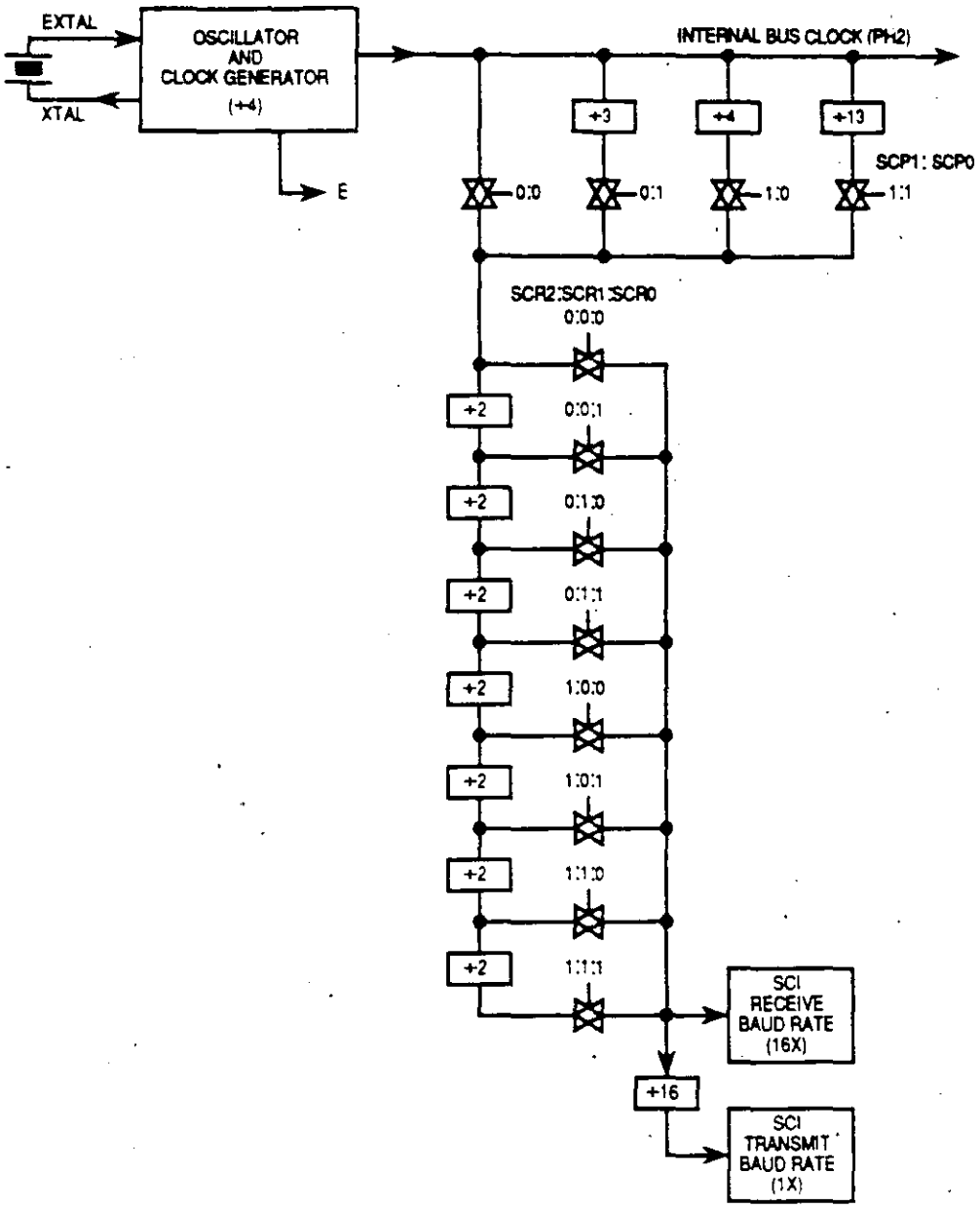
SCR2, SCR1, and SCR0 — SCI Baud Rate Selects
Selects receiver and transmitter baud rate. Refer to the illustration of the baud rate clock divider chain.

Baud Rate Prescaler Sets Highest Rate

SCP [1:0]	Divide Internal Clock By	Crystal Frequency in MHz			
1 0		4.0 MHz (Baud)	8.0 MHz (Baud)	10.0 MHz (Baud)	12.0 MHz (Baud)
0 0	1	62.50K	125.0K	156.25K	187.5K
0 1	3	20.83K	41.67K	52.08K	62.5K
1 0	4	15.625K	31.25K	38.4K	46.88K
1 1	13	4800	9600	12.02K	14.42K

Baud Rate Selection Table

SCR Bit [2:0]	Divide Prescaler By	Highest Baud Rate (Prescaler Output from Previous Table)		
		4800	9600	38.4K
0 0 0	1	4800	9600	38.4K
0 0 1	2	2400	4800	19.2K
0 1 0	4	1200	2400	9600
0 1 1	8	600	1200	4800
1 0 0	16	300	600	2400
1 0 1	32	150	300	1200
1 1 0	64	—	150	600
1 1 1	128	—	—	300



Baud Rate Generator Block Diagram

SCCR1 — SCI Control 1 **$102C**

	Bit 7	6	5	4	3	2	1	Bit 0
	R8	T8	0	M	WAKE	0	0	0
RESET:	U	U	0	0	0	0	0	0

R8 — Receive Data Bit 8

If M bit is set, R8 stores the ninth bit in the receive data character.

T8 — Transmit Data Bit 8

If M bit is set, T8 stores the ninth bit in the transmit data character.

Bits 5 and 2–0 — Not implemented; always read zero

M — Mode (Select Character Format)

0 = Start, 8 data bits, 1 stop bit
1 = Start, 9 data bits, 1 stop bit

WAKE — Wake Up by Address Mark/Idle

0 = Wake up by IDLE line recognition
1 = Wake up by address mark

SCCR2 — SCI Control 2 **$102D**

	Bit 7	6	5	4	3	2	1	Bit 0
	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:	0	0	0	0	0	0	0	0

TIE — Transmit Interrupt Enable

0 = TDRE interrupts disabled
1 = SCI interrupt requested when TDRE status flag is set

TCIE — Transmit Complete Interrupt Enable

0 = TC interrupts disabled
1 = SCI interrupt requested when TDRE status flag is set

RIE — Receiver Interrupt Enable

0 = RDRF and OR interrupts disabled
1 = SCI interrupt requested when RDRF flag or the OR status flag is set

ILIE — Idle Line Interrupt Enable

0 = IDLE interrupts disabled
1 = SCI interrupt requested when IDLE status flag is set

TE — Transmitter Enable

0 = Transmitter disabled
1 = Transmitter enabled

RE — Receiver Enable

0 = Receiver disabled
1 = Receiver enabled

RWU — Receiver Wake Up Control

0 = Normal SCI receiver
1 = Wake up enabled and receiver interrupt inhibited

SBK — Send Break

0 = Break generator off
1 = Break codes generated as long as SBK = 1

SCSR — SCI Status **$102E**

	Bit 7	6	5	4	3	2	1	Bit 0
	TDRE	TC	RDRF	IDLE	OR	NF	FE	0
RESET:	1	1	0	0	0	0	0	0

TDRE — Transmit Data Register Empty Flag

Set if transmit data can be written to SCDR; if TDRE = 0, transmit data register is busy. Cleared by SCSR read with TDRE set, followed by SCDR write.

TC — Transmit Complete Flag

Set if transmitter is idle — no data, preamble, or break transmission in progress. Cleared by SCSR read with TC set, followed by SCDR write.

RDRF — Receive Data Register Full Flag

Set if a received character is ready to be read from SCDR. Cleared by SCSR read with RDRF set, followed by SCDR read

IDLE — Idle Line Detected Flag

Set if the RxD line is idle. Idle flag is inhibited when RWU = 1. Cleared by SCSR read with IDLE set, followed by SCDR read. Once cleared, IDLE is not set again until the RxD line has been active and becomes idle again.

OR — Overrun Error Flag

Set if a new character is received before a previously received character is read from SCDR. Cleared by SCSR read with OR set, followed by SCDR read.

NF — Noise Error Flag

Set if majority sample logic detects anything other than a unanimous decision. Cleared by SCSR read with NF set, followed by SCDR read.

FE — Framing Error

Set if a 0 is detected where a stop bit was expected. Cleared by SCSR read with FE set, followed by SCDR read.

Bit 0 — Not implemented; always reads zero

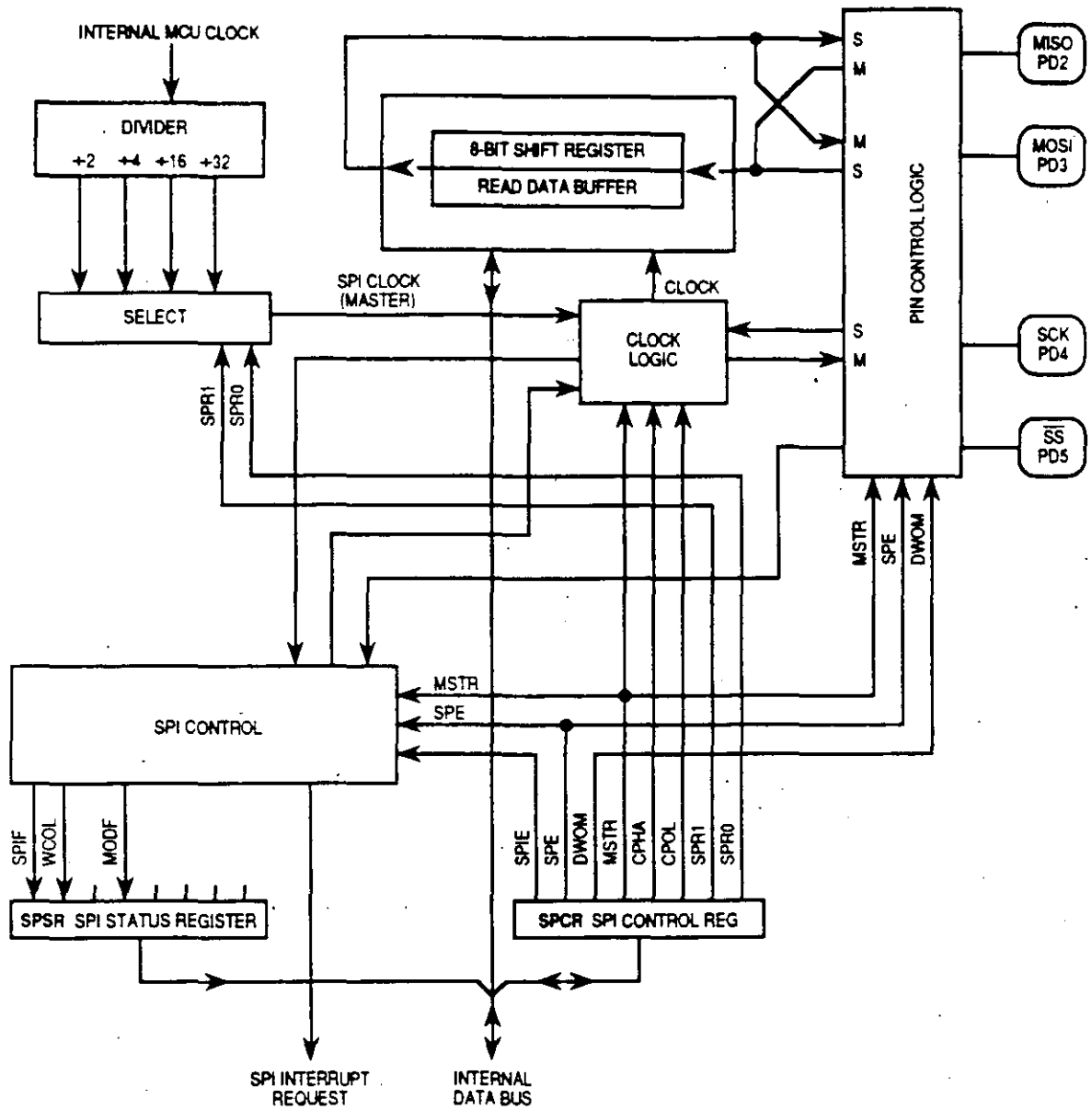
SCDR — Serial Communications Data Register **$102F**

	Bit 7	6	5	4	3	2	1	Bit 0
	Bit 7	6	5	4	3	2	2	Bit 0
RESET:	U	U	U	U	U	U	U	U

Reads access buffered receive data. Writes access transmit data buffer register. Receive and transmit are double buffered.

Serial Peripheral Interface (SPI)

The SPI allows the MCU to communicate synchronously with peripheral devices and other microprocessors. The SPI protocol facilitates rapid exchange of serial data between devices in a control system. Each SPI-compatible component in a system can be set up for master or slave operation. Data rates can be as high as one half of the E-clock rate when configured as master, and as fast as the E-Clock when configured as slave.



SPI Block Diagram

SPCR — Serial Peripheral Control **$1028**

	Bit 7	6	5	4	3	2	1	Bit 0
	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0
RESET:	0	0	0	0	0	1	U	U

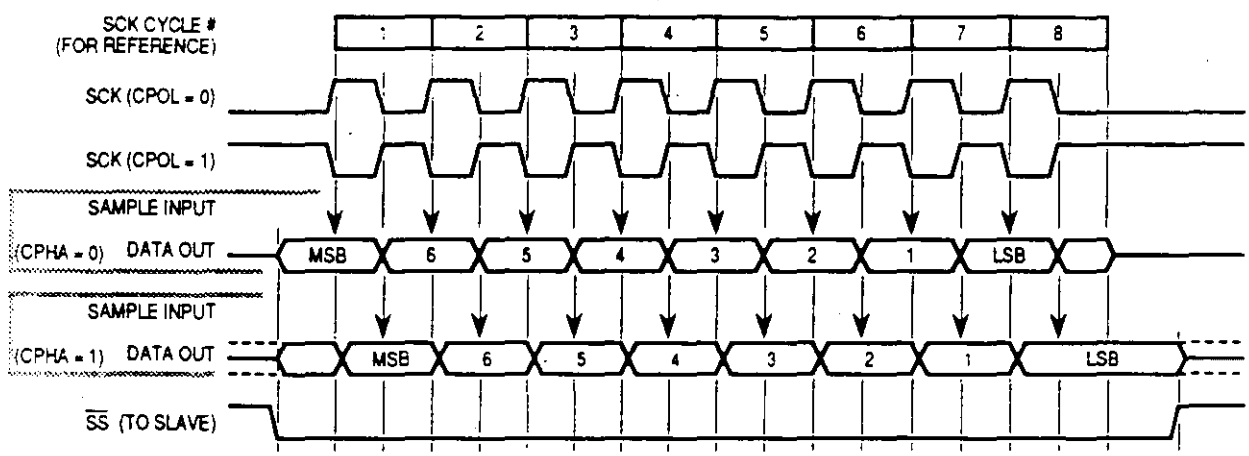
SPIE — Serial Peripheral Interrupt Enable
- 0 = SPI interrupt disabled
- 1 = SPI interrupt enabled

SPE — Serial Peripheral System Enable
- 0 = SPI off
- 1 = SPI on

DWOM — Port D Wired-OR Mode Option for SPI Pins PD5–PD2
- 0 = Normal CMOS outputs
- 1 = Open-drain outputs

MSTR — Master Mode Select
- 0 = Slave mode
- 1 = Master mode

CPOL, CPHA — Clock Phase, Clock Polarity (Refer to following figure, **SPI Transfer Format**.)



NOTE: This figure shows the LSBF = 0 default case. If LSBF = 1, data is transferred in reverse order (LSB first).

SPR1 and SPR0 — SPI Clock Rate Selects

SPI Clock Rate Selects

SPR [1:0]	Divide E Clock By	Frequency at E = 2 MHz (Baud)	Frequency at E = 3 MHz (Baud)	Frequency at E = 4 MHz (Baud)
0 0	2	1 MHz	1.5 MHz	2.0 MHz
0 1	4	500 kHz	750 kHz	1.0 kHz
1 0	16	125 kHz	187.5 kHz	250 kHz
1 1	32	62.5 kHz	93.8 kHz	125 kHz

SPSR — Serial Peripheral Status **$1029**

	Bit 7	6	5	4	3	2	1	Bit 0
	SPIF	WCOL	0	MODF	0	0	0	0
RESET:	0	0	0	0	0	0	0	0

SPIF — SPI Transfer Complete Flag

Set when an SPI transfer is complete. Cleared by reading SPSR with SPIF set, followed by SPDR access.

WCOL — Write Collision

Set when SPDR is written while transfer is in progress. Cleared by SPSR with WCOL set, followed by SPDR access.

Bits 5 and 3–0 — Not Implemented; always read zero

MODF — Mode Fault

A mode fault terminates SPI operation. Set when $\overline{SS}$ is pulled low while MSTR = 1. Cleared by SPSR read with MODF set, followed by SPCR write.

SPDR — SPI Data **$102A**

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

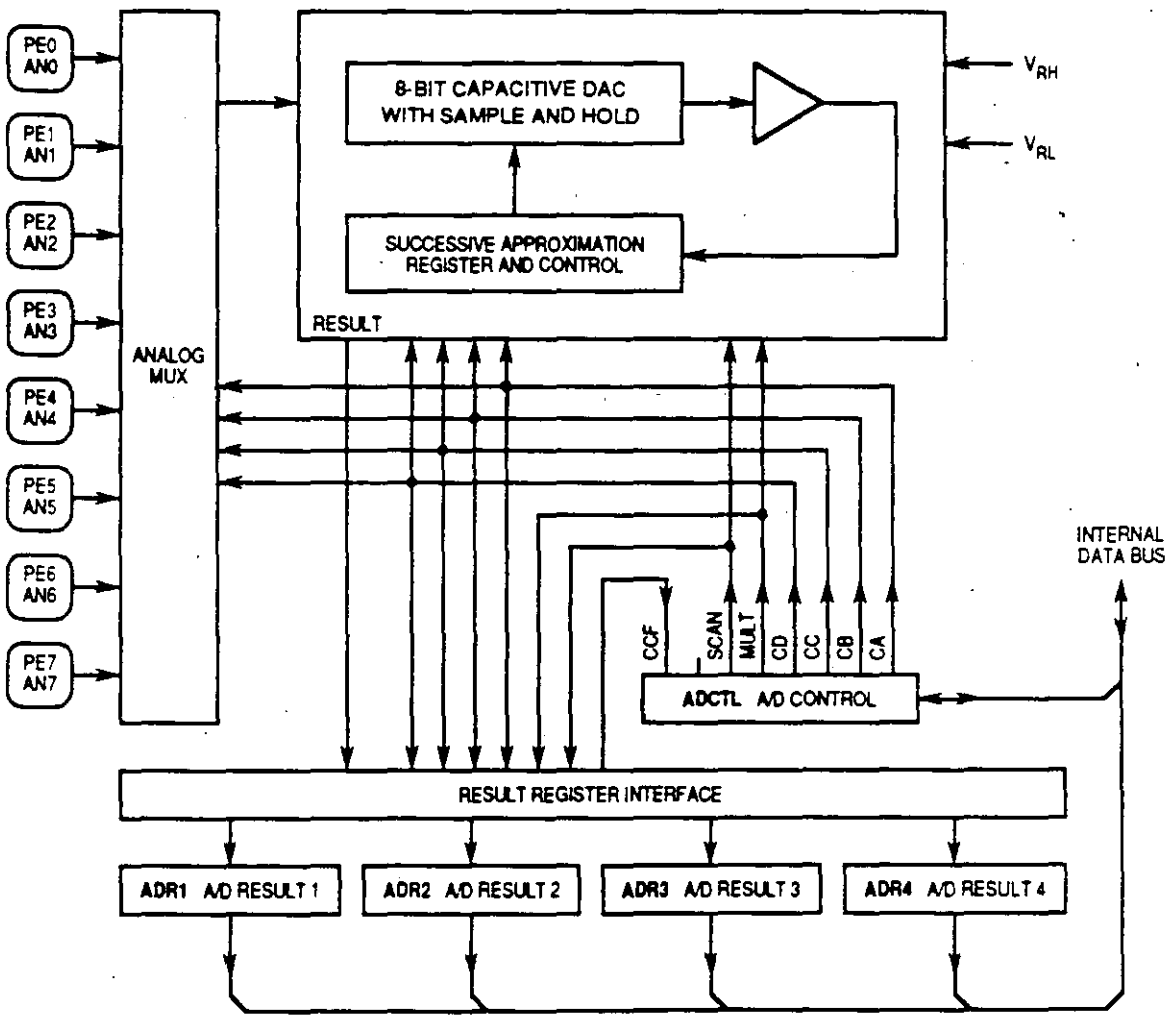
SPI is double buffered in, single buffered out.

Analog-to-Digital Converter

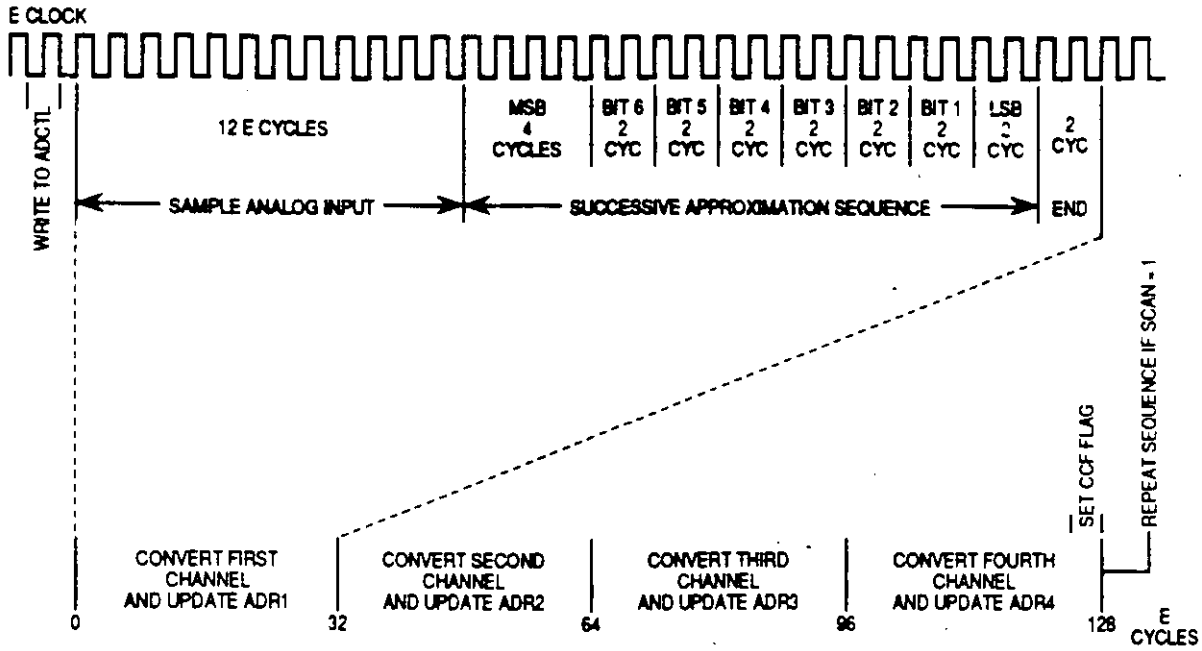
The MC68HC11F1 analog-to-digital (A/D) converter system uses an all-capacitive charge-redistribution technique to convert analog signals to digital values. The A/D system is an 8-channel, 8-bit, multiplexed-input, successive-approximation converter, accurate to ±1 least significant bit (LSB). Because the capacitive charge redistribution technique used includes a built-in sample-and-hold, no external sample-and-hold is required.

Dedicated lines V_{RH} and V_{RL} provide the reference supply voltage inputs. Systems operating at clock rates of 750 kHz or below must use an internal RC oscillator. Refer to the CSEL bit in the OPTION register.

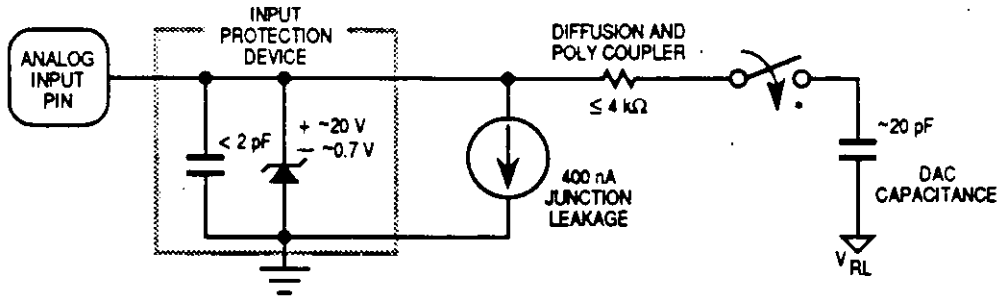
A multiplexer allows the single A/D converter to select one of 16 analog signals, as shown in the table of A/D converter channel assignments.



A/D Converter Block Diagram



A/D Conversion Sequence



* This analog switch is closed only during the 12-cycle sample time.

Electrical Model of an Analog Input Pin (Sample Mode)

ADCTL — A/D Control/Status **$1030**

	Bit 7	6	5	4	3	2	1	Bit 0
	CCF	0	SCAN	MULT	CD	CC	CB	CA
RESET:	U	0	U	U	U	U	U	U

CCF — Conversions Complete Flag
Set after the fourth A/D conversion in a conversion sequence. Cleared when ADCTL is written.

Bit 6 – Not implemented; always reads zero

SCAN — Continuous Scan Control
0 = Do four conversions and stop
1 = Convert four channels in selected group continuously

MULT — Multiple Channel/Single Channel Control
0 = Convert single channel selected
1 = Convert four channels in selected group

CD–CA — Channel Select D through A

A/D Converter Channel Assignments

Channel Select Control Bits				Channel	Result in ADRx if
CD	CC	CB	CA	Signal	MULT = 1
0	0	0	0	AN0	ADR1
0	0	0	1	AN1	ADR2
0	0	1	0	AN2	ADR3
0	0	1	1	AN3	ADR4
0	1	0	0	AN4	ADR1
0	1	0	1	AN5	ADR2
0	1	1	0	AN6	ADR3
0	1	1	1	AN7	ADR4
1	0	X	X	Reserved	ADR1–ADR4
1	1	0	0	V_{RH}*	ADR1
1	1	0	1	V_{RL}*	ADR2
1	1	1	0	$(V_{RH})/2$*	ADR3
1	1	1	1	Reserved*	ADR4

*Used for factory testing

ADR1–ADR4 — A/D Results **$1031–$1034**

	Bit 7	6	5	4	3	2	1	Bit 0	
$1031	Bit 7	6	5	4	3	2	1	Bit 0	ADR1
$1032	Bit 7	6	5	4	3	2	1	Bit 0	ADR2
$1033	Bit 7	6	5	4	3	2	1	Bit 0	ADR3
$1034	Bit 7	6	5	4	3	2	1	Bit 0	ADR4

Analog Input to 8-Bit Result Translation Table

	Bit 7	6	5	4	3	2	1	Bit 0
% (1)	50%	25%	12.5%	6.25%	3.12%	1.56%	0.78%	0.39%
Volts (2)	2.500	1.250	0.625	0.3125	0.1562	0.0781	0.0391	0.0195

(1) % of V_{RH}–V_{RL} (2) Volts for V_{RL} = 0; V_{RH} = 5.0 V

OPTION — System Configuration Options **$1039**

	Bit 7	6	5	4	3	2	1	Bit 0
	ADPU	CSEL	IRQE*	DLY*	CME*	0*	CR1*	CR0*
RESET:	0	0	0	1	0	0	0	0

Can be written only once in first 64 cycles out of reset in normal modes, or at any time in special modes

ADPU — A/D Power Up
- 0 = A/D powered down
- 1 = A/D powered up

CSEL — Clock Select
- 0 = A/D and EEPROM use system E-Clock
- 1 = A/D and EEPROM use internal RC clock

Bits 5–0 — Refer to **Resets and Interrupts** and **Electrically Erasable Programmable ROM.**

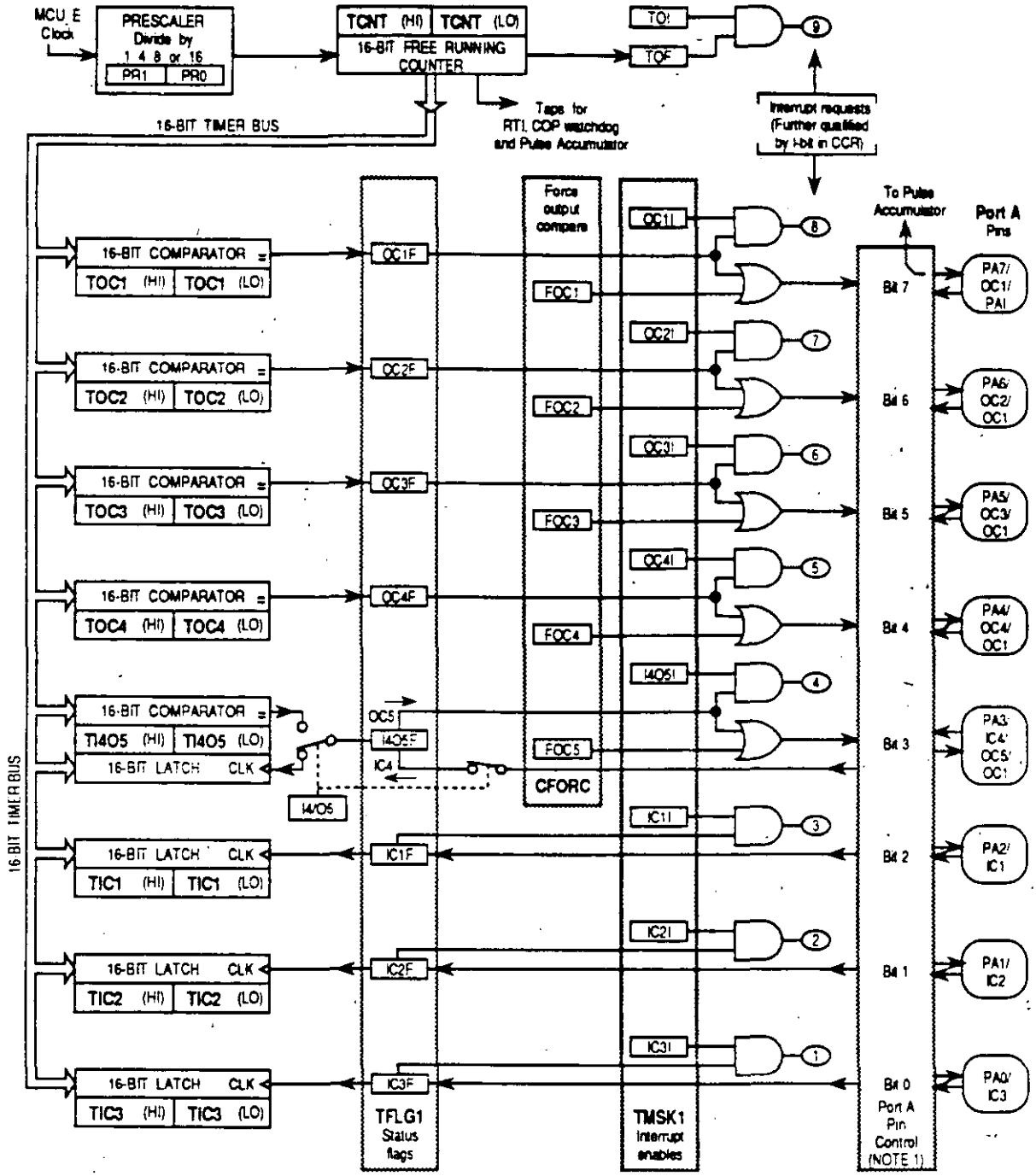
Main Timer

The main timer is based on a free-running 16-bit counter with a four-stage programmable prescaler. The timer shares PORTA. It has three channels of input capture and four channels of output compare, in addition to a channel that can be configured as either an input capture or an output compare (I4/O5). The real-time interrupt circuit is based on the main timer's 16-bit counter.

Refer to the the following table for a summary of crystal-related frequencies and periods.

Timer Summary

Control Bits	XTAL Frequencies			
	4.0 MHz	8.0 MHz	12.0 MHz	Other Rates
	1.0 MHz	2.0 MHz	3.0 MHz	(E)
	1000 ns	500 ns	333 ns	(1/E)
PR [1:0]	Main Timer Count Rates			
0 0 1 count overflow	1.0 µs 65.536 ms	500 ns 32.768 ms	333 ns 21.845 ms	(E/1) ($E/2^{15}$)
0 1 1 count overflow	4.0 µs 262.14 ms	2.0 µs 131.07 ms	1.333 µs 87.381 ms	(E/4) ($E/2^{18}$)
1 0 1 count overflow	8.0 µs 524.29 ms	4.0 µs 262.14 ms	2.667 µs 174.76 ms	(E/8) ($E/2^{19}$)
1 1 1 count overflow	16.0 µs 1.049 s	8.0 µs 524.29 ms	5.333 µs 349.52 ms	(E/16) ($E/2^{20}$)
RTR [1:0]	Periodic (RTI) Interrupt Rates			
0 0	8.192 ms	4.096 ms	2.731 ms	($E/2^{13}$)
0 1	16.384 ms	8.192 ms	5.461 ms	($E/2^{14}$)
1 0	32.768 ms	16.384 ms	10.923 ms	($E/2^{15}$)
1 1	65.536 ms	32.768 ms	21.845 ms	($E/2^{16}$)
CR [1:0]	COP Watchdog Timeout Rates			
0 0	32.768 ms	16.384 ms	10.923 ms	($E/2^{15}$)
0 1	131.07 ms	65.536 ms	43.691 ms	($E/2^{17}$)
1 0	524.29 ms	262.14 ms	174.76 ms	($E/2^{19}$)
1 1	2.097 s	1.049 s	699.05 ms	($E/2^{21}$)
Timeout Tolerance (–0 ms/+...)	32.768 ms	16.4 ms	10.9 ms	($E/2^{15}$)



NOTE: Port A pin actions controlled by DDRA, OC1M, OC1D, PACTL, TCTL1, and TCTL2 registers

Main Timer

CFORC — Timer Compare Force **$100B**

	Bit 7	6	5	4	3	2	1	Bit 0
	FOC1	FOC2	FOC3	FOC4	FOC5	0	0	0
RESET:	0	0	0	0	0	0	0	0

FOC5–FOC1 — Force Output Compare x Action

0 = Not affected

1 = Output compare x action occurs, but OCxF flag bit is not set

Bits 2 – 0 — Not implemented; always read zero

OC1M — Output Compare 1 Mask **$100C**

	Bit 7	6	5	4	3	2	1	Bit 0
	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0
RESET:	0	0	0	0	0	0	0	0

Set bit(s) to enable OC1 to control corresponding pin(s) of port A.

OC1M7–OC1M3 — Output Compare Masks

0 = OC1 is disabled

1 = OC1 is enabled to control the corresponding pin of port A

Bits 2–0—Not implemented; always read zero

OC1D — Output Compare 1 Data **$100D**

	Bit 7	6	5	4	3	2	1	Bit 0
	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0
RESET:	0	0	0	0	0	0	0	0

If OC1Mx is set, data in OC1Dx is output to port A bit x on successful OC1 compares.

Bits 2–0 — Not implemented; always read zero

TCNT — Timer Count **$100E, $100F**

$100E	Bit 15	14	13	12	11	10	9	Bit 8	High	TCNT
$100F	Bit 7	6	5	4	3	2	1	Bit 0	Low	
RESET:	0	0	0	0	0	0	0	0		

In normal modes, TCNT is read-only.

TIC1–TIC3 — Timer Input Capture **$1010–$1015**

$1010	Bit 15	14	13	12	11	10	9	Bit 8	High	TIC1
$1011	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$1012	Bit 15	14	13	12	11	10	9	Bit 8	High	TIC2
$1013	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$1014	Bit 15	14	13	12	11	10	9	Bit 8	High	TIC3
$1015	Bit 7	6	5	4	3	2	1	Bit 0	Low	

TICx not affected by reset.

TOC1–TOC4 — Timer Output Compare **$1016–$101D**

$1016	Bit 15	14	13	12	11	10	9	Bit 8	High	TOC1
$1017	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$1018	Bit 15	14	13	12	11	10	9	Bit 8	High	TOC2
$1019	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$101A	Bit 15	14	13	12	11	10	9	Bit 8	High	TOC3
$101B	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$101C	Bit 15	14	13	12	11	10	9	Bit 8	High	TOC4
$101D	Bit 7	6	5	4	3	2	1	Bit 0	Low	
$101E	Bit 15	14	13	12	11	10	9	Bit 8	High	TOC5
$1001F	Bit 7	6	5	4	3	2	1	Bit 0	Low	
RESET:	1	1	1	1	1	1	1	1		

All TOCx register pairs reset to ones ($FFFF).

TI4O5 — Timer Input Capture 4/Output Compare 5 **$101E, $101F**

$101E	Bit 15	14	13	12	11	10	9	Bit 8	High TI4O5
$101F	Bit 7	6	5	4	3	2	1	Bit 0	Low

All TI4O5 register pairs reset to ones ($FFFF).

TCTL1 — Timer Control 1 **$1020**

	Bit 7	6	5	4	3	2	1	Bit 0
	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5
RESET:	0	0	0	0	0	0	0	0

OM2–OM5 — Output Mode

OL2–OL5 — Output Level

OMx	OLx	Action Taken on Successful Compare
0	0	Timer disconnected from output pin logic
0	1	Toggle OCx output line
1	0	Clear OCx output line to 0
1	1	Set OCx output line to 1

TCTL2 — Timer Control 2 **$1021**

	Bit 7	6	5	4	3	2	1	Bit 0
	EDG4B	EDG4A	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A
RESET:	0	0	0	0	0	0	0	0

Timer Control Configuration

EDGxB	EDGxA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge

TMSK1 — Timer Interrupt Mask 1 **$1022**

	Bit 7	6	5	4	3	2	1	Bit 0
	OC1I	OC2I	OC3I	OC4I	I4O5I	IC1I	IC2I	IC3I
RESET:	0	0	0	0	0	0	0	0

OC1I–OC4I — Output Compare x Interrupt Enable

I4O5I — Input Capture 4 or Output Compare 5 Interrupt Enable

IC1I–IC3I — Input Capture x Interrupt Enable

NOTE

Bits in TMSK1 correspond bit for bit with flag bits. Ones in TMSK1 enable the corresponding interrupt sources.

TFLG1 — Timer Interrupt Flag 1 **$1023**

	Bit 7	6	5	4	3	2	1	Bit 0
	OC1F	OC2F	OC3F	OC4F	I4O5F	IC1F	IC2F	IC3F
RESET:	0	0	0	0	0	0	0	0

Cleared by writing a one to the corresponding bit position(s).

OC1F–OC5F — Output Compare x Flag
Set each time the counter matches output compare x value.

I4O5F — Input Capture 4/Output Compare 5 Flag
Set by IC4 or OC5, depending on which function was enabled by I4O5 of PACTL.

IC1F–IC3F — Input Capture x Flag
Set each time a selected active edge is detected on the ICx input line.

TMSK2 — Timer Interrupt Mask 2 **$1024**

	Bit 7	6	5	4	3	2	1	Bit 0
	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0
RESET:	0	0	0	0	0	0	0	0

TOI — Timer Overflow Interrupt Enable

RTII — Real-time Interrupt Enable

PAOVI — Pulse Accumulator Overflow Interrupt Enable

PAII — Pulse Accumulator Interrupt Enable

Bits in TMSK2 correspond bit for bit with flag bits in TFLG2. Ones in TMSK2 enable the corresponding interrupt sources.

Bits 3–2 — Not implemented; always read zero

PR1 and PR0 — Timer Prescaler Select

In normal modes, PR1 and PR0 can only be written once, and the write must be within 64 cycles after reset. Refer to table, **Timer Summary**, for specific timing values.

PR [1:0]	Prescaler
0 0	1
0 1	4
1 0	8
1 1	16

TFLG2 — Timer Interrupt Flag 2 **$1025**

	Bit 7	6	5	4	3	2	1	Bit 0
	TOF	RTIF	PAOVF	PAIF	0	0	0	0
RESET:	0	0	0	0	0	0	0	0

Cleared by writing a one to the corresponding bit position(s).

TOF — Timer Overflow Flag
Set when TCNT changes from $FFFF to $0000.

RTIF — Real-Time (Periodic) Interrupt Flag
Set periodically (Refer to RTR1:0 bits in PACTL register).

Bits 5–4 — Refer to **Pulse Accumulator.**

Bits 3–0 — Not implemented; always read zero

PACTL — Pulse Accumulator Control **$1026**

	Bit 7	6	5	4	3	2	1	Bit 0
	0	PAEN	PAMOD	PEDGE	0	I4/O5	RTR1	RTR0
RESET:	0	0	0	0	0	0	0	0

Bits 7 and 3 — Not implemented; always read zero

Bits 6–4 — Refer to **Pulse Accumulator.**

I4/O5 — Configure TI4O5 Register for IC or OC
0 = OC5 function enabled
1 = IC4 function enabled

RTR1–RTR0 — RTI Interrupt Rate Selects
These two bits select one of four rates for the real-time periodic interrupt circuit. Refer to the table of real-time interrupt rates for additional detail.

Real-Time Interrupt Rates

RTR [1:0]	Divide E By	XTAL = 4.0 MHz	XTAL = 8.0 MHz	XTAL = 12.0 MHz
0 0	2^{13}	8.19 ms	4.096 ms	2.731 ms
0 1	2^{14}	16.38 ms	8.192 ms	5.461 ms
1 0	2^{15}	32.77 ms	16.384 ms	10.923 ms
1 1	2^{16}	65.54 ms	32.768 ms	21.845 ms
	E =	1.0 MHz	2.0 MHz	3.0 MHz

PACNT — Pulse Accumulator Count **$1027**

	Bit 7	6	5	4	3	2	1	Bit 0
	Bit 7	6	5	4	3	2	1	Bit 0
RESET:	U	U	U	U	U	U	U	U

Readable and writable.

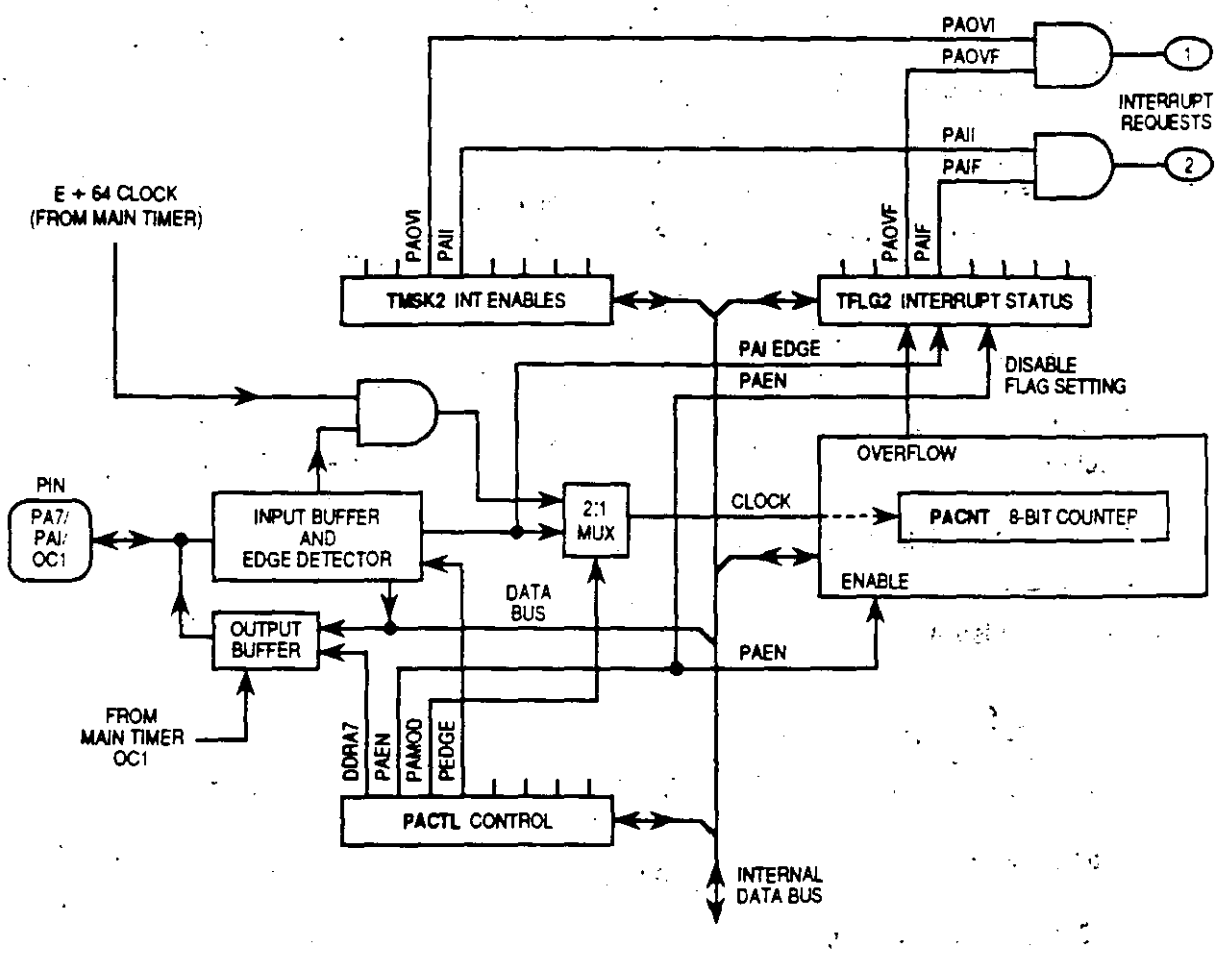
Pulse Accumulator

The MC68HC11F1 has an 8-bit counter that can be configured to operate as a simple event counter or as a gated time accumulator, depending on the PAMOD bit in the PACTL register.

The port A bit-7 I/O pin acts as a clock input in event counting mode, or as a gate signal to enable a free-running clock (E divided by 64) in gated time accumulation mode. While the external PAI input pin is activated, the free-running clock increments the 8-bit counter in gated time accumulation mode. An overflow status flag and an interrupt are available to extend the range of the 8-bit counter.

The maximum clocking rate for the external event counting mode is the E-Clock divided by two.

The pulse accumulator counter can be read or written at any time.



Pulse Accumulator Block Diagram

	Selected Crystal	Common XTAL Frequencies		
		8.0 MHz	12.0 MHz	16.0 MHz
CPU Clock	(E)	2.0 MHz	3.0 MHz	4.0 MHz
Cycle Time	(1/E)	500 ns	333 ns	250 ns
Pulse Accumulator (in Gated Mode)				
$(E/2^6)$ $(E/2^{14})$	1 count overflow	32.0 µs 8.192 ms	21.33 µs 5.461 ms	16.0 µs 4.096 ms

TMSK2 — Timer Interrupt Mask 2 **$1024**

	Bit 7	6	5	4	3	2	1	Bit 0
	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0
RESET:	0	0	0	0	0	0	0	0

Bits 7–6 and 1–0 — Refer to **Main Timer**.

PAOVI — Pulse Accumulator Overflow Interrupt Enable
0 = Pulse accumulator overflow interrupt disabled
1 = Interrupt requested when bit PAOVF of TFLG2 is set

PAII — Pulse Accumulator Interrupt Enable
0 = Pulse accumulator interrupt disabled
1 = Interrupt requested when bit PAIF of TFLG2 is set

Bits in TMSK2 correspond bit for bit with flag bits. Ones in TMSK2 enable the corresponding interrupt sources.

Bits 3–2 — Not implemented; always read zero

TFLG2 — Timer Interrupt Flag 2 **$1025**

	Bit 7	6	5	4	3	2	1	Bit 0
	TOF	RTIF	PAOVF	PAIF	0	0	0	0
RESET:	0	0	0	0	0	0	0	0

Cleared by writing a one to the corresponding bit position(s).

Bits 7–6 — Refer to **Main Timer.**

PAOVF — Pulse Accumulator Overflow Flag
Set when PACNT changes from $FF to $00.

PAIF — Pulse Accumulator Input Edge Flag
Set each time a selected active edge is detected on the PAI input line.

Bits 3–0 — Not implemented; always read zero

PACTL — Pulse Accumulator Control **$1026**

	Bit 7	6	5	4	3	2	1	Bit 0
	0	PAEN	PAMOD	PEDGE	0	I4/05	RTR1	RTR0
RESET:	0	0	0	0	0	0	0	0

Bits 7 and 3 — Not implemented; always read zero

PAEN — Pulse Accumulator System Enable
- 0 = Pulse Accumulator disabled
- 1 = Pulse Accumulator enabled

PAMOD — Pulse Accumulator Mode
- 0 = Event counter
- 1 = Gated time accumulation

PEDGE — Pulse Accumulator Edge Control
- 0 = Falling edges increment counter: high level enables accumulation
- 1 = Rising edges increment counter: low level enables accumulation

Bits 2–0 — Refer to **Main Timer.**

PACNT — Pulse Accumulator Count **$1027**

	Bit 7	6	5	4	3	2	1	Bit 0
	Bit 7	6	5	4	3	2	1	Bit 0
RESET:	U	U	U	U	U	U	U	U

Readable and writable.

DIRECTORIO DE ALUMNOS DEL CURSO
LOS MICROPROCESADORES Y SUS APLICACIONES
DEL 14 DE AGOSTO AL 12 DE SEPTIEMBRE DE 1992.

1.- CARREON MENDEZ ALFREDO OCTAVIO
ASTURIAS 245-6, COL. ALAMOS, DELEG. B. JUAREZ, C.P.03400
TEL. 579 37 27 DOM.

2.- DAZA PIMENTEL JUAN VICENTE
JEFE DE MANTENIMIENTO
INDUSTRIAS VINICOLAS PEDRO DOMECQ S.A. DE C.V.
KM 17.5 CARR. FED. MEXICO PUEBLA, LOS REYES LA PAZ,
EDO. MEX. C.P. 56400, TEL. 326 52 52 OFNA.

3.- DOLORES ANGUIANO RAMSES
OCOTEPEC 214, COL. MARAVILLAS, CUERNAVACA 91-73
TEL. 13 74 87 DOM.

4.- FLORES MARROQUIN JUAN MANUEL
ING. DE CALIDAD DE CAMPO
ELEVADORES OTIS, S.A. DE C.V.
ABEDULES No. 75, COL. STA. MA. INSURGENTES
DELEG. CUAUHTEMOC, C.P. 06430, TEL. 326 53 00 OFNA.

5.- GARCIA GARCIA ENRIQUE GABINO
ACADEMICO
UNAM
TEL. 91 595 42 056 DOM.

6.- GONZALEZ PRADO JAVIER
EMPLEADO
COMISION FEDERAL DE ELECTRICIDAD
AUGUSTO RODIN 265, COL. NOCHE BUENA, DELEG. B. JUAREZ
C.P. 03820, TEL. 563 37 00 EXT. 246 OFNA.

7.- GONZALEZ RAMIREZ PATRICIA
JEFE DE LABORATORIO
MICROTECNOLOGIA DE MEXICO, S.A. DE C.V.
RIO PANUCO No. 55, COL. CUAUHTEMOC, DELEG. CUAUHTEMOC
TEL. 703 32 44 OFNA.

8.- HERNANDEZ FLORES ENRIQUE
AV. MORELOS 827-EDIF. 3-104, COL. JARDIN BALBUENA,
DELEG. V. CARRANZA, C.P. 15900, TEL. 552 08 94 DOM.

9.- LEON OROZCO VICENTE
CATEDRATICO
INST. TECNOLOGICO DE TUXTLA GUTIERREZ
CARRETERA PANAMERICANA KM 1079, TUXTLA GUTIERREZ CHIAPAS
TEL. 208 86 DOM.

10.- MARIN JOYA RICARDO
GERENTE GENERAL
INSTRUQUIM, S.A. DE C.V.
VALLEJO 1722, COL. STA. ROSA, DELEG. G. A. MADERO
C.P. 07620, TEL. 391 54 47 OFNA.

11.- RAMIREZ MANDUJANO NORBERTO
POTRERO 13, COL. COLINA DEL SUR, DELEG. A.OBREGON
C.P. 01430, TEL. 651 08 93 DOM.

12.- RAMIREZ ZEPEDA MARTIN
TECNICO
MICROTECNOLOGIA DE MEXICO, S.A. DE C.V.
RIO PANUCO No. 55, COL. CUAUHTEMOC, DELEG. CUAUHTEMOC
TEL. 703 32 44 OFNA.

13.- RIVERA HUERTA GUILLERMO
INGENIERO "C"
COMISION FEDERAL DE ELECTRICIDAD
AUGUSTO RODIN No. 265, COL. NOCHEBUENA
TEL. 563 37 00 OFNA.

14.- SANCHEZ HERNANDEZ VALENTIN
JEFE DE INGENIERIA
INMER
CARDENALES 63, COL. LOS ANGELES, DELEG. A. OBREGON
TEL. 660 19 43 OFNA.

15.- URIBE GARCIA NICOLAS ENRIQUE
INGENIERO EN ELECTRONICA (EMPLEADO)
COMISION FEDERAL DE ELECTRICIDAD
AUGUSTO RODIN No. 265, COL. NOCHEBUENA, DELEG. B. JUAREZ
C.P. 03820, TEL. 563 37 00 EXT. 246 OFNA.