DIRECTORIO DE PROFESORES
MICROPROCESADORES: TEORIA Y APLICACIONES

ING. JORGE GIL MENDIETA
JEFE DEL DEPARTAMENTO DE DISEÑO
DE SISTEMAS DIGITALES
I I M A S, UNAM
CIUDAD UNIVERSITARIA
MEXICO 20, D.F.
TEL: 548.33.60

M. EN C. ALEJANDRO GUARDA AURAS
INVESTIGADOR
DEPARTAMENTO DE DISEÑO DE SISTEMAS DIGITALES
I I M A S, UNAM
CIUDAD UNIVERSITARIA
MEXICO 20, D.F.
TEL: 550.52.15 ext. 4573

M. EN C. ANGEL KURI MORALES
INVESTIGADOR
INSTITUTO DE INVESTIGACIONES
MATEMATICAS APLICADAS Y
EN SISTEMAS, UNAM
MEXICO 20, D.F.
TEL: 550. 52.15 ext. 4580
548.33.60

ING. MARIO RODRIGUEZ MANZANERA
INVESTIGADOR
I I M A S, UNAM
MEXICO 20, D.F.
TEL: 550.52.15 ext. 4581 ó 4582

'pmc.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES: TEORIA Y APLICACIONES

TEMA 2: INTRODUCCION A LA ARQUITECTURA DE LOS MICROPROCESADORES

ING. JORGE GIL MENDIETA

ABRIL, 1978.

Palacio de Minería Calle de Tacuba 5, primer piso. México 1, D. F.

2. ARQUITECTURA BASICA DE UNA MICROCOMPUTADORA.

Un microprocesador es una unidad central de proceso en un solo "chip" de tecnología LSI. Como se ilustra en la figura 2.1 una microcomputadora es un

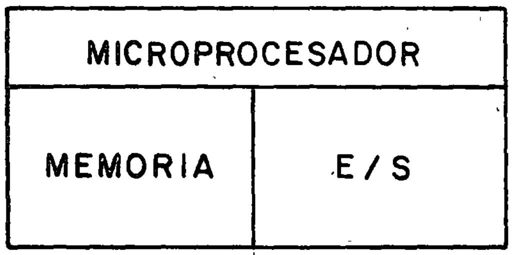


Fig. 2.1. Representación conceptual de una microcomputadora.

microprocesador combinado con memoria y entrada/salida. Hay casos en que un procesador se construye en varias componentes LSI. El aspecto crucial de las definiciones es que un microprocesador es solamente una unidad central de proceso, en tanto que una microcomputadora incluye memoria y entrada/salida.

2.1. Arquitectura de las microcomputadoras

La arquitectura de las microcomputadoras representa un compromiso difícil entre los recursos de la tecnología y las necesidades de los usuarios. A la fecha, la preocupación que domina el diseño ha sido el construir procesadores fáciles de usar. Las limitaciones sobre complejidad, velocidad y empaquetamiento (número de patas) han definido los tipos fundamentales de procesadores que pueden construirse. Existe una variedad de procesadores diseñados primariamente para construir calculadoras. Por lo menos uno de ellos (4004) ha sido usado en el área de controladores, algunos han sido construídos con capacidad para realizar operaciones aritméticas. Una tercera generación de procesadores es el procesador de 8 bits; por ejemplo el 8008 y el 8080 de INTEL y el M6800 de Motorola son ejemplos de esta clase.

Existen procesadores con longitud de palabra mayor y la clase actual que tiene en un solo estrato procesador, memoria ROM y RAM y capacidad de entrada/salida.

Hay una clase de procesador por bit que para formar palabras usan varios chips, se pueden formar procesadores con longitud de palabra de 16 bits o mayores. Una de las aplicaciones primarias de esos procesadores es la emulación de minicomputadoras.

2.1.1. Características generales de las microcomputadoras.

- 'Software' que requiere de poco apoyo
- Documentación limitada
- Recursos pequeños de 'Hardware'
- La arquitectura del procesador influye fuertemente al programa y al diseño de sistema
- Mecanismos limitados para responder a eventos asíncronos o dependientes del tiempo.
- Las interfases son programadas más que puro hardware
- La memoria de datos debe manejarla el programador como un dispositivo de entrada/salida
- La baja velocidad de ejecución impone restricciones de algunas aplicaciones.
- El pequeño tamaño de las palabras hace muy costosa la aplicación de operaciones con datos con alta precisión.

En la figura 2.2 observamos un diagrama a bloques de una microcomputadora en donde se especifican sus características básicas.

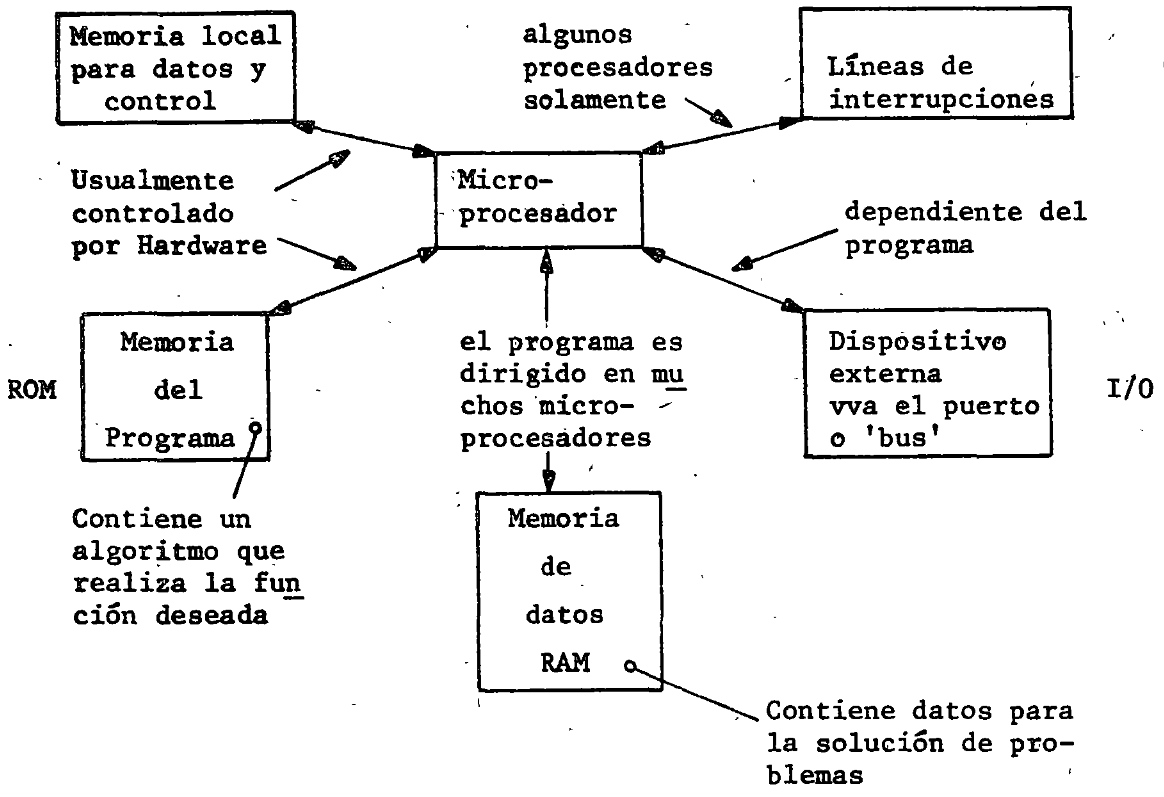

Memoria local para datos y control
algunos procesadores solamente
Líneas de interrupciones
Usualmente controlado por Hardware
Micro-procesador
dependiente del programa
ROM
Memoria del Programa
el programa es dirigido en muchos micro-procesadores
Dispositivo externa vva el puerto o 'bus'
I/O
Contiene un algoritmo que realiza la función deseada
Memoria de datos RAM
Contiene datos para la solución de problemas

Fig. 2.2 Características básicas de una microcomputadora.

En la figura 2.2 presentamos el diagrama a bloques de una microcomputadora típica, en donde se indican las secciones de que comprende al sistema.

La memoria de solo lectura (ROM) se usa para almacenar microprogramas o un programa fijo dependiendo de la microprogramabilidad del CPU. El microprograma proporciona la traducción de los comandos de alto nivel del usuario, tales como ADD, SUBTR, etc., a una serie de códigos de control reconocibles por el microprocesador para ejecución. El tamaño del ROM varía de acuerdo a los requerimientos del usuario dentro de la máxima capacidad permisible por el direccionamiento del microprocesador. En la tabla 2.1 podemos ver algunos de los microprocesadores más comunes.

	Intel 4004	Intel 8008	Intel 8080	AMI 7300
Tamaño de palabra	4 bits	8 bits	8 bits	8 bits
tamaño del cómputo de instrucciones	45	48	74	150
Formato de instrucciones	1,2 bytes	1,2,3 bytes	1,2,3 bytes	24 bits
Capacidad de memoria	4Kx8 ROM 1280Kx4 RAM	16Kx8 ROM 16Kx8 RAM	64Kx8 ROM 64Kx8 RAM	512Kx24 ROM 64Kx16 RAM
Profundidad del Stach	3 niveles	7 niveles	memoria	32 niveles
Capacidad de interrupción	NO	SI	SI	SI
Aritmética	Paralela	Paralela	Paralela	Paralela
Registros	16x4	6x8	6x8	49x8
Tiempo del ciclo de instrucción	10.8μs	7.5μs	2μsec	4μs

TABLA 2.1 Algunos microprocesadores en el mercado

La memoria de lectura-escritura (RAM) consiste de un conjunto de dispositivos de memoria de acceso aleatorio que se usan para almacenar el macrocódigo (programa del usuario) bajo ejecución. Como se indica en la tabla 2.1, su tamaño varía dependiendo de los requerimientos del usuario y está limitada por la capacidad de direccionamiento del microprocesador.

La sección de entrada/salida (I/O) consiste del almacenamiento y control de interfase necesario para conectar al sistema a dispositivos de entrada/salida tal como teletipos, terminales, y otro tipo de dispositivos periféricos.

La entrada/salida es una área que debe considerarse cuidadosamente durante el proceso de selección de un microprocesador. La entrada/salida puede producir una limitación seria para sistemas pequeños que requieren de una actividad de entrada/salida pesada.

La cuarta sección de la microcomputadora es el microprocesador que se designa como CPU. El CPU se caracteriza por una gran variedad de unidades disponibles en el mercado.

2.2. Arquitectura del CPU 8080 de INTEL.

La unidad central de proceso 8080 es un dispositivo que maneja 8 bits en paralelo y se usa para construir computadoras digitales de propósito general. Se construye en un solo estrato usando el proceso MOS canal-n. La 8080 transfiere datos e información sobre su estado interno vía un BUS de datos bidireccional del tipo tres-estados (3-State) (DO-D7). La dirección de memoria y la de los dispositivos periféricos se transmiten sobre un BUS de direcciones del tipo tres-estados (3-State). Bus de direcciones (AO-A15). Dispone de seis salidas de señales de control y de tiempo (SYNC, DBIN,WAIT,$\overline{\text{WR}}$,HLDA e INTE) esas son las señales que salen del 8080, hay cuatro entradas de potencia (+12V, +5V, -5V y TIERRA) y dos entradas de relój ($\emptyset_1$ y $\emptyset_2$). (Ver figu. 2.3)

2.2.1. Unidades funcionales del CPU

El CPU 8080 tiene las siguientes unidades funcionales:

- Direccionamiento y arreglo de registros
- Unidad Aritmética y Lógica
- Sección de control y del registro de instrucciones
- Bus de datos bidireccional (3-state).

Véase la figura 2. en dode se ilustra el diagrama funcional a bloques del CPU 8080.

Registros

Los registros están formados por un arreglo de RAM estática organizada en seis registros de 16 bits.

- Contador de programa (PC)
- Apuntador del Stack (SP)
- Seis registros de propósito general de 8 bits arreglados en pares, se designan como B,C; D, E; y H, L
- Un par de registros temporales W y Z.

Unidad Aritmética y Lógica.

La ULA contiene los siguientes registros

- Un acumulador de 8 bits
- Un acumulador temporal de 8 bits (ACT)
- Un registro de banderas de 5 bits: cero, carry, sign, parity y carry auxiliar
- Un registro temporal de 8 bits.

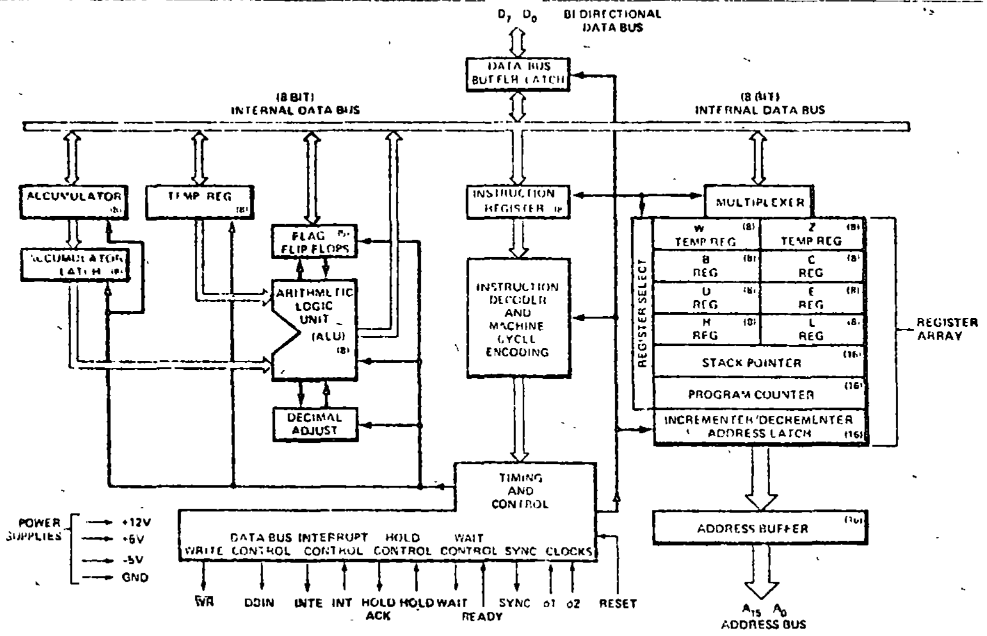


FIG. 2.3. Diagrama funcional a bloques del CPU 8080

ARCHITECTURE OF THE 8080 CPU

The 8080 CPU consists of the following functional units:

- Register array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bi-directional, 3-state data bus buffer

Figure 2-2 illustrates the functional blocks within the 8080 CPU.

Registers:

The register section consists of a static RAM array organized into six 16-bit registers:

- Program counter (PC)
- Stack pointer (SP)
- Six 8-bit general purpose registers arranged in pairs, referred to as B,C; D,E; and H,L
- A temporary register pair called W,Z

The program counter maintains the memory address of the current program instruction and is incremented automatically during every instruction fetch. The stack pointer maintains the address of the next available stack location in memory. The stack pointer can be initialized to use any portion of read-write memory as a stack. The stack pointer is decremented when data is "pushed" onto the stack and incremented when data is "popped" off the stack (i.e., the stack grows "downward").

The six general purpose registers can be used either as single registers (8-bit) or as register pairs (16-bit). The temporary register pair, W,Z, is not program addressable and is only used for the internal execution of instructions.

Eight-bit data bytes can be transferred between the internal bus and the register array via the register-select multiplexer. Sixteen-bit transfers can proceed between the register array and the address latch or the incrementer/decrementer circuit. The address latch receives data from any of the three register pairs and drives the 16 address output buffers (A_0-A_{15}), as well as the incrementer/decrementer circuit. The incrementer/decrementer circuit receives data from the address latch and sends it to the register array. The 16-bit data can be incremented or decremented or simply transferred between registers.

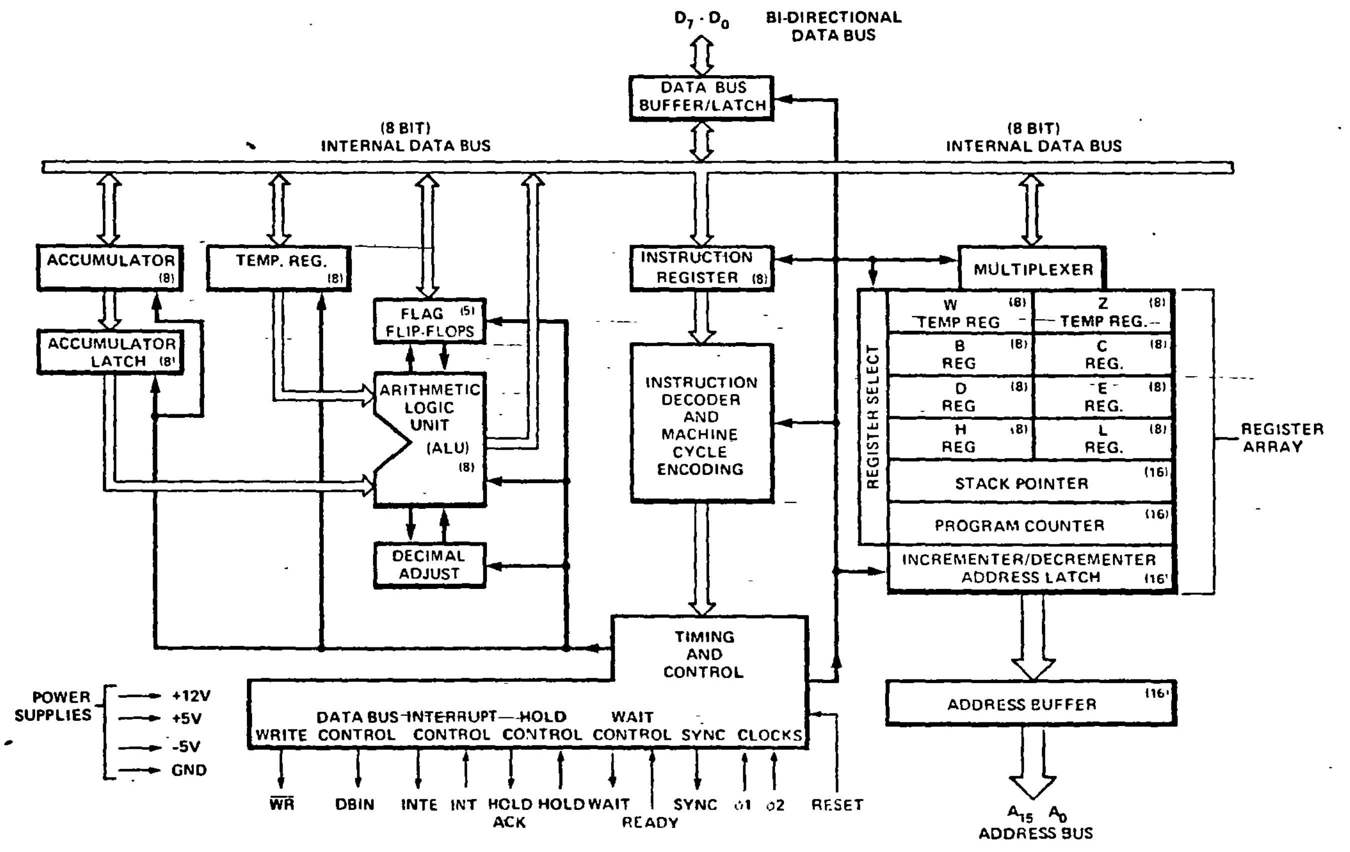


Figure 2-2. 8080 CPU Functional Block Diagram

2.2.2. Definición de los estados del CPU.

Las instrucciones para el CPU requieren de uno a cinco ciclos de máquina para su ejecución completa. La 8080 envía una palabra de 8 bits de información del estado sobre el bus de datos al principio de cada ciclo (durante la duración de SYNC). La definición de los estados es la siguiente:

INTA	DO	Reconocimiento de la señal para una petición de INTERRUPT. La señal debe usarse para disparar una instrucción de reinicio sobre el bus de datos cuando DBIN está activa.
$\overline{\text{WO}}$	D1	Indica que la operación en el ciclo corriente de máquina será una escritura a memoria (WRITE) o una función de salida (OUT PUT) ($\overline{\text{WO}}$ =0). De otra forma una lectura (READ) de memoria o se ejecutará una operación de entrada (INPUT).
STACK	D2	Indicará que el bus de direcciones mantendrá la dirección de 'pushdown the stack' del apuntador del 'stack'.
HLTA	D3	Reconoce la señal de la instrucción de HALT.
OUT	D4	Indica que el BUS de direcciones contiene la dirección de un dispositivo de salida y que el BUS de datos contiene el dato de salida cuando $\overline{\text{WR}}$ está activado.
M1	D5	Proporciona una señal para indicar que el CPU está en el ciclo de traer el primer byte de una instrucción.
IMP	D6	Indica que el BUS de direcciones contiene la dirección de un dispositivo de entrada y que el dato de entrada debe colocarse sobre el bus de datos cuando DBIN está activo.
MEMR	D7	Designa que el BUS de datos será usado para leer datos de memoria.

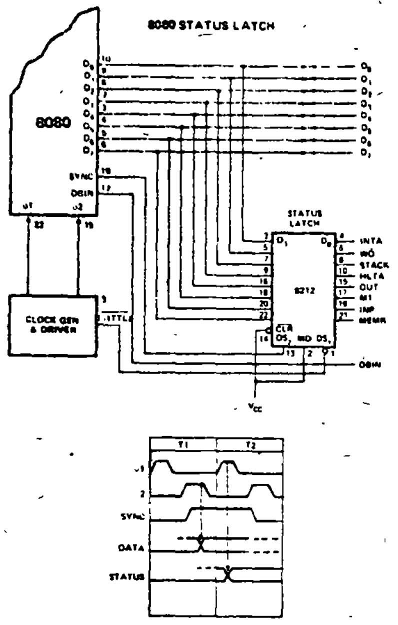


FIG. 2.4 Estado del Procesador

STATUS WORD CHART

DATA BUS BIT	STATUS INFORMATION	INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

TYPE OF MACHINE CYCLE

Ⓝ STATUS WORD

TABLA 2.2 Estados de la palabra

2.2.2. Definición de los estados del CPU.

Las instrucciones para el CPU requieren de uno a cinco ciclos de máquina para su ejecución completa. La 8080 envía una palabra de 8 bits de información del estado sobre el bus de datos al principio de cada ciclo (durante la duración de SYNC). La definición de los estados es la siguiente:

INTA	DO	Reconocimiento de la señal para una petición de INTERRUPT. La señal debe usarse para disparar una instrucción de reinicio sobre el bus de datos cuando DBIN está activa.
$\overline{\text{WO}}$	D1	Indica que la operación en el ciclo corriente de máquina será una escritura a memoria (WRITE) o una función de salida (OUT PUT) ($\overline{\text{WO}}$ =0). De otra forma una lectura (READ) de memoria o se ejecutará una operación de entrada (INPUT).
STACK	D2	Indicará que el bus de direcciones mantendrá la dirección de 'pushdown the stack' del apuntador del 'stack'.
HLTA	D3	Reconoce la señal de la instrucción de HALT.
OUT	D4	Indica que el BUS de direcciones contiene la dirección de un dispositivo de salida y que el BUS de datos contiene el dato de salida cuando $\overline{\text{WR}}$ está activado.
M1	D5	Proporciona una señal para indicar que el CPU está en el ciclo de traer el primer byte de una instrucción.
IMP	D6	Indica que el BUS de direcciones contiene la dirección de un dispositivo de entrada y que el dato de entrada debe colocarse sobre el bus de datos cuando DBIN está activo.
MEMR	D7	Designa que el BUS de datos será usado para leer datos de memoria.

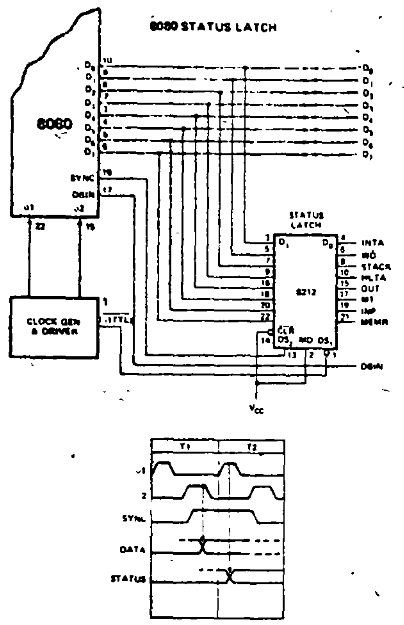


FIG. 2.4 Estado del Procesador

STATUS WORD CHART

DATA BUS BIT	STATUS INFORMATION	INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
		TYPE OF MACHINE CYCLE									
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{\text{WO}}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Ⓝ STATUS WORD

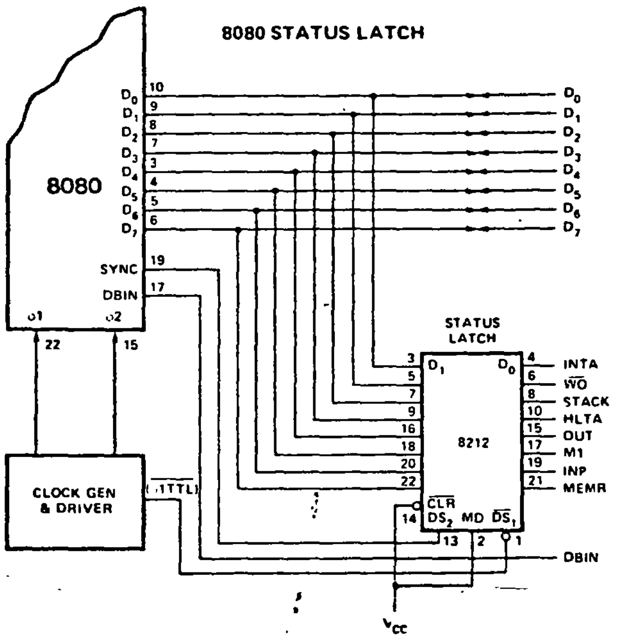
TABLA 2.2 | Estados de la palabra

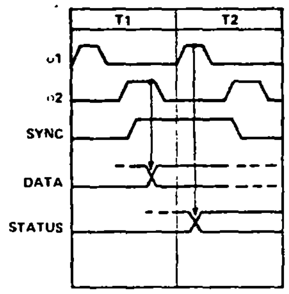
Instructions for the 8080 require from one to five machine cycles for complete execution. The 8080 sends out 8 bit of status information on the data bus at the beginning of each machine cycle (during SYNC time). The following table defines the status information.

STATUS INFORMATION DEFINITION

Symbols	Data Bus Bit	Definition
INTA*	D_0	Acknowledge signal for INTERRUPT request. Signal should be used to gate a restart instruction onto the data bus when DBIN is active.
$\overline{\text{WO}}$	D_1	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function ($\overline{\text{WO}}$ = 0). Otherwise, a READ memory or INPUT operation will be executed.
STACK	D_2	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
HLTA	D_3	Acknowledge signal for HALT instruction.
OUT	D_4	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when $\overline{\text{WR}}$ is active.
M_1	D_5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
INP*	D_6	Indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when DBIN is active.
MEMR*	D_7	Designates that the data bus will be used for memory read data.

*These three status bits can be used to control the flow of data onto the 8080 data bus.





STATUS WORD CHART

		TYPE OF MACHINE CYCLE									
DATA BUS BIT	STATUS INFORMATION	INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{\text{WO}}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Ⓝ STATUS WORD

Table 2-1. 8080 Status Bit Definitions

2.3. Interconexión al CPU de memoria y dispositivos de entrada salida.

El diseño de una microcomputadora con las componentes actuales le permite a uno realizarlo de una manera simple en vista de la modularidad de las componentes. Un sistema completo consiste básicamente de tres módulos, comúnes a cualquier sistema de cómputo: CPU, memoria y módulo de entrada/salida. En la figura 2.5 se presenta un sistema típico.

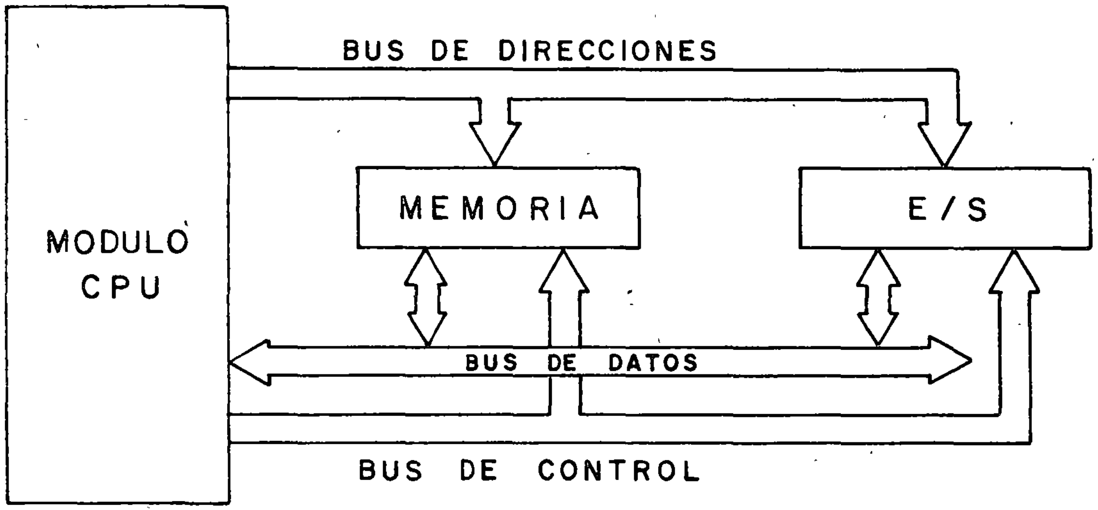


FIG. 2.5 Sistema de cómputo típico.

CPU (módulo)	Contiene al CPU, el reloj y los circuitos para el interfaz de memoria y los dispositivos de entrada salida.
Memoria	Contiene memoria de sólo lectura (ROM) y memoria de lectura/escritura (RAM) para almacenar programas y datos.
E/s	Contiene los circuitos que permiten al sistema de cómputo comunicarse con los circuitos o estructuras que estén fuera del CPU o el arreglo de memoria. Por ejemplo terminales, Discos, etc.

Existen tres buses para interconectar esos módulos.

BUS DE DATOS	Es una trayectoria bidireccional por el cual fluyen los datos entre el CPU y la memoria o E/s.
BUS DE DIRECCIONES	Es un grupo de líneas unidireccionales que identifican una dirección particular de memoria o un dispositivo de E/s.
BUS DE CONTROL	Es un conjunto unidireccional de señales que indican el tipo de actividad que se procesa corrientemente.

Tipo de Actividades

1. Leer de memoria
2. Escribir a memoria
3. Lectura de E/s
4. Escritura de E/s
5. Reconocimiento de una interrupción.

The following pages will cover the detailed design of the CPU Module with the 8080. The three Busses (Data, Address and Control) will be developed and the interconnection to Memory and I/O will be shown.

Design philosophies and system architectures presented in this manual are consistent with product development programs underway at INTEL for the MCS-80. Thus, the designer who uses this manual as a guide for his total system engineering is assured that all new developments in components and software for MCS-80 from INTEL will be compatible with his design approach.

CPU Module Design

The CPU Module contains three major areas:

1. The 8080 Central Processing Unit
2. A Clock Generator and High Level Driver
3. A bi-directional Data Bus Driver and System Control Logic

The following will discuss the design of the three major areas contained in the CPU Module. This design is presented as an alternative to the Intel® 8224 Clock Generator and Intel 8228 System Controller. By studying the alternative approach, the designer can more clearly see the considerations involved in the specification and engineering of the 8224 and 8228. Standard TTL components and Intel general purpose peripheral devices are used to implement the design and to achieve operational characteristics that are as close as possible to those of the 8224 and 8228. Many auxiliary timing functions and features of the 8224 and 8228 are too complex to practically implement in standard components, so only the basic functions of the 8224 and 8228 are generated. Since significant benefits in system timing and component count reduction can be realized by using the 8224 and 8228, this is the preferred method of implementation.

1. 8080 CPU

The operation of the 8080 CPU was covered in previous chapters of this manual, so little reference will be made to it in the design of the Module.

2. Clock Generator and High Level Driver

The 8080 is a dynamic device, meaning that its internal storage elements and logic circuitry require a timing reference (Clock), supplied by external circuitry, to refresh and provide timing control signals.

The 8080 requires two (2) such Clocks. Their waveforms must be non-overlapping, and comply with the timing and levels specified in the 8080 A.C. and D.C. Characteristics, page 5-15.

Clock Generator Design

The Clock Generator consists of a crystal controlled,

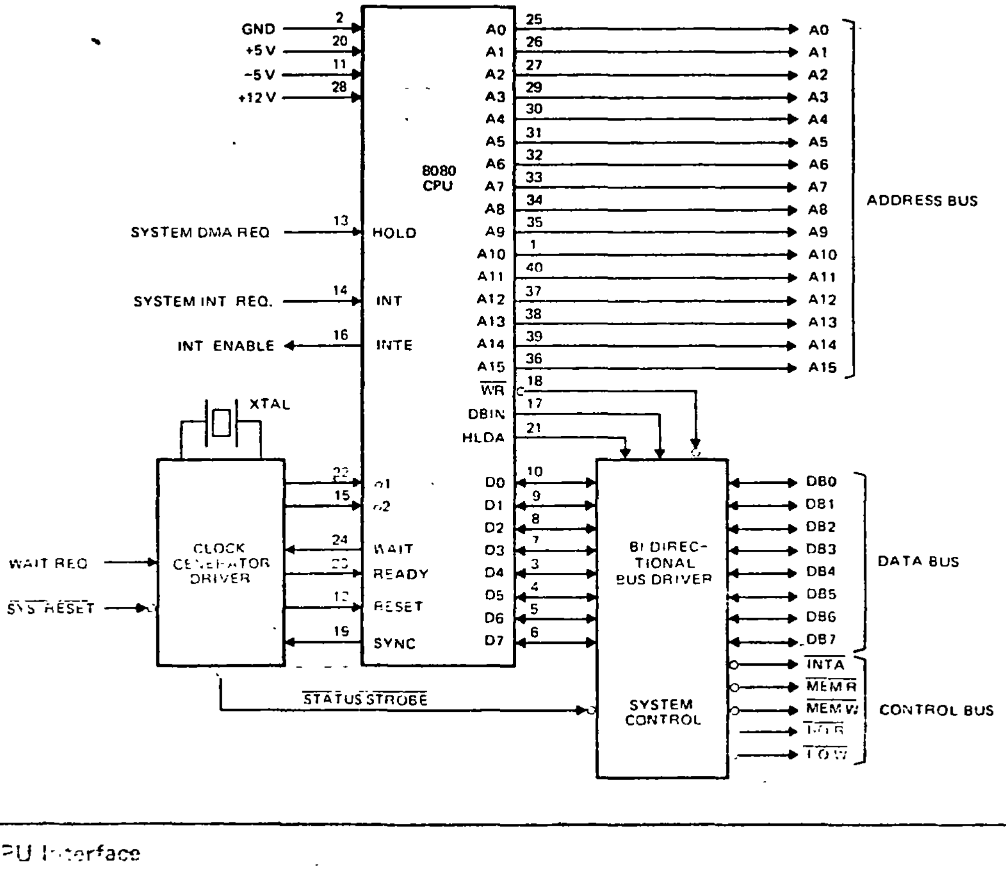


Figure 3-2. 8080 CPU Interface

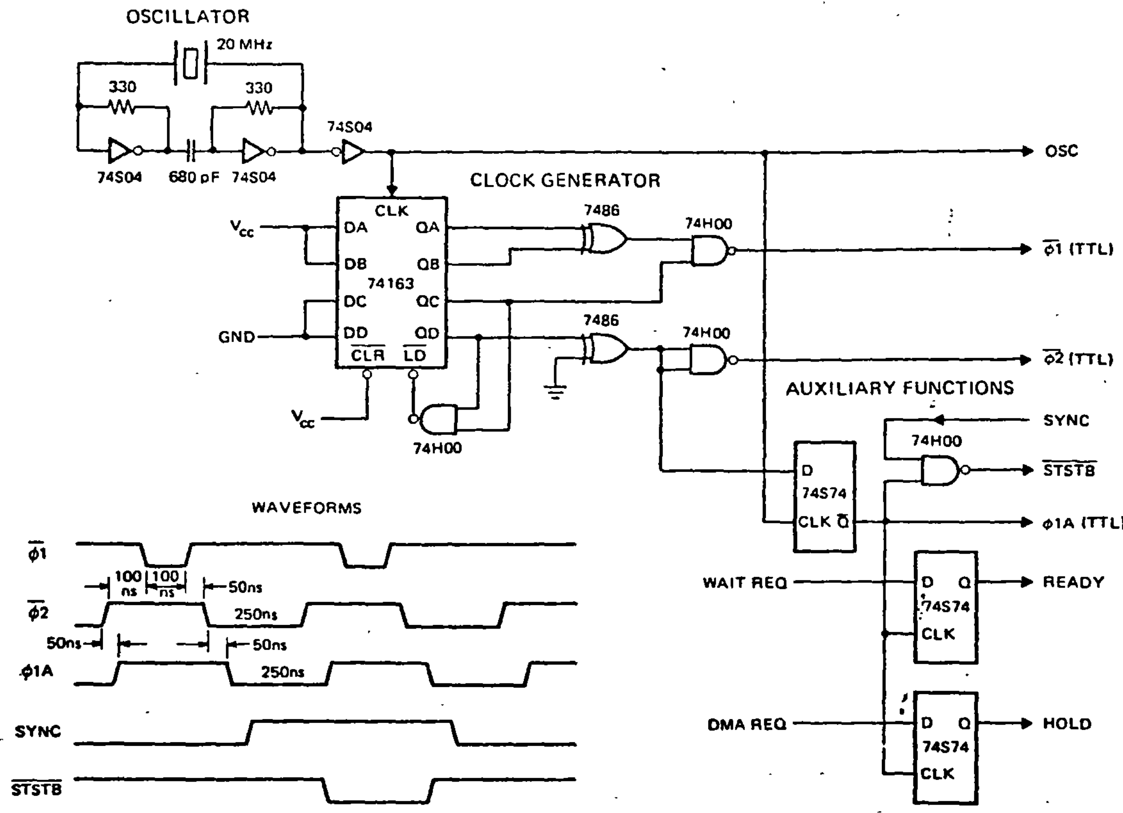


Figure 3-3. 8080 Clock Generator

20 MHZ oscillator, a four bit counter, and gating circuits.

The oscillator provides a 20 MHZ signal to the input of a four (4) bit, presettable, synchronous, binary counter. By presetting the counter as shown in figure 3-3 and clocking it with the 20 MHZ signal, a simple decoding of the counters outputs using standard TTL gates, provides proper timing for the two (2) 8080 clock inputs.

Note that the timing must actually be measured at the output of the High Level Driver to take into account the added delays and waveform distortions within such a device.

High Level Driver Design

The voltage level of the clocks for the 8080 is not TTL compatible like the other signals that input to the 8080. The voltage swing is from .6 volts (V_{ILC}) to 11 volts (V_{IHC}) with risetimes and falltimes under 50 ns. The Capacitive Drive is 20 pf (max.). Thus, a High Level Driver is required to interface the outputs of the Clock Generator (TTL) to the 8080.

The two (2) outputs of the Clock Generator are capacitivity coupled to a dual- High Level clock driver. The driver must be capable of complying with the 8080 clock input specifications, page 5-15. A driver of this type usually has little problem supplying the positive transition when biased from the 8080 V_{DD} supply (12V) but to achieve the low voltage specification (V_{ILC}) .8 volts Max. the driver is biased to the 8080 V_{BB} supply (-5V). This allows the driver to swing from GND to V_{DD} with the aid of a simple resistor divider.

A low resistance series network is added between the driver and the 8080 to eliminate any overshoot of the pulsed waveforms. Now a circuit is apparent that can easily comply with the 8080 specifications. In fact rise and falltimes of this design are typically less than 10 ns.

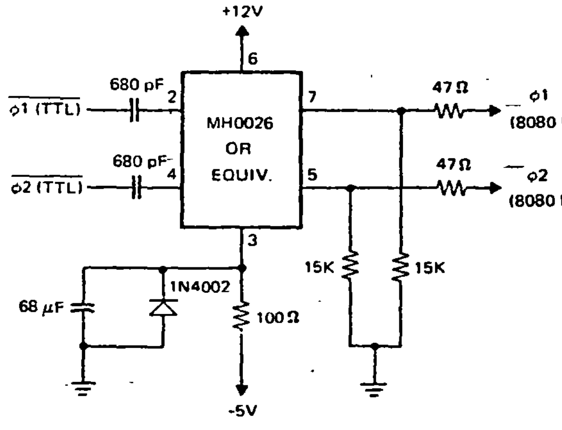


Figure 3-4. High Level Driver

2.4 Comparación entre microprocesadores Z80 vs 8080A y 6800

CPU CHIP COMPARISONS

- ☐ Similarities
- ☐ Differences
- ☐ Registers
- ☐ Pin definitions
- ☐ Clock requirements
- ☐ Clock waveforms
- ☐ Dynamic memory refresh
- ☐ Interrupts
- ☐ Technology
- ☐ Second source

MOSTEK Z80

CPU CHIP SIMILARITIES

The Z80, 8080 and 6800 all have:

- ☐ 8 bit data and 16-bit address paths
- ☐ Stack addressing
- ☐ Address 65K of memory
- ☐ Interruptable architecture
- ☐ 40 pin DIP
- ☐ Independent tri-state data and address buses
- ☐ N-channel MOS

MOSTEK Z80

CPU CHIP DIFFERENCES

	Z80	8080	6800
Number of:			
Instructions	158	78	72
8-bit registers	14	7	2
16-bit registers*	8	5	2
Index registers	2	0	1
Address modes	10	7	8
I/O addresses	256	256	0 **
Flag bits	6	5	6

*Counts 8-bit register pairs as 16 bit registers

**Treats I/O as memory

MOSTEK Z80

CPU CHIP DIFFERENCES (continued)

	Z80	8080	6800
Voltage required	+5V	+5, −5, +12V	+5V
Standard clock rate	DC-2.5MHz (1)	0.5-2MHz	0.1−1MHz
Non-maskable interrupt	Yes	No	Yes
TTL compatible inputs	Yes	No (2)	Yes
Asynchronous control inputs	Yes	No (3)	No
3 state control lines	Yes	No	Yes/No

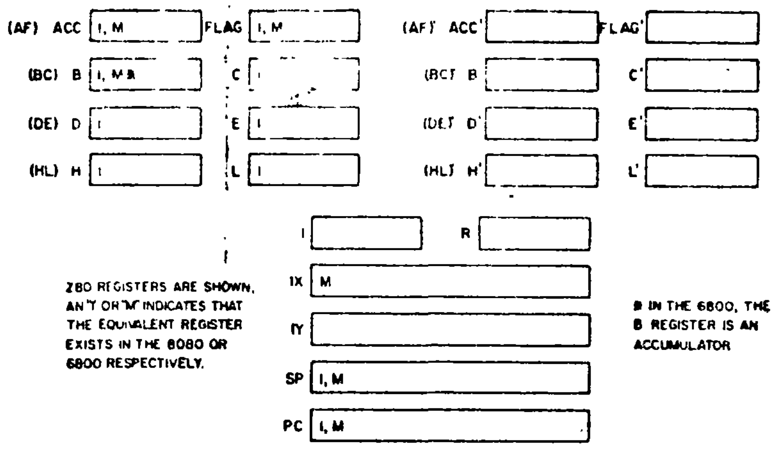
(1) DC capability allows designer to step thru each clock cycle during debug.

(2) Requires expensive interface buffers for TTL compatibility.

(3) Some inputs must be synchronized with external logic.

MOSTEK Z80

REGISTERS



Z80 REGISTERS ARE SHOWN, AN 'I' OR 'M' INDICATES THAT THE EQUIVALENT REGISTER EXISTS IN THE 8080 OR 6800 RESPECTIVELY.

✱ IN THE 6800, THE B REGISTER IS AN ACCUMULATOR

MOSTEK Z80

CLOCK REQUIREMENTS

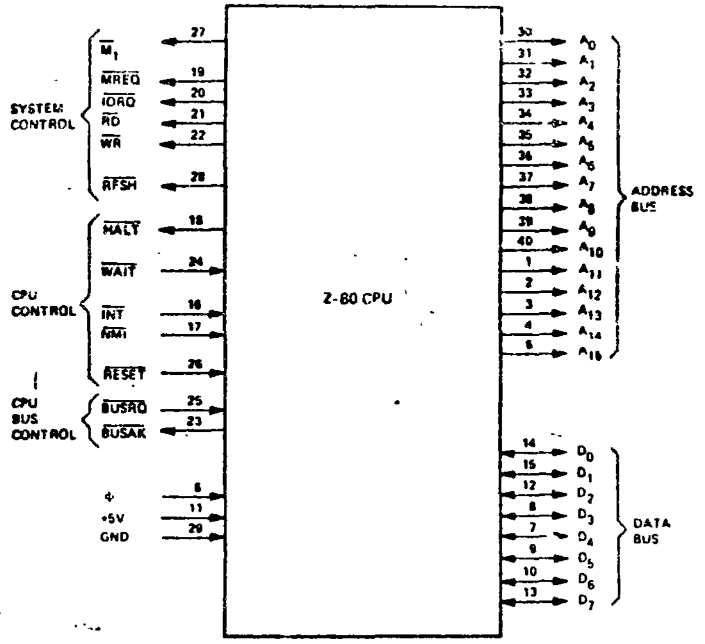
The 8080 requires two 8.1V asymmetrical non-overlapping clock inputs with a minimum delay between the end of one input and the beginning of the other.

The 6800 requires two 4.4V asymmetrical, non-overlapping clock inputs ($V_{SS} + 0.3 \geqslant V_{clock} \geqslant V_{cc} - 0.3$)

The Z80 requires a single 4.2V square wave. However, the clock can be held in the high state at any time, for any length of time. ($0.45 \geqslant V_{clock} \geqslant 4.65$).

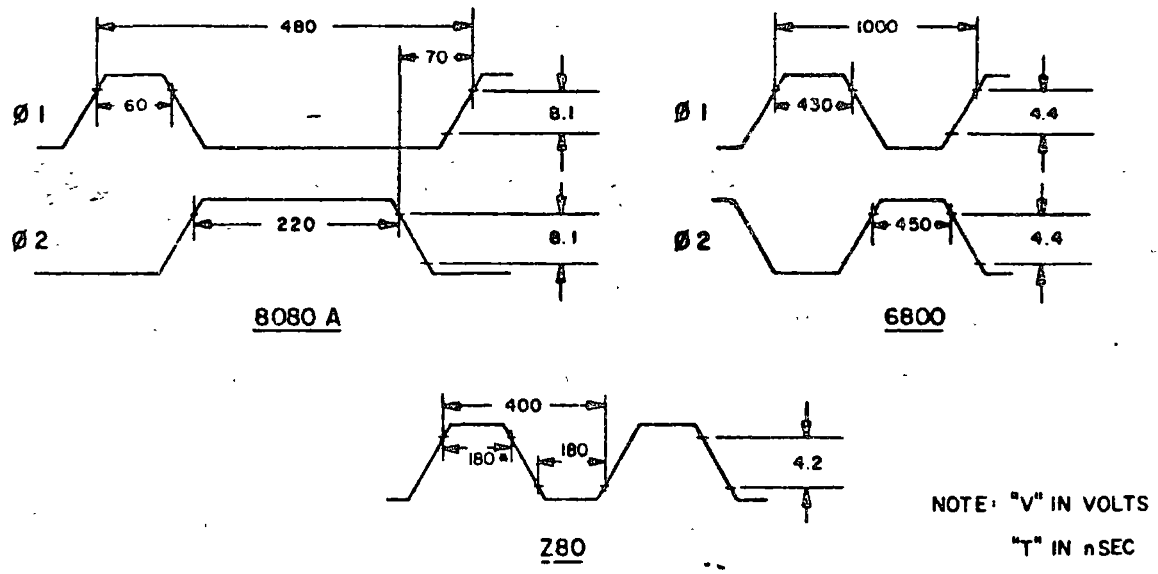
MOSTEK Z80

Z80 CPU PIN CONFIGURATION



MOSTEK Z80

MINIMUM CLOCK WAVEFORMS



NOTE: "V" IN VOLTS
"T" IN nSEC

*NOTE: NO MAXIMUM ON "T" CLOCK HIGH

MOSTEK Z80

STATIC OPERATION

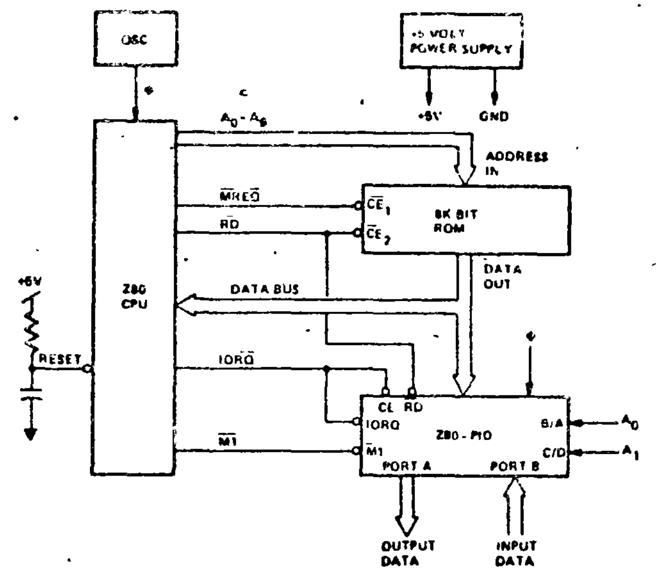
8080	No
6800	No
Z80	Yes

The Z80 clock can be left in the high state for an indefinitely long period without losing any information.

This means that the CPU clock can be slowed down or single clocked during development and debug.

The clock can also be stopped for fast-response DMA transfers.

MINIMUM Z80 SYSTEM

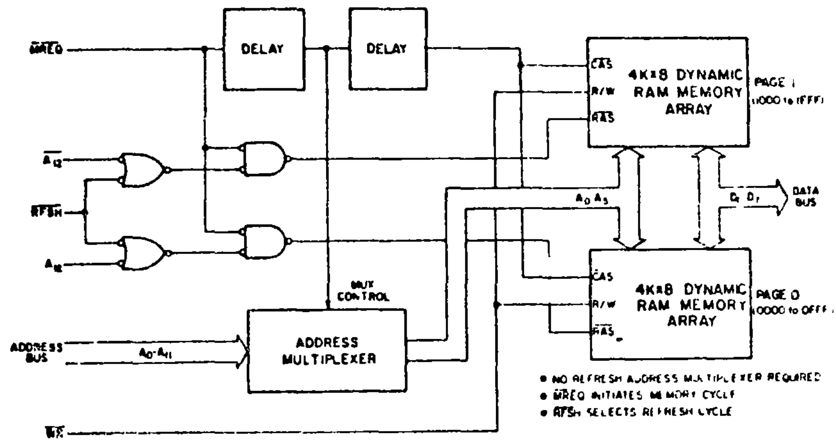


DYNAMIC MEMORY REFRESH

Neither the 8080 nor the 6800 have provisions for refreshing dynamic memories. This must be accomplished by an external controller which requires a special interface to the system.

The Z80 processor has built-in dynamic memory refresh capabilities, which occur between memory operations and are, therefore, transparent to system operation and do not slow down the processor.

INTERFACING DYNAMIC RAMs



MREQ
DELAY
DELAY
A12
RFSH
A12
CAS
R/W
RAS
4K×8 DYNAMIC RAM MEMORY ARRAY
PAGE 1 (1000 to 1FFF)
A0 A5
D0 D7
DATA BUS
MUX CONTROL
CAS
R/W
RAS
4K×8 DYNAMIC RAM MEMORY ARRAY
PAGE 0 (0000 to 0FFF)
ADDRESS BUS
A0-A11
ADDRESS MULTIPLEXER
WR
• NO REFRESH ADDRESS MULTIPLEXER REQUIRED
• MREQ INITIATES MEMORY CYCLE
• RFSH SELECTS REFRESH CYCLE

INTERRUPT CAPABILITY

The purpose of an interrupt is to allow peripheral devices to suspend ongoing CPU operation and force the CPU to start a peripheral service routine.

Once the service routine is completed the CPU returns to the operation it was performing when it was interrupted.

Interrupt capability is measured by:

A) How fast the CPU can begin servicing the peripheral
B) How many interrupts can be handled by the system
C) Whether one device can interrupt another's service routine
D) How much external hardware is needed to support interrupts

MOSTEK Z80

NON-MASKABLE INTERRUPTS

A non-maskable interrupt allows features such as power fail/auto restart to be implemented into the users system.

When the 6800 receives a non-maskable interrupt request, it completes its current instruction then performs an interrupt call to the location specified in memory locations FFFC & FFFD.

When the Z80 receives a non maskable interrupt request, it completes its current instruction, then performs a call to location 0066$_H$

The 8080 does not have provisions for a non-maskable interrupt

MOSTEK Z80

Z80 INTERRUPT MODE 0

When the 8080 recognizes an interrupt, it receives its next instruction (usually a call or restart) from the interrupting device via the data bus.

In mode 0, the Z80 responds exactly like the 8080.

There is no equivalent response in the 6800.

MOSTEK Z80

Z80 INTERRUPT MODE 1

- The 6800 responds to a maskable interrupt by performing an interrupt call to the location specified in memory locations FFF8 and FFF9.
- In mode 1, the Z80 responds by performing a restart to location 0038$_H$.
- There is no equivalent response in the 8080.

MOSTEK Z80

Z80 INTERRUPT MODE 2

This mode allows for very simple vectoring to the proper interrupt service routine anywhere in memory.

In mode 2, the Z80 programmer establishes an interrupt vector table in memory. An indirect call through this table is made in response to an interrupt. The high order address bits for the pointer to the table are placed into the interrupt vector register by the programmer. The low order seven bits are supplied by the interrupting device, thus allowing 128 interrupt routines.

There is no equivalent response in either the 6800 or the 8080.

This mode of interrupt response allows a single seven bit vector from the interrupting device to vector the CPU to the proper service routine very quickly. Furthermore, since the interrupt table may be located in RAM memory, the programmer can dynamically reallocate how the CPU responds to interrupts from the same device.

MOSTEK Z80

SAVING REGISTERS ON INTERRUPT

- The 6800 automatically pushes the index register, program counter, accumulators and condition code onto the stack. This is possible because the 6800 has very few registers.
- In the 8080 and the Z80, a normal subroutine call is executed, leaving it to the interrupt routine to save only those registers necessary.
- The Z80 programmer has the option of swapping register banks for very fast context switching or using the 8080 approach.

MOSTEK Z80

TECHNOLOGY

The Z80 makes use of ion-implant, N-channel, silicon gate depletion load technology. Using various implant levels, it is easy to trade speed for power. Thus, we will offer a wide range of high speed or low power Z80 devices in the future.

MOSTEK Z80

SECOND SOURCE

MOSTEK has a complete second source agreement with ZILOG for the Z80 family. This includes mask and technology interchange for all Z80 parts. This leads to the ultimate in compatibility in parts between the two sources.

MOSTEK Z80

MOSTEK Z80
INSTRUCTION SET AND ADDRESSING MODES

MOSTEK Z80

INSTRUCTION SET COMPARISON

- ☐ 8080 compatibility
- ☐ Instruction formats
- ☐ Addressing modes
- ☐ Types of instructions
- ☐ 8080 and 6800 programs equivalent to single Z80 instructions

MOSTEK Z80

8080 COMPATIBILITY

- ☐ The Z80 instruction is a true superset of the 8080 instruction set.
- ☐ The 8080 instructions are compatible at the binary machine code level.
- ☐ This means that programs which run on the 8080 will also run on the Z80 without modification.
- ☐ In other words, "anything the 8080 can do, the Z80 can do (better)".

MOSTEK Z80

INSTRUCTION FORMATS

		Z80	6800	8080	
OPCODE		X	X	X	One byte instructions
OPCODE	OPCODE	X	–	–	Two byte instructions
OPCODE	OPERAND	X	X	X	
OPCODE OPERAND	OPCODE	X	–	–	Three byte instructions
OPCODE OPERAND	OPERAND	X	X	X	
OPCODE OPERAND	OPCODE OPERAND	X	–	–	Four byte instructions

MOSTEK Z80

ADDRESSING MODES – DEFINITIONS

- ☐ IMMEDIATE. In this mode of addressing the byte following the OP code in memory contains the actual operand.
- ☐ IMMEDIATE EXTENDED. This mode is merely an extension of immediate addressing in that the two bytes following the OP codes are the operand.
- ☐ MODIFIED PAGE ZERO ADDRESSING. The Z-80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero.
- ☐ RELATIVE ADDRESSING. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur.
- ☐ EXTENDED ADDRESSING. Extended addressing provides for two bytes (16 bits) of address to be included in the instruction.

MOSTEK Z80

[illegible] (CONTINUED)

INDEXED ADDRESSING. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory

☐ REGISTER ADDRESSING. Many of the Z80 codes contain bits of information that specify which CPU register is to be used for an operation

☐ IMPLIED ADDRESSING. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands.

☐ REGISTER INDIRECT ADDRESSING. This type of addressing specifies a CPU register pair (such as HL) to be used as a pointer to any location in memory.

☐ BIT ADDRESSING. The Z80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation while three bits in the OP code specify which of the eight bits is to be manipulated

MOSTEK Z80

ADDRESSING MODES

		Z80	6800	8080
	IMMEDIATE	X	X	X
EXTENDED	IMMEDIATE	X	X	X
	PAGE ZERO	X	X	X
	RELATIVE	X	X	—
	EXTENDED	X	X	X
	INDEXED	X	X	—
	REGISTER	X	—	X
	IMPLIED	X	X	X
REGISTER	INDIRECT	X	X	X
	BIT	X	—	—

MOSTEK Z80

TYPES OF INSTRUCTIONS

8-bit load	Rotate/shift
16 bit load, push and pop	Bit set, reset & test
Exchange	Jump, call & return
Block transfer	Loop
Block search	Restart
8 bit arithmetic & logic	Byte input/output
General purpose arithmetic and flag	Block input/output
16 bit arithmetic	Misc CPU control

MOSTEK Z80

INSTRUCTION SET COMPARISON

For each type of instruction used, a verbal comparison of the major similarities and differences between the 6800, the 8080 and the Z80 is given.

Then, a chart showing a detailed instruction-by-instruction comparison between the Z80 and the 8080 is given.

The hexadecimal number in each square denotes the OP code for the Z80 instruction. A shaded square indicates the instruction is also an 8080 instruction.

Note that all 8080 instructions are included in the Z80 instruction set, and, in all cases, the OP codes are identical, thus providing program interchangability at the machine level.

MOSTEK Z80

8 BIT LOAD INSTRUCTIONS

The Z80 can move a byte between registers and between memory and registers exactly as the 8080 can.

The Z80 is also capable of indexed addressed moves to or from any one of seven registers, or from an immediate operand, using either of two index registers. (The 6800 only has two registers and one index register; the 8080 has no index registers.)

Index registers are valuable, especially when working with blocks of data. The second index register is particularly useful when keeping track of two tables simultaneously.

In addition, the Z80 can move data between the interrupt vector register, the refresh address register and the accumulator. These registers are unique to the Z80.

16 BIT LOAD INSTRUCTIONS, PUSH AND POP

- ☐ The Z80 is capable of all the 16 bit moves that the 8080 can perform.
- ☐ The Z80 also allows extended addressing of 16-bit operands. This allows more flexibility in accessing 16 bit operands for subsequent processing by the Z80 16-bit instructions.
- ☐ In addition, the Z80 has the ability to load and store its index registers like the 6800.
- ☐ The Z80 can also push and pop the index registers, not provided in the 6800. This extends the usefulness of the stack by allowing several tasks to use both index registers (re-entrent programs).

EXCHANGE INSTRUCTIONS

	Z80	8080	6800
A ⟶ B	–	–	X*
AF ⟶ AF'	X	–	–
BC ⟶ BC' DE ⟶ DE' HL ⟶ HL'	X	–	–
DE ⟶ HL	X	X	–
HL ⟶ (SP)	X	X	–
IX ⟶ (SP)	X	–	–
IY ⟶ (SP)	X	–	–

The additional Z80 exchange instructions allow fast context switching without using "PUSH" and "POP".

* In the 6800, A and B are each 8 bit accumulator.

EXCHANGE – Z80 VS 8080

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

LEGEND: XX Z80 INSTRUCTION (XX = OPCODE)
XX Z80 AND 8080 INSTRUCTION (XX = OPCODE)

EQUIVALENT PROGRAMS

The following examples demonstrate the power of some of the unique new Z80 instructions. For each case, the 8080 and 6800 programs equivalent to a single Z80 instruction are presented and compared.

MOVE BLOCK

MOVE A BLOCK OF DATA OF ARBITRARY SIZE FROM ANYWHERE IN MEMORY TO ANYWHERE ELSE IN MEMORY.

				CYCLES	BYTES
1.	8080A				
	MOV	LD	A,(HL)	7	1
		LD	(DE),A	7	1
		INC	HL	5	1
		INC	DE	5	1
		DEC	C	5	1
		JP	NZ,MOV	10	3
		DEC	B	5	1
		JP	NZ,MOV	10	3
				54 cycles	12 bytes

CYCLE TIME = 54 x 0.5 = 27 μS/BYTE

				CYCLES	BYTES
2.	6800				
	MOV	LDAA	0,X	5	2
		INX		4	1
		STX	SOURCE	5	2
		LDX	DEST	4	2
		STAA	0,X	6	2
		INX		4	1
		STX	DEST	5	2
		LDX	BYTCNT	4	2
		DEX		4	1
		STX	BYTCNT	5	2
		BNE	MOV	4	2
				50 cycles	19 bytes

CYCLE TIME = 50 x 1.0 = 50 μS/BYTE

			CYCLES	BYTES
3.	Z80	LDIR	21 cycles	2 bytes

CYCLE TIME · 21 x 0.4 = 8.4 μS/BYTE

BLOCK SEARCH

SEARCH A BLOCK OF DATA OF ARBITRARY SIZE, LOCATED ANYWHERE IN MEMORY FOR A BYTE MATCH

```
1.   8080A                                  CYCLES    BYTES
     FIND     CMP       (HL)                7         1
              JP        Z,FOUND             10        3
              INC       HL                  5         1
              DEC       C                   5         1
              JP        NZ,FIND             10        3
              DEC       B                   5         1
              JP        NZ,FIND             10        3
     NO FIND. . .                           52 cycles 13 bytes
              CYCLE TIME = 52 x 0.5 μs = 26 μs/BYTE

2.   6800                                   CYCLES    BYTES
     FIND     LDX       LOC                 4         2
              CMPA      0,X                 5         2
              BEQ       FOUND               4         2
              INX                           4         1
              STX       LOC                 5         2
              LDX       BYTCNT              4         4
              DEX                           4         1
              STX       BYTCNT              5         2
              BNE       FIND                4         2
                                            39 cycles 18 bytes
              CYCLE TIME = 39 x 1.0 μs = 39 μs/BYTE

3.   Mostek   Z80
     FIND     CPIR                          21        2
              (JP Z,FOUND)
     NO FIND. . .
              CYCLE TIME = 21 x 0.4 μs = 8.4 μs/BYTE
```

BLOCK I/O TRANSFER

READ A BLOCK OF DATA, UP TO 256 BYTES, FROM ANY I/O DEVICE TO ANYWHERE IN MEMORY.

```
                                            CYCLES      BYTES
1.   8080A
     IOMOV    IN        A, (PORT)           10          2
              LD        (HL), A             7           1
              INC       HL                  5           1
              DEC       C                   5           1
              JP        NZ,IOMOV            10          3
                                            37 cycles   8 bytes

          CYCLE TIME = 37 x 0.5 = 18.5 μs/BYTE
                                            CYCLES      BYTES
2.   6800
     IOMOV    LDAA      PORT                4           3
              STAA      0,X                 6           2
              INX                           4           1
              DEC       B                   2           1
              BNE       IOMOV               4           2
                                            20 cycles   9 bytes
              CYCLE TIME = 20 x 1.0 μs/BYTE = 20 μs/BYTE

3.   Mostek   Z80
     IOMOV    INIR                          20 cycles   2 bytes

              CYCLE TIME = 20 x 0.4 μs = 8 μs/BYTE
```

BENCHMARKS

- ☐Triple-precision binary multiply
- ☐Move a block of data
- ☐Search a memory block for a substring
- ☐Interrupt driven I/O transfer

MOSTEK Z80

NOTE ON BENCHMARK PROGRAMS

The following benchmark programs were selected for the purpose of objectively comparing the Z80, 8080 and the 6800 in important application areas.

MOSTEK realizes that the best benchmark is one written for your specific application. The following are only offered for your evaluation as being representative of the general class of problems being solved by microcomputers.

MOSTEK Z80

TRIPLE PRECISION BINARY MULTIPLY – DEFINITION

Multiply two 24 bit 2's complement binary numbers in memory (starting, MSB first, at APART and BPART) to form a 48 bit 2's complement product in memory (at PROD).

MOSTEK Z80

TRIPLE PRECISION BINARY MULTIPLY – RESULTS

	Z80	8080	6800
Execution speed* (μs)	2504	5061	3532
(Relative to Z80)	(1.0)	(2.0)	(1.4)
Memory required (Bytes)	85	112	121
(Relative to Z80)	(1.0)	(1.3)	(1.4)

*Worst case, multiplier = 0101 0101 0101 0101 0101 0101

MOSTEK Z80

INTERFACING

- System design considerations
- Interface comparisons
- Minimum chip set for functional processor
- Basic Z80 interfacing
- Dynamic refresh comparisons

SYSTEM DESIGN CONSIDERATIONS

☐ The 6800 is designed to use 6800-series parts to form a tight-knit-small to medium size system. This design must be significantly modified for larger size systems containing more standard parts.

☐ The 8080 is actually a three-chip microprocessor, once the 8224 and 8228 have been included in the design. This limits its usefulness in small systems. Also, the need for external non TTL compatible buffers is a definite dis advantage.

☐ The Z80 is quite efficient in small systems, due to its clean interface design. However, its efficiency actually increases as system size increases, because it was designed with large systems of standard components in mind. The CPU controlled dynamic memory refresh is just one example of the design philosophy.

INTERFACE CONSIDERATIONS

Provisions for:	Z80	8080	6800
DMA	YES	YES	YES
INTERRUPTS			
MASKED	YES	YES	YES
NON-MASKED	YES	NO	YES
SYNC'ING SLOW DEVICES	YES	YES	NO *
SEPARATE I/O	YES	YES	NO
MEMORY REFRESH	YES	NO	NO
DYNAMIC 4K RAM TIMING	YES	NO	NO

* Can be done in clock circuitry

MORE INTERFACE CONSIDERATIONS

	Z80	8080	6800
VOLTAGE REQUIRED	+5	+5,–5,+12	+5
CLOCK PHASES	1	2	2
CLOCK VOLTAGE Δ	4.2V	8.1V	4.4V
INTERNAL SYNC OF INPUTS	YES	NO	NO
DIRECT STROBE OUTPUTS	YES	NO	NO
NEED EXTERNAL STATUS LATCHES	NO	YES	NO
TTL COMPATIBLE	YES	NO	YES

2.5 Memorias RAM Y ROM

1024 BIT (256 x 4) STATIC MOS RAM WITH SEPARATE I/O

- 256 x 4 Organization to Meet Needs for Small System Memories
- Access Time — 850 nsec Max.
- Single +5V Suppl Voltage
- Directly TTL Compatible — All Inputs and Output
- Static MOS — No Clocks or Refreshing Required
- Simple Memory Expansion — Chip Enable Input
- Inputs Protected — All Inputs Have Protection Against Static Charge
- Low Cost Packaging — 22 Pin Plastic Dual-In-Line Configuration
- Low Power — Typically 150 mW
- Three-State Output — OR-Tie Capability
- Output Disable Provided for Ease of Use in Common Data Bus Systems

The Intel® 8101-2 is a 256 word by 4 bit static random access memory element using normally off N-channel MOS devices integrated on a monolithic array. It uses fully DC stable (static) circuitry and therefore requires no clocks or refreshing to operate. The data is read out nondestructively and has the same polarity as the input data.

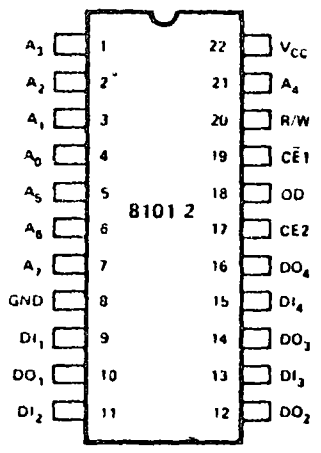
The 8101-2 is designed for memory applications where high performance, low cost, large bit storage, and simple interfacing are important design objectives.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. Two chip-enables allow easy selection of an individual package when outputs are OR-tied. An output disable is provided so that data inputs and outputs can be tied for common I/O systems. Output disable is then used to eliminate any bidirectional logic.

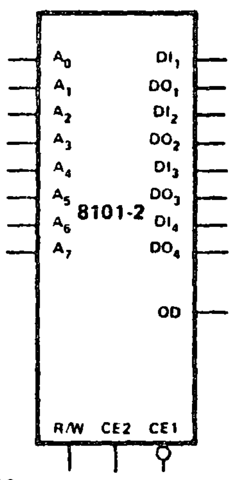
The Intel® 8101-2 is fabricated with N-channel silicon gate technology. This technology allows the design and production of high performance, easy-to-use MOS circuits and provides a higher functional density on a monolithic chip than either conventional MOS technology or P-channel silicon gate technology.

Intel's silicon gate technology also provides excellent protection against contamination. This permits the use of low cost silicone packaging.

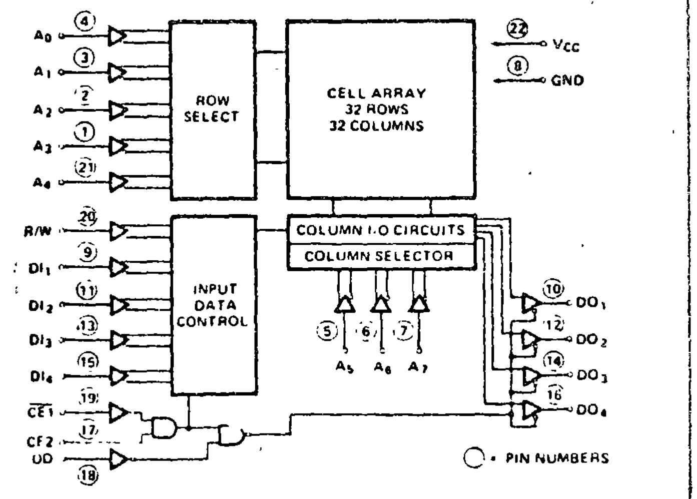
PIN CONFIGURATION


8101 2
A3 1, A2 2, A1 3, A0 4, A5 5, A6 6, A7 7, GND 8, DI1 9, DO1 10, DI2 11
22 VCC, 21 A4, 20 R/W, 19 CE1, 18 OD, 17 CE2, 16 DO4, 15 DI4, 14 DO3, 13 DI3, 12 DO2

LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

D_{IN}	DATA INPUT	OD	OUTPUT DISABLE
A_0 – A_7	ADDRESS INPUTS	D_{OUT}	DATA OUTPUT
R/W	READ/WRITE INPUT	V_{CC}	POWER (+5V)
$\overline{CE1}$ CE2	CHIP ENABLE		

Silicon Gate MOS 8111-2

1024 BIT (256 x 4) STATIC MOS RAM WITH COMMON I/O AND OUTPUT DISABLE

- **Organization 256 Words by 4 Bits**
- **Access Time — 850 nsec Max.**
- **Common Data Input and Output**
- **Single +5V Supply Voltage**
- **Directly TTL Compatible — All Inputs and Output**
- **Static MOS — No Clocks or Refreshing Required**
- **Simple Memory Expansion — Chip Enable Input**
- **Fully Decoded — On Chip Address Decode**
- **Inputs Protected — All Inputs Have Protection Against Static Charge**
- **Low Cost Packaging — 18 Pin Plastic Dual-In-Line Configuration**
- **Low Power — Typically 150 mW**
- **Three-State Output — OR-Tie Capability**

The Intel®8111-2 is a 256 word by 4 bit static random access memory element using normally off N-channel MOS devices integrated on a monolithic array. It uses fully DC stable (static) circuitry and therefore requires no clocks or refreshing to operate. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

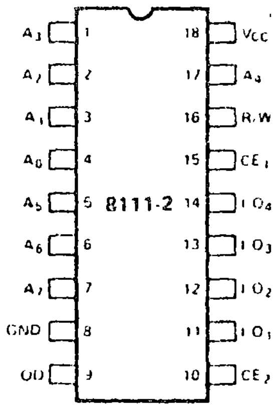
The 8111-2 is designed for memory applications in small systems where high performance, low cost, large bit storage, and simple interfacing are important design objectives.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. Separate chip enable ($\overline{CE}$) leads allow easy selection of an individual package when outputs are OR-tied.

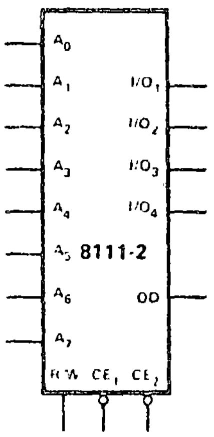
The Intel®8111-2 is fabricated with N-channel silicon gate technology. This technology allows the design and production of high performance, easy-to-use MOS circuits and provides a higher functional density on a monolithic chip than either conventional MOS technology or P-channel silicon gate technology.

Intel's silicon gate technology also provides excellent protection against contamination. This permits the use of low cost silicone packaging.

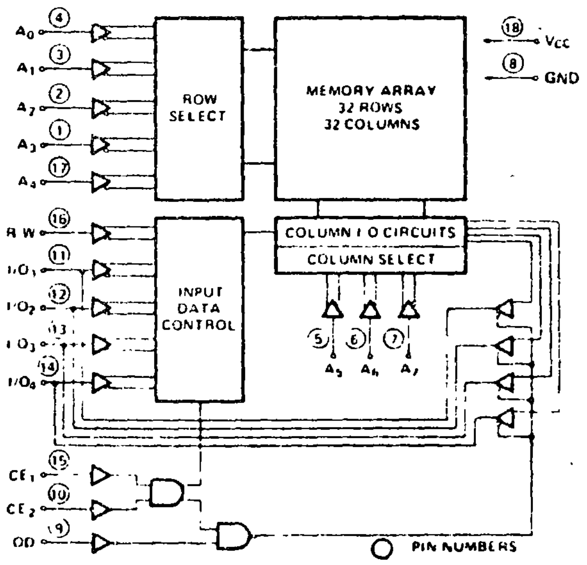
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_7	ADDRESS INPUTS
$\overline{OD}$	OUTPUT DISABLE
R/W	READ/WRITE INPUT
$\overline{CE}_1$	CHIP ENABLE 1
CE_2	CHIP ENABLE 2
I/O_1–I/O_4	DATA INPUT/OUTPUT

Silicon Gate MOS 8102-2

1024 BIT FULLY DECODED STATIC MOS RANDOM ACCESS MEMORY

- **Access Time — 850ns Max.**
- **Single +5 Volts Supply Voltage**
- **Directly TTL Compatible — All Inputs and Output**
- **Static MOS — No Clocks or Refreshing Required**
- **Low Power — Typically 150 mW**
- **Three-State Output — OR-Tie Capability**
- **Simple Memory Expansion — Chip Enable Input**
- **Fully Decoded — On Chip Address Decode**
- **Inputs Protected — All Inputs Have Protection Against Static Charge**
- **Low Cost Packaging — 16 Pin Plastic Dual-In-Line Configuration**

The Intel®8102-2 is a 1024 word by one bit static random access memory element using normally off N-channel MOS devices integrated on a monolithic array. It uses fully DC stable (static) circuitry and therefore requires no clocks or refreshing to operate. The data is read out nondestructively and has the same polarity as the input data.

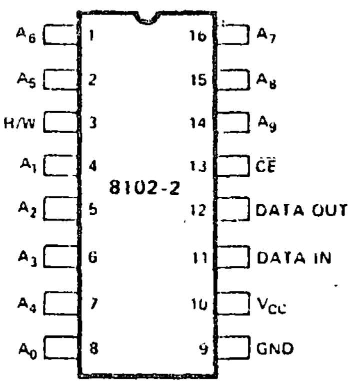
The 8102-2 is designed for microcomputer memory applications where high performance, low cost, large bit storage, and simple interfacing are important design objectives.

It is directly TTL compatible in all respects: inputs, output, and a single +5 volt supply. A separate chip enable (CE) lead allows easy selection of an individual package when outputs are OR-tied.

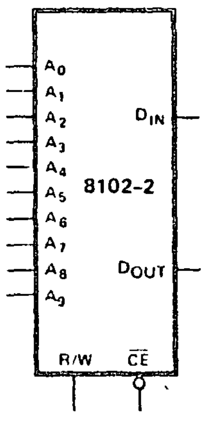
The Intel®8102-2 is fabricated with N-channel silicon gate technology. This technology allows the design and production of high performance, easy-to-use MOS circuits and provides a higher functional density on a monolithic chip than either conventional MOS technology or P-channel silicon gate technology.

Intel's silicon gate technology also provides excellent protection against contamination. This permits the use of low cost silicone packaging.

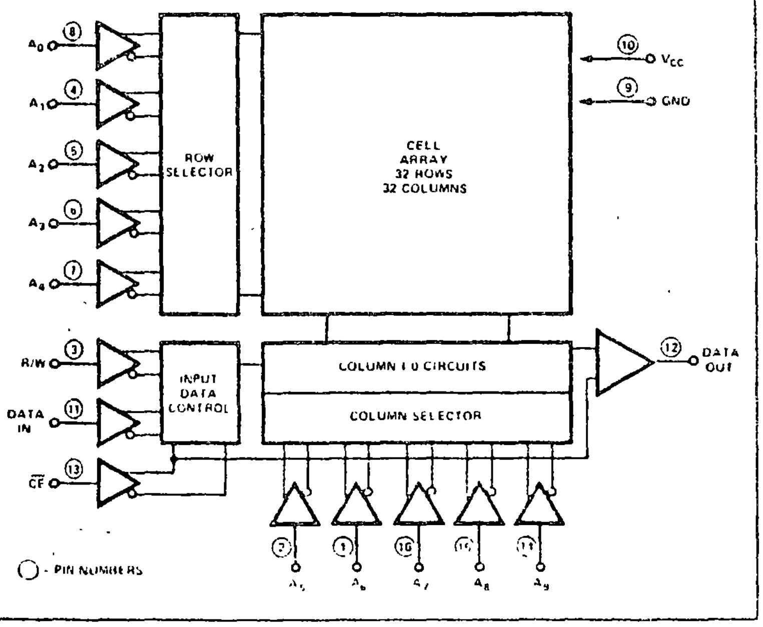
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

D_{IN}	DATA INPUT	CE	CHIP ENABLE
A_0-A_9	ADDRESS INPUTS	D_{OUT}	DATA OUTPUT
R/W	READ/WRITE INPUT	V_{CC}	POWER (+5V)

Silicon Gate MOS 8102A-4

1024 BIT FULLY DECODED STATIC MOS RANDOM ACCESS MEMORY

- Access Time — 450 ns Max.
- Single +5 Volts Supply Voltage
- Directly TTL Compatible — All Inputs and Output
- Static MOS — No Clocks or Refreshing Required
- Low Power — Typically 150 mW
- Three-State Output — OR-Tie Capability
- **Simple Memory Expansion — Chip Enable Input**
- **Fully Decoded — On Chip Address Decode**
- **Inputs Protected — All Inputs Have Protection Against Static Charge**
- **Low Cost Packaging — 16 Pin Plastic Dual-In-Line Configuration**

The Intel® 8102A-4 is a 1024 wrod by one bit static random access memory element using normally off N-channel MOS devices integrated on a monolithic array. It uses fully DC stable (static) circuitry and therefore requires no clocks or refreshing to operate. The data is read out nondestructively and has the same polarity as the input data.

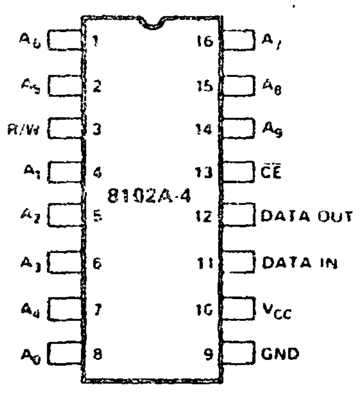
The 8102A-4 is designed for microcomputer memory applications where high performance, low cost, large bit storage, and simple interfacing are important design objectives.

It is directly TTL compatible in all respects: inputs, output, and a single +5 volt supply. A separate chip enable ($\overline{CE}$) lead allows easy selection of an individual package when outputs are OR-tied.

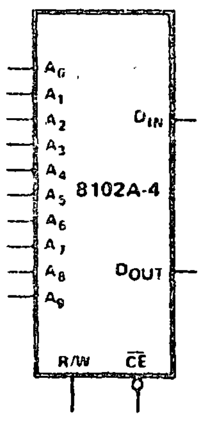
The Intel® 8102A-4 is fabricated with N-channel silicon gate technology. This technology allows the design and production of high performance, easy-to-use MOS circuits and provides a higher functional density on a monolithic chip than either conventional MOS technology or P-channel silicon gate technology.

Intel's silicon gate technology also provides excellent protection against contamination. This permits the use of low cost silicone packaging.

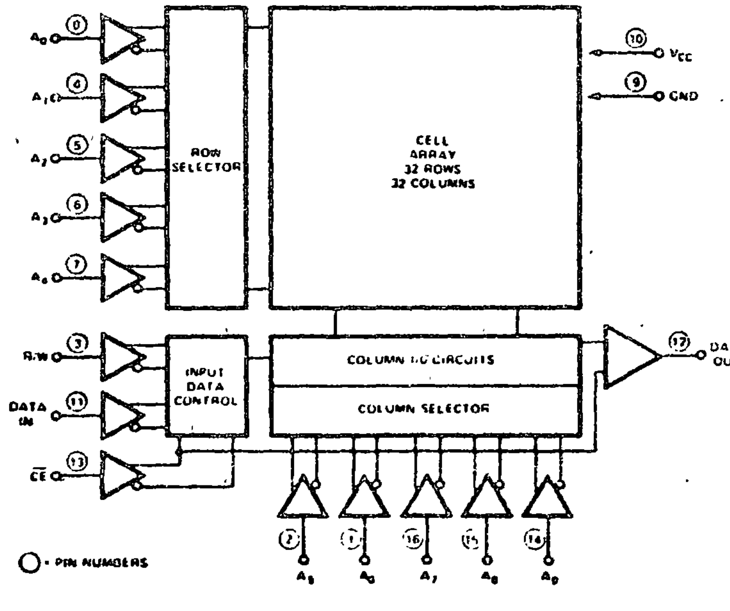
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

D_{IN}	DATA INPUT	$\overline{CE}$	CHIP ENABLE
A_0-A_9	ADDRESS INPUTS	D_{OUT}	DATA OUTPUT
R/W	READ/WRITE INPUT	V_{CC}	POWER (+5V)

Silicon Gate MOS 8107A

FULLY DECODED RANDOM ACCESS 4096 BIT DYNAMIC MEMORY

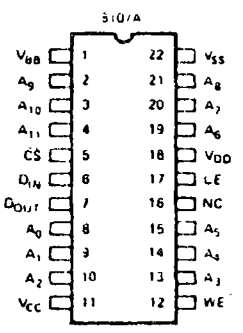
- Access Time -- 420 ns max.
- Refresh Period -- 2 ms
- Low Cost Per Bit
- Low Standby Power
- Easy System Interface
- Only One High Voltage Input Signal—Chip Enable
- Low Level Address, Data, Write Enable, Chip Select Inputs
- Address Registers Incorporated on the Chip
- Simple Memory Expansion—Chip Select Input Lead
- Fully Decoded—On Chip Address Decode
- Output is Three State and TTL Compatible
- Ceramic and Plastic 22-Pin DIPs

The Intel® 8107A is a 4096 word by 1 bit dynamic n-channel MOS RAM. It was designed for memory applications where very low cost and large bit storage are important design objectives. The 8107A uses dynamic circuitry which reduces the operation and standby power dissipation.

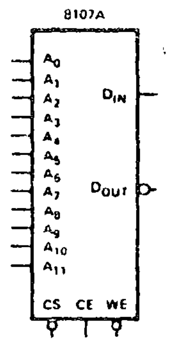
Reading information from the memory is non-destructive. Refreshing is accomplished by performing one read cycle on each of the 64 row addresses. Each row address must be refreshed every two milliseconds. The memory is refreshed whether Chip Select is a logic one or a logic zero.

The 8107A is fabricated with n-channel silicon gate technology. This technology allows the design and production of high performance, easy to use MOS circuits and provides a higher functional density on a monolithic chip than other MOS technologies.

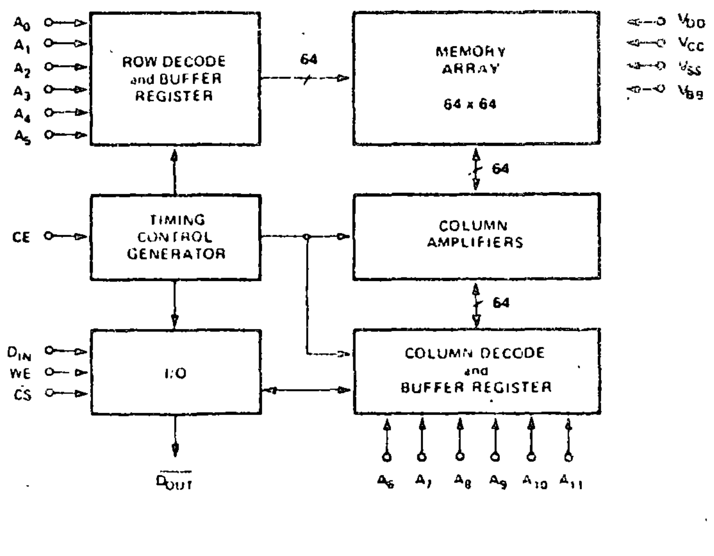
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

D_{IN}	DATA INPUT	CE	CHIP ENABLE
A_0-A_{11}	ADDRESS INPUTS*	$\overline{D_{OUT}}$	DATA OUTPUT
$\overline{WE}$	WRITE ENABLE	V_{CC}	POWER (+5V)
$\overline{CS}$	CHIP SELECT	NC	NOT CONNECTED

*Refresh Addresses A_0-A_5.

intel® Silicon Gate CMOS 5101, 5101-3, 5101L, 5101L-3

1024 BIT (256 x 4) STATIC CMOS RAM

*Ultra Low Standby Current: 15 nA/Bit for the 5101

- Fast Access Time — 650 ns
- Single +5 V Power Supply
- CE_2 Controls Unconditional Standby Mode
- Directly TTL Compatible — All Inputs and Outputs
- Three-State Output

The Intel® 5101 and 5101-3 are ultra-low power 1024 bit (256 words x 4-bits) static RAMs fabricated with an advanced ion-implanted silicon gate CMOS technology. The devices have two chip enable inputs. When CE_2 is at a low level, the minimum standby current is drawn by these devices, regardless of any other input transitions on the addresses and other control inputs. Also, when $\overline{CE}_1$ is at a high level and address and other control transitions are inhibited, the minimum standby current is drawn by these devices. When in standby the 5101 and 5101-3 draw from the single 5 volt supply only 15 microamps and 200 microamps, respectively. These devices are ideally suited for low power applications where battery operation or battery backup for non-volatility are required.

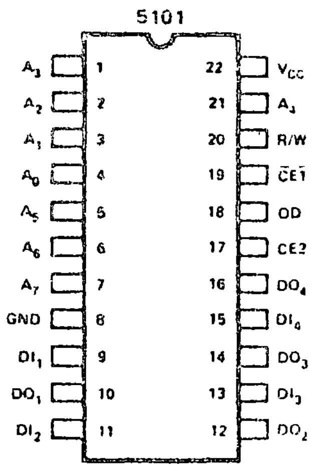
The 5101 and 5101-3 use fully DC stable (static) circuitry; it is not necessary to pulse chip select for each address transition. The data is read out non-destructively and has the same polarity as the input data. All inputs and outputs are directly TTL compatible. The 5101 and 5101-3 have separate data input and data output terminals. An output disable function is provided so that the data inputs and outputs may be wire OR-ed for use in common data I/O systems.

The 5101L and 5101L-3 are identical to the 5101 and 5101-3, respectively, with the additional feature of guaranteed data retention at a power supply voltage as low as 2.0 volts.

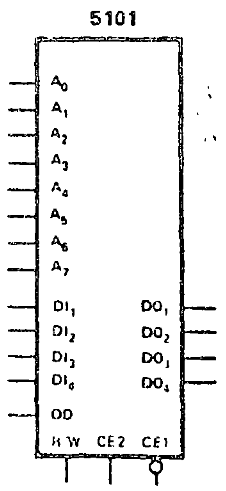
A pin compatible N-channel static RAM, the Intel 2101, is also available for low cost applications where a 256 x 4 organization is needed.

The Intel ion-implanted, silicon gate, complementary MOS (CMOS) allows the design and production of ultra-low power, high performance memories.

PIN CONFIGURATION



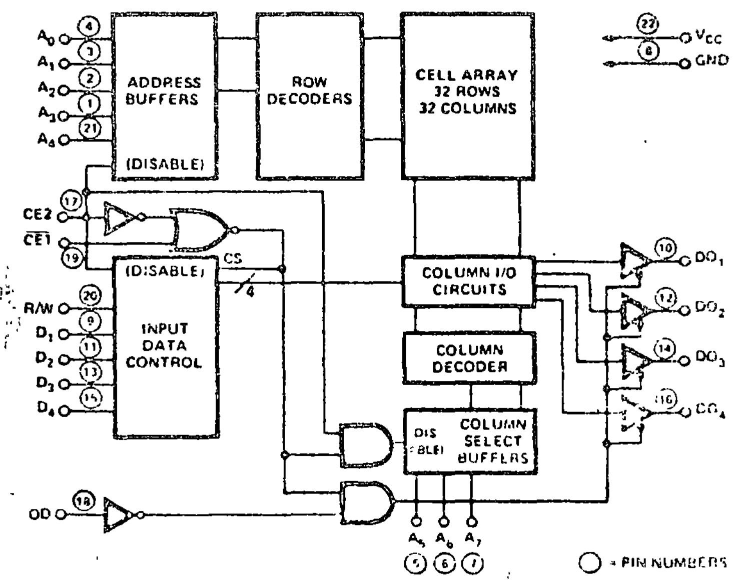
LOGIC SYMBOL



PIN NAMES

DI_1–DI_4	DATA INPUT	OD	OUTPUT DISABLE
A_0–A_7	ADDRESS INPUTS	DO_1–DO_4	DATA OUTPUT
R/W	READ/WRITE INPUT	V_{CC}	POWER (+5V)
$\overline{CE1}$, CE2	CHIP ENABLE		

BLOCK DIAGRAM



Silicon Gate MOS 8702A

2048 BIT ERASABLE AND ELECTRICALLY REPROGRAMMABLE READ ONLY MEMORY

- **Access Time — 1.3 μsec Max.**
- **Fast Programming — 2 Minutes for All 2048 Bits**
- **Fully Decoded, 256 x 8 Organization**
- **Static MOS — No Clocks Required**
- **Inputs and Outputs TTL Compatible**
- **Three-State Output — OR-Tie Capability**
- **Simple Memory Expansion Chip Select Input Lead**

The 8702A is a 256 word by 8 bit electrically programmable ROM ideally suited for microcomputer system development where fast turn-around and pattern experimentation are important. The 8702A undergoes complete programming and functional testing on each bit position prior to shipment, thus insuring 100% programmability.

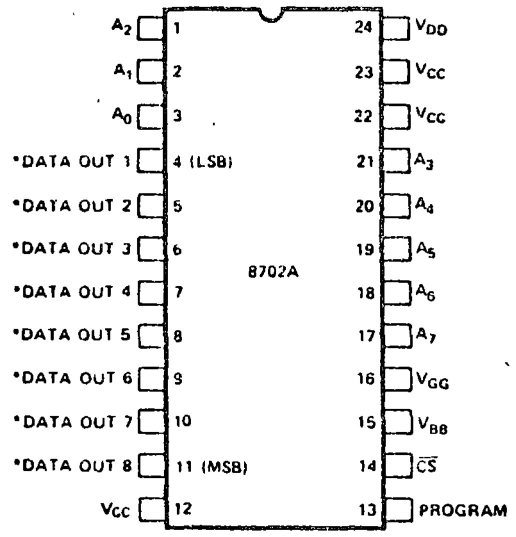
The 8702A is packaged in a 24 pin dual-in line package with a transparent quartz lid. The transparent quartz lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the device. This procedure can be repeated as many times as required.

The circuitry of the 8702A is entirely static; no clocks are required.

A pin-for-pin metal mask programmed ROM, the Intel 8302, is ideal for large volume production runs of systems initially using the 8702A.

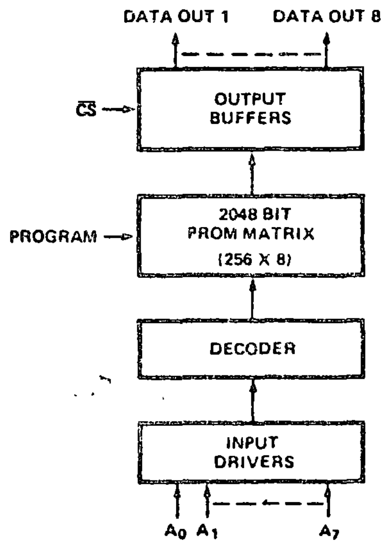
The 8702A is fabricated with silicon gate technology. This low threshold technology allows the design and production of higher performance MOS circuits and provides a higher functional density on a monolithic chip than conventional MOS technologies.

PIN CONFIGURATION



•THIS PIN IS THE DATA INPUT LEAD DURING PROGRAMMING.

BLOCK DIAGRAM



PIN NAMES

A_0-A_7	ADDRESS INPUTS
$\overline{CS}$	CHIP SELECT INPUT
DO_1-DO_2	DATA OUTPUTS

Silicon Gate MOS 8708/8704

8192/4096 BIT ERASABLE AND ELECTRICALLY REPROGRAMMABLE READ ONLY MEMORY

- 8708 1024x8 Organization
- 8704 512x8 Organization

- Fast Programming — Typ. 100 sec. For All 8K Bits
- Low Power During Programming
- Access Time — 450 ns
- Standard Power Supplies — +12V, ±5V
- Static — No Clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Modes
- Three-State Output — OR-Tie Capability

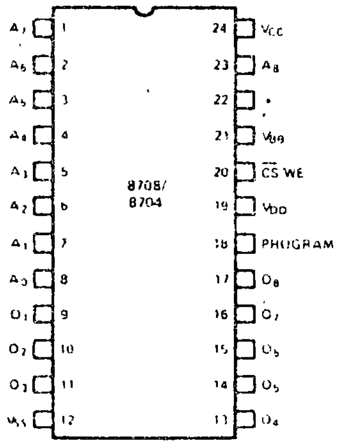
The Intel® 8708/8704 are high speed 8192/4096 bit erasable and electrically reprogrammable ROM's (EPROM) ideally suited where fast turn around and pattern experimentation are important requirements.

The 8708/8704 are packaged in a 24 pin dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the devices.

A pin for pin mask programmed ROM, the Intel® 8308, is available for large volume production runs of systems initially using the 8708.

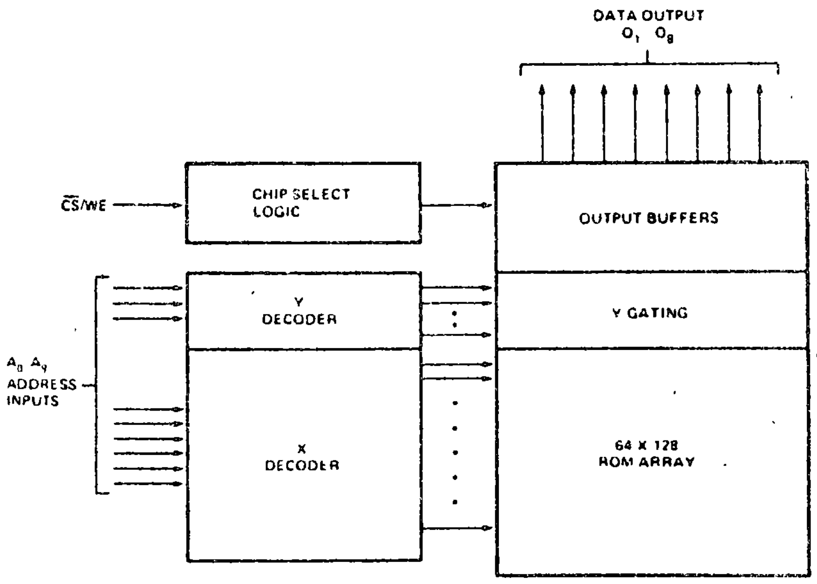
The 8708/8704 is fabricated with the time proven N-channel silicon gate technology.

PIN CONFIGURATIONS



*8704 - V_{SS}
8708 - A_9

BLOCK DIAGRAM



PIN NAMES

A_0 A_9	ADDRESS INPUTS
O_1 O_8	DATA OUTPUTS
$\overline{CS}$/WE	CHIP SELECT/WRITE ENABLE INPUT

Silicon Gate MOS 8302

2048 BIT MASK PROGRAMMABLE READ ONLY MEMORY

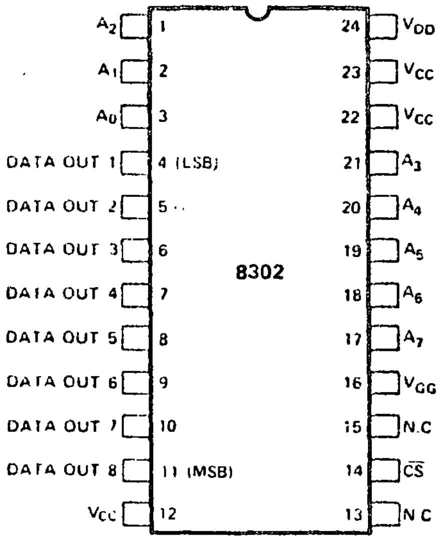
- **Access Time — 1 sec Max.**
- **Fully Decoded, 256 x 8 Organization**
- **Inputs and Outputs TTL Compatible**
- **Three-State Output — OR-Tie Capability**
- **Static MOS — No Clocks Required**
- **Simple Memory Expansion — Chip Select Input Lead**
- **24-Pin Dual-In-Line Hermetically Sealed Ceramic Package**

The Intel®8302 is a fully decoded 256 word by 8 bit metal mask ROM. It is ideal for large volume production runs of microcomputer systems initially using the 8702A erasable and electrically programmable ROM. The 8302 has the same pinning as the 8702A.

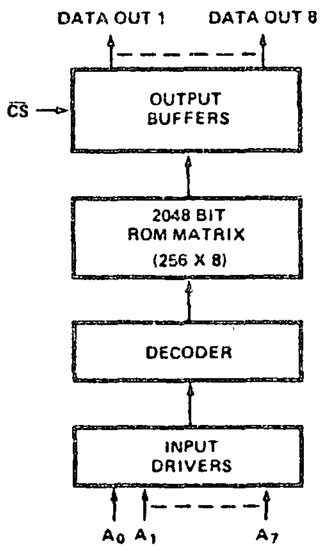
The 8302 is entirely static – no clocks are required. Inputs and outputs of the 8302 are TTL compatible. The output is three-state for OR-tie capability. A separate chip select input allows easy memory expansion. The 8302 is packaged in a 24 pin dual-in-line hermetically sealed ceramic package.

The 8302 is fabricated with p-channel silicon gate technology. This low threshold allows the design and production of higher performance MOS circuits and provides a higher functional density on a monolithic chip than conventional MOS technologies.

PIN CONFIGURATION



BLOCK DIAGRAM



PIN NAMES

A_0–A_7	ADDRESS INPUTS
$\overline{CS}$	CHIP SELECT INPUT
DO_1–DO_8	DATA OUTPUTS

Silicon Gate MOS 8308

8192 BIT STATIC MOS READ ONLY MEMORY
Organization — 1024 Words x 8 Bits

- Fast Access — 450 ns
- Directly Compatible with 8080 CPU at Maximum Processor Speed
- Two Chip Select Inputs for Easy Memory Expansion
- Directly TTL Compatible — All Inputs and Outputs
- Three State Output — OR-Tie Capability
- Fully Decoded
- Standard Power Supplies +12V DC, ±5V DC

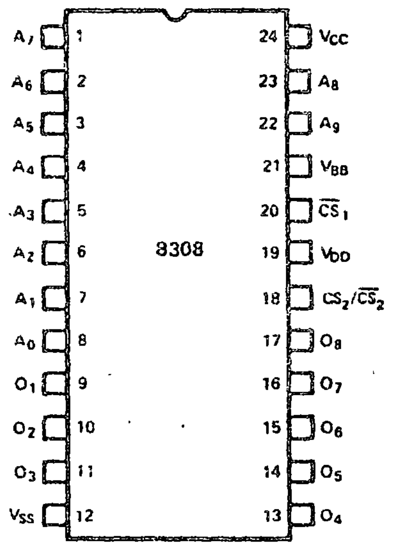
The Intel® 8308 is an 8,192 bit static MOS mask programmable Read Only Memory organized as 1024 words by 8-bits. This ROM is designed for 8080 microcomputer system applications where high performance, large bit storage, and simple interfacing are important design objectives. The inputs and outputs are fully TTL compatible.

A pin for pin compatible electrically programmed erasable ROM, the Intel® 8708, is available for system development and small quantity production use.

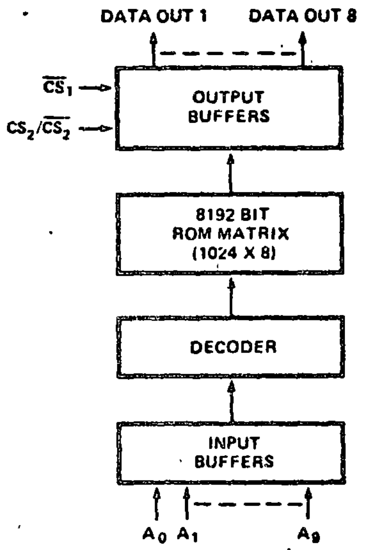
Two Chip Selects are provided — $\overline{CS}_1$ which is negative true, and $CS_2/\overline{CS}_2$ which may be programmed either negative or positive true at the mask level.

The 8308 read only memory is fabricated with N-channel silicon gate technology. This technology provides the designer with high performance, easy-to-use MOS circuits.

PIN CONFIGURATION



BLOCK DIAGRAM



PIN NAMES

A_0-A_9	ADDRESS INPUTS
O_1-O_8	DATA OUTPUTS
$\overline{CS}_1$, CS_2	CHIP SELECT INPUTS

intel Silicon Gate MOS ROM 8316A

16,384 BIT STATIC MOS READ ONLY MEMORY
Organization—2048 Words x 8 Bits
Access Time-850 ns max

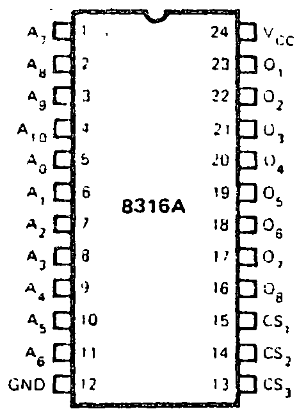
- Single +5 Volts Power Supply Voltage
- Directly TTL Compatible — All Inputs and Outputs
- Low Power Dissipation of 31.4 μW/Bit Maximum
- Three Programmable Chip Select Inputs for Easy Memory Expansion
- Three-State Output — OR-Tie Capability
- Fully Decoded — On Chip Address Decode
- Inputs Protected — All Inputs Have Protection Against Static Charge

The Intel® 8316A is a 16,384-bit static MOS read only memory organized as 2048 words by 8 bits. This ROM is designed for microcomputer memory applications where high performance, large bit storage, and simple interfacing are important design objectives.

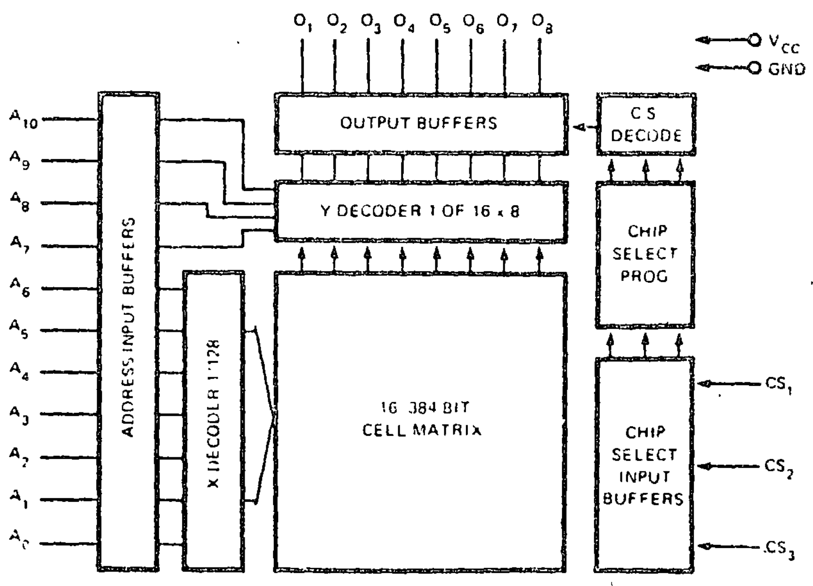
The inputs and outputs are fully TTL compatible. This device operates with a single +5V power supply. The three chip select inputs are programmable. Any combination of active high or low level chip select inputs can be defined and the desired chip select code is fixed during the masking process. These three programmable chip select inputs, as well as OR-tie compatibility on the outputs, facilitate easy memory expansion.

The 8316A read only memory is fabricated with N-channel silicon gate technology. This technology provides the designer with high performance, easy-to-use MOS circuits. Only a single +5V power supply is needed and all devices are directly TTL compatible.

PIN CONFIGURATION



BLOCK DIAGRAM



PIN NAMES

A_0–A_{10}	ADDRESS INPUTS
O_1–O_8	DATA OUTPUTS
CS_1–CS_3	PROGRAMMABLE CHIP SELECT INPUTS

2.6 Interfases.

Schottky Bipolar 8212

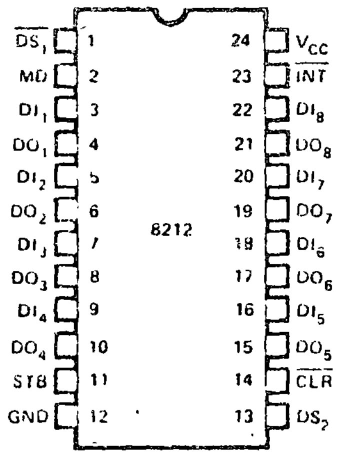
EIGHT-BIT INPUT/OUTPUT PORT

- **Fully Parallel 8-Bit Data Register and Buffer**
- **Service Request Flip-Flop for Interrupt Generation**
- **Low Input Load Current — .25 mA Max.**
- **Three State Outputs**
- **Outputs Sink 15 mA**
- **3.65V Output High Voltage for Direct Interface to 8080 CPU or 8008 CPU**
- **Asynchronous Register Clear**
- **Replaces Buffers, Latches and Multiplexers in Micro-computer Systems**
- **Reduces System Package Count**

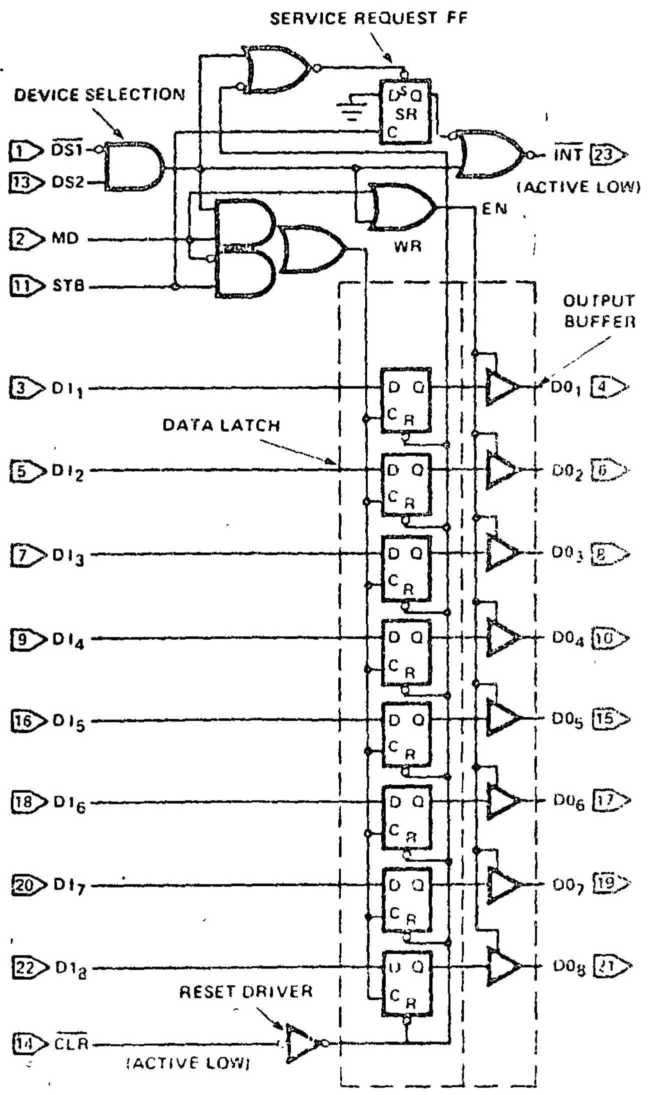
The 8212 input/output port consists of an 8-bit latch with 3-state output buffers along with control and device selection logic. Also included is a service request flip-flop for the generation and control of interrupts to the microprocessor.

The device is multimode in nature. It can be used to implement latches, gated buffers or multiplexers. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with this device.

PIN CONFIGURATION



LOGIC DIAGRAM



PIN NAMES

DI_1 DI_8	DATA IN
DO_1 DO_8	DATA OUT
DS_1 DS_2	DEVICE SELECT
MD	MODE
STB	STROBE
$\overline{INT}$	INTERRUPT (ACTIVE LOW)
$\overline{CLR}$	CLEAR (ACTIVE LOW)

Functional Description

Data Latch

The 8 flip-flops that make up the data latch are of a "D" type design. The output (Q) of the flip-flop will follow the data input (D) while the clock input (C) is high. Latching will occur when the clock (C) returns low.

The data latch is cleared by an asynchronous reset input ($\overline{CLR}$). (Note: Clock (C) Overides Reset ($\overline{CLR}$).)

Output Buffer

The outputs of the data latch (Q) are connected to 3-state, non-inverting output buffers. These buffers have a common control line (EN); this control line either enables the buffer to transmit the data from the outputs of the data latch (Q) or disables the buffer, forcing the output into a high impedance state. (3-state)

This high-impedance state allows the designer to connect the 8212 directly onto the microprocessor bi-directional data bus.

Control Logic

The 8212 has control inputs $\overline{DS1}$, DS2, MD and STB. These inputs are used to control device selection, data latching, output buffer state and service request flip-flop

$\overline{DS1}$, DS2 (Device Select)

These 2 inputs are used for device selection. When $\overline{DS1}$ is low and DS2 is high ($\overline{DS1} \cdot DS2$) the device is selected. In the selected state the output buffer is enabled and the service request flip-flop (SR) is asynchronously set.

MD (Mode)

This input is used to control the state of the output buffer and to determine the source of the clock input (C) to the data latch

When MD is high (output mode) the output buffers are enabled and the source of clock (C) to the data latch is from the device selection logic ($\overline{DS1} \cdot DS2$).

When MD is low (input mode) the output buffer state is determined by the device selection logic ($\overline{DS1} \cdot DS2$) and the source of clock (C) to the data latch is the STB (Strobe) input

STB (Strobe)

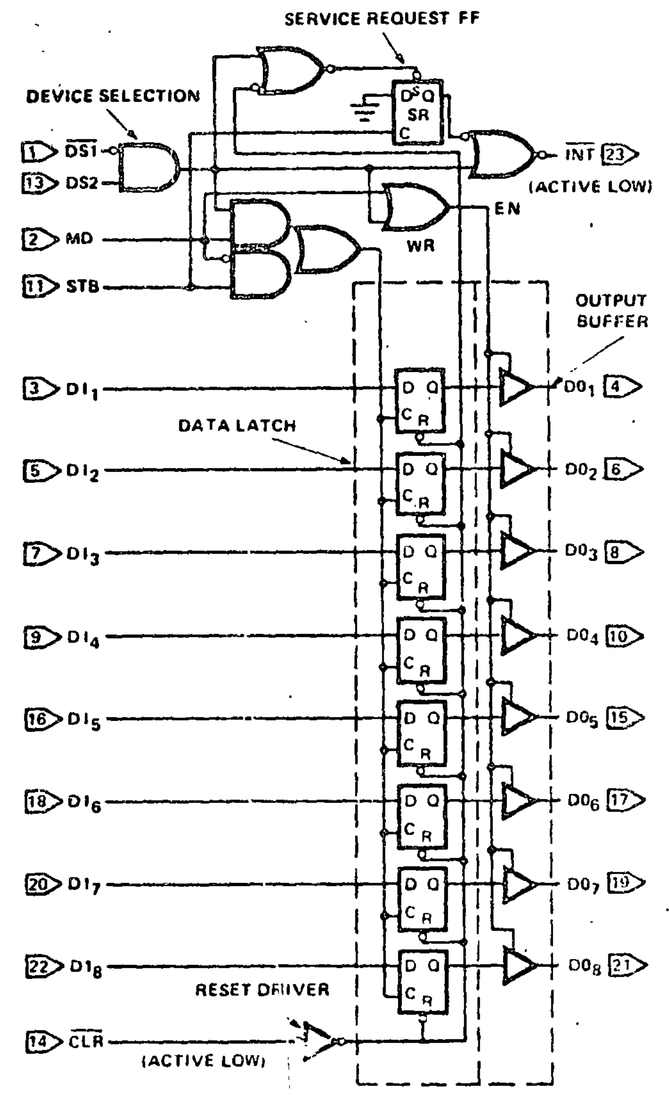
This input is used as the clock (C) to the data latch for the input mode MD = 0) and to synchronously reset the service request flip-flop (SR).

Note that the SR flip flop is negative edge triggered.

Service Request Flip-Flop

The (SR) flip-flop is used to generate and control interrupts in microcomputer systems. It is asynchronously set by the $\overline{CLR}$ input (active low). When the (SR) flip-flop is set it is in the non-interrupting state.

The output of the (SR) flip-flop (Q) is connected to an inverting input of a "NOR" gate. The other input to the "NOR" gate is non-inverting and is connected to the device selection logic ($\overline{DS1} \cdot DS2$). The output of the "NOR" gate ($\overline{INT}$) is active low (interrupting state) for connection to active low input priority generating circuits.



STB	MD	$(\overline{DS_1} \cdot DS_2)$	DATA OUT EQUALS
0	0	0	3 STATE
1	0	0	3 STATE
0	1	0	DATA LATCH
1	1	0	DATA LATCH
0	0	1	DATA LATCH
1	0	1	DATA IN
0	1	1	DATA IN
1	1	1	DATA IN

$\overline{CLR}$ – RESETS DATA LATCH
SETS SR FLIP FLOP
(NO EFFECT ON OUTPUT BUFFER)

$\overline{CLR}$	$(\overline{DS_1} \cdot DS_2)$	STB	*SR	$\overline{INT}$
0	0	0	1	1
0	1	0	1	0
1	1	¯_	0	0
1	1	0	1	0
1	0	0	1	1
1	1	¯_	1	0

*INTERNAL SR FLIP FLOP

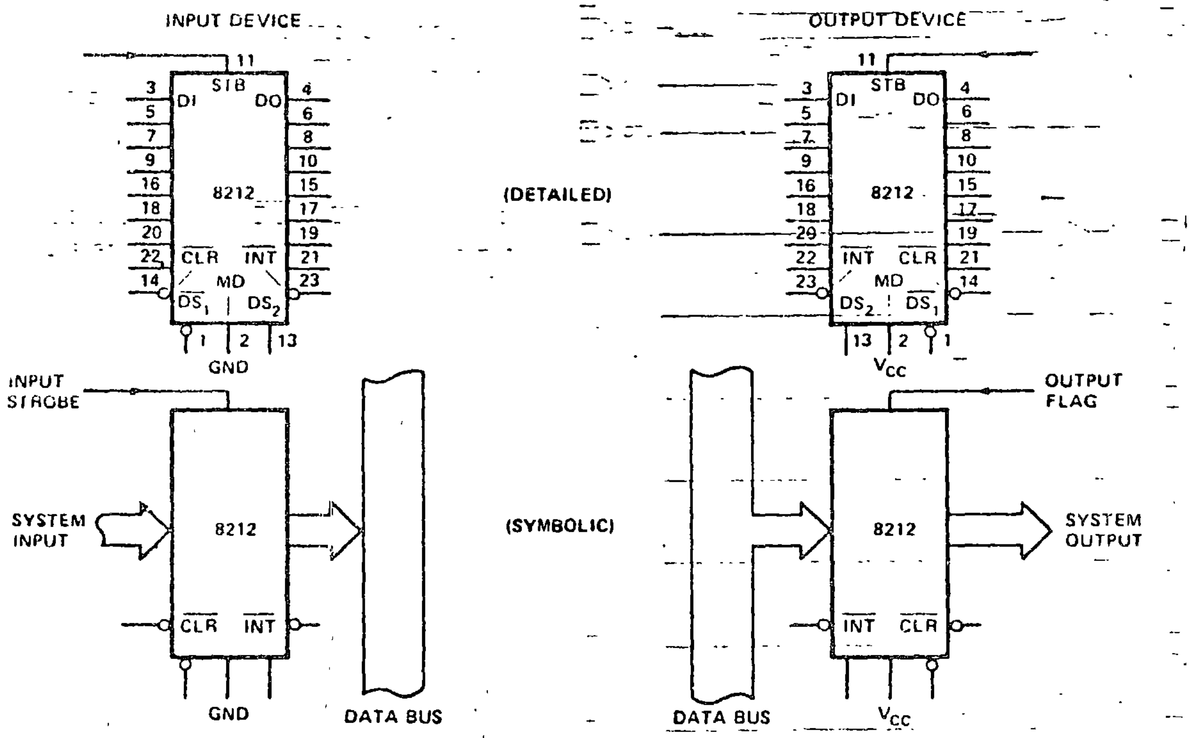
Applications Of The 8212 -- For Microcomputer Systems

I	Basic Schematic Symbol	VII	8080 Status Latch
II	Gated Buffer	VIII	8008 System
III	Bi-Directional Bus Driver	IX	8080 System:
IV	Interrupting Input Port		8 Input Ports
V	Interrupt Instruction Port		8 Output Ports
VI	Output Port		8 Level Priority Interrupt

I. Basic Schematic Symbols

Two examples of ways to draw the 8212 on system schematics—(1) the top being the detailed view showing pin numbers, and (2) the bottom being the symbolic view showing the system input or output as a system bus (bus containing 8 parallel lines). The output to the data bus is symbolic in referencing 8 parallel lines.

BASIC SCHEMATIC SYMBOLS



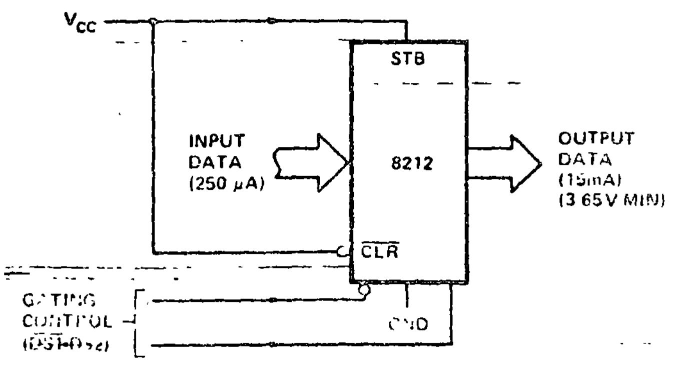
II. Gated Buffer (3 - STATE)

The simplest use of the 8212 is that of a gated buffer. By tying the mode signal low and the strobe input high, the data latch is acting as a straight through gate. The output buffers are then enabled from the device selection logic $\overline{DS1}$ and DS2.

When the device selection logic is false, the outputs are 3-state.

When the device selection logic is true, the input data from the system is directly transferred to the output. The input data load is 250 micro amps. The output data can sink 15 milli amps. The minimum high output is 3.65 volts

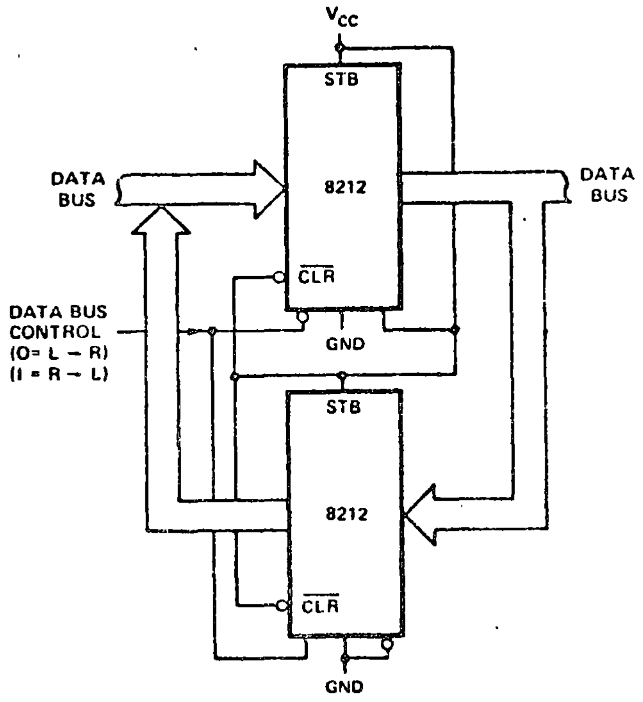
GATED BUFFER
3-STATE



III. Bi-Directional Bus Driver

A pair of 8212's wired (back-to-back) can be used as a symmetrical drive, bi-directional bus driver. The devices are controlled by the data bus input control which is connected to $\overline{DS1}$ on the first 8212 and to DS2 on the second. One device is active, and acting as a straight thr[illegible]gh buffer the other is in 3-state mode. This is a very useful circuit in small system-design.

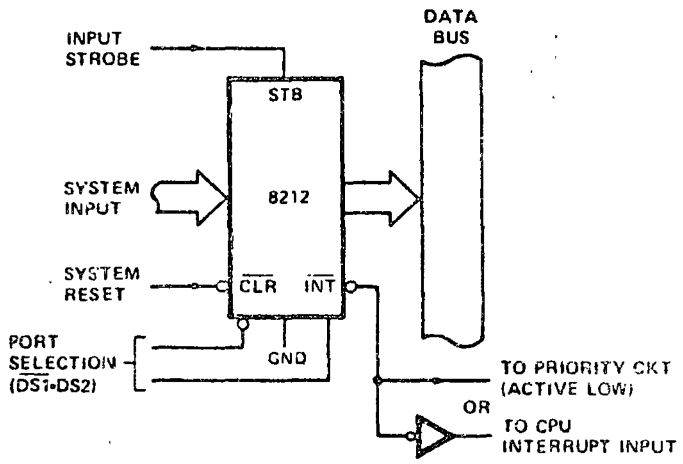
BI-DIRECTIONAL BUS DRIVER



IV. Interrupting Input Port

This use of an 8212 is that of a system input port that accepts a strobe from the system input source, which in turn clears the service request flip-flop and interrupts the processor. The processor then goes through a service routine, identifies the port, and causes the device selection logic to go true — enabling the system input data onto the data bus.

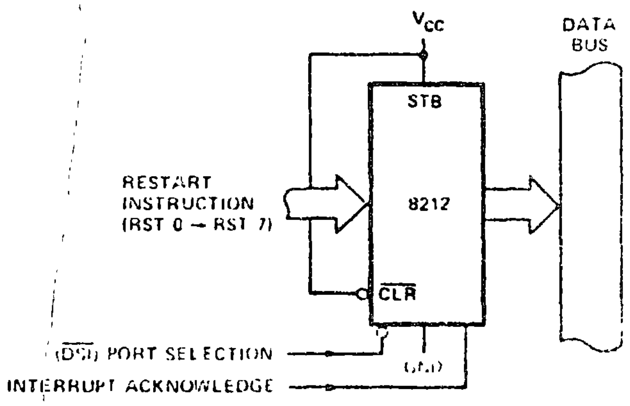
INTERRUPTING INPUT PORT



V. Interrupt Instruction Port

The 8212 can be used to gate the interrupt instruction, normally RESTART instructions, onto the data bus. The device is enabled from the interrupt acknowledge signal from the microprocessor and from a port selection signal. This signal is normally tied to ground. ($\overline{DS1}$ could be used to multiplex a variety of interrupt instruction ports onto a common bus)

INTERRUPT INSTRUCTION PORT

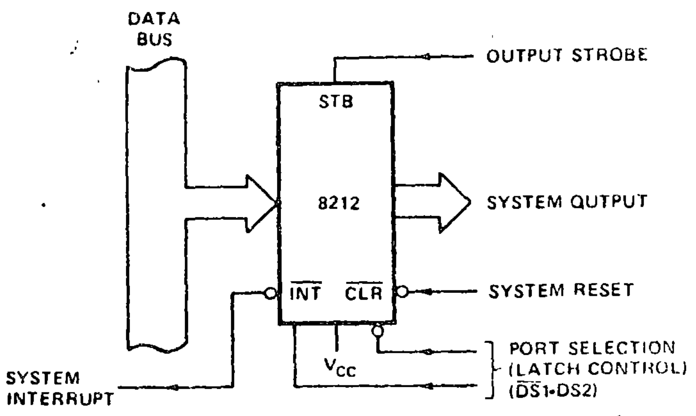


V_{CC}
DATA BUS
STB
RESTART INSTRUCTION (RST 0 – RST 7)
8212
CLR
($\overline{DS1}$) PORT SELECTION
GND
INTERRUPT ACKNOWLEDGE

VI. Output Port (With Hand-Shaking)

The 8212 can be used to transmit data from the data bus to a system output. The output strobe could be a hand-shaking signal such as "reception of data" from the device that the system is outputting to. It in turn, can interrupt the system signifying the reception of data. The selection of the port comes from the device selection logic. ($\overline{DS1}$ • DS2)

OUTPUT PORT (WITH HAND-SHAKING)



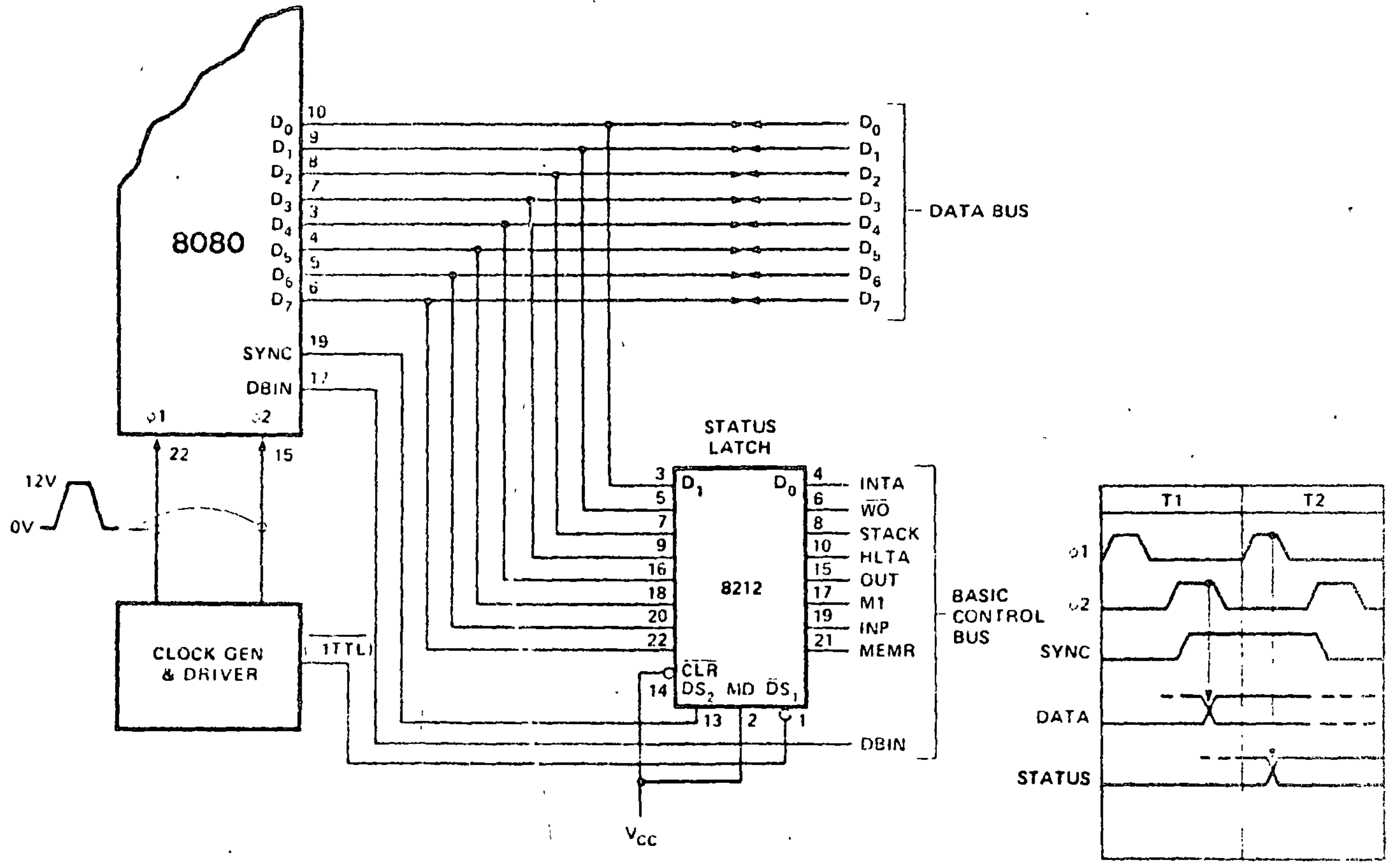
VII. 8080 Status Latch

Here the 8212 is used as the status latch for an 8080 microcomputer system. The input to the 8212 latch is directly from the 8080 data bus. Timing shows that when the SYNC signal is true, which is connected to the DS2 input and the phase 1 signal is true, which is a TTL level coming from the clock generator; then, the status data will be latched into the 8212.

Note: The mode signal is tied high so that the output on the latch is active and enabled all the time.

It is shown that the two areas of concern are the bidirectional data bus of the microprocessor and the control bus.

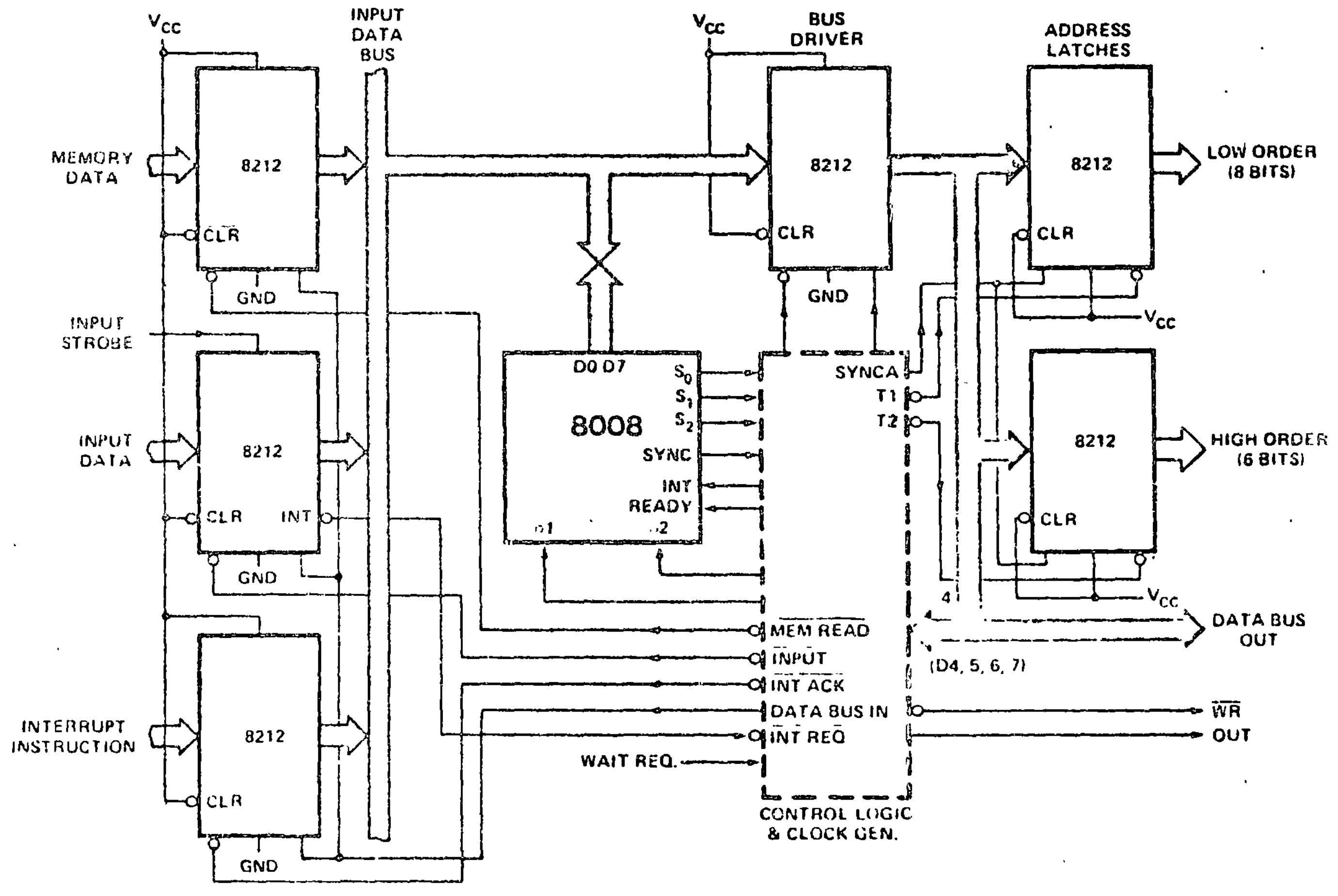
8080 STATUS LATCH



VIII. 8008 System

This shows the 8212 used in an 8008 microcomputer system. They are used to multiplex the data from three different sources onto the 8008 input data bus. The three sources of data are: memory data, input data, and the interrupt instruction. The 8212 is also used as the uni-directional bus driver to provide a proper drive to the address latches (both low order and high order are also 8212's) and to provide adequate drive to the output data bus. The control of these six 8212's in the 8008 system is provided by the control logic and clock generator circuits. These circuits consist of flip-flops, decoders, and gates to generate the control functions necessary for 8008 microcomputer systems. Also note that the input data port has a strobe input. This allows the processor to be interrupted from the input port directly. The control of the input bus consists of the data bus input signal, control logic, and the appropriate status signal for bus discipline whether memory read, input, or interrupt acknowledge. The combination of these four signals determines which one of these three devices will have access to the input data bus. The bus driver, which is implemented in an 8212, is also controlled by the control logic and clock generator so it can be 3-stated when necessary and also as a control transmission device to the address latches. Note: The address latches can be 3-stated for DMA purposes and they provide 15 milli amps drive, sufficient for large bus systems.

8008 SYSTEM



IX. 8080 System

This drawing shows the 8212 used in the I/O section of an 8080 microcomputer system. The system consists of 8 input ports, 8 output ports, 8 level priority systems, and a bidirectional bus driver. (The data bus within the system is darkened for emphasis). Basically, the operation would be as follows: The 8 ports, for example, could be connected to 8 keyboards, each keyboard having its own priority level. The keyboard could provide a strobe input of its own which would clear the service request flip-flop. The $\overline{INT}$ signals are connected to an 8 level priority encoding circuit. This circuit provides a positive true level to the central processor (INT) along with a three-bit code to the interrupt instruction port for the generation of RESTART instructions. Once the processor has been interrupted and it acknowledges the reception of the interrupt, the Interrupt Acknowledge signal is generated. This signal transfers data in the form of a RESTART instruction onto the buffered data bus. When the DBIN signal is true this RESTART instruction is gated into the microcomputer, in this case, the 8080 CPU. The 8080 then performs a software controlled interrupt service routine, saving the status of its current operation in the push-down stack and performing an INPUT instruction. The INPUT instruction thus sets the INP status bit, which is common to all input ports.

Also present is the address of the device on the 8080 address bus which in this system is connected to an 8205, one out of eight decoder with active low outputs. These active low outputs will enable one of the input ports, the one that interrupted the processor, to put its data onto the buffered data bus to be transmitted to the CPU when the data bus input signal is true. The processor can also output data from the 8080 data bus to the buffered data bus when the data bus input signal is false. Using the same address selection technique from the 8205 decoder and the output status bit, we can select with this system one of eight output ports to transmit the data to the system's output device structure.

Note: This basic I/O configuration for the 8080 can be expanded to 256 input devices and 256 output devices all using 8212 and, of course, the appropriate decoding.

Note that the 8080 is a 3.3-volt minimum high input requirement and that the 8212 has a 3.65-volt minimum high output providing the designer with a 350 milli volt noise margin worst case for 8080 systems when using the 8212.

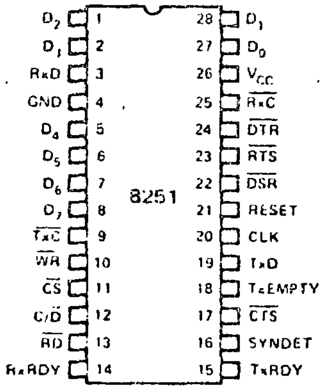
Silicon Gate MOS 8251

PROGRAMMABLE COMMUNICATION INTERFACE

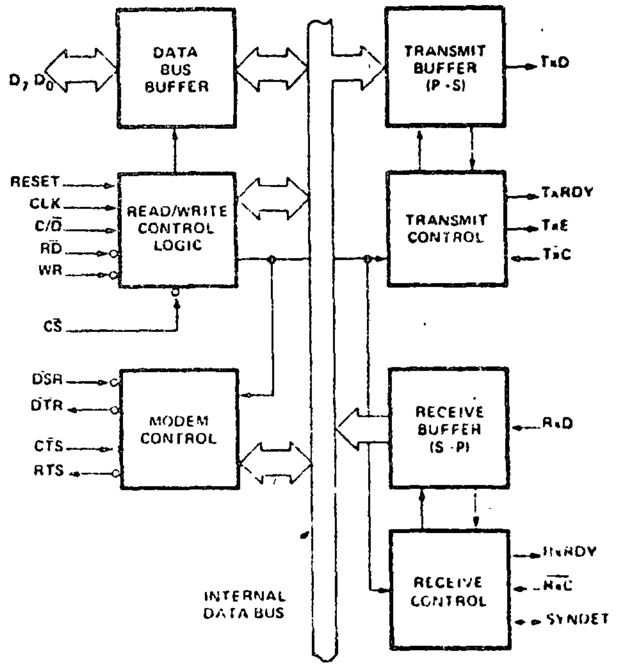
- Synchronous and Asynchronous Operation
  - Synchronous:
    5-8 Bit Characters
    Internal or External Character Synchronization
    Automatic Sync Insertion
  - Asynchronous:
    5-8 Bit Characters
    Clock Rate — 1, 16 or 64 Times Baud Rate
    Break Character Generation
    1, 1½, or 2 Stop Bits
    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



BLOCK DIAGRAM



Pin Name	Pin Function
D7-D0	Data Bus (8 bits)
C/D	Control or Data is to be Written or Read
RD	Read Data Command
WR	Write Data or Control Command
CS	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
TxC	Transmitter Clock
TxD	Transmitter Data
RxC	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for character from 8080)

Pin Name	Pin Function
DSR	Data Set Ready
DTR	Data Terminal Ready
SYNDET	Sync Detect
RTS	Request to Send Data
CTS	Clear to Send Data
TxE	Transmitter Empty
Vcc	+5 Volt Supply
GND	Ground

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 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)

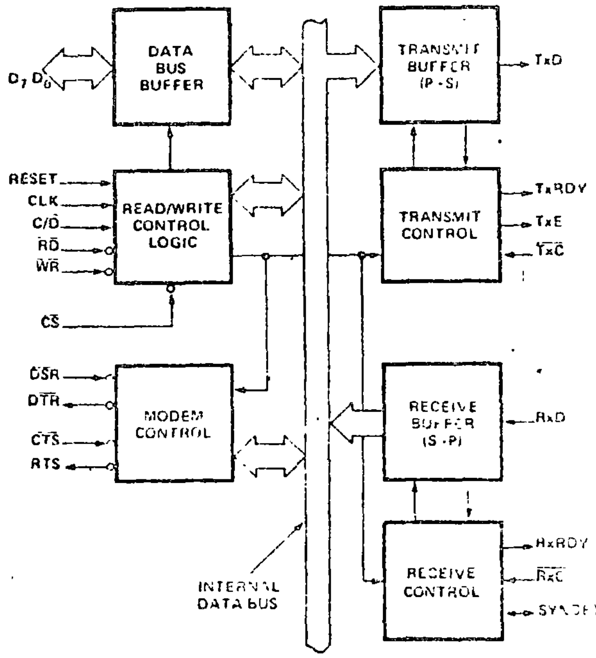
A "low" on this input informs the 8251 that the CPU is inputting data or status information, in essence, the CPU is reading from the 8251.

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.

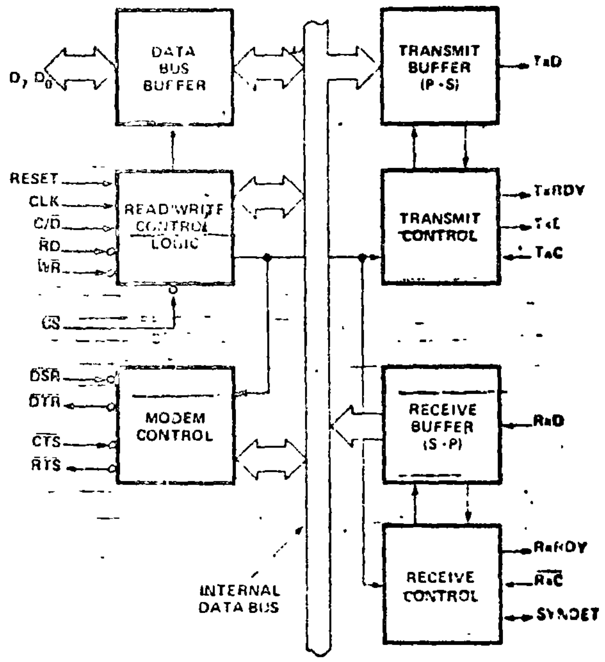
TxRDY (Transmitter Ready)

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 has no characters to transmit, the TxE output will go "high". It resets automatically upon receiving a character from the CPU. TxE can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplexed operational mode.

In SYNChronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be transmitted automatically as "fillers".



$\overline{TxC}$ (Transmitter Clock)

The Transmitter Clock 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 equals 110 Baud,
$\overline{TxC}$ equals 110 Hz (1x)
$\overline{TxC}$ equals 1.76 kHz (16x)
$\overline{TxC}$ equals 7.04 kHz (64x).
If Baud Rate equals 9600 Baud,
$\overline{TxC}$ equals 614.4 kHz (64x).

The falling edge of $\overline{TxC}$ shifts the serial data out 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.

$\overline{\text{RxC}}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the frequency of $\overline{\text{RxC}}$ is equal to the actual Baud Rate (1x). In Asynchronous Mode, the frequency of $\overline{\text{RxC}}$ 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,
	$\overline{\text{RxC}}$ equals 300 Hz (1x)
	$\overline{\text{RxC}}$ equals 4800 Hz (16x)
	$\overline{\text{RxC}}$ equals 19.2 kHz (64x).
	If Baud Rate equals 2400 Baud,
	$\overline{\text{RxC}}$ equals 2400 Hz (1x)
	$\overline{\text{RxC}}$ equals 38.4 kHz (16x)
	$\overline{\text{RxC}}$ equals 153.6 kHz (64x).

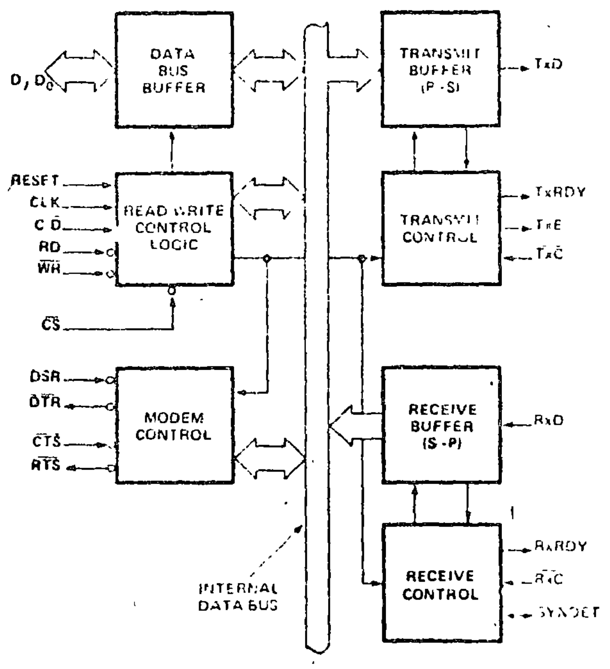
Data is sampled into the 8251 on the rising edge of $\overline{\text{RxC}}$.

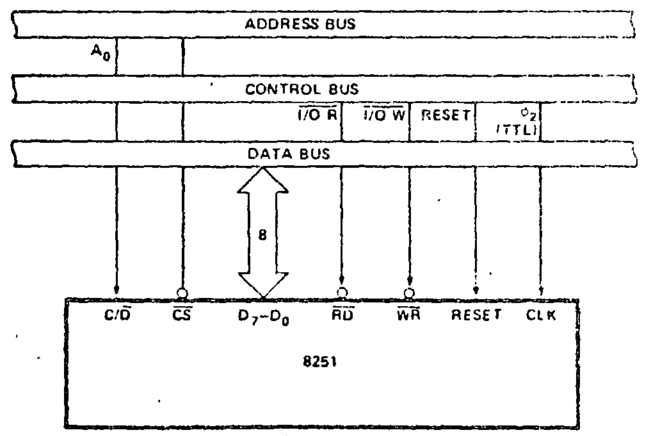
NOTE: In most communications systems, the 8251 will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$ will require identical frequencies for this operation 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 $\overline{\text{RxC}}$. 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 $\overline{\text{RxC}}$.





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

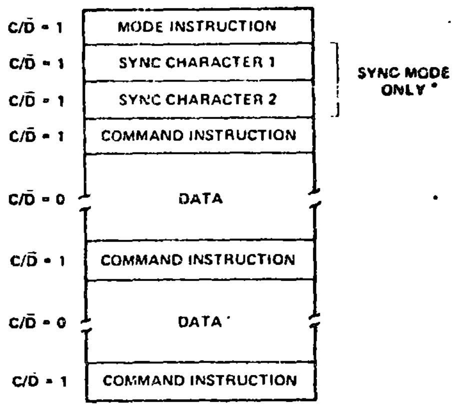
This format defines the general operational characteristics of the 8251. It must follow a Reset operation (internal or external). Once the Mode instruction has been written into the 8251 by the CPU, SYNC characters or Command instructions may be inserted.

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.

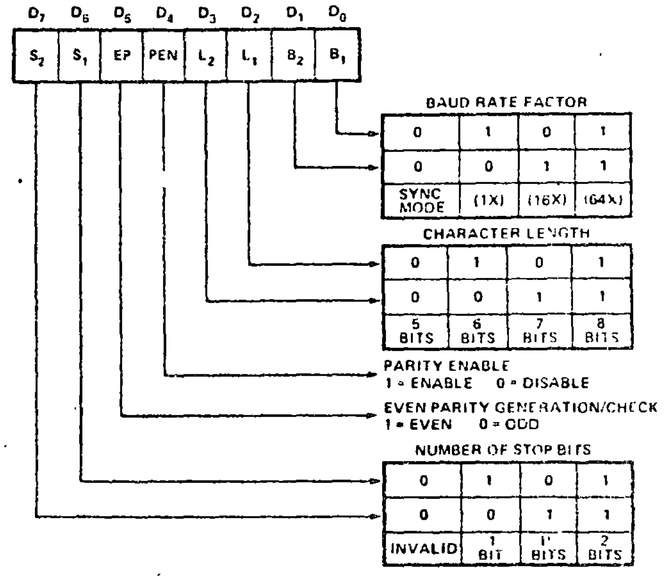
Asynchronous Mode (Transmission)

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.

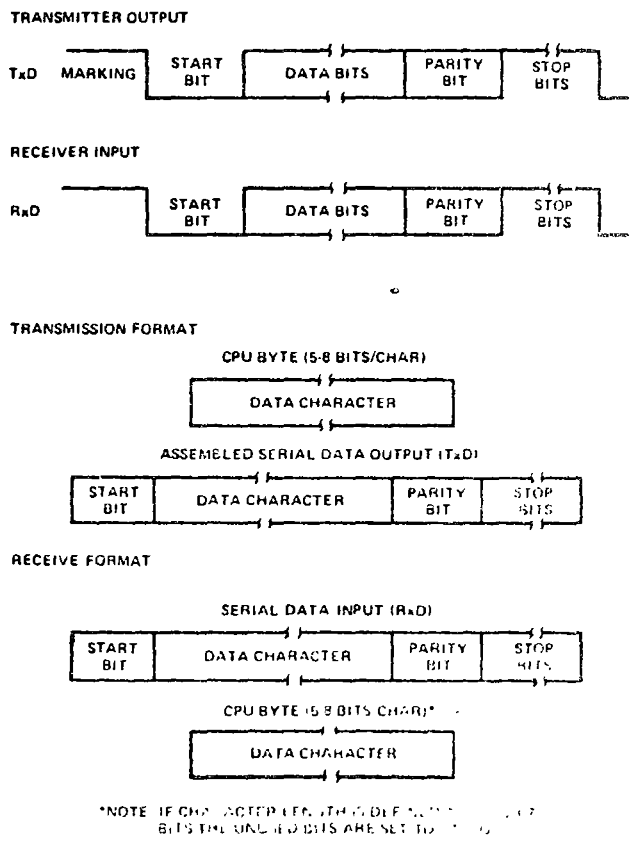
When no data characters have loaded into the 8251 the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

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.



Mode Instruction Format, Asynchronous Mode



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.

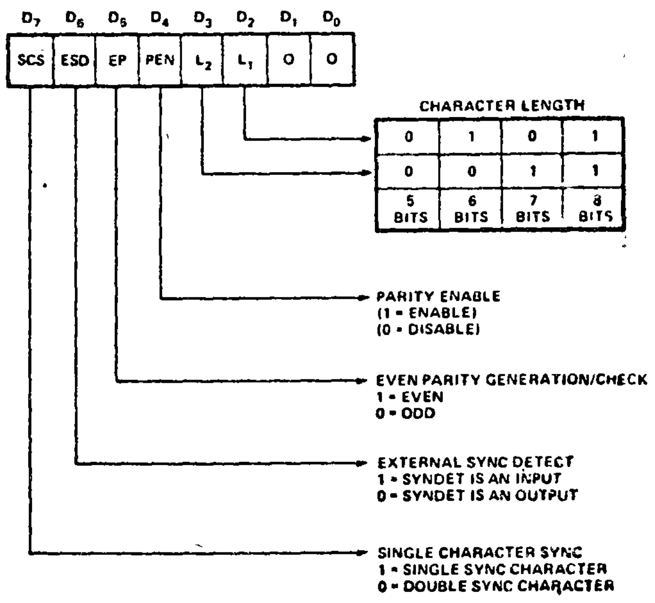
Synchronous Mode (Receive)

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.

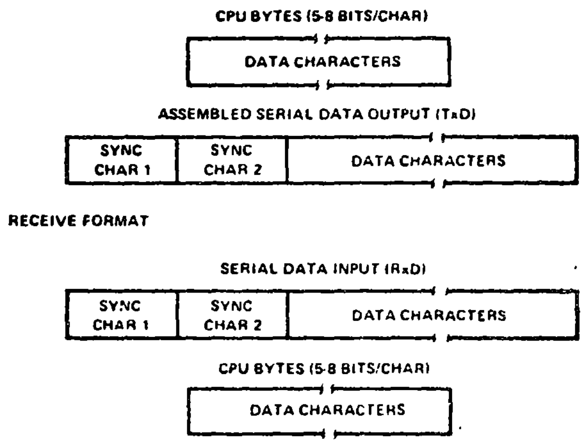
Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost.



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SCS ESD EP PEN L_2 L_1 0 0
CHARACTER LENGTH
0 1 0 1
0 0 1 1
5 BITS 6 BITS 7 BITS 8 BITS
PARITY ENABLE (1 = ENABLE) (0 = DISABLE)
EVEN PARITY GENERATION/CHECK 1 = EVEN 0 = ODD
EXTERNAL SYNC DETECT 1 = SYNDET IS AN INPUT 0 = SYNDET IS AN OUTPUT
SINGLE CHARACTER SYNC 1 = SINGLE SYNC CHARACTER 0 = DOUBLE SYNC CHARACTER

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.

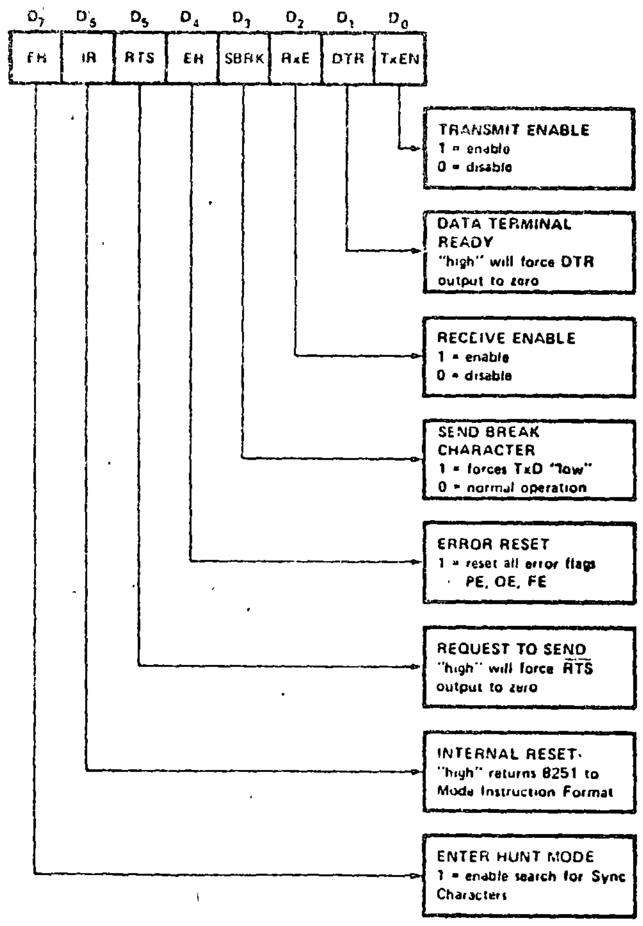
Once the Mode Instruction has been written into the 8251 and Sync characters inserted, if necessary, then all further "control writes" (C/$\overline{D}$ = 1) will load the Command Instruction. A Reset operation (internal or external) will return the 8251 to the Mode Instruction Format.

STATUS READ DEFINITION

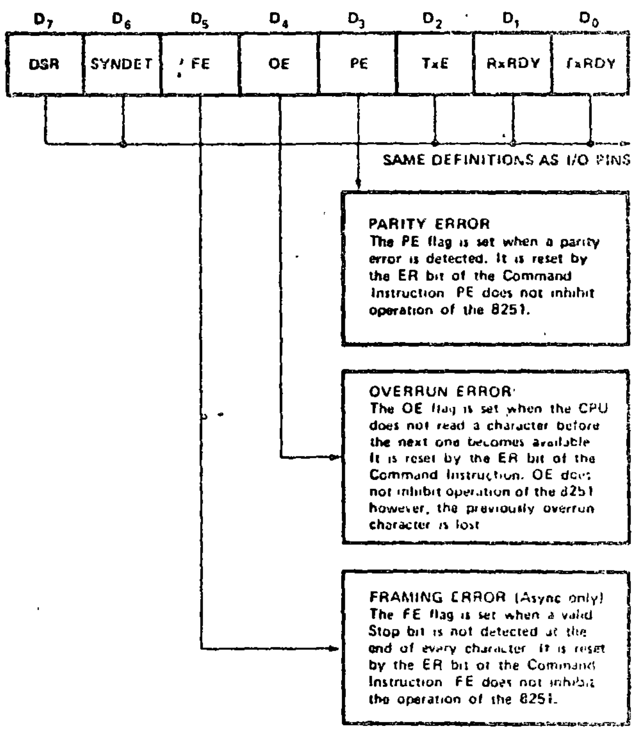
In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251 has facilities that allow the programmer to "read" the status of the device at any time during the 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.

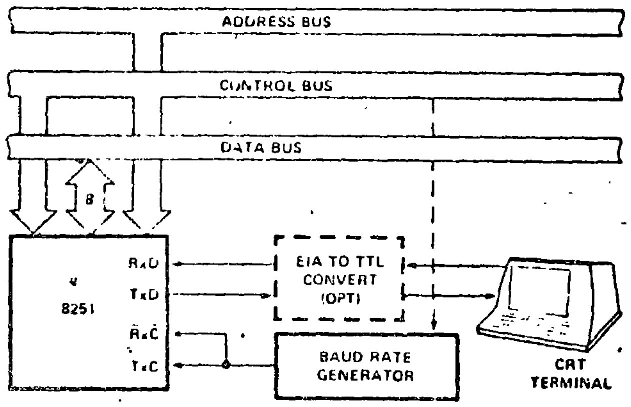

D7 D6 D5 D4 D3 D2 D1 D0
EH
IR
RTS
ER
SBRK
RxE
DTR
TxEN
TRANSMIT ENABLE 1 = enable 0 = disable
DATA TERMINAL READY "high" will force DTR output to zero
RECEIVE ENABLE 1 = enable 0 = disable
SEND BREAK CHARACTER 1 = forces TxD "low" 0 = normal operation
ERROR RESET 1 = reset all error flags PE, OE, FE
REQUEST TO SEND "high" will force RTS output to zero
INTERNAL RESET "high" returns 8251 to Mode Instruction Format
ENTER HUNT MODE 1 = enable search for Sync Characters

Command Instruction Format

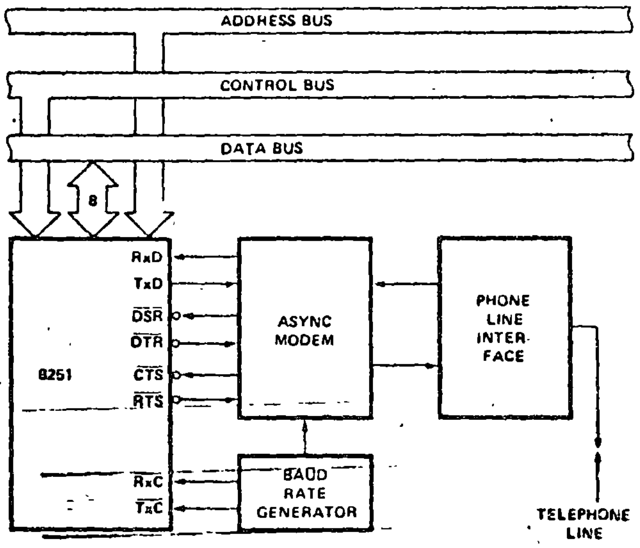

D7 D6 D5 D4 D3 D2 D1 D0
DSR
SYNDET
FE
OE
PE
TxE
RxRDY
TxRDY
SAME DEFINITIONS AS I/O PINS
PARITY ERROR The PE flag is set when a parity error is detected. It is reset by the ER bit of the Command Instruction. PE does not inhibit operation of the 8251.
OVERRUN ERROR The OE flag is set when the CPU does not read a character before the next one becomes available. It is reset by the ER bit of the Command Instruction. OE does not inhibit operation of the 8251 however, the previously overrun character is lost.
FRAMING ERROR (Async only) The FE flag is set when a valid Stop bit is not detected at the end of every character. It is reset by the ER bit of the Command Instruction. FE does not inhibit the operation of the 8251.

Status Read Format

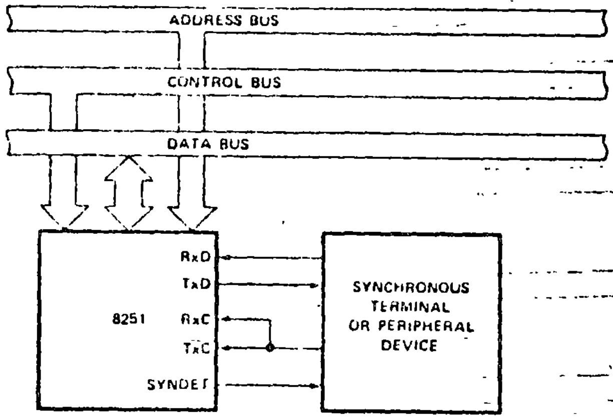
APPLICATIONS OF THE 8251



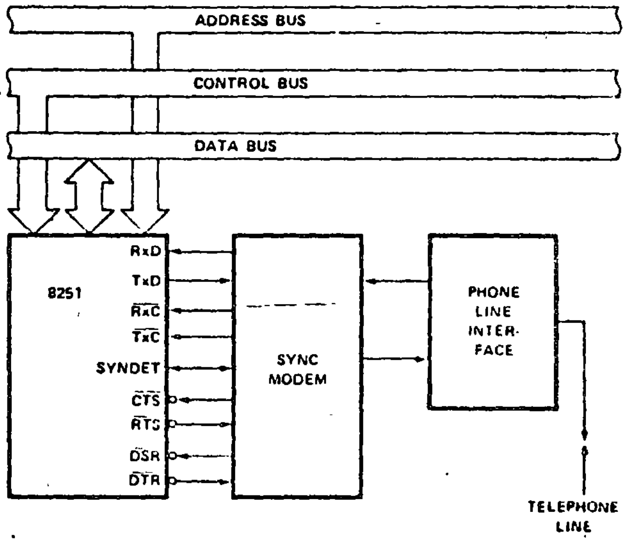
Asynchronous Serial Interface to CRT Terminal, DC-9600 Baud



Asynchronous Interface to Telephone Lines



Synchronous Interface to Terminal or Peripheral Device



Synchronous Interface to Telephone Lines

Silicon Gate MOS 8255

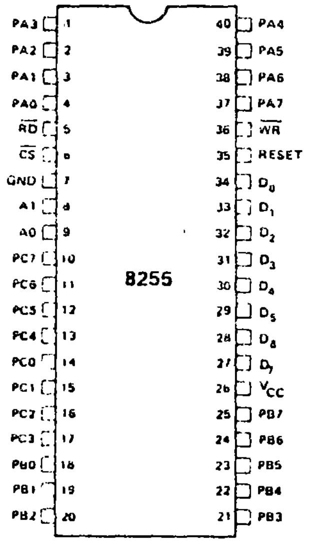
PROGRAMMABLE PERIPHERAL INTERFACE

- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with MCS™-8 and MCS™-80 Microprocessor Families
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40 Pin Dual In-Line Package
- Reduces System Package Count

The 8255 is a general purpose programmable I/O device designed for use with both the 8008 and 8080 microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bidirectional Bus mode which uses 8 lines for a bidirectional bus, and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255 include bit set and reset capability and the ability to source 1mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

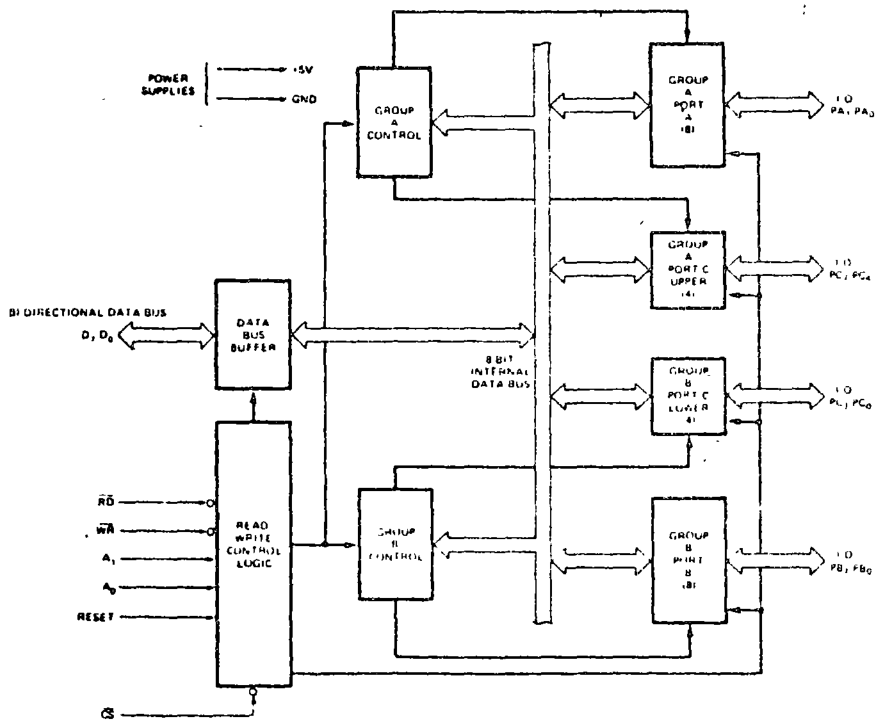
PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
[illegible]	+5 VOLTS
GND	0 VOLTS

8255 BLOCK DIAGRAM



8255 BASIC FUNCTIONAL DESCRIPTION

General

The 8255 is a Programmable Peripheral Interface (PPI) device designed for use in 8080 Microcomputer Systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the 8080 system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state, bi-directional, eight bit buffer is used to interface the 8255 to the 8080 system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions by the 8080 CPU. Control Words and Status information are also transferred through the Data Bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the 8080 CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select: A "low" on this input pin enables the communication between the 8255 and the 8080 CPU.

($\overline{RD}$)

Read: A "low" on this input pin enables the 8255 to send the Data or Status information to the 8080 CPU on the Data Bus. In essence, it allows the 8080 CPU to "read from" the 8255.

($\overline{WR}$)

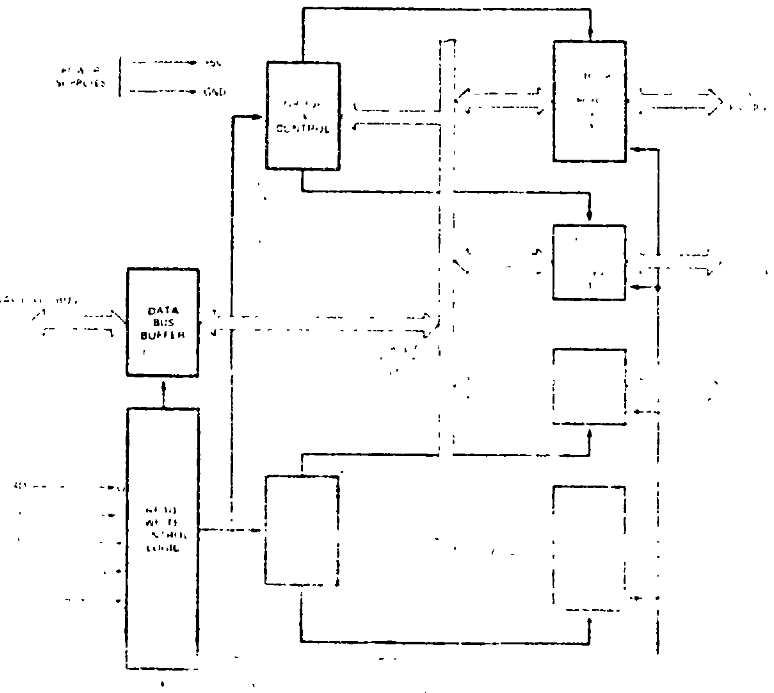
Write: A "low" on this input pin enables the 8080 CPU to write Data or Control words into the 8255.

(A_0 and A_1)

Port Select 0 and Port Select 1: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the Control Word Register. They are normally connected to the least significant bits of the Address Bus (A_0 and A_1).

8255 BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A ⇒ DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS ⇒ PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS ⇒ PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS ⇒ 3-STATE
1	1	0	1	0	ILLEGAL CONDITION



8255 Block Diagram

(RESET)

Reset: A "high" on this input clears all internal registers including the Control Register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the 8080 CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset" etc. that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

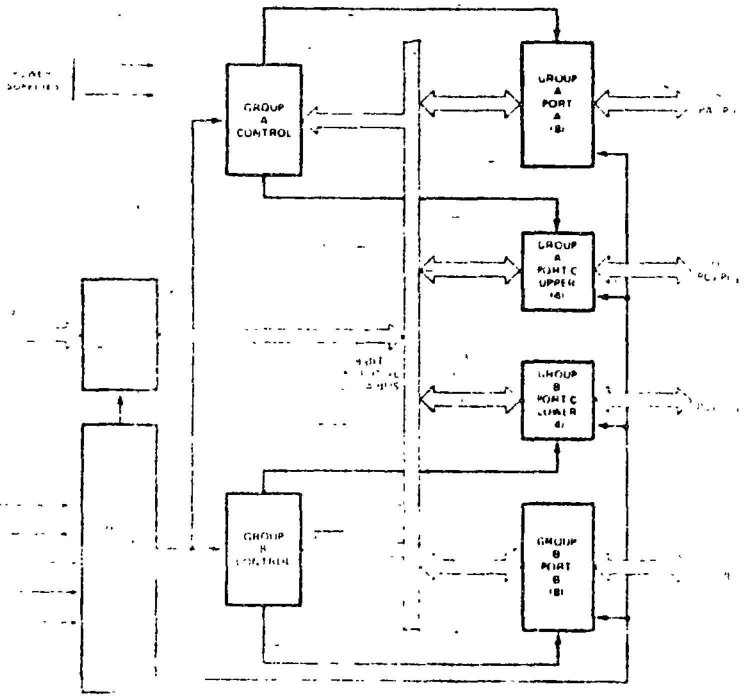
The 8255 contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

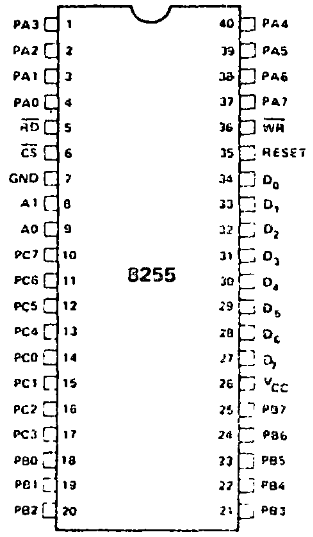
Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with Ports A and B.

8255 BLOCK DIAGRAM



PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 DETAILED OPERATIONAL DESCRIPTION

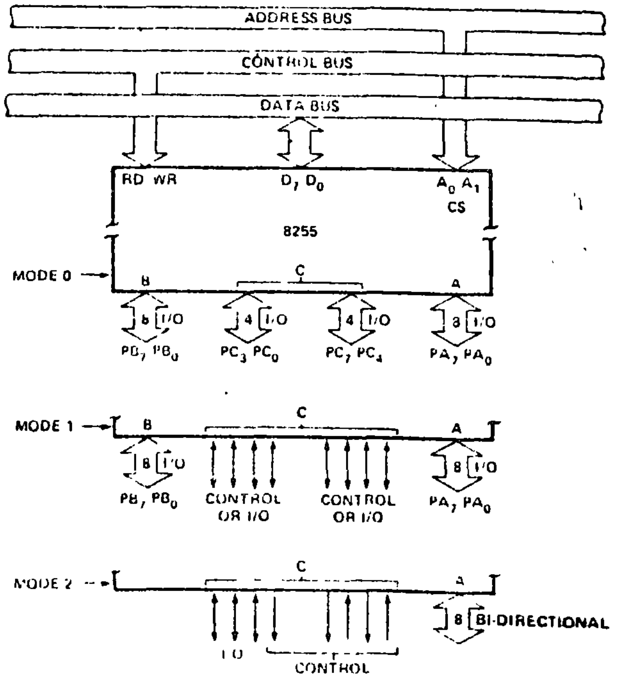
Mode Selection

There are three basic modes of operation that can be selected by the system software.

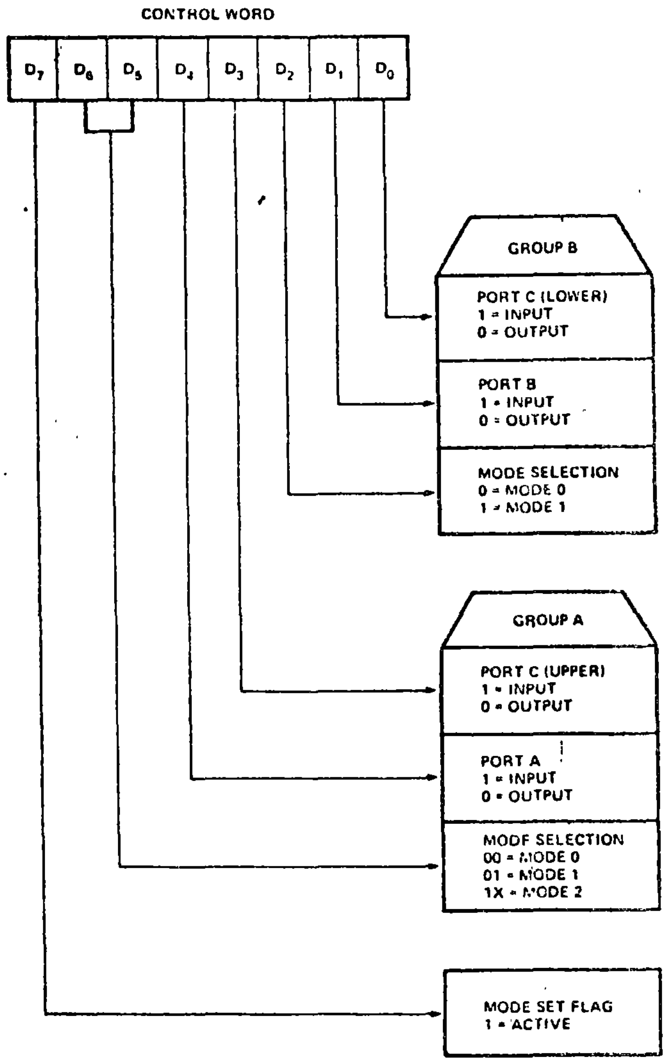
Mode 0 – Basic Input/Output
Mode 1 – Strobed Input/Output
Mode 2 – Bi-Directional Bus

When the RESET input goes "high" all ports will be set to the Input mode (i.e., all 24 lines will be in the high impedance state) After the RESET is removed the 8255 can remain in the Input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single OUTput instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.



Basic Mode Definitions and Bus Interface



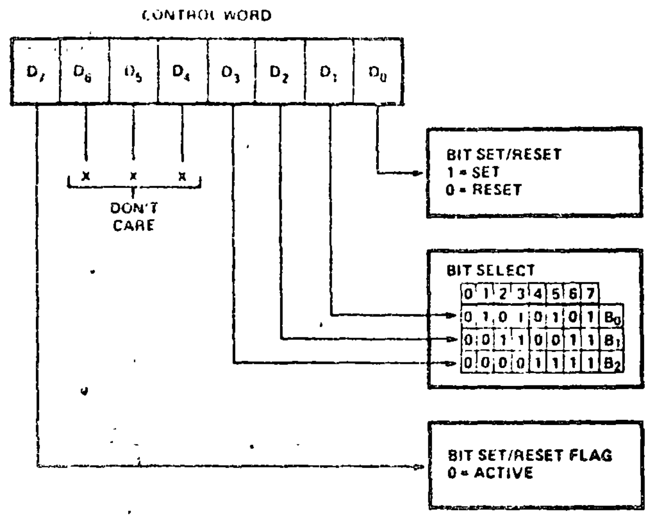
CONTROL WORD
D7
D6
D5
D4
D3
D2
D1
D0
GROUP B
PORT C (LOWER) 1 = INPUT 0 = OUTPUT
PORT B 1 = INPUT 0 = OUTPUT
MODE SELECTION 0 = MODE 0 1 = MODE 1
GROUP A
PORT C (UPPER) 1 = INPUT 0 = OUTPUT
PORT A 1 = INPUT 0 = OUTPUT
MODE SELECTION 00 = MODE 0 01 = MODE 1 1X = MODE 2
MODE SET FLAG 1 = ACTIVE

Mode Definition Format

The Mode definitions and possible Mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255 has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.



Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the Bit set/reset function of Port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without effecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) – INTE is SET – Interrupt enable
(BIT-RESET) – INTE is RESET – Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

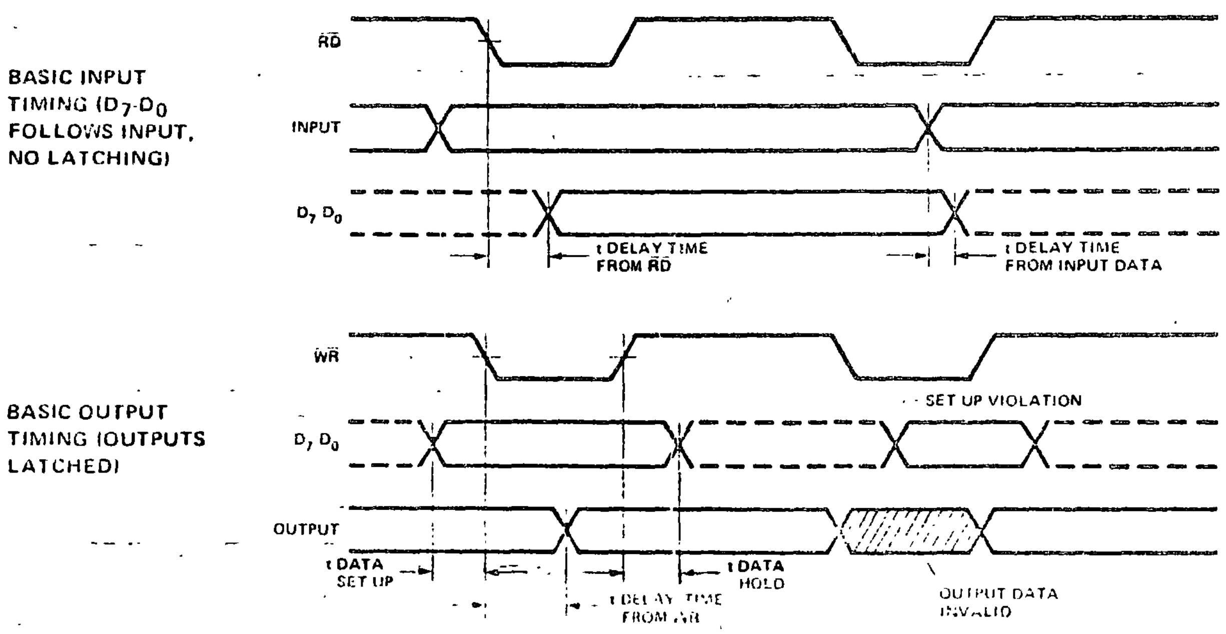
Operating Modes

Mode 0 (Basic Input/Output)

This functional configuration provides simple Input and Output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



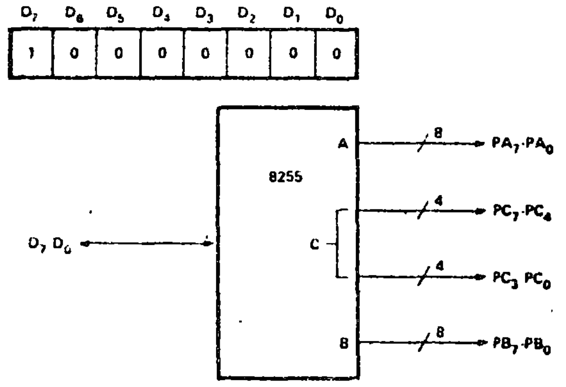
Mode 0 Timing

MODE 0 PORT DEFINITION CHART

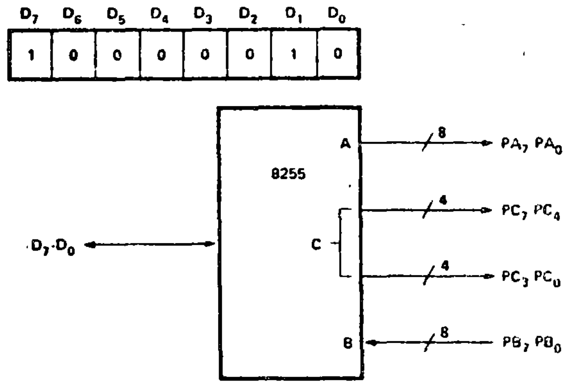
A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE 0 CONFIGURATIONS

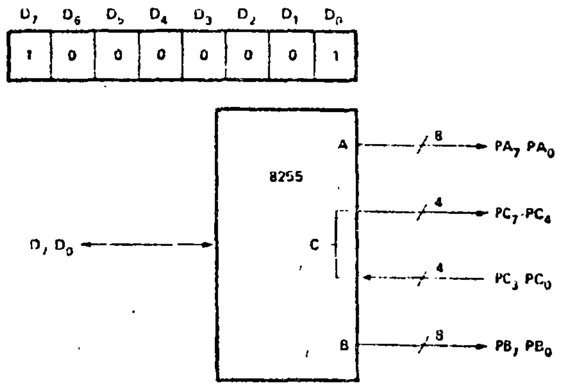
CONTROL WORD #0



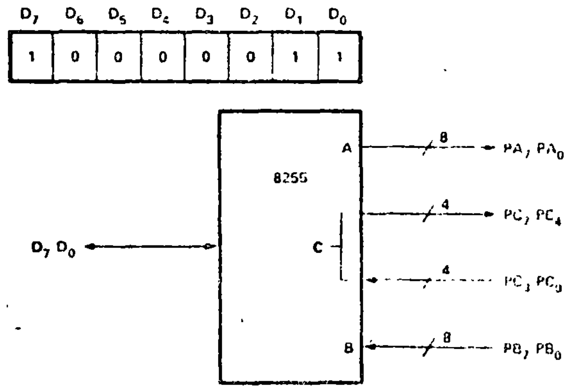
CONTROL WORD #2



CONTROL WORD #1

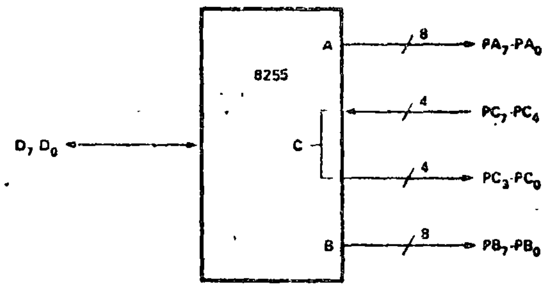


CONTROL WORD #3



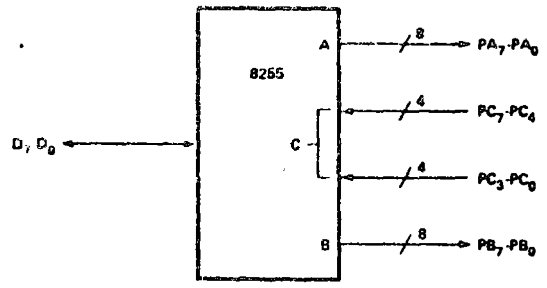
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



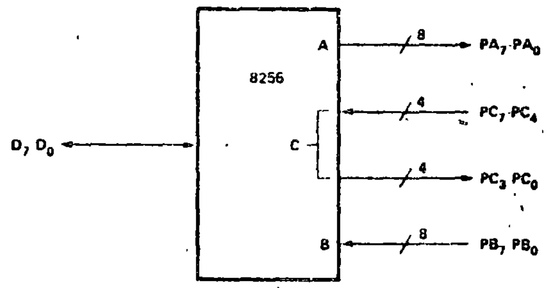
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



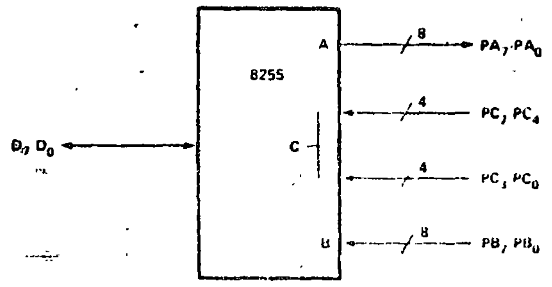
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



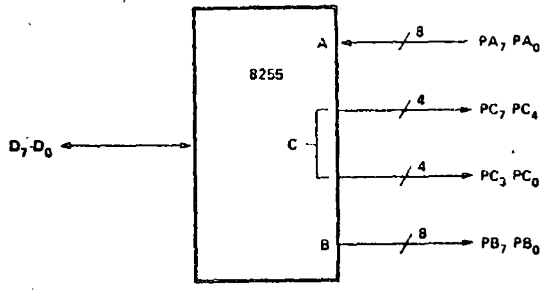
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



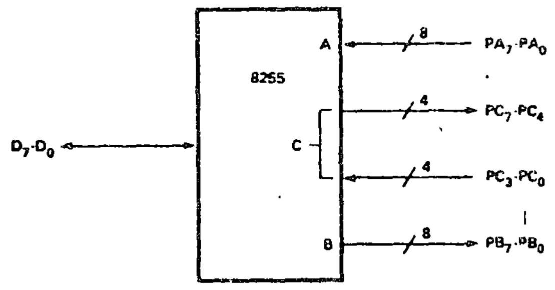
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



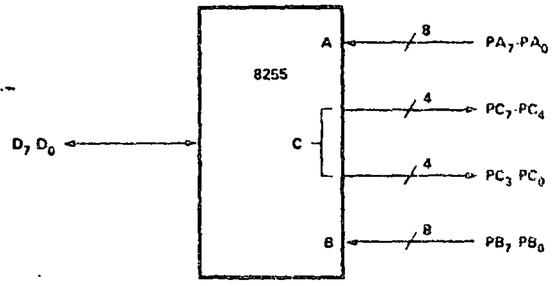
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



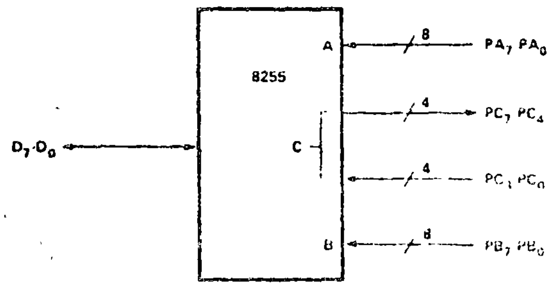
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



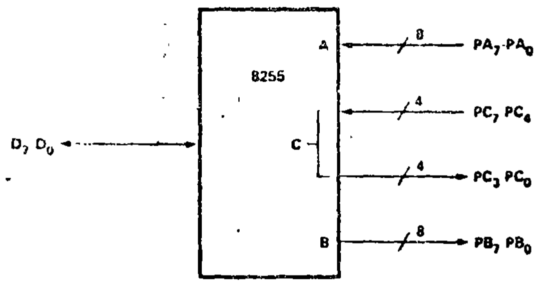
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



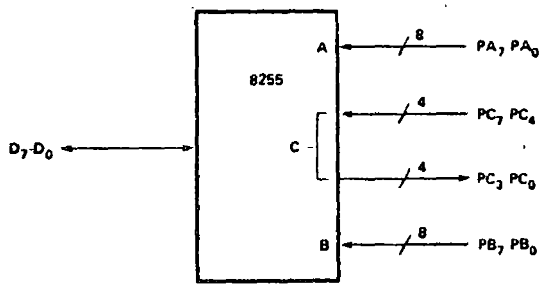
CONTROL WORD #12

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0



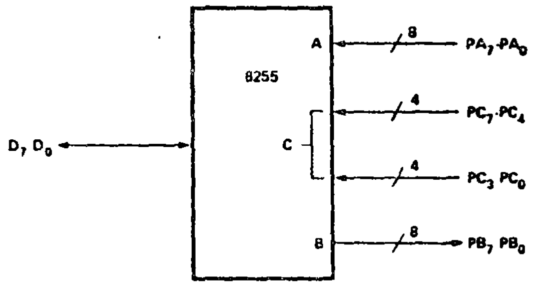
CONTROL WORD #14

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	0



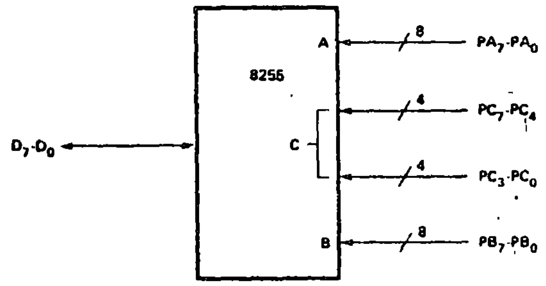
CONTROL WORD #13

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	1



CONTROL WORD #15

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	1



Operating Modes

Mode 1 (Strobed Input/Output)

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

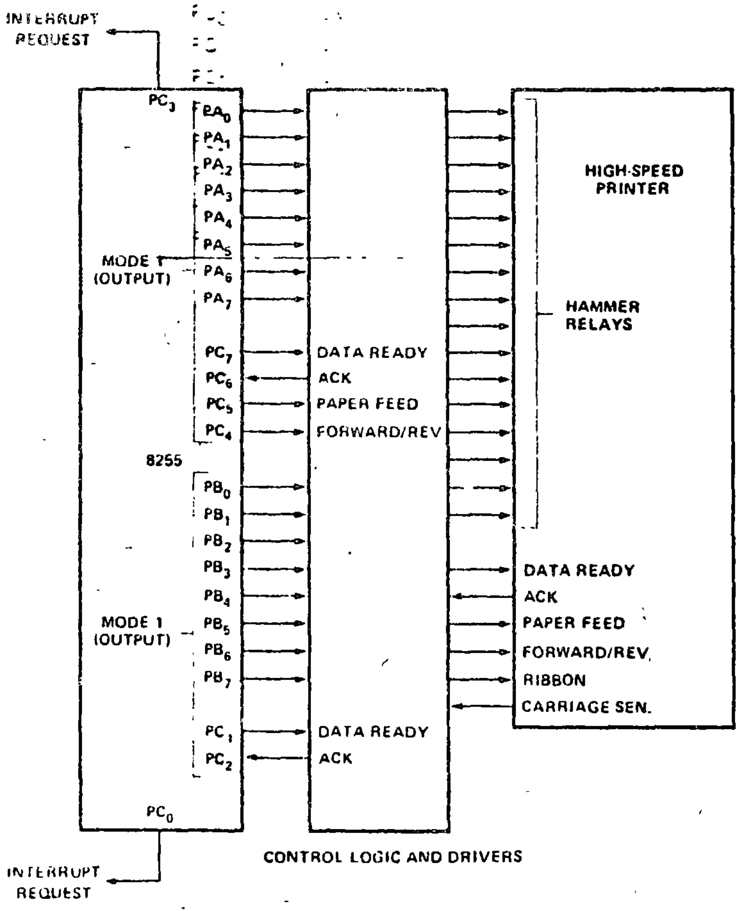
Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

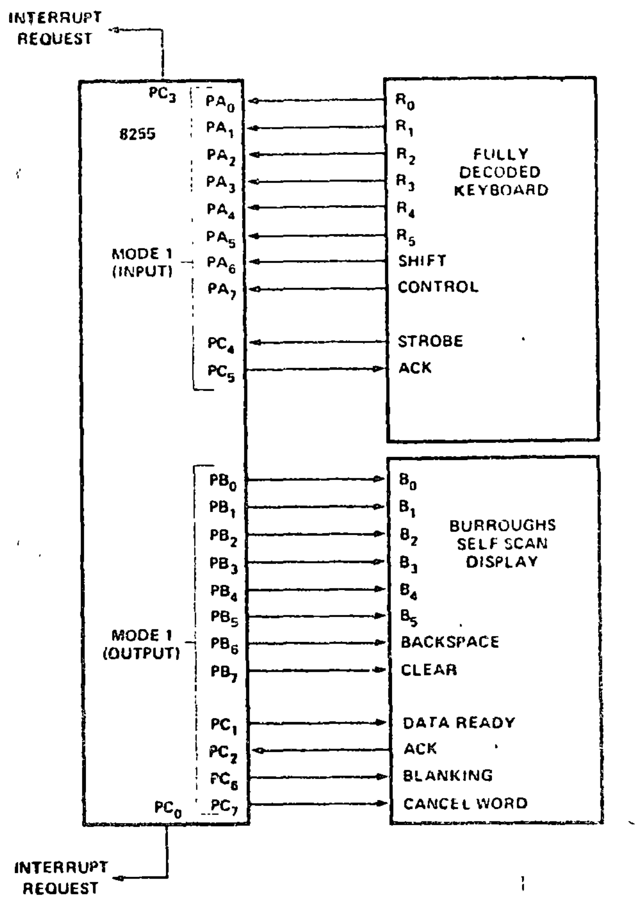
APPLICATIONS OF THE 8255

The 8255 is a very powerful tool for interfacing peripheral equipment to the 8080 microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

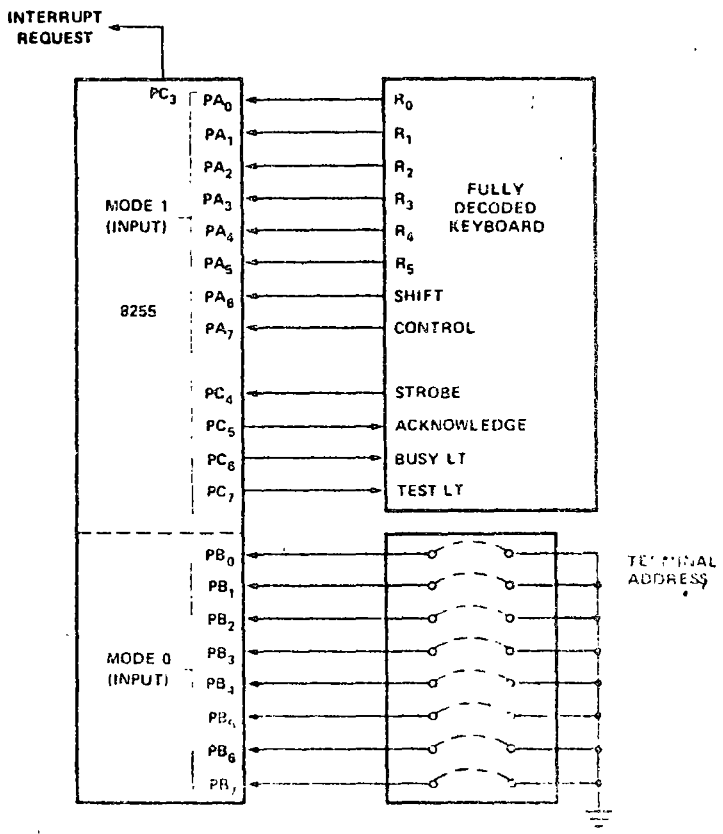
Each peripheral device in a Microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255 is programmed by the I/O service routine and becomes an extension of the systems software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the Detailed Operational Description, a control word can easily be developed to initialize the 8255 to exactly "fit" the application. Here are a few examples of typical applications of the 8255.



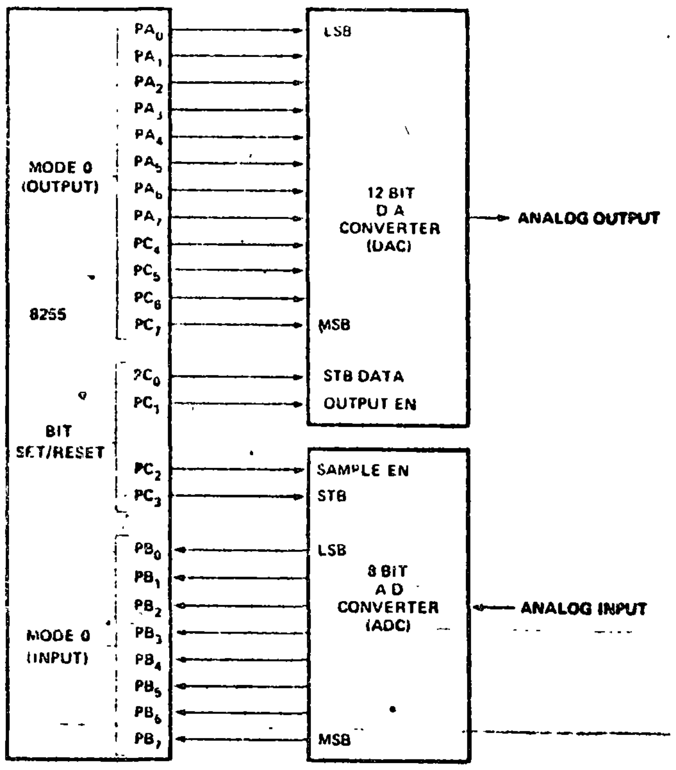
Printer Interface



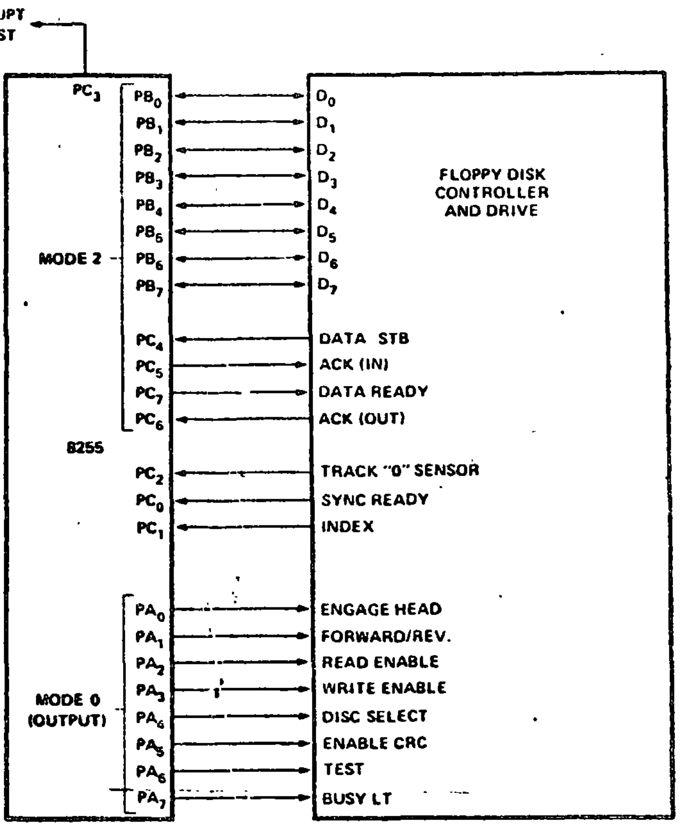
Keyboard and Display Interface



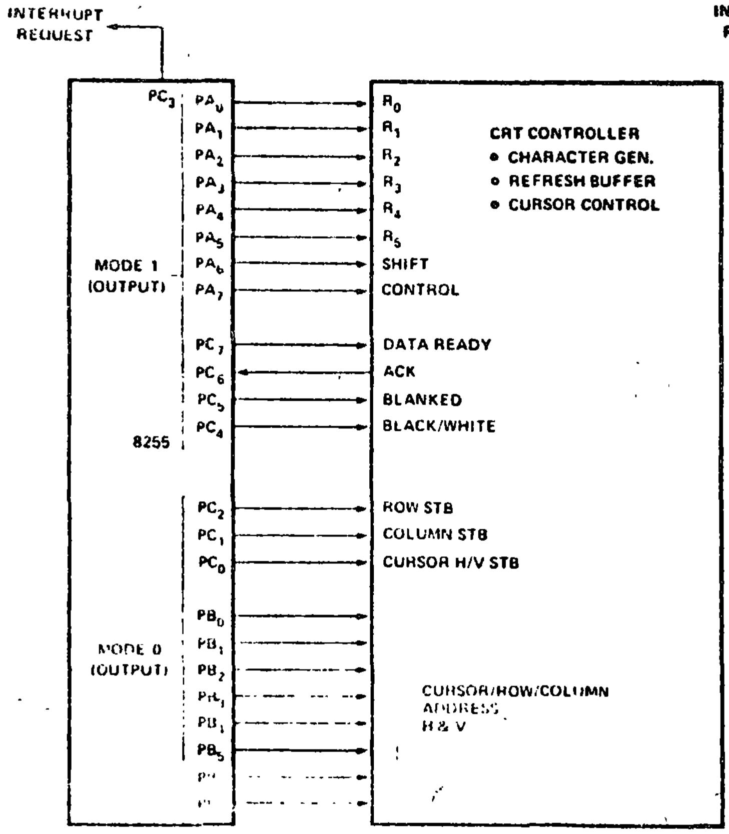
Keyboard and Terminal Address Interface


8255
MODE 0 (OUTPUT)
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 PC4 PC5 PC6 PC7
LSB
MSB
12 BIT D A CONVERTER (DAC)
ANALOG OUTPUT
BIT SET/RESET
PC0 PC1 PC2 PC3
STB DATA
OUTPUT EN
SAMPLE EN
STB
MODE 0 (INPUT)
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
8 BIT A D CONVERTER (ADC)
ANALOG INPUT

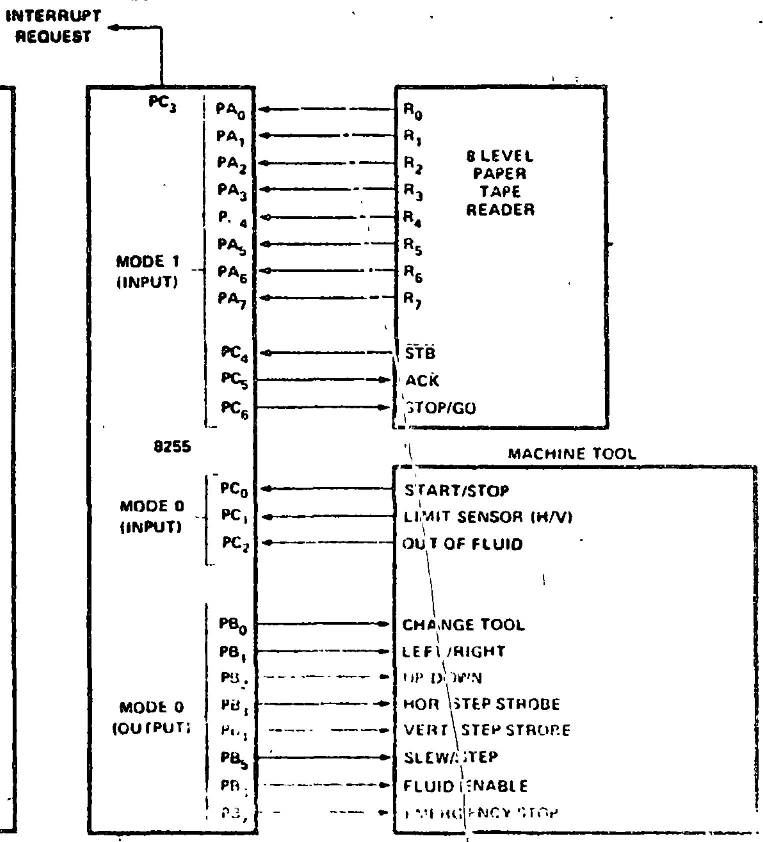
Digital to Analog, Analog to Digital



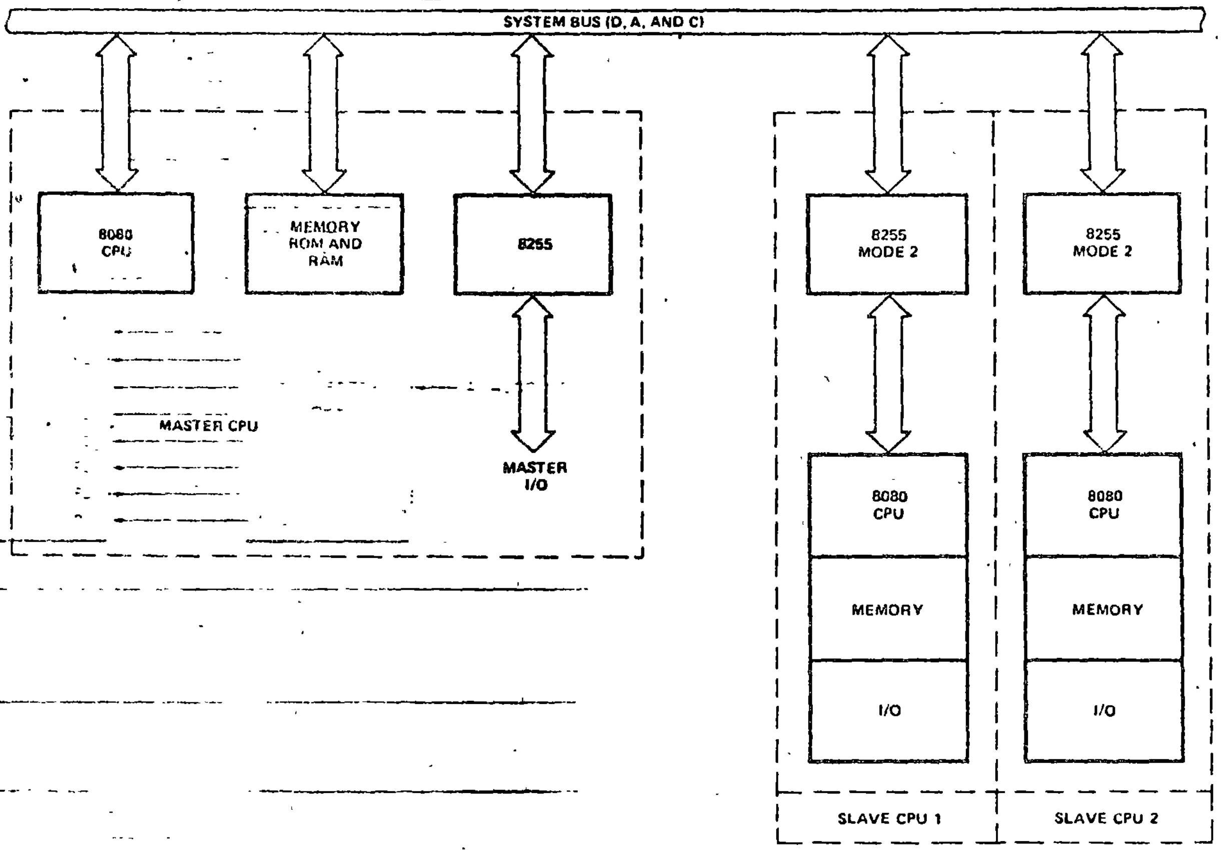
Basic Floppy Disc Interface



Basic CRT Controller Interface



Machine Tool Controller Interface



Distributed Intelligence Multi-Processor Interface

2.7 Perspectivas

Hay evidencia del poder de la industria de los semiconductores de empujar a la tecnología de las computadoras hacia nuevos circuitos integrados -algunos inclusive más complejos que los microprocesadores que han revolucionado la computación durante los años pasados- empezaron a ser una realidad en el control de periféricos, y también como un medio para estandarizar las capacidades de software.

Los nuevos dispositivos en el area del control de periféricos, por ejemplo, muy por afuera de la complejidad de los CPU's y de las memorias. En tanto que un microprocesador típico tiene alrededor de 7000 transistores, varios CRT en un CI, comunicaciones y controlador de discos contienen del orden de 22000 transistores c/u. Uno de esos el chip de National MM57109 (Number Cruncher) -diseñado para conectarse a un microprocesador estandar- contiene reloj, lógica para control y aritmética E/s, y 12 K bits de microinstrucciones en ROM para habilitar los cálculos de un huesped de funciones matemáticas, incluyendo las funciones trigonométricas, exponenciales y logarítmicas.

El cruncher también proporciona manipulación de registros, control para ramificar, banderas de error, etc. para un total de 64 instrucciones. Cada instrucción toma 20μs de tiempo de ciclo a 400 K Hz de frecuencia de reloj. La mayor parte son instrucciones de un solo dígito (entonces arcseno es una continuación de las instrucciones de INV y SEN). El formado para los datos se puede seleccionar que sea punto flotante a instrucción científica, bajo el control de progama. Entrada rápida al dispositivo usualmente toma 2.4 ms, pero este método puede usarse solamente si otras rutinas no estan operando durante la entrada de dígitos. Las entradas controladas por instrucciones toma 6 ms. para mantiza de 8 bits signada en un exponente signado de dos dígitos. Con 'handshaking, la misma entrada toma 30 ms. El dispositivo en TTL compatible y puede ser operado de una fuente de +5 y -4 volts.

El adaptador de interfases de propósito general (GPIA Motorola MC68488 proveé la interfase entre el bus instrumental IEEE488 y la familia de microprocesadores 6800, y con alguna lógica adicional, se puede interfacear con otros microprocesadores también.

Las microcomputadoras se ve que parecen destinadas a volverse la parte fundamental en los sistemas de telefonía en los Estados Unidos. De hecho, las telecomunicaciones futuras probablemente serán realizadas por micros hablando con otras micros, con chips enterrados en cada nodo del sistema. El microprocesador MAC-8 fue diseñado en los laboratorios Bell con esa aplicación en mente. Es esencialmente un microprocesador que puede compilar eficientemente un lenguaje de alto nivel, desigual ahora simplemente como e. Este lenguaje, desarrollado originalmente en los laboratorios Bell, se ha adoptado rápidamente en los sistemas Bell. Su objetivo es prevenirse de una Torre de Babel ofreciendo un medio estandar de diseñar software y para operar sistemas operativos. C es un lenguaje interactivo. Comparado con el código a nivel de ensamblador, el pago por uso es de un 20% de incremento en tiempo de corrida y espacio de memoria.

Adicionalmente a correr y compilar eficientemente C, MAC-8 tiene algunas características únicas. Una de ellas es la instrucción llamada "encuentra el que se encuentra en el extremo izquierdo". (Find the left most) (FLO) que se usa ampliamente en el proceso de identificación del siguiente item del negocio. MAC-8 tiene 16 acumuladores/registros de índice realizados en RAM. Se proporciona una variedad de módulos de memoria, incluyendo auto-incremento como son operaciones de memoria a memoria. La máquina tiene arriba de 400 instrucciones fundamentales. Debido a que la mayoría son de 16 bits de longitud (aunque tomando 8 bits a un tiempo), hay alrededor de 32000 combinaciones de instrucciones.

Otro desarrollo básico ha sido la aparición de interpretes en un chip. Con eso, los usuarios de microprocesadores solamente necesitan conocer un lenguaje de alto nivel (como Basic) y no requiere trabajar con lenguaje de máquina, o a nivel de ensamblador. Los înterpretes no son, desde luego, nada nuevo. Lo que ha pasado es que han sido puestos en un chip, y en algunos casos en medio chip.

BASIC Industiral es un lenguaje de programación que ha sido escrito en un solo chip (ROM). El lenguaje viene a ser un înterprete del lenguaje y fue especialmente diseñado para usarse con los microprocesadores National SC/MP. Se interfasa, a través de sus propias rutinas de entrada-salida con un teletipo o terminal. Su ûnica modificación incluye un DD/UNTU que es una estructura de control adoptada del lenguaje PASCAL, y un operador de indirección para un acercamiento flexible de lenguaje de máquina para control de procesos.

Adicionalmente a NIBL, un înteprete en BASIC en un chip, ha sido desarrollado en los Laboratorios Lawrence de Livermore para usarse con un microprocesador del tipo 8080; y el înterprete TRS-80 desarrollado por Tandy, Inc., para usarse con el microprocesador Zilog-Z80.

La posibilidad de disponer de înterpretes en un chip abre la computación a muchos nuevos usuarios.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES : TEORIA Y APLICACIONES

TEMA: COMPLEMENTO AL TEMA 2
MICROPROCESSOR QUESTIONNAIRE.

PROF. ING. JORGE GIL MENDIETA.

Abril,1978.

MICROPROCESSOR QUESTIONNAIRE

Please use the Reader Service Card in the back of the magazine to answer this questionnaire. Even if you are not the slightest bit interested in microprocessors, your answers could be illuminating; if you are slightly interested, but know little or nothing about the subject, your answers are also valuable. If you do know what you need, here's your opportunity to spell it out. In the event that you are working on more than one micro-based system, or have requirements for different types of products, please make a copy (or copies) of the reply card and respond for each of your different specifications. (You'll have to put the copies in an envelope, however.) Room is allotted on the reply card for comments; if this is not sufficient, letters are welcome. If you come across a question which you don't understand or which is not applicable to your situation (such as questions directed at current users), leave the answer blank. Don't try to guess at an answer. "No preference" answers are provided in places where we think it is an honest answer to a question. Tabulated results and an analysis of this questionnaire will be printed in a future issue of **InTech**.

Section I: What do you know about micros?

1. Do you understand microprocessor terminology and nomenclature?
   a) Yes b) No c) Some of it d) Most of it

2. How do you stay up to date on the subject?
   a) Read magazines b) Send for literature
   c) Talk to reps d) I work with micros
   e) Attend workshops or special microprocessor sessions at shows or seminars.

3. Do you feel that it is a necessary part of your job to stay up to date on microprocessors?
   a) Yes b) No c) Partly d) Intellectually

4. What would help you the most in learning about microprocessors?
   a) More articles in trade magazines
   b) Better sales literature
   c) Technical documentation from vendors
   d) Traveling microprocessor workshops
   e) Hands-on experience with actual systems

5. What is your personal opinion about the entire microprocessor field?
   a) Fabulous opportunities for cost savings, efficiency, more powerful devices, new instruments, and a plethora of applications
   b) They will never replace minicomputers, but micros definitely have a place in the process control industry
   c) Has possibilities, but the technology has not matured enough to make any predictions
   d) I will wait until commercial products are available, MTBF studies have been run, and a number of systems actually installed before I even consider using a micro-based product
   e) Has possibilities, but I consider much of the sales literature to be misleading

Section II: Would you buy one?

6. Are you using any microprocessor-based products now? (check all that apply)
   a) No b) Instrument, scope or meter
   c) Controller, data logger, or PLC
   d) Microcomputer development system, assembled kit, or full system, which is undergoing development or evaluation
   e) Process control or data processing system

7. If you have been using a microprocessor-based product or system, how do you feel about it?
   a) It surpasses my expectations
   b) It meets my expectations
   c) I am disappointed
   d) Haven't decided
   e) Does not apply

8. Do you anticipate buying a microprocessor in the future?
a) Yes b) No c) Maybe

9. How will you purchase it? (Or, how did you purchase it?)
a) A set of chips b) Board-mounted chips
c) A kit with all necessary components
d) Assembled microcomputer, no peripherals
e) Packaged development system (with TTY)
f) As a complete data processing or turnkey system, with software

10. What criteria do you use in selecting a vendor?
a) A major vendor who offers service, second-sourcing of components, training, software support and documentation
b) Any major "name brand" manufacturer
c) Any vendor with a product that fits my hardware and software specifications
d) The most inexpensive source
e) A "systems house" (nonmanufacturer) with a packaged system

11. How do you plan to use the component or product?
a) To replace, or instead of, a commercial nonmicro product or device such as a meter, scope, data logger, annunciator, etc.
b) As an element in a control system such as a multiplexer, controller, data concentrator, communications module, PLC, etc.
c) As a number-crunching or data processing computer, supervisory control system, process control system or management information system
d) As part of a product (as described in a or b) which my company will build and market
e) As a full system (as described in c) which my company will build and market

12. How many micro-based products, such as scopes, data loggers, meters, stand-alone controllers, etc. will your company build or purchase in the next five years?
a) None b) 1-5 c) 1-10 d) 1-25
e) more than 25

13. How many micro-based control systems, such as intelligent controllers, programmable controllers, multiplexers, communication devices, or small process controllers will your company build or purchase in the next five years?
a) None b) 1-5 c) 1-10 d) 1-25
e) More than 25

14. How many complete microcomputer systems, such as data processors, number crunchers, process control systems, distributed control systems, or supervisory systems will your company build or purchase in the next five years?
a) None b) 1-5 c) 1-10 d) 1-25
e) More than 25

Section III: Hardware specifications
In Sections III, IV, V and VI, please limit your answers to a specific product or application. If you have more than one product or application in mind, which requires different specifications, please answer on a separate card.

15. What size CPU fits your needs?
a) 4-bit b) 8-bit c) 12-bit d) 16-bit
e) 24-bit f) n-slice

16. What word size fits your needs?
a) 8-bit b) 12-bit c) 16-bit d) 24-bit

17. What technology do you prefer?
a) PMOS b) NMOS c) CMOS
d) Bipolar e) No preference

18. What features do you need? (check all that apply)
a) Interrupts
b) Direct memory access (DMA)
c) Microprogramming capability
d) Arithmetic unit (add, subtract)
e) Arithmetic unit (multiply, divide)
f) Floating point module
g) Logic instructions (AND, OR, XOR)
h) Clock driver
i) BCD arithmetic
j) Memory protect

19. How many general registers do you need?
a) 1 b) 1-4 c) 1-8 d) 1-16
e) More than 16

20. What type of memory do you prefer or need? (For mixed systems, check all that apply)
a) Core b) Semiconductor RAM c) ROM
d) PROM e) No preference

21. How much memory do you need? (8-bit bytes)
a) Less than 1K b) 2K c) 4K d) 8K
e) More than 8K

Section IV: Peripherals
22. Check all of the peripheral devices you need.
a) Teletype b) CRT c) "Smart" terminal
d) Line printer e) High-speed paper tape
f) Programming panel or data entry panel
g) Cassette tape h) Mag tape
i) Floppy disc j) Standard disc
k) Strip printer l) Others (Use comment section on card)

23. How many digital inputs and outputs?

a) 1-8 b) 1-16 c) 1-32 d) 1-64
e) More than 64

24. How many analog inputs and outputs?
a) 1-8 b) 1-16 c) 1-32 d) 1-64
e) More than 64

25. Do you need a computer-to-computer link?
a) Yes b) No c) No, but may need it in future

26. What type of computer-to-computer link do you prefer?
a) TTY interface b) ASCII bus
c) CAMAC d) Direct, single word
e) DMA, bulk transfer

Section V: Software development

27. What is your software background?
a) Familiar with assembler, machine language can write complete control programs
b) Familiar with assembler
c) FORTRAN, BASIC or COBOL
d) Have used a process control language
e) Limited

28. What type of assembler support do you need?
a) Resident assembler
b) Cross-assembler (for IBM 360, etc.)
c) PROM programmer
d) High-level language compiler
e) Resident development system

29. What software aids do you need? (check all that apply)
a) Hardware simulator b) Debug program
c) Cross-simulator (runs in another computer)
d) Trace program e) Other

30. What application programs or operating systems do you need? (check all that apply)
a) Real-time exec (skeleton)
b) Real-time operating system
c) Peripheral driver programs
d) Network communication package
e) Batch operating system
f) Disc or tape operating system
g) Editor program (paper tape, disc or tape)
h) Library of programs and subroutines

31. In what medium would you prefer to work with source code?
a) Paper tape b) Cassette tape c) Cards
d) Disc file e) Mag tape file

32. In what medium would you prefer to work with object code?
a) Paper tape b) Cassette tape c) Cards
d) Disc file e) Mag tape file f) ROM

33. What kind of loader would you like?
a) Relocating b) Absolute c) Linking
d) No preference

34. Who will develop the software?
a) Myself b) Programming department
c) Outside contractor (software house)
d) Microprocessor vendor
e) Consultant (job shopper, etc.)

Section VI: Hardware development

35. What is your hardware background?
a) Extensive experience in micros and minis
b) Experience in minis or other computers
c) Process devices and control hardware only
d) Limited

36. What type of hardware support do you need? (check all that apply)
a) Technical manuals and logic diagrams
b) Maintenance and diagnostic software
c) Hardware simulator d) Prototyping cards
e) Application notes

37. What kind of assistance do you expect from the vendor? (check all that apply)
a) Factory engineer available by phone
b) Authorized service in your area
c) Hardware school d) Little or none

38. What components do you expect to buy off the shelf? (check all that apply)
a) Digital I/O boards
b) Analog I/O components and boards
c) TTY-compatible interface
d) Multiplexers, communication devices
e) Peripheral device interfaces
f) All necessary components

39. Who will build and develop the hardware?
a) Myself b) Engineering department
c) Outside contractor (systems house)
d) Microprocessor vendor
e) Consultant (job shopper, etc.)

40. Who will maintain the hardware?
a) Vendor b) Authorized service company
c) Our maintenance department
d) Operators and engineers
f) Ship it back for factory service

41. How many hardware functions are you or your company able to perform with a reasonable amount of assistance from the vendor? (check all that apply)
a) Hardware design
b) Component installation
c) Checkout d) Maintenance
e) Future expansion/modification

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES: TEORIA Y APLICACIONES

TEMA 3 Y 4 : LA MICROCOMPUTADORA 8080

M. EN C. ALEJANDRO GUARDA AURAS

ABRIL, 1978.

CAPITULO 3: LA MICROCOMPUTADORA 8080

Alejandro Guarda Auras

3.1 Introducción a la familia INTEL 8080

Desde que en 1972 se anunció la introducción al mercado del microprocesador 8008 de INTEL, la proliferación de este tipo de dispositivo ha sido tan grande que, sólo cinco años después, se estima que habían más de 120 microprocesadores distintos en el mercado.

Una porción muy importante del mercado de microprocesadores de 8 bits, ha sido captada por el 8080 de INTEL. Las razones que se pueden aducir son varias. Sin embargo, las más importantes son las siguientes:

A) INTEL ha sido el pionero en el desarrollo de microprocesadores, siendo sus dispositivos los que han establecido estandares para cada generación. En la tabla 3.1.1 a continuación se resumen los productos más importantes y algunas de sus características:

Procesador	Año de Introducción	Ciclo de Introducción µseg.	N°de Instruc.
8008	1972 - 1a. generación	12.5 - 20.0	48
8080	1974 - 2a. generación	1.3 - 2.0	78
8048	1976	2.5 - 5.0	96
8085	1977 - 3a. generación	1.3	80
8086	1978 - 4a. generación	0.5 - 0.8	?

TABLA 3.1.1

B) Disponibilidad. Este factor es muy importante en la relación de un procesador. Aún cuando un producto pueda parecer muy poderoso, rápido y versátil en la hoja de especificación. de nada sirve si no se puede conseguir comercialmente. El 8080, es un producto ya establecido en el mercado, con amplia distribución y "second-sourcing" (producido por otras compañías).

C) Soporte. En este aspecto vamos a diferenciar en: soporte de circuitos periféricos, soporte en sistemas de desarrollo, soporte en software y soporte en módulos para "O.E.M." (Original Equipment Manufacturer = Fabricantes de Equipos Originales).

C.1. Soporte en circuitos periféricos: Un μprocesador sólo es totalmen inútil. (distingue entre μprocesador y μcomputadora). Requiere de una serie de circuitos periféricos que provean funciones de secuenciación, control e interface. Estos circuitos pueden ser implementados en forma discreta (hard-wired logic), con los consecuentes problemas de sincronía, costo de desarrollo y confiabilidad que ello implica, o bién, como es el caso general, el fabricante provee de circutis LSI que realizan tales funciones. Una característica de la familia 8080 es la gran cantidad de circuitos periféricos de soporte. En la tabla 3.1.2 se incluyen los circuitos periféricos de la familia 8080

Circuitos de Soporte al C.P.U.	
8224	Generador de reloj y driver
8228	Controlador y Bus driver
Circuitos Periféricos de Propósito general	
8205	Decodificador Binario
8212	Puerto de E/S de 8 bits
8214	Control de Prioridad de Interrupción
8251	Interface programable de comunicación
8253	Secuenciador programable de intervalos
8255	Interface de E/S Programable
8257	Controlador Programable de DMA
8259	Controlador Programable de interrupciones
Circuitos Periféricos Dedicados	
8271	Controlador Programable de discos flexibles
8273	Controlador de Protocolo de comunicación de datos
8275	Controlador Programable de CRT
8279	Interface Programable de teclados y displays.

TABLA 3.1.2

C.2. Soporte en sistemas de desarrollo. El proceso de desarrollo de un sistema basado en un microprocesador, requiere de estaciones de desarrollo que agilicen dicho proceso. Estas estaciones, dependiendo de necesidades y recursos, pueden ir desde microcomputadoras muy simples con costos de $ 5,000.00 M.N. hasta sistemas completos y de propósito general con costos desde $ 25,000.00.
Para el procesador 8080 ofrece una gran variedad de opiniones que van desde el sistema SDK-80 hasta el MDS-80.

C.3. Soporte de Software. Una computadora funciona en base a códigos binarios el cual debe ser cargado en memoria para luego iniciar su ejecución. Para facilitar el desarrollo de programas de aplicación los fabricantes ofrecen paquetes de programs que varían de complejidad:
Monitores. Permiten: cargar en memoria, vía teletipo o similar, un programa codificado en hexadecimal, iniciar su ejecución y en forma muy primitiva, limpiar de errores el programa.
Ensambladores: Permite al usuario codificar progrmas utilizando nemónicos, y direcciones relativas mediante el uso de etiquetas, produciendo como salida el código binario con direcciones absolutas.
Compiladores: Permiten el uso de lenguajes de alto nivel para la generación de programs. Entre estos lenguajes destacan FORTRAN, BASIC, PL/M y actualmente se habla mucho de PASCAL. El compilador traduce el programa escrito en lenguaje de alto nivel a código máquina. Su empleo resulta beneficioso, sólo cuando el programa a realizar es muy complejo.
Sistemas Operativos: Permiten el manejo de archivos, la reubicación y ligado de programs, así como el manejo de programotecas residentes en disco.
Sistemas operativos en Tiempo Real: Permiten la generación de programas a controlar eventos asíncronosy concurrentes, como es el caso del control de procesos industriales y adquisición de datos.
Intel ofrece todos estos programas, espcíficamente: Monitores para SDK-80 y para MDS-80; Macro Ensamblador para MDS-80; compilador PL/M para MDS-80, Sistema operativo ISIS-II para MDS-80 y Sistema operativo en Tiempo Real, RMX-80 para microcomputadoras modulares.

:.4. Soporte en Módulos para O.E.M. Existe una enorme cantidad de aplicaciones de microcomputadores, en las que el interés se centra en disminuir el costo de diseño en vez del costo de partes. Tal es el caso de industrias en proceso, comunicaciones, instrumentación, etc. Es estos casos, lo más deseable para el fabricante es contar con módulos: computadoras en un circuito impreso, a las que pueda conectar a su vez otros módulos, tales como memoria, entradas analógica, salida analógica, etc.
Basados en el 8080, Intel ofrece varios de estos módulos que varían en complejidad y precios. Tales son los "Single Board Computer" SBC-80, de los cuales existen, en orden de complejidad creciente el SBC 80/04, SBC 80/05, SBC 80/10 SBC 80/20, SBC 80/20-4, SBC 310. Además de estos módulos de computadora, ofrece módulos de: Expansión de memora.
E/S digital
E/S analógica
De esta forma, el diseñador no tiene que preocuparse de particulares como secuenciación de señales, niveles de alimentación, circuitos impresos y todos los problemas que invariablemente surgen al pasar un prototipo de su forma original al circuito impreso.

Con lo anterior no se ha pretendido decir que Intel e la única compañía que ofrece los productos señalados anteriormente. Otra compañía que a juicio del autor, merece mucha atención es Zilog. Sin embargo, y nuevamente a juicio del autor, es conveniente trabajar con productos de una compañía bien establecida, que esté entre los líderes en su campo y que tenga un buen número de otras compañías haciendo "second-source" de sus productos. Estas características las cumple Intel y por haber sido la pionera en este campo hay muchos ingenieros que prefieren seguir con sus productos.
Finalmente, cabe mencionar algo sobre la obsolescenciade productos, cosa que nos preocupa mucho en un campo en el que entre una generación y la siguiente transcurre menos de dos años y donde aún queda mucho por verse. Afortunadamente, los fabricantes han sido cooperativo en este sentido y cada vez que introducen un nuevo producto que obsolece al anterior, lo hacen compatible en software; es decir, mantienen el mismo conjunto de instruccio

nes que tenía la versión anterior, agregando algunas nuevas que mejoren la eficiencia del procesador. De esta forma, puede introducirse la nueva versión manteniendo los programas desarrollados para la anterior sin grandes gastos en desarrollo. Esto es lo que los fabricantes dicen; sin embargo no es del todo cierto como será el caso entre el 8080/8085 y el 8086, lo cual aún está por verse ya que el dispositivo fue anunciado en un congreso (ISSCC San Francisco, California, febrero de 1978) y descrito en una revista (Electronics febrero 16, 1978) pero aún no está disponible.

El material impreso que sigue a continuación, son copias del manual de usuario del 8080, puesto que no hay nada publicado que mejore o agregue a lo que se dice en dicho manual. Se incluyen además, copias de descripción y especidifaciones de algunos de los dispositivos periféricos que serán descritos en el curso, así como una breve selección de artículos que pueden ser útiles para lectura adicional.

Programmable peripheral interface IC's boost your µC's flexibility

The popular 8255 software programmable I/O device is versatile, has an interrupt/handshake capability and can reduce system complexity.

Alan Ebright, Intel Corp.

Hardly is a new microcomputer system produced than a second generation or improved version is announced. The trend in µP's has been to add more functions and to improve control of these functions. At first, many features were fixed; then they became hardware alterable and finally software programmable. This trend applies equally to other devices in the various microprocessor families.

Among today's more flexible µC system-oriented LSI devices are the programmable parallel input/output types such as those offered by Motorola, Intel and others. These units allow designers to interface with parallel I/O peripherals under the varying conditions required for a specific peripheral, with device operating modes selectable under program control. This article describes the popular 8255 developed by Intel and now also available from AMD, NEC and National.

The 8255 provides three operating modes

Directly compatible with 8080-type CPU modules, the 8255 programmable peripheral interface (PPI) is a 40 pin device with a total of 24 control/data lines and an 8080 bus interface. Three 8 bit I/O ports can be programmed to function in three different modes, two of which offer interrupt capability. One of the ports is actually two 4 bit ports, and its bits can be individually set and reset.

In the first operating mode, the two 8 bit ports can be either input or output, as can the two 4 bit sections of the third port. In the second mode, the two 8 bit ports are strobed and again can be either input or output, utilizing bits from the two 4 bit ports for control. In the third mode, one 8 bit port is used as a bidirectional bus, the other 8 bit port can be in either of the other two modes, and bits from the two 4 bit ports are used for control.

For purposes of discussion we will break the 8255 into three sections: CPU interface, peripheral interface and internal logic (**Fig. 1**).

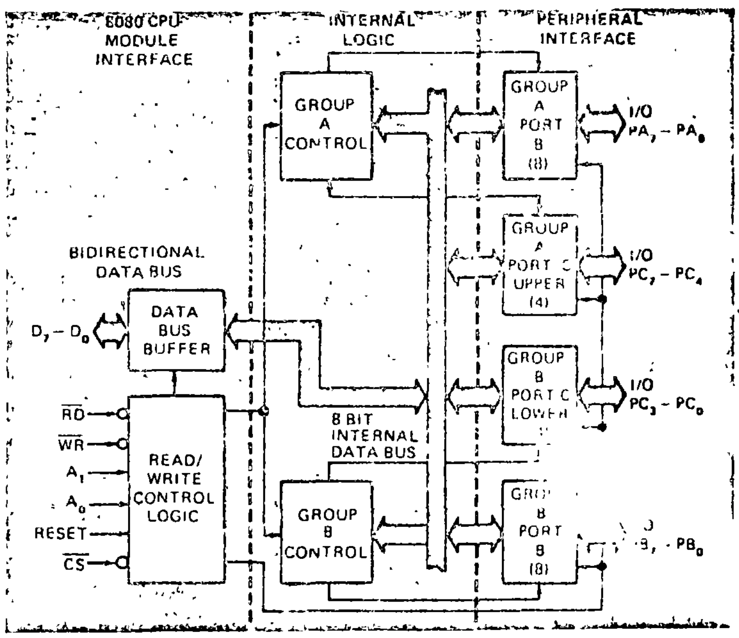


Fig. 1—Operation of the 8255 can best be explained by breaking the device's block diagram into three sections: CPU interface, peripheral interface and internal logic.

CPU interface—direct communications

The CPU interface section is rather straightforward. **Fig. 2** shows a simple system in which a CPU, clock generator and system controller all communicate directly with I/O devices. This circuit dedicates address lines A_0 and A_1 to select either the control register or one of the 8255's ports, and address lines A_2 and A_3 to select one of the two I/O devices. Signal levels required to accomplish port and device selection are shown in **Fig. 3.**

When the CPU selects an I/O via an address line signal to the 8255's $\overline{CS}$ input, that I/O unit then communicates directly with the CPU. With this particular system, the CPU can select up to six I/O devices. If additional I/O devices are necessary, bits A_2 through A_7 on the 8255 can be encoded to generate up to 64 device select lines

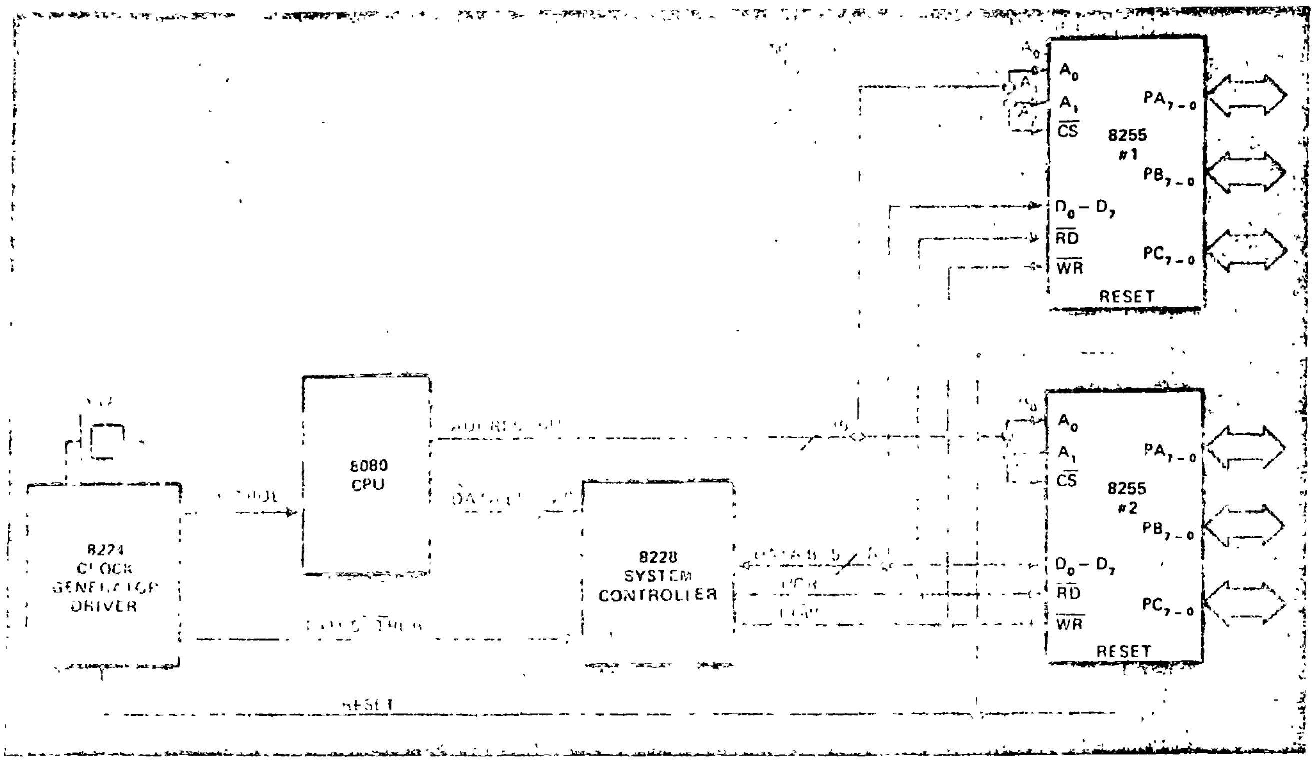
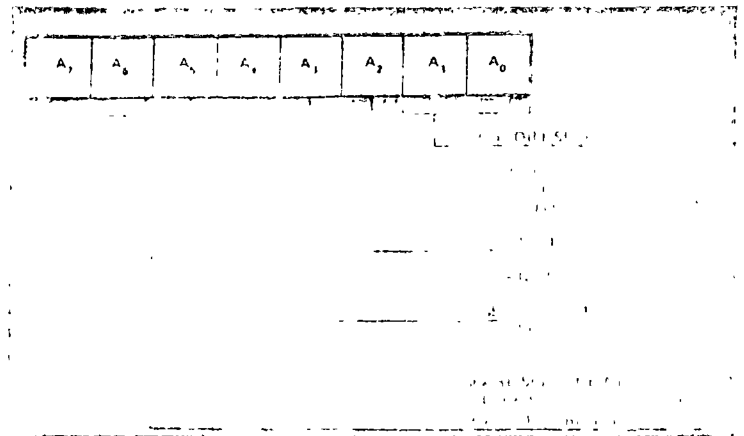


Fig. 2—By encoding bits A2-A7, the 8255 can generate up to 64 device select lines. In this linear select interconnect system, the CPU can select up to six I/O devices.

The system controller drives the I/O device read and write lines directly. With the resulting isolated I/O architecture, the CPU I/O instructions reference independent I/O address locations. Note that this system uses no external logic; even in larger systems the CPU-to-I/O interface requires only minimum external logic.



PORT SELECTED	HEXADECIMAL PORT SELECT CHARACTER (USED WITH IN OR OUT INSTRUCTIONS)
PORT A 8255 1	F8
PORT B 8255 1	F9
PORT C 8255 1	FA
CONTROL WORD REGISTER 8255 1	FB
PORT A 8255 2	F4
PORT B 8255 2	F5
PORT C 8255 2	F6
CONTROL WORD REGISTER 8255 2	F7

Fig. 3—This I/O port select coding applies to the system shown in Fig. 2.

Peripheral interface—strobed I/O or bus

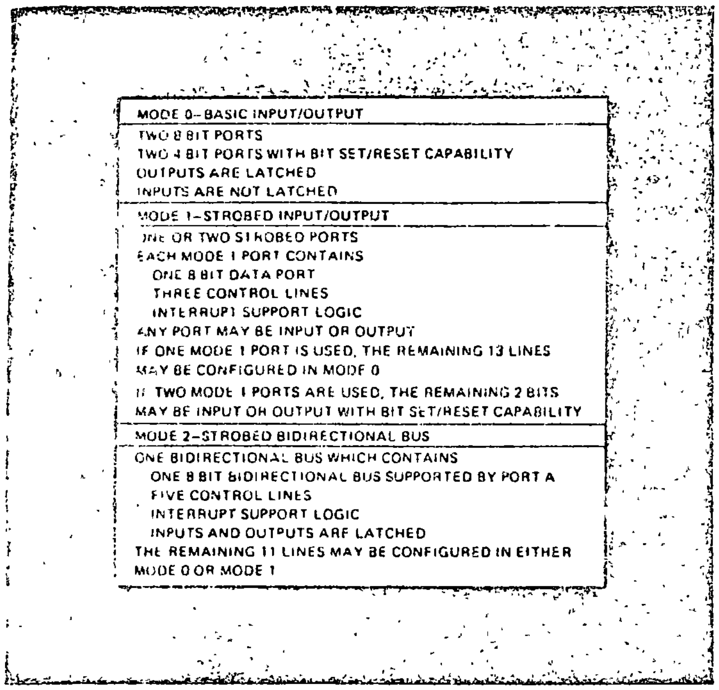
One of the greatest features of a programmable I/O unit is its flexibility; the 8255's flexibility stems from the fact that it can be programmed to operate in any of three different and internally flexible modes. Because the characteristics of the interface lines vary with each mode, you should be aware of those changes. **Table 1** lists the basic features of the interface lines for each of the available modes, while **Fig. 4** illustrates the groupings of these lines for each mode.

Both Table 1 and Fig. 4 point out an interesting aspect of the 8255: Port C is arranged as two 4 bit ports, so you can use it as I/O or control depending on the mode selected. Additionally, any of port C's bits can be set or reset independently, so you can generate device strobes by software without any external logic.

In both Mode 1 and Mode 2, several lines on port C handle interrupt control. This means that a significant portion of the logic required for interrupt drive I/O is on the I/O device itself, simplifying the design of interrupt-driven hardware. We will present an example of this type of application after discussing the internal logic and software considerations.

Internal logic — MSB determines function

Internal logic on the 8255's internal data bus accomplishes transfer of data and control information. When lines A_0 and A_1 select port A, B or



MODE 0–BASIC INPUT/OUTPUT
TWO 8 BIT PORTS
TWO 4 BIT PORTS WITH BIT SET/RESET CAPABILITY
OUTPUTS ARE LATCHED
INPUTS ARE NOT LATCHED

MODE 1–STROBED INPUT/OUTPUT
ONE OR TWO STROBED PORTS
EACH MODE 1 PORT CONTAINS
ONE 8 BIT DATA PORT
THREE CONTROL LINES
INTERRUPT SUPPORT LOGIC
ANY PORT MAY BE INPUT OR OUTPUT
IF ONE MODE 1 PORT IS USED, THE REMAINING 13 LINES MAY BE CONFIGURED IN MODE 0
IF TWO MODE 1 PORTS ARE USED, THE REMAINING 2 BITS MAY BE INPUT OR OUTPUT WITH BIT SET/RESET CAPABILITY

MODE 2–STROBED BIDIRECTIONAL BUS
ONE BIDIRECTIONAL BUS WHICH CONTAINS
ONE 8 BIT BIDIRECTIONAL BUS SUPPORTED BY PORT A
FIVE CONTROL LINES
INTERRUPT SUPPORT LOGIC
INPUTS AND OUTPUTS ARE LATCHED
THE REMAINING 11 LINES MAY BE CONFIGURED IN EITHER MODE 0 OR MODE 1

Table 1—Interface line characteristics vary for each of the 8255's three distinctly different operating modes.

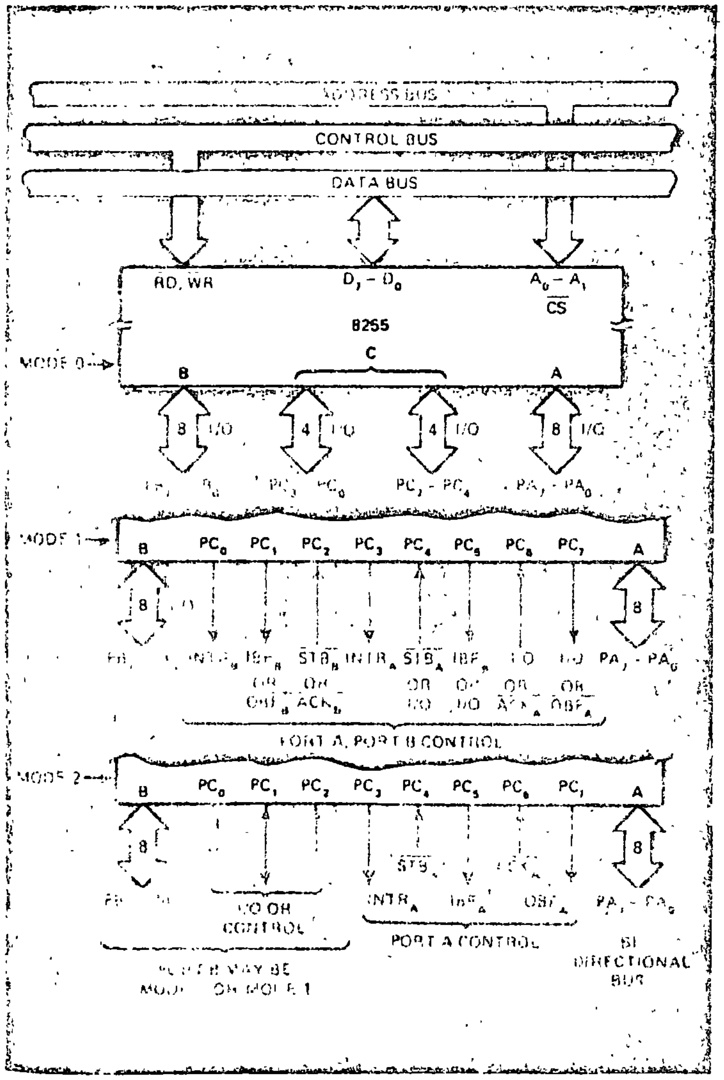


Fig. 4—Because port C is arranged as two 4 bit ports, you can use it for I/O control.

C, the operation is an I/O port data transfer; i.e., internal logic selects the specified port and performs the transfer. If the control-word register is selected, internal logic performs the operations described by the control word.

In the latter case, the most significant bit (MSB) of the control word determines the function to be performed. If the MSB is a ONE, the control word is interpreted to mean mode definition; subsequent bits determine mode and port assignment (**Fig. 5**). For example, the control word 10010101 means the function is mode definition; port A is to be Mode 0 as an input port; port C higher order bits are to be outputs; port B is to be Mode 1 as an output port; and port C bit 3 is an input. (Bits 0 through 2 on port C are used for port B control.)

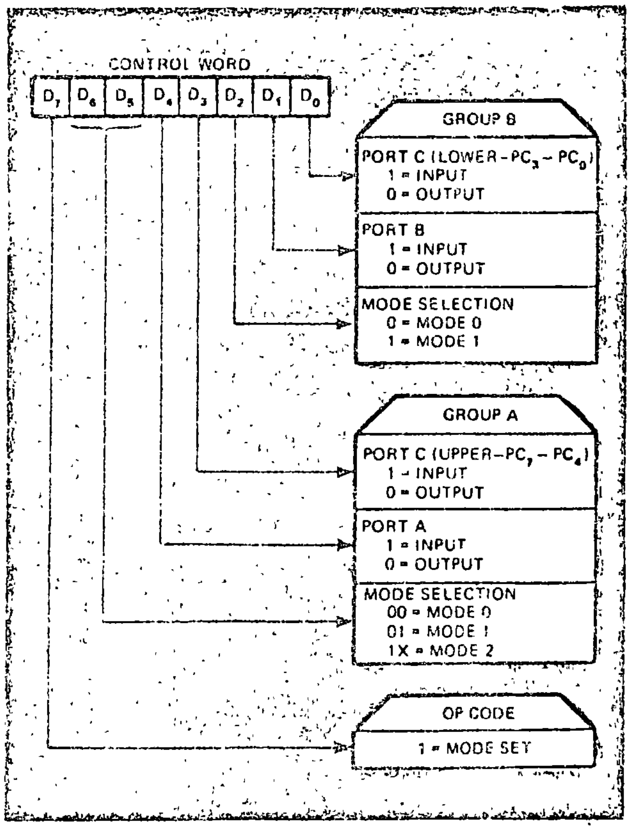
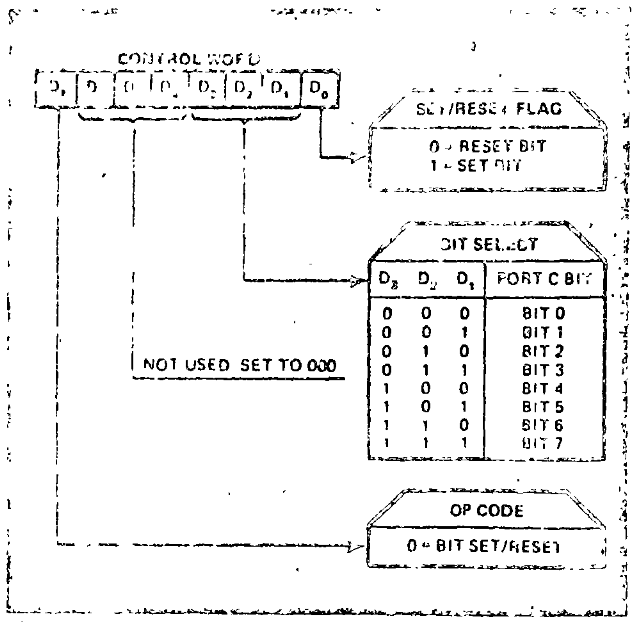


Fig. 5—When control lines A_0 and A_1 select the control-word register, the MSB of the control word determines the function performed. Here the MSB is a ONE, so the control word provides mode definition.

If the control word's MSB is a ZERO, the 8255 interprets it to be a port C bit set/reset command; subsequent bits determine which bits are to be set or reset. Any of the eight bits of port C can be independently set or reset. **Fig. 6** illustrates the format of the control word to accomplish bit set/reset. Since bits 4 through 6 of the control word are not needed, they should be set to zero. Thus, in this operating mode, the control word 00001001 indicates that bit 4 of port C is to be set.



D3	D2	D1	PORT C BIT
0	0	0	BIT 0
0	0	1	BIT 1
0	1	0	BIT 2
0	1	1	BIT 3
1	0	0	BIT 4
1	0	1	BIT 5
1	1	0	BIT 6
1	1	1	BIT 7

Fig. 6—Setting the control word's MSB to ZERO lets you independently set or reset any of the eight bits of port C.

Interrupt control logic status words

When the 8255 functions in either Mode 1 or Mode 2, it issues a status word to the software to reflect the state of the interrupt logic that it's supporting. In Mode 1, a status word is issued when port C is read; this word appears in the format shown in **Fig. 7**. Here the bits in the status word correspond to the states of the associated port C lines (buffer full, interrupt request, etc.).

Of course, the status word issued by the 8255 varies in structure depending on the function of the mode. **Fig. 8** shows the status word that results from a read of port C when the 8255 is programmed to have port A in Mode 1 as an input and port B in Mode 1 as an output.

A status word issued when the 8255 is functioning in Mode 2 is different, because in Mode 2, port B can operate in either Mode 0, Mode 1 or Mode 2. Status word bits D_2, D_1 and D_0 reflect the mode of port B (**Fig. 9**).

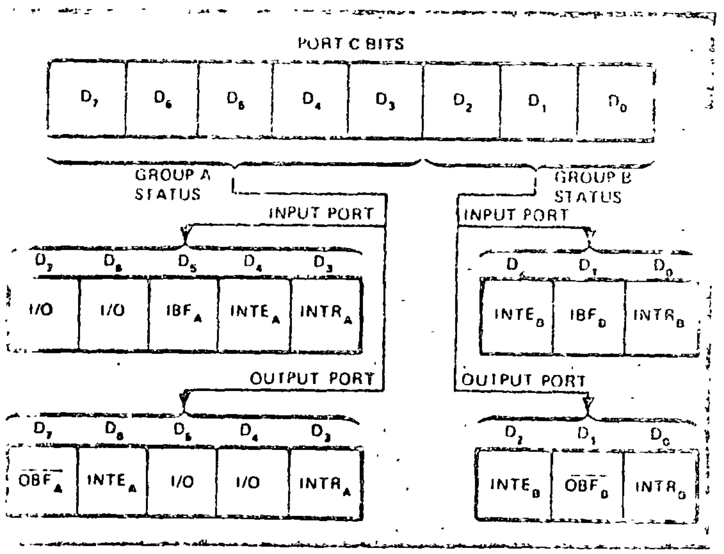


Fig. 7—In Mode 1, the 8255 issues a status word in this format when port C is read.

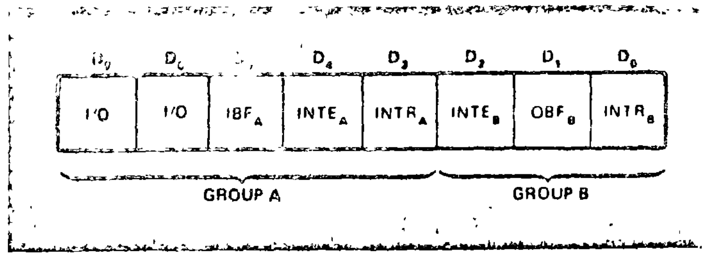


Fig. 8—Structure of the Mode 1 status word varies depending on the function of the mode. Here is what you get from a read of port C when you program the 8255 to have port A in Mode 1 as an input and port B in Mode 1 as an output.

Software considerations for the PPI

All the flexibility and versatility of the hardware is not much good unless some software issues the correct control word and supporting information to the programmable I/O device. The 8255's software requirements vary according to its operating mode. Generally, however, you need an initialization routine that issues the correct mode control word, sets up the initial states of the control lines and initializes program internal data.

For simple status-driven device interfaces with no interrupts, software reflecting the flowcharts in **Fig. 10** proves quite adequate. Most devices support BUSY/READY signals to determine I/O

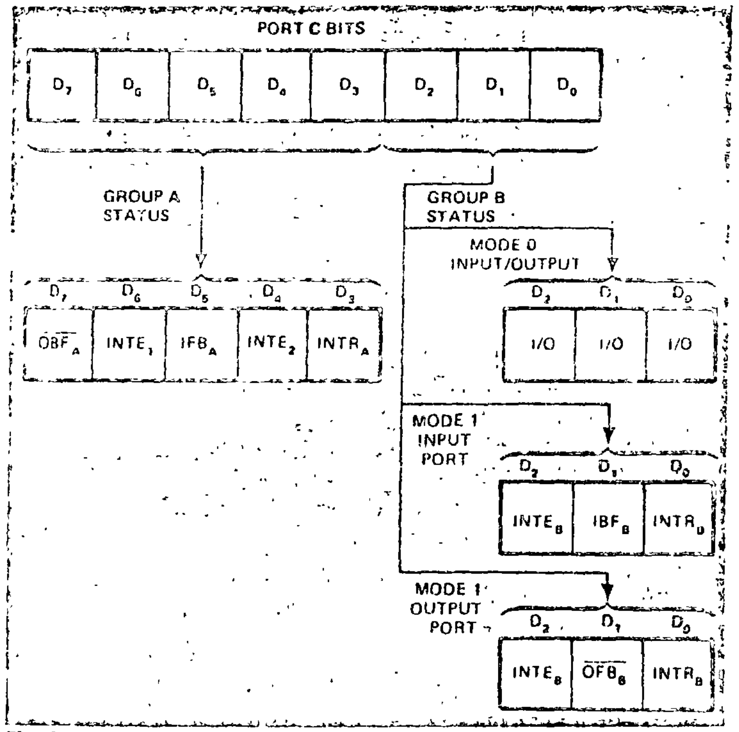


Fig. 9—Port B can operate in Mode 0, Mode 1 or Mode 2 when the 8255 functions in Mode 2, and status word bits D_0, D_1 and D_2 indicate port B's operation.

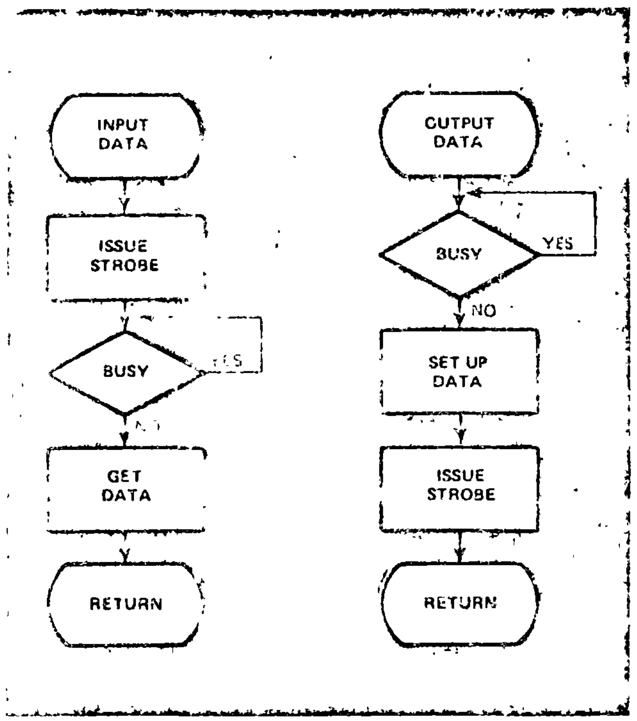


Fig. 10—Software flowcharts such as these suffice for simple status-driven device interfaces with no interrupts.

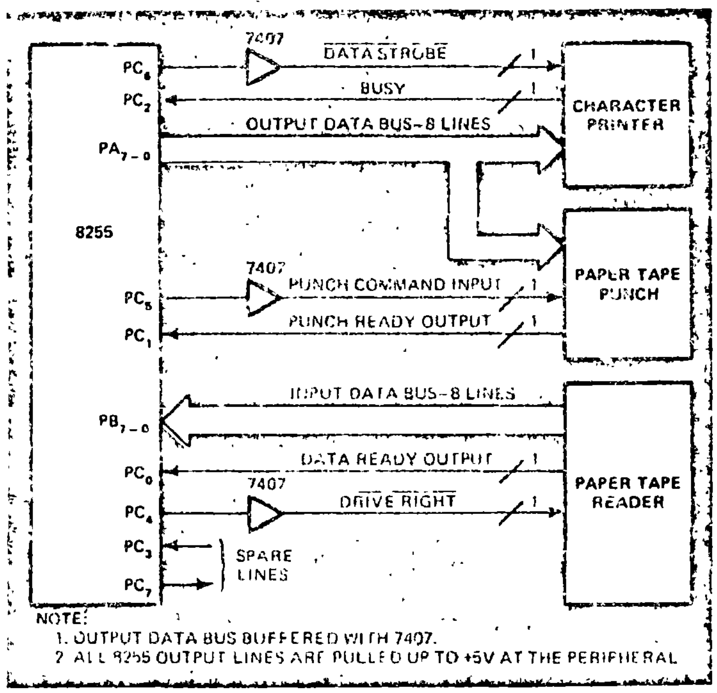


Fig. 11—The three peripherals in this status-driven interface system support the standard BUSY/DATA STROBE signals

readiness, and a DATA STROBE to request data transfer. If necessary, you can easily generate the DATA STROBE with the 8255's port C bit set/reset capability.

With Mode 1 and Mode 2 operation, the 8255 provides interrupt-driven peripheral capability. This usually means that the software must be able to support interrupts. Unfortunately, routines for interrupt-driven situations tend to be relatively complex because they usually are constructed to allow other software tasks to be performed while the I/O operation occurs. A common approach breaks the routine down into a command processor and an interrupt service routine.

The command processor initiates program requests and passes information to the interrupt service routine. Upon validating the transaction and starting the operation, it returns control to the requesting part of the program. Meanwhile, the interrupt service routine processes the transaction with the information passed.

For an interrupt service routine to be effective, it must perform several functions: It must save the state of registers, status, etc., so that they can be restored after the interrupt is processed; determine the source of the interrupt; and be able to pass data to and from the 8255. Further, it must return control to the requesting routine at the end of the I/O operation and restore the state of the machine prior to returning to the interrupted program.

With these factors in mind, you can use one of several methods to write command processor and interrupt service routines.

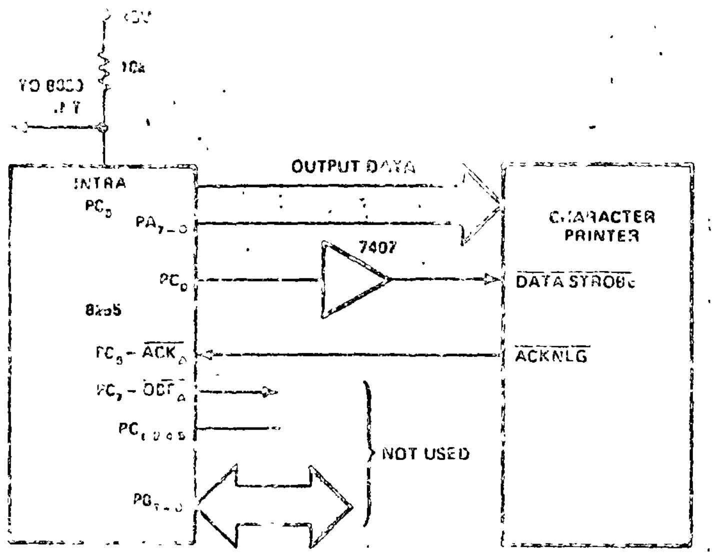
A peripheral driven by status

Let's now examine a typical application of the 8255: **Fig. 11** shows an interface block diagram for a status-driven interface where the peripherals are a character printer, a paper-tape punch and a paper-tape reader. All of these units support the standard BUSY/DATA STROBE signals previously mentioned.

In this interface, both port A and port B of the 8255 are Mode 0 (simple 8 bit I/O). Port A is an output port; port B, an input port. Three of the lower order port C bits (PC_0-PC_2) input peripheral busy signals, while three of the higher order bits (PC_4-PC_6) output strobe signals to the peripherals. Because the drive requirements of these output lines depend on the peripherals, length of interconnections and operating environment, the 7407 that we have used may not be the appropriate buffer for all applications.

Addressing the 8255 is accomplished with the linear select approach illustrated earlier in **Fig. 2.** All of the CPU-to-8255 interface lines are driven directly by the CPU module.

Although fairly simple, **Fig. 11's** application illustrates the principles behind Mode 0 operation of the 8255. You could enhance the system somewhat by using one of the spare port C output lines to control punch direction—a modification that would require only a minor change in software.



NOTES:
1 DATA ARE BUFFERED WITH 7407
2 ALL 8255 OUTPUT LINES ARE PULLED UP TO +5V AT THE PERIPHERAL.

Fig. 12—Character printer can be handled with an interrupt-driven interface by operating the 8255 in Mode 1.

An interrupt-driven peripheral

In a status-driven peripheral the software driver must poll the peripherals to determine if an operation has been completed. By contrast, in an interrupt-driven peripheral interface, an interrupt can indicate the end of an operation, freeing the processor from the time-wasting polling task. The 8255 can be programmed to operate in Mode 1 to provide an interrupt-driven interface, as Fig. 12's interface for a character printer shows.

Here, port A operates in Mode 1, and instead of a BUSY/DATA STROBE interface, we make a DATA STROBE/ACK interface by using bit zero of port C as a data strobe, bit 6 of port C as an acknowledgement line from the printer and bit 3 of port C as the interrupt line to the CPU. Port B and the other bits of port C are not necessary in this application.

Interrupt support is provided by an 8228 system controller. In a small system requiring only one interrupt vector, this device can automatically provide an RST 7 instruction to the data bus at the appropriate time.

In operation, a character is transferred to the printer on port A upon a DATA STROBE signal on bit zero, port C. The ACK signal from the printer notifies the 8255 that the character has been transferred.

Software for this interrupt-driven application is based on the typical command processor/interrupt service routine mentioned earlier. The initialization routine issues the mode control word after the 8255 is reset. This routine also places a jump to the interrupt service routine in the interrupt location for an RST 7. A user routine starts the command processor and checks whether or not an I/O operation is in progress. If not, it starts the I/O, enables the interrupt and returns to the calling routine so that other processing can take place.

The data-transfer operation places the character in the output buffer, then a DATA STROBE causes that character to be sent to the printer. The printer next generates an ACK signal to the 8255, clearing the "buffer full" indicator. Then the 8255 generates an interrupt, in turn processed by the CPU. Once the CPU determines the location of the interrupting unit, the interrupt service routine passes control to the user's routine, with the state of the processor restored to the conditions that existed prior to the interrupt.

This type of interface illustrates the capability of the 8255 to support an interrupt-driven hardware interface without any external logic. In larger systems, you may want to provide hardware to locate the source of the interrupt request.

As previously mentioned, software required by an interrupt-driven system is inherently more complex than that needed for a status-driven interface, but the added complexity is usually a small price to pay for a significant increase in system throughput.

A challenge to the user

Space limitations preclude us from illustrating all of the many types of interface applications that the 8255 can handle in its three operating modes. Even within these modes, variations are possible. We have shown the basic I/O capabilities of Mode 0 and the more sophisticated capability of a strobed I/O port possible with Mode 1. In addition, a bidirectional bus configuration, Mode 2, offers even more flexibility to designers. □

Author's biography

Alan Ebright, a MOS design engineer in Intel's MCD division, is currently involved in the design of µP peripheral support chips. He received his BSEE degree from the Univ. of California at Berkeley. As a hobby, Alan collects exotic tropical fish.

Article evaluation: Please circle one on R.S. card.
Read Some—No. 307 Read Most—No. 308

USART—a universal μP interface for serial data communications

With everything conveniently packaged on one chip, these receiver/transmitter devices reduce cost and complexity of interface design.

Lionel Smith, Intel Corp.

Today's microprocessors handle and store data in parallel form. In many cases, data transfer to and from peripherals can be effected in this format. However, in certain data-transfer applications—via modems or devices with modem-like interfaces—a bit serial format is necessary.

To satisfy these needs, μP manufacturers like Signetics, TI, Motorola, Western Digital, General Instruments and Intel have developed universal interfaces for serial communications. While some of these circuits, **Table 1**, operate in either the asynchronous or synchronous transmission modes, others can operate in both.

One device which falls into the latter category is the 8251 USART (universal synchronous/asynchronous receiver transmitter) (Fig. 1). Because it satisfies data-transfer requirements in all transmission modes, a functional description of this device will cover the general operation of most programmable interfaces. Signal designations will of course be unique to the 8251 in most cases, but the theory of operation is universal.

Let's start our trip inside the chip

In the synchronous mode, the USART operates with 5-, 6-, 7- or 8-bit characters. An additional bit can provide even or odd parity for error checking. Synchronization can be achieved externally via added hardware, or internally via sync-character detection. To prevent loss of synchronization in the event software fails to supply data in time, single or double sync characters are automatically inserted into the data stream.

Asynchronous operation uses the same data and parity structures. A START and one, one and a half, or two STOP bits are appended to this data. The receiver checks proper framing, and a status flag is set if an error occurs.

TABLE 1
COMPARISON OF SERIAL INTERFACE ICs

	INTEL 8251	MOTOROLA 6850	AMI 7350	WESTERN DIGITAL TR1402A	WESTERN DIGITAL UR1671B
SYNCHRONOUS/ASYNCHRONOUS	BOTH	ASYNCHRONOUS	SYNCHRONOUS	ASYNCHRONOUS	BOTH
5 TO 8 BIT CHARACTERS	YES	NO	YES	YES	YES
FULL DUPLEX	YES	YES	YES	YES	YES
ERROR DETECTION (PARITY, ETC.)	YES	YES	YES	YES	YES
BUS ORGANIZED	YES	YES	NO	NO	YES
NUMBER OF PINS	28	24	40	40	40
POWER SUPPLIES	+5V	+5V	+5V	+5V -12V	+12V +5V -12V
INT EXT SYNC DETECTION	YES	NA	INTERNAL	NA	INTERNAL
AUTO SYNC CHARACTER INSERT	YES	NA	YES	NA	YES
ONE, ONE AND A HALF, OR TWO STOP BITS	YES	1 OR 2	NA	1 OR 2	YES

This USART can transmit in half- or full-duplex mode and is double-buffered internally (i.e., the software has a complete character time to respond to a service request). Although it supports such basic data-set control signals as DTR (data-terminal-ready) and RTS (request-to-send), it does not support such signals as CF (carrier-detect) and CE (ring-indicator). An additional port will also be required to implement secondary channel signals. Finally, additional drivers and receivers are needed to interface to the voltage levels required by EIA RS232C.

In order to keep the hardware as flexible as possible (both at the chip and end-product level), all specific operating characteristics required to

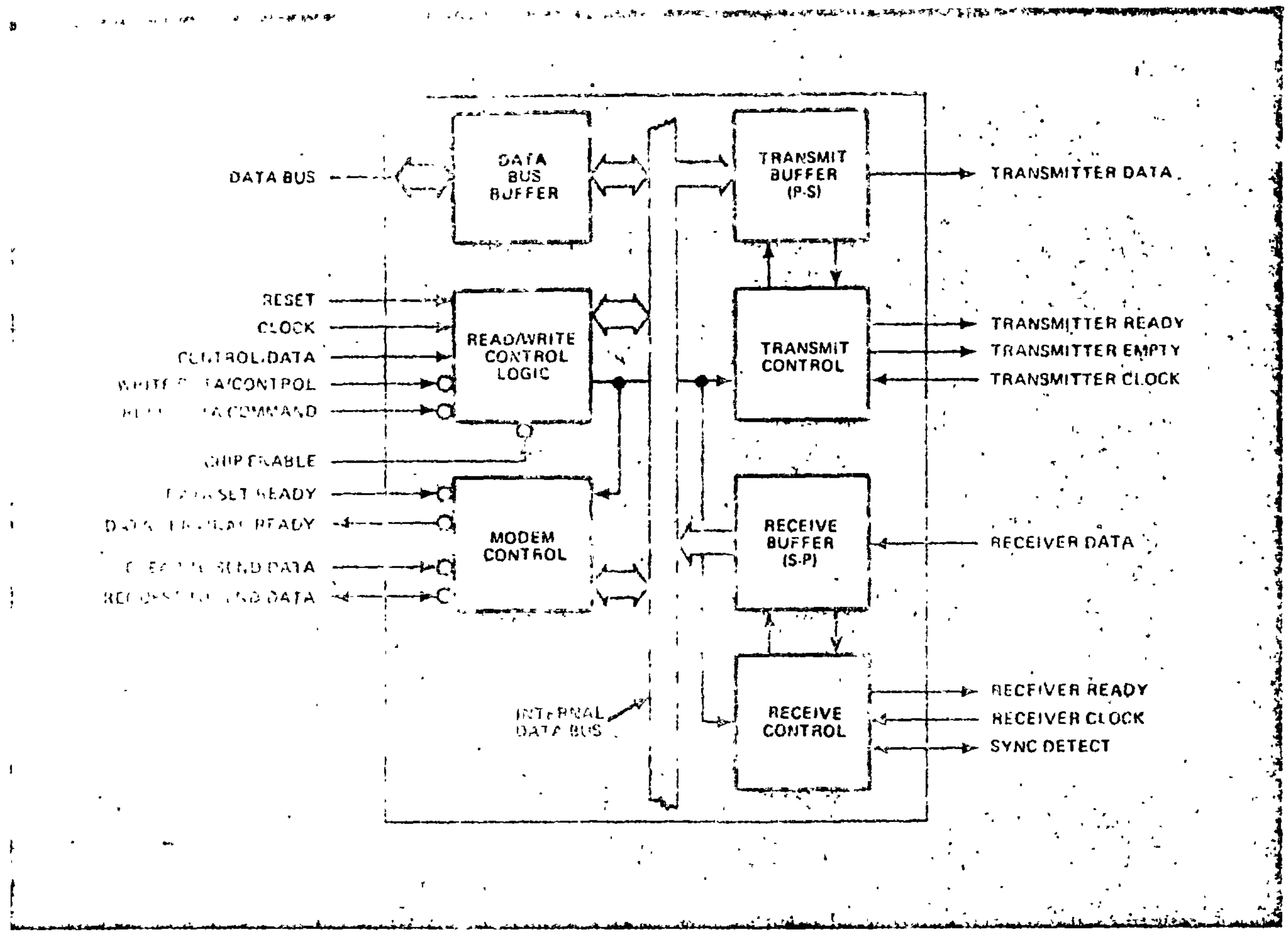


Fig. 1—Designed to maximize user flexibility, the USART can be programmed to operate using virtually any serial data-transmission technique in use today. The 8251 is double-buffered internally and operates in either a half- or full-duplex mode.

support a desired communications format are programmed into the USART by the systems software of the CPU. This programming takes place immediately following a reset operation (internal or external). The CPU sends out a set of control words consisting of both mode instructions and command instructions to program baud rate, character length, number of stop bits, synchronous or asynchronous operation, even or odd parity, etc. In the synchronous mode, options are provided to select either internal or external character synchronization.

Once programmed, the USART is ready to perform its communication functions, and so signals the CPU by raising the transmitter-ready output HIGH. This output is reset automatically when the CPU writes in a character. In the reverse operation, serial data can be received from the modem or I/O device. After receiving an entire character, the receiver-ready output is raised HIGH to signal the CPU that a complete character is ready for the CPU to fetch. Again, reset (of receiver-ready) is automatic upon completion of the CPU read operation.

Transmission mode selection comes next

As the initialization flowchart (**Fig. 2a**) shows, the first operation that must occur following a reset is the loading of the mode control register. Since the USART needs this information to structure its internal logic, it is essential to complete the initialization before any attempts are made at data transfers (including reading status).

Following a reset, the mode control register is loaded by the first control output. The mode control instruction, **Fig. 2b**, can be considered as four 2-bit fields. The first 2-bit field (D_1, D_0) determines whether the USART operates in the synchronous or asynchronous mode. In the asynchronous mode this field also controls the clock scaling factor. As an example, if D_1 and D_0 are ONE's, both the receiver and transmitter clocks will be divided by 64 to establish the baud rate.

The second field (D_3, D_2) determines the number of data bits in the character length, while the third (D_5, D_4) controls parity generation. When setting up the character length, note that

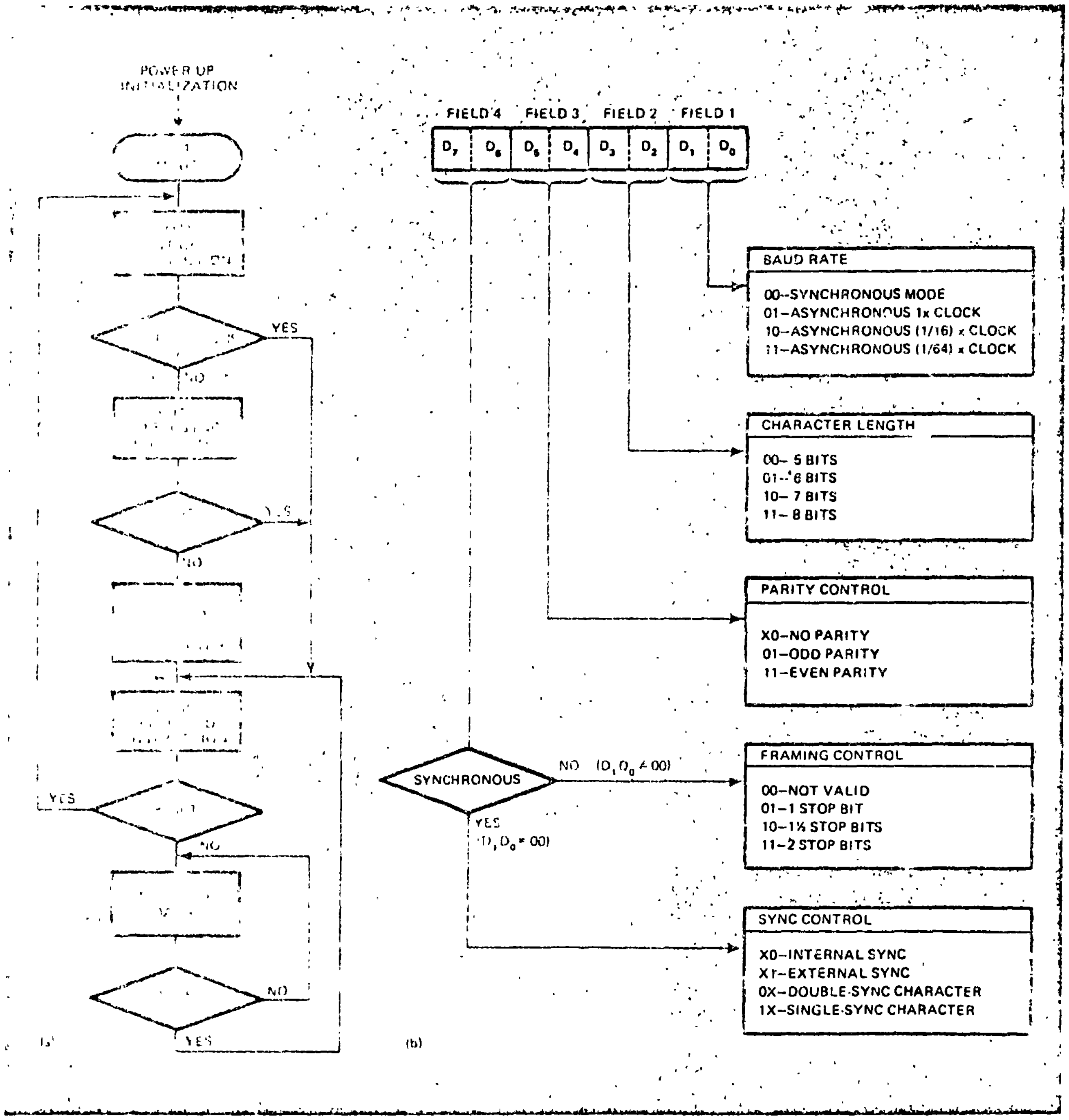


Fig. 2—Loading the mode control register is the first operation following a reset. The first two bits of the mode control instruction (b) determine both the operating mode and clock scaling factor

the parity bit (if enabled) is added to the data bits and is not considered as part of them.

Depending on the mode of operation, the last field (D_7, D_6) has two meanings. For the asynchronous mode, it controls the number of STOP bits to be transmitted with the character. Since the receiver will always operate with only one STOP bit, D_7 and D_6 really only control the transmitter. In the synchronous mode, this last field controls the synchronizing process. Note that the choice of single or double sync characters is independent of the choice of internal or external synchronization. Even though the receiver may operate with external synchronization logic, the transmitter must still know whether to send one or two sync characters should the CPU fail to supply a character in time.

If synchronous mode has been specified, the appropriate sync character (or characters) must be loaded following the loading of the mode instruction. It is loaded by the same control instruction used to load the mode instruction.

At completion of sync character(s) loading (or after the mode instruction in the asynchronous mode), a command character (Fig. 3) is issued to the USART. This command instruction controls the operation of the USART within the basic framework established by the mode instruction. A status register, which can be read under program control, indicates the state of the USART.

When operating the receiver, it is important to realize that receive-enable (bit D_2 of the command instruction) only inhibits the assertion of receiver-ready and does not inhibit the actual reception of characters. Since the receiver is constantly running, it can contain extraneous data when enabled. To avoid problems, this data should be read from the USART and discarded. The read should be done immediately following the setting of receive-enable in the asynchronous mode and following the setting of enter-hunt in the synchronous mode. It is not necessary to wait for receiver-ready before executing the dummy read.

The parallel μP meets the serial modem

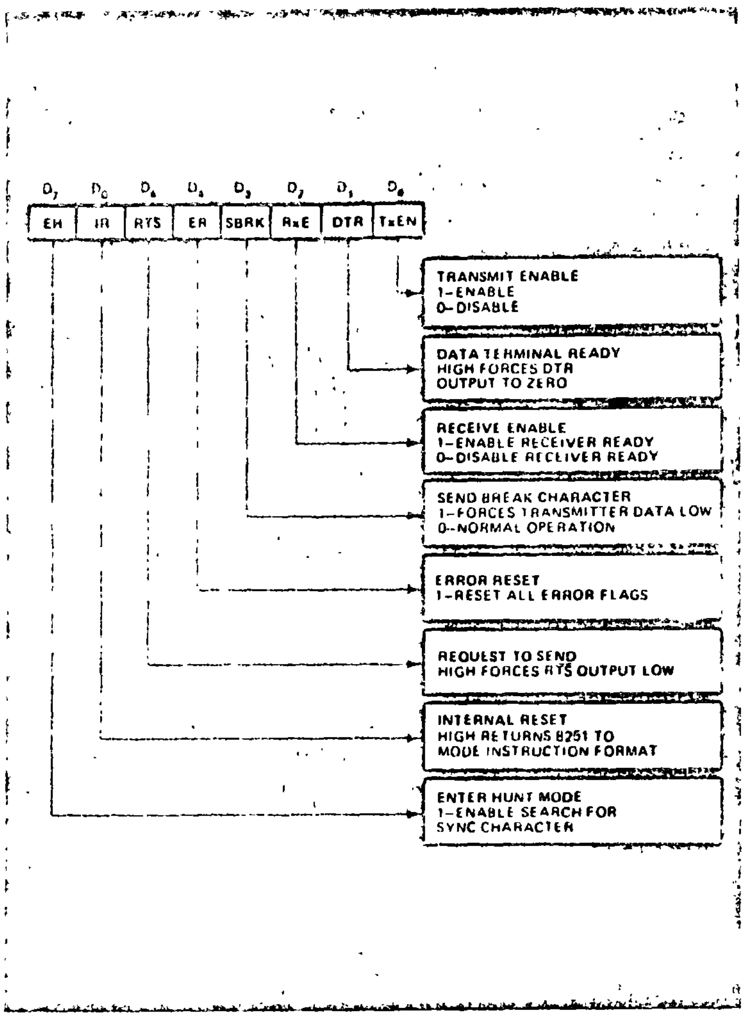
Fig. 4 shows a complete system. For ease in following its operation, we will consider first the interrelationship between the 8080A CPU and the USART and then between the USART and either a synchronous or asynchronous modem.

TABLE 2
TRUTH TABLE FOR INTERNAL BUS AND EXTERNAL I/O BUS

[illegible]	[illegible]	[illegible]	[illegible]	FUNCTION
[illegible]	[illegible]	[illegible]	[illegible]	READS DATA FROM USART
[illegible]	[illegible]	[illegible]	[illegible]	WRITES DATA TO USART
[illegible]	[illegible]	[illegible]	[illegible]	[illegible] STATUS FROM USART
[illegible]	[illegible]	[illegible]	[illegible]	WRITES COMMAND TO USART
[illegible]	[illegible]	[illegible]	[illegible]	(BUS FLOATS, NO OPERATION)

A 3-state, bi-directional, 8-bit buffer interfaces the USART to the CPU. Control words, command words and status information are also transferred through the data bus buffer. Note that while two registers store data for transfer to the CPU (STATUS and RECEIVE DATA), only one register stores data being transferred to the USART. This sharing of the input register, for both transmit data and commands, dictates that this register be clear of any data before any command is sent to the device. Such assurance can be provided by monitoring the transmitter-ready line. If it is low, neither data nor command transfer should be attempted. Erroneous data transmission can result if this check is not performed.

Information on the CPU's control bus (Table 2) is decoded by the READ/WRITE control logic into signals which gate data on and off the USART's internal bus and control the external I/O bus. If neither READ nor WRITE is ZERO, then the USART will not perform the I/O function. READ and WRITE being a ZERO at the same time is an illegal state with undefined results. The READ/WRITE control logic contains synchronization circuits so that the READ and WRITE pulses can occur at anytime with respect to the clock inputs to the USART.



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
EH IR RTS ER SBRK RxE DTR TxEN
TRANSMIT ENABLE 1–ENABLE 0–DISABLE
DATA TERMINAL READY HIGH FORCES DTR OUTPUT TO ZERO
RECEIVE ENABLE 1–ENABLE RECEIVER READY 0–DISABLE RECEIVER READY
SEND BREAK CHARACTER 1–FORCES TRANSMITTER DATA LOW 0–NORMAL OPERATION
ERROR RESET 1–RESET ALL ERROR FLAGS
REQUEST TO SEND HIGH FORCES RTS OUTPUT LOW
INTERNAL RESET HIGH RETURNS 8251 TO MODE INSTRUCTION FORMAT
ENTER HUNT MODE 1–ENABLE SEARCH FOR SYNC CHARACTER

Fig. 3—An 8-bit command instruction controls the operation of the USART within the framework established by the mode instruction.

Directing transmissions to the modem

The USART's modem-control section generates the request-to-send data and receives the clear-to-send data signals. In addition, two general-purpose terminals are provided—a data-terminal-ready output and a data-set-ready input. Although the USART attaches no special significance to these signals, the former is normally assigned to the modem to indicate a ready terminal while the latter indicates a ready modem.

The USART transmitter accepts parallel data from the CPU, adds the appropriate framing information, serializes it, and transmits it on the transmitter-data pin. In the asynchronous mode the transmitter always adds a START bit. Depend-

Some practical design hints

Several areas involving signal timing and signal interaction can cause operational problems. Based on our experience with users, the following design hints will help avoid such problems.

- A command output to the USART will destroy the integrity of any transmission in progress. Any character stored in the buffer, waiting for transfer to the parallel-to-serial converter, will be written over. This problem can be avoided by waiting for transmitter-ready to be asserted before sending a command. Transmission will also be disturbed if a command is sent while a sync character is being generated by the USART. To prevent this problem, commands should be transferred only when a positive-going edge is detected on the transmitter-ready line.
- Receive-enable does not control receiver operation; it only acts as a mask for receiver-ready. The receiver could contain one or two characters when it is eanbled. If this condition will cause system problems, these characters should be read and discarded when the receive-enable bit is first set.
- Loss of clear-to-send data or transmitter-enable will immediately clamp the serial output line. These signals should remain asserted until the transmission is complete, and complete in this case means clear of the modem. A delay of 1 msec, following detection of transmitter-empty, is usually sufficient. In the synchronous mode, loss of transmit-enable will clamp the data in at a SPACE instead of the normal MARK. An external gate, combining request-to-send data and the serial output data, can correct this problem.
- Extraneous transitions could occur on the transmitter-empty line during transfer of data (including USART-generated sync's) to the parallel-to-serial converter. To insure no such transitions, allow time for transmitter-empty to occur through several consecutive status reads before assuming that the transmitter is truly in the empty state.
- A BREAK detected by the receiver results in a string of characters which have framing errors. If reception is to continue after a break, care must be taken to ensure that the received data is valid. This is a special problem with the last character perceived during a BREAK since its value, including any associated framing error, is indeterminate.

ing on how the unit is programmed, it also adds an optional even or odd parity bit, and either one, one and a half, or two STOP bits. In the synchronous mode, the transmitter does not generate any extra bits (other than parity if enabled) unless the computer fails to send a character to the USART. In that case, the USART will transmit a sync character.

If the USART operates in the dual sync mode, both sync characters will be transmitted before the message can be resumed. No sync character will be generated until the software has supplied at least one character (i.e., the USART will fill "holes" in the transmission but will not initiate transmission itself). Until transmit-enable and clear-to-send are asserted, transmission is inhibited.

An additional feature of the transmitter is the ability to transmit a BREAK. A BREAK is a period of continuous SPACE's on the communication line and is used in full-duplex communication to interrupt the transmitting terminal. A BREAK condition will be transmitted as long as the D_3 bit of the command register is set.

Listening to the outside world

The USART receiver accepts serial data on the receiver-data pin and converts it to parallel data according to the appropriate format. When the USART is in the asynchronous mode and ready to accept a character, it looks for a low level on the receiver-data line. When it sees the low level it assumes a START bit and enables an internal counter. At a count equivalent to one half of a bit time the receiver-data line is sampled again. If the line is still low, reception of a valid START bit is assumed and assembly of the character begins.

On the other hand, if the second sampling detects a ONE, then either a noise pulse has occurred on the line or the receiver has become enabled in the middle of the transmission of a character. In either case, the receiver aborts its operation and prepares itself to accept a new character. After the successful reception of a START bit, data, parity and STOP bits are clocked in and then transferred on the internal data bus to the receive-data register. The transmitter-ready signal is then asserted to indicate that a character is available.

In the synchronous mode, the receiver simply clocks in the specified number of data bits and transfers them to the receiver buffer register, setting receiver-ready. Since the receiver blindly groups data bits into characters, there must be a means of synchronizing the receiver to the transmitter so that the proper character bounda-

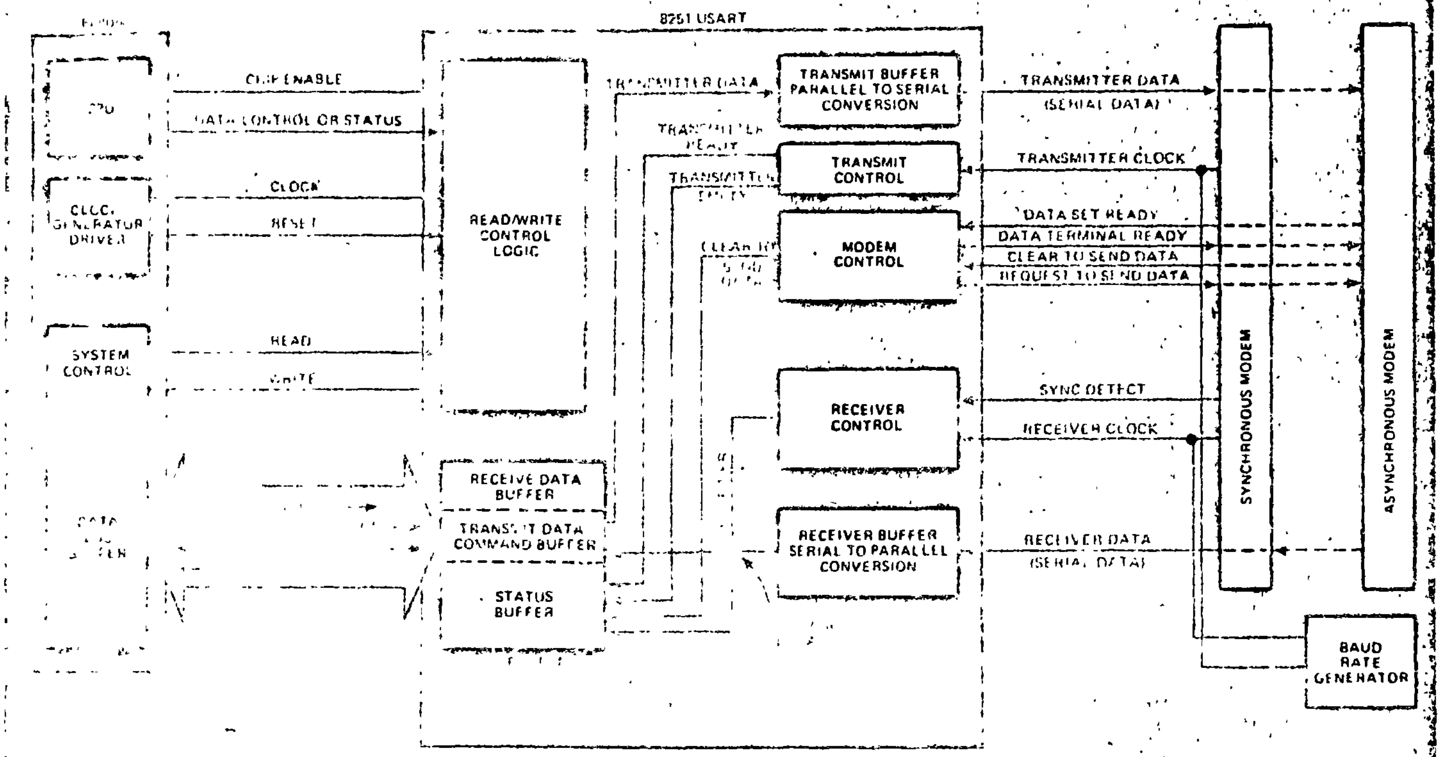


Fig. 4—Erroneous data transmission can result if data or command transfers to the USART are attempted before the transmit data/command buffer is clear. No transfer should be attempted if the transmitter-ready line is low.

...s are maintained in the serial data stream. This synchronization is achieved in the HUNT mode.

In the HUNT mode, data is shifted in on the receiver-data line one bit at a time. Upon receipt of each bit, the receiver register is compared to a register holding the sync character (program loaded). If the two registers are not equal, another bit is shifted in and the comparison is repeated. When the registers compare, the USART ends the HUNT mode and raises the sync-detect line to indicate synchronization. If 2-sync character operation has been programmed, two contiguous characters from the line must compare to the two stored sync characters before synchronization is declared. Parity is not checked. If the USART has been programmed to accept external synchronization, the sync-detect pin operates as an input to synchronize the receiver. The USART enters the HUNT mode when it is initialized into the synchronous mode or when it is commanded to do so by the command instruction. Before the receiver is operated, it must be enabled by the receive-enable bit (D_2) of the command instructions. If this bit is not set, the receiver will not assert the receiver-ready bit.□

Author's biography

Lionel Smith is a Senior Application Engineer in Intel's MCD div., Santa Clara, CA. A graduate of Case Institute of Technology, he has been with Intel for one year. Lionel's spare time is spent pursuing a wide range of interests. He is an avid science-fiction reader and a devotee of Shoto-Kan karate. His outdoor interests include field archery and backpacking.

Article evaluation: Please circle one on R.S. card.
Read Some—No. 307 Read Most—No. 308

CHAPTER 1 THE FUNCTIONS OF A COMPUTER

This chapter introduces certain basic computer concepts. It provides background information and definitions which will be useful in later chapters of this manual. Those already familiar with computers may skip this material, at their option.

A TYPICAL COMPUTER SYSTEM

A typical digital computer consists of:

a) A central processor unit (CPU)
b) A memory
c) Input/output (I/O) ports

The memory serves as a place to store **Instructions**, the coded pieces of information that direct the activities of the CPU, and **Data**, the coded pieces of information that are processed by the CPU. A group of logically related instructions stored in memory is referred to as a **Program**. The CPU "reads" each instruction from memory in a logically determined sequence, and uses it to initiate processing actions. If the program sequence is coherent and logical, processing the program will produce intelligible and useful results.

The memory is also used to store the data to be manipulated, as well as the instructions that direct that manipulation. The program must be organized such that the CPU does not read a non-instruction word when it expects to see an instruction. The CPU can rapidly access any data stored in memory; but often the memory is not large enough to store the entire data bank required for a particular application. The problem can be resolved by providing the computer with one or more **Input Ports**. The CPU can address these ports and input the data contained there. The addition of input ports enables the computer to receive information from external equipment (such as a paper tape reader or floppy disk) at high rates of speed and in large volumes.

A computer also requires one or more **Output Ports** that permit the CPU to communicate the result of its processing to the outside world. The output may go to a display, for use by a human operator, to a peripheral device that produces "hard-copy," such as a line-printer, to a peripheral storage device, such as a floppy disk unit, or the output may constitute process control signals that direct the operations of another system, such as an automated assembly line. Like input ports, output ports are addressable. The input and output ports together permit the processor to communicate with the outside world.

The CPU unifies the system. It controls the functions performed by the other components. The CPU must be able to fetch instructions from memory, decode their binary contents and execute them. It must also be able to reference memory and I/O ports as necessary in the execution of instructions. In addition, the CPU should be able to recognize and respond to certain external control signals, such as INTERRUPT and WAIT requests. The functional units within a CPU that enable it to perform these functions are described below.

THE ARCHITECTURE OF A CPU

A typical central processor unit (CPU) consists of the following interconnected functional units:

- Registers
- Arithmetic/Logic Unit (ALU)
- Control Circuitry

Registers are temporary storage units within the CPU Some registers, such as the program counter and instru[illegible] register, have dedicated uses. Other registers, such as the accumulator, are for more general purpose use.

Accumulator:

The accumulator usually stores one of the operands to be manipulated by the ALU. A typical instruction might direct the ALU to add the contents of some other register to the contents of the accumulator and store the result in the accumulator itself. In general, the accumulator is both a source (operand) and a destination (result) register.

Often a CPU will include a number of additional general purpose registers that can be used to store operands or intermediate data. The availability of general purpose registers eliminates the need to "shuffle" intermediate results back and forth between memory and the accumulator, thus improving processing speed and efficiency.

Program Counter (Jumps, Subroutines and the Stack):

The instructions that make up a program are stored in the system's memory. The central processor references the contents of memory, in order to determine what action is appropriate. This means that the processor must know which location contains the next instruction.

Each of the locations in memory is numbered, to distinguish it from all other locations in memory. The number which identifies a memory location is called its Address.

The processor maintains a counter which contains the address of the next program instruction. This register is called the Program Counter. The processor updates the program counter by adding "1" to the counter each time it fetches an instruction, so that the program counter is always current (pointing to the next instruction).

The programmer therefore stores his instructions in numerically adjacent addresses, so that the lower addresses contain the first instructions to be executed and the higher addresses contain later instructions. The only time the programmer may violate this sequential rule is when an instruction in one section of memory is a Jump instruction to another section of memory.

A jump instruction contains the address of the instruction which is to follow it. The next instruction may be stored in any memory location, as long as the programmed jump specifies the correct address. During the execution of a jump instruction, the processor replaces the contents of its program counter with the address embodied in the Jump. Thus, the logical continuity of the program is maintained.

A special kind of program jump occurs when the stored program "Calls" a subroutine. In this kind of jump, the processor is required to "remember" the contents of the program counter at the time that the jump occurs. This enables the processor to resume execution of the main program when it is finished with the last instruction of the subroutine.

A Subroutine is a program within a program. Usually it is a general-purpose set of instructions that must be executed repeatedly in the course of a main program. Routines which calculate the square, the sine, or the logarithm of a program variable are good examples of functions often written as subroutines. Other examples might be programs designed for inputting or outputting data to a particular peripheral device.

The processor has a special way of handling subroutines, in order to insure an orderly return to the main program. When the processor receives a Call instruction, it increments the Program Counter and stores the counter's contents in a reserved memory area known as the Stack. The Stack thus saves the address of the instruction to be executed after the subroutine is completed. Then the processor loads the address specified in the Call into its Program Counter. The next instruction fetched will therefore be the first step of the subroutine.

The last instruction in any subroutine is a Return. Such an instruction need specify no address. When the processor fetches a Return instruction, it simply replaces the current contents of the Program Counter with the address on the top of the stack. This causes the processor to resume execution of the calling program at the point immediately following the original Call Instruction.

Subroutines are often Nested; that is, one subroutine will sometimes call a second subroutine. The second may call a third, and so on. This is perfectly acceptable, as long as the processor has enough capacity to store the necessary return addresses, and the logical provision for doing so. In other words, the maximum depth of nesting is determined by the depth of the stack itself. If the stack has space for storing three return addresses, then three levels of subroutines may be accommodated.

Processors have different ways of maintaining stacks. Some have facilities for the storage of return addresses built into the processor itself. Other processors use a reserved area of external memory as the stack and simply maintain a Pointer register which contains the address of the most recent stack entry. The external stack allows virtually unlimited subroutine nesting. In addition, if the processor provides instructions that cause the contents of the accumulator and other general purpose registers to be "pushed" onto the stack or "popped" off the stack via the address stored in the stack pointer, multi-level interrupt processing (described later in this chapter) is possible. The status of the processor (i.e., the contents of all the registers) can be saved in the stack when an interrupt is accepted and then restored after the interrupt has been serviced. This ability to save the processor's status at any given time is possible even if an interrupt service routine, itself, is interrupted.

Instruction Register and Decoder:

Every computer has a Word Length that is characteristic of that machine. A computer's word length is usually determined by the size of its internal storage elements and interconnecting paths (referred to as Busses); for example, a computer whose registers and busses can store and transfer 8 bits of information has a characteristic word length of 8-bits and is referred to as an 8-bit parallel processor. An eight-bit parallel processor generally finds it most efficient to deal with eight-bit binary fields, and the memory associated with such a processor is therefore organized to store eight bits in each addressable memory location. Data and instructions are stored in memory as eight-bit binary numbers, or as numbers that are integral multiples of eight bits: 16 bits, 24 bits, and so on. This characteristic eight-bit field is often referred to as a Byte.

Each operation that the processor can perform is identified by a unique byte of data known as an Instruction

Code or Operation Code. An eight-bit word used as an instruction code can distinguish between 256 alternative actions, more than adequate for most processors.

The processor fetches an instruction in two distinct operations. First, the processor transmits the address in its Program Counter to the memory. Then the memory returns the addressed byte to the processor. The CPU stores this instruction byte in a register known as the Instruction Register, and uses it to direct activities during the remainder of the instruction execution.

The mechanism by which the processor translates an instruction code into specific processing actions requires more elaboration than we can here afford. The concept, however, should be intuitively clear to any logic designer. The eight bits stored in the instruction register can be decoded and used to selectively activate one of a number of output lines, in this case up to 256 lines. Each line represents a set of activities associated with execution of a particular instruction code. The enabled line can be combined with selected timing pulses, to develop electrical signals that can then be used to initiate specific actions. This translation of code into action is performed by the Instruction Decoder and by the associated control circuitry.

An eight-bit instruction code is often sufficient to specify a particular processing action. There are times, however, when execution of the instruction requires more information than eight bits can convey.

One example of this is when the instruction references a memory location. The basic instruction code identifies the operation to be performed, but cannot specify the object address as well. In a case like this, a two- or three-byte instruction must be used. Successive instruction bytes are stored in sequentially adjacent memory locations, and the processor performs two or three fetches in succession to obtain the full instruction. The first byte retrieved from memory is placed in the processor's instruction register, and subsequent bytes are placed in temporary storage; the processor then proceeds with the execution phase. Such an instruction is referred to as Variable Length.

Address Register(s):

A CPU may use a register or register-pair to hold the address of a memory location that is to be accessed for data. If the address register is Programmable, (i.e., if there are instructions that allow the programmer to alter the contents of the register) the program can "build" an address in the address register prior to executing a Memory Reference instruction (i.e., an instruction that reads data from memory, writes data to memory or operates on data stored in memory).

Arithmetic/Logic Unit (ALU):

All processors contain an arithmetic/logic unit, which is often referred to simply as the ALU. The ALU, as its name implies, is that portion of the CPU hardware which performs the arithmetic and logical operations on the binary data.

The ALU must contain an Adder which is capable of combining the contents of two registers in accordance with the logic of binary arithmetic. This provision permits the processor to perform arithmetic manipulations on the data it obtains from memory and from its other inputs.

Using only the basic adder a capable programmer can write routines which will subtract, multiply and divide, giving the machine complete arithmetic capabilities. In practice, however, most ALUs provide other built-in functions, including hardware subtraction, boolean logic operations, and shift capabilities.

The ALU contains Flag Bits which specify certain conditions that arise in the course of arithmetic and logical manipulations. Flags typically include Carry, Zero, Sign, and Parity. It is possible to program jumps which are conditionally dependent on the status of one or more flags. Thus, for example, the program may be designed to jump to a special routine if the carry bit is set following an addition instruction.

Control Circuitry:

The control circuitry is the primary functional unit within a CPU. Using clock inputs, the control circuitry maintains the proper sequence of events required for any processing task. After an instruction is fetched and decoded, the control circuitry issues the appropriate signals (to units both internal and external to the CPU) for initiating the proper processing action. Often the control circuitry will be capable of responding to external signals, such as an interrupt or wait request. An Interrupt request will cause the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program. A Wait request is often issued by a memory or I/O element that operates slower than the CPU. The control circuitry will idle the CPU until the memory or I/O port is ready with the data.

COMPUTER OPERATIONS

There are certain operations that are basic to almost any computer. A sound understanding of these basic operations is a necessary prerequisite to examining the specific operations of a particular computer.

Timing:

The activities of the central processor are cyclical. The processor fetches an instruction, performs the operations required, fetches the next instruction, and so on. This orderly sequence of events requires precise timing, and the CPU therefore requires a free running oscillator clock which furnishes the reference for all processor actions. The combined fetch and execution of a single instruction is referred to as an Instruction Cycle. The portion of a cycle identified with a clearly defined activity is called a State. And the interval between pulses of the timing oscillator is referred to as a Clock Period. As a general rule, one or more clock periods are necessary for the completion of a state, and there are several states in a cycle.

Instruction Fetch:

The first state(s) of any instruction cycle will be dedicated to fetching the next instruction. The CPU issues a read signal and the contents of the program counter are sent to memory, which responds by returning the next instruction word. The first byte of the instruction is placed in the instruction register. If the instruction consists of more than one byte, additional states are required to fetch each byte of the instruction. When the entire instruction is present in the CPU, the program counter is incremented (in preparation for the next instruction fetch) and the instruction is decoded. The operation specified in the instruction will be executed in the remaining states of the instruction cycle. The instruction may call for a memory read or write, an input or output and/or an internal CPU operation, such as a register-to-register transfer or an add-registers operation.

Memory Read:

An instruction fetch is merely a special memory read operation that brings the instruction to the CPU's instruction register. The instruction fetched may then call for data to be read from memory into the CPU. The CPU again issues a read signal and sends the proper memory address; memory responds by returning the requested word. The data received is placed in the accumulator or one of the other general purpose registers (not the instruction register).

Memory Write:

A memory write operation is similar to a read except for the direction of data flow. The CPU issues a write signal, sends the proper memory address, then sends the data word to be written into the addressed memory location.

Wait (memory synchronization):

As previously stated, the activities of the processor are timed by a master clock oscillator. The clock period determines the timing of all processing activity.

The speed of the processing cycle, however, is limited by the memory's Access Time. Once the processor has sent a read address to memory, it cannot proceed until the memory has had time to respond. Most memories are capable of responding much faster than the processing cycle requires. A few, however, cannot supply the addressed byte within the minimum time established by the processor's clock.

Therefore a processor should contain a synchronization provision, which permits the memory to request a Wait state. When the memory receives a read or write enable signal, it places a request signal on the processor's READY line, causing the CPU to idle temporarily. After the memory has had time to respond, it frees the processor's READY line, and the instruction cycle proceeds.

Input/Output:

Input and Output operations are similar to memory read and write operations with the exception that a peripheral I/O device is addressed instead of a memory location. The CPU issues the appropriate input or output control signal, sends the proper device address and either receives the data being input or sends the data to be output.

Data can be input/output in either parallel or serial form. All data within a digital computer is represented in binary coded form. A binary data word consists of a group of bits; each bit is either a one or a zero. Parallel I/O consists of transferring all bits in the word at the same time, one bit per line. Serial I/O consists of transferring one bit at a time on a single line. Naturally serial I/O is much slower, but it requires considerably less hardware than does parallel I/O.

Interrupts:

Interrupt provisions are included on many central processors, as a means of improving the processor's efficiency. Consider the case of a computer that is processing a large volume of data, portions of which are to be output to a printer. The CPU can output a byte of data within a single machine cycle but it may take the printer the equivalent of many machine cycles to actually print the character specified by the data byte. The CPU could then remain idle waiting until the printer can accept the next data byte. If an interrupt capability is implemented on the computer, the CPU can output a data byte then return to data processing. When the printer is ready to accept the next data byte, it can request an interrupt. When the CPU acknowledges the interrupt, it suspends main program execution and automatically branches to a routine that will output the next data byte. After the byte is output, the CPU continues with main program execution. Note that this is, in principle, quite similar to a subroutine call, except that the jump is initiated externally rather than by the program.

More complex interrupt structures are possible, in which several interrupting devices share the same processor but have different priority levels. Interruptive processing is an important feature that enables maximum untilization of a processor's capacity for high system throughput.

Hold:

Another important feature that improves the throughput of a processor is the Hold. The hold provision enables Direct Memory Access (DMA) operations.

In ordinary input and output operations, the processor itself supervises the entire data transfer. Information to be placed in memory is transferred from the input device to the processor, and then from the processor to the designated memory location. In similar fashion, information that goes from memory to output devices goes by way of the processor.

Some peripheral devices, however, are capable of transferring information to and from memory much faster than the processor itself can accomplish the transfer. If any appreciable quantity of data must be transferred to or from such a device, then system throughput will be increased by having the device accomplish the transfer directly. The processor must temporarily suspend its operation during such a transfer, to prevent conflicts that would arise if processor and peripheral device attempted to access memory simultaneously. It is for this reason that a hold provision is included on some processors.

MICROPROCESADORES: TEORIA Y APLICACIONES

TEMA 5 Y 6 : EL SISTEMA SDK - 80

TEORIA

M. EN C. ANGEL KURI MORALES

ABRIL, 1978.

CURSO DE MICROS

EL SISTEMA SDK-80.

Ante la gran demanda que tuvieron los microprocesadores (en particular el I8080), surgió la necesidad de producir sistemas ya configurados, es decir, que estuvieran ya integrados en una unidad funcional. Este tipo de unidades se conocen en el medio como 'kits'.

La ventaja de un kit, por encima de la compra de las partes discretas que lo componen, es que, en la gran mayoría de los casos, existe, para empezar, un desconocimiento parcial o total de la mecánica y la lógica implícita en un procesador. Ante esta situación, es de gran conveniencia empezar a trabajar con un sistema ya integrado, probado y diseñado por el fabricante, de bajo precio, con un sistema de 'software' mínimo, confiable y que, además, es de alta calidad y resistencia al desgaste.

El kit que ahora nos ocupa, es conocido como el SDK-80 (System Development Kit 8080) y es de gran popularidad. Consta, como es de suponer, de todas las partes básicas que conforman a un sistema 8080, tales como:

a) Un procesador central (CPU) 8080.
b) Un controlador de sistema (8228).
c) Un sistema de reloj (8224) y su correspondiente reloj de cuarzo de 2 Megahertz.
d) Dos kilobytes de ROM (2 8708).
e) Un kilobyte de RAM (2 8111).
f) Decodificadores para las memorias (2 8205).
g) Una interfaz programable para comunicación serie/paralelo (8251) conocida como USART (Universal Synchronous Asynchronous Receiver Transmitter).
h) Una interfaz programable para comunicación paralelo/paralelo, con tres puertos de entrada/salida (8255).
i) Elementos de lógica SSI (Small Scale Integration) para la generación de las frecuencias de comunicación que requiere el sistema.

j) Una tarjeta de circuito impreso en la cual se acomodan las componentes anteriormente mencionadas. En esta tarjeta, además, hay espacio para agregar un 8255 más, dos 8212 (buffers/drivers) para arreglos de expansión de memoria, y dos Kbytes más de ROM. Asimismo, hay un área de expansión para arreglos alambrados tipo 'wire wrap'.

k) El sistema incluye también conectores 'amphenol' para la comunicación de la micro con el mundo exterior, y un regulador de voltaje auxiliar para lograr los voltajes de polarización de la 8080.

l) Finalmente, todas las componentes discretas adicionales (tales como resistencias, capacitores y transistores) vienen incluídas en este sistema mínimo.

Por otro lado, el sistema sería incapaz de ejecutar programa alguno si no contara, al menos con un cargador para inicializar el sistema. El fabricante ofrece, pues, un sistema 'MONITOR' mínimo que permite el exámen del sistema, así como la carga y ejecución de ciertos programas sencillos (de menos 1 K).

El Monitor posee los comandos X, I, S, M, D y G. Estos comandos se discuten, brevemente, a continuación.

Comando 'X'. Examina el contenido de(l) (los) registro(s). Por medio de este comando es posible obtener el valor contenido en cualquiera de los registros del CPU (A, B, C, D, E, H, L y banderas; PC, SP).

Comando 'I'. Inserta código en memoria. Por medio de este comando es posible cargar en memoria código de máquina (expresado en hexadecimal).

Comando 'S'. Examina memoria. Por medio de este comando es posible examinar y/o modificar el contenido de alguna(s) localidad(es) de memoria.

Comando 'M'. Transfiere el contenido de una localidad de memoria hasta otra localidad de memoria. Por medio de este comando es posible transferir bloques de memoria a otros lugares de memoria distintos a los originales.

Comando 'D'. Despliega un bloque de memoria en la unidad de entrada/salida. Por medio de este comando es posible 'vaciar' en el dispositivo de consola del sistema un bloque predeterminado de información contenida en memoria.

Comando 'G'. Ejecuta un programa a partir de la localidad designada. Por medio de este comando es posible inicializar el PC (contador de programa) de modo que empiece a correr a partir de una localidad arbitrariamente seleccionada por el usuario. Esto tiene el efecto de permitir la ejecución de un programa a partir de cualquier localidad de memoria preasignada.

Para montar un sistema de cómputo (si bien elemental) no es necesario más que el siguiente conjunto de elementos: 1 sistema de kit como el descrito; un arreglo de fuentes de potencia; una terminal de video o teletipo. El costo mínimo de este arreglo es de aproximadamente unos veinte mil pesos. Es fácil convencerse de que ésta es una de las principales razones por las que este tipo de computadoras personales estén en pleno apogeo en la actualidad.

El tiempo de armado de un SDK-80 es de aproximadamente cinco horas. La experiencia personal del autor indica que las probabilidades de falla son extremadamente reducidas si se observan las precauciones adecuadas de seguridad y limpieza. En cinco sistemas armados, nunca ha habido fallas de funcionamiento. La computadora empieza a responder a la consola inmediatamente.

Aplicaciones. El sistema SKD-80 fue diseñado para ser lo más versátil posible. El autor ha implementado, con pequeñas variaciones en el 'hardware', concentradores, estaciones muestreadoras, sistemas de vigilancia, sistemas de multiproceso, sistemas de recolección de información, sistemas de memoria y DMA (direct memory access). Con excepción hecha del último caso (y tal vez ni en éste) el punto crítico del diseño ha sido la parte 'software'. Esto, por supuesto, no es motivo de sorpresa. La gran flexibilidad de una computadora reside en su parte programática. Esta es, en la opinión del autor, la segunda razón por la cual las micros han proliferado: un sistema único se puede aplicar en una amplia gama de problemas. Ello reduce los tiempos de diseño en una forma dramática, lo cual, a su vez, redunda en costos mucho menores en el diseño en general.

A continuación, se ha incluído una copia de la información pertinente de un sistema SDK-80. Esta información, aparentemente, está dirigida a un sector no-técnico del mercado. Como se observa, el lenguaje es claro y no utiliza tecnicismos complicados.

CHAPTER 3 THEORY OF OPERATION

Now that you have assembled the structure of the SDK-80 it is time to discuss the internal composition of the design. We will do this by presenting the functional organization of the SDK-80 logic and, in the process, bring in the decisions that you, as the user, must make before completing the kit.

Figure 3-1 is a functional block diagram of the SDK-80. It has been purposely drawn as simple as possible in order to give a basis for discussion. You will note that this figure shows only the major signals in the unit. For this reason, some occasional reference to the SDK-80 schematics (Appendix B) will be in order.

The text to follow describes each of the elements in the block diagram.

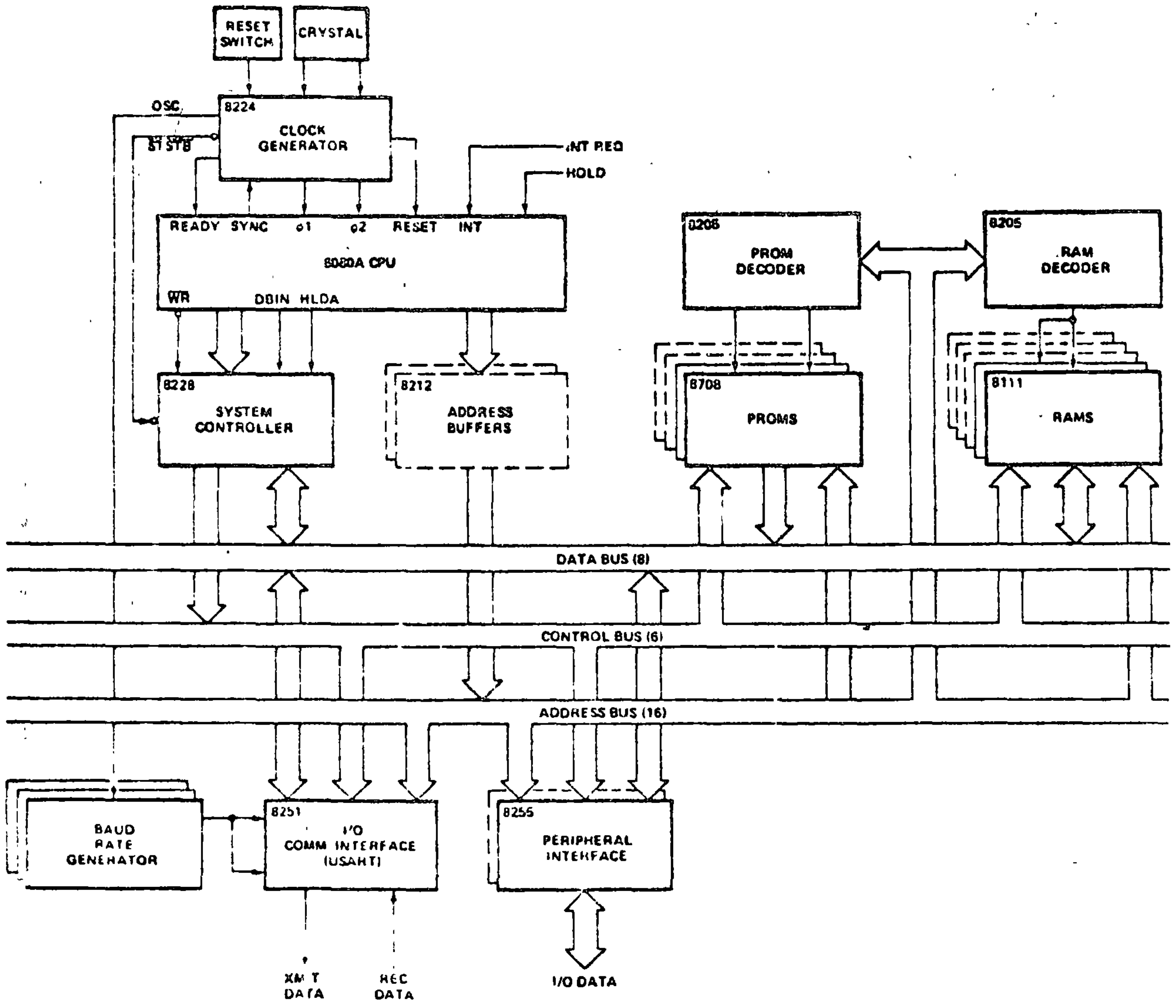


Figure 3-1. SDK-80 Functional Block Diagram.

SYSTEM BUSSES

The SDK-80 logic is built around three system busses: the data bus, the address bus and the control bus. All of the MCS-80 components communicate via these three busses.

The system busses can be selectively enabled/disabled from the user system if the board is jumpered for that capability. Bus enable jumpering is described in the System Bus Enable section of Chapter 4.

Each bus is more fully described in the 8080 Microcomputer Systems User's Manual.

RESET SWITCH

The Reset Switch gives you the capability of forcing a reset to the SDK-80 logic at any time. When the switch is pressed, the Clock Generator will send a RESET signal throughout the system. The Reset Switch should be pressed each time you power-up the system.

CLOCK GENERATOR AND CLOCK CRYSTAL

The 8224 Clock Generator provides the primary timing to the system. It generates the high-level clocks necessary to drive the 8080A CPU, synchronizes the READY signal into the CPU, and transmits the power-up (and Reset Switch) reset signal.

Ø1 and Ø2 Clocks

Ø1 and Ø2 are 2.048 MHz clocks for the CPU. They are derived from OSC using an internal divide-by-nine function.

RESET Signal

RESET is the primary reset signal to the system logic. It is asserted both at power-up and when the Reset Switch is pressed. RESET clears the CPU, disables the RAM Decoder, and resets the USART. RESET is available to the user system at pad V.

READY Signal

READY can provide a synchronized READY to the CPU, derived from an external asynchronous RDYIN signal (pad P).

OSC Signal

OSC provides an 18.432 MHz input to the Baud Rate Generator. This 18.432 MHz rate was chosen for two reasons. First, it permits the 8080A CPU to run at very close to its maximum speed. Second, it is a convenient rate to use in designing a simple, but highly stable, Baud Rate Generator.

$\overline{\text{STSTB}}$ (Status Strobe) Signal

At the beginning of each machine cycle, the CPU issues status information on its data bus. $\overline{\text{STSTB}}$ causes the 8228 System Controller to store this information into its status latch. $\overline{\text{STSTB}}$ is available to the user system as $\overline{\text{STATUS STROBE}}$ at pad J.

8080A CPU

The 8080A CPU is thoroughly described in the Intel 8080 Microcomputer Systems User's Manual and need not be repeated here.

The CPU clocks, Ø1 and Ø2, will be supplied (at 2.048 MHz) by the Clock Generator.

The data bus will interface directly to the System Controller and the address bus will enter the system through the Address Buffers, if applicable.

There are two separate jumper-wire options with the CPU. The first option allows an external HOLD signal to be presented to the CPU via pad R. The second option allows an external READY signal to force a Wait state in the CPU. It should be pointed out, however, that the 8080A and SDK-80 memory chips have been designed to operate without Wait states. The option permits you to force a Wait if desired, though. Both of these jumper options are described in the Hold And Wait Options section of Chapter 4.

SYSTEM CONTROLLER

The 8228 System Controller generates the control bus signals that provide read and write functions for I/O and memory.

They are available to the user system as shown below:

- $\overline{\text{I/O W}}$ is at pad E
- $\overline{\text{I/O R}}$ is at pad L
- $\overline{\text{MEMW}}$ is at pad U
- $\overline{\text{MEMR}}$ is at pad T

The System Controller also buffers the data bus.

Interrupt

A single-level interrupt structure is provided such that whenever pad H ($\overline{\text{INT REQ}}$) is grounded, the System Controller causes a Restart instruction (RST 7) to be inserted into the CPU. This feature provides a single interrupt vector without using additional components, such as an interrupt instruction port.

Multiple level interrupts will require additional chips to be installed into the wire-wrap area.

ADDRESS BUFFERS (OPTIONAL)

The 8212 Address Buffers are not included in the System Design Kit, but must be added if more than a nominal amount of memory (more than 1024 bytes of RAM and more than 4K bytes of PROM) is used. The Address Buffers are tri-state TTL buffers that provide 15mA drive.

The address bus level can be forced to the high-impedance state by inputting a high level on pad S (SYSTEM BUS ENABLE), if the board is jumpered for this capability.

SDK-80 MEMORY

The SDK-80 has two types of memory. Its PROM Memory can accomodate from 1K to 4K bytes, where the lower 1K bytes are dedicated to the system monitor. Its RAM Memory can accomodate from 256 to 1K bytes, in which all but the uppermost 30 bytes (addresses 13E2-13FF) are useable by your system. Figure 3-2 is a map of SDK-80 memory.

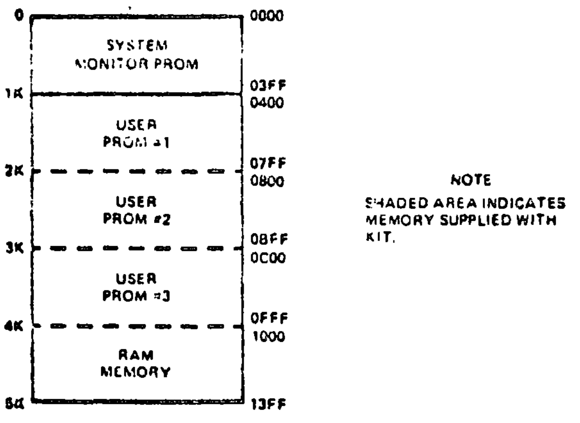


Figure 3-2. SDK-80 Memory Map.

PROM DECODER AND PROM MEMORY

The SDK-80 can accomodate up to four 1024x8 Electrically Programmable Read Only Memory (PROM) chips. Two of these chips are supplied in the System Design Kit.

The 8708 that installs into board location A14 has been pre-programmed with the SDK-80 system monitor.

The 8708 that installs into board location A15 can be used to hold a program that you have developed and checked out in RAM.

Locations A16 and A17 are useable for additional PROMs if required.

The 8205 PROM Decoder selects the PROM chip being addressed. Figure 3-3 shows the PROM address format.

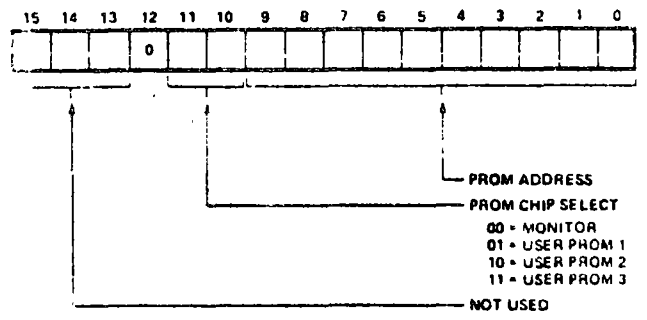


Figure 3-3. PROM Address Format.

RAM DECODER AND RAM MEMORY

In the standard configuration, the SDK-80 can accommodate up to eight 256x4 Static MOS Random Access Memory (RAM) chips. Two of these chips are supplied in the System Design Kit, so users requiring only 256 bytes of memory need not install additional RAM chips.

The 8205 RAM Decoder selects the RAM chip pair being addressed. Figure 3-4 shows the RAM address format.

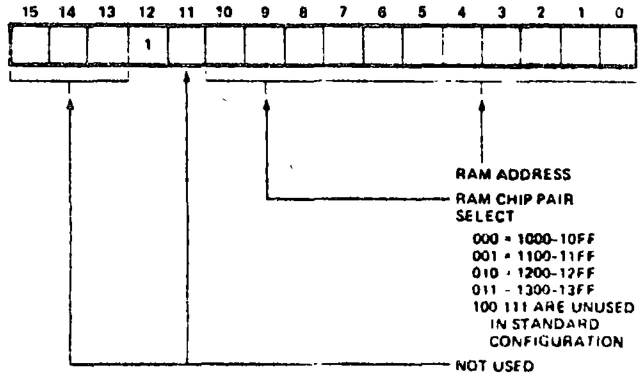


Figure 3-4. RAM Address Format.

RAM access is disabled whenever the RESET signal from the Clock Generator is asserted.

BAUD RATE GENERATOR

The Baud Rate Generator circuit supplies the transmitter and receiver clocks to the I/O Communication Interface. This circuit is made up of three IC chips: one 93S16 and two 74161s.

The Baud Rate Generator takes the 18.432 MHz OSC signal from the Clock Generator and, by internal division, generates a series of signals which represent baud rates between 75 and 4800. The baud rate that will be presented to the I/O Communication Interface is determined by jumper-wiring or a rotary switch. This selection will be discussed in the Baud Rate Selection section of Chapter 4.

I/O COMMUNICATION INTERFACE

The 8251 I/O Communication Interface is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) chip that accommodates any data communications required by the SDK-80 system. The I/O Communication Interface can accept parallel data from the data bus and send it serially to an external device. It can also accept serial data from an external device and put it onto the data bus in parallel form when eight bits have been collected. Figure 3-5 shows the address format for communications.

The baud rate at which the I/O Communication Interface will transmit and receive data is governed by the Baud Rate Generator.

The I/O Communication Interface circuit on the board also includes some jumpers that select the communication input/output level. Any of three levels may be selected.

- RS-232 level, which is typically used for CRT applications
- Current-loop level, for TTY applications
- TTL level.

The input/output level jumpering is discussed in the Communication Level Selection section of Chapter 4.

PERIPHERAL INTERFACES

The 8255 Programmable Peripheral Interfaces provide the user's primary access point to the SDK-80 data bus. One 8255 chip is supplied in the System Design Kit.

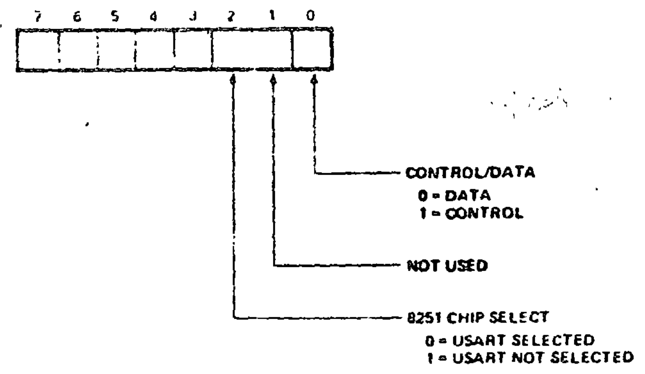


Figure 3-5. I/O Communication Interface Address Format.

Each Peripheral Interface chip provides three 8-bit parallel I/O ports, each of which is independently addressable. Figure 3-6 shows the address format for I/O port selection.

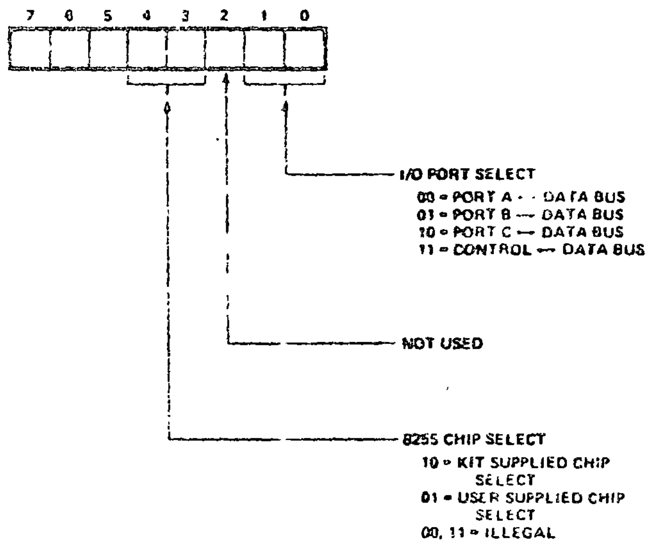


Figure 3-6. Peripheral Interface Address Format.

The output pins of the Peripheral Interfaces are totally uncommitted and may be jumper-wired to best suit your particular application. For example, they might be wired directly to the interface plugs or, alternately, they might be wired to standard TTL buffers in the wire-wrap area before coming back to the plugs. This wiring is further discussed in the Output Wiring section of Chapter 4.

CHAPTER 4
FINAL ASSEMBLY AND CHECKOUT

At this point in the manual you should have completed the preliminary assembly and read the theory of operation. You can now finish the board assembly and begin a checkout sequence.

JUMPER-WIRING THE BOARD

The SDK-80 is designed to be used in virtually any evaluation application and can be jumpered to suit your particular requirements. These questions will help you decide what jumpers are needed:

1. Will you ever want the CPU to enter a Hold or Wait state?
2. Will you ever want to disable the system busses?
3. What type of input device will you use to communicate with the SDK-80 (e.g., CRT, Teletype)?
4. What is its baud rate?
5. Will you be using 8212 Address Buffer chips?
6. What kind of information will be transferred to/from the SDK-80?

If you have a fairly good idea of the answers to all of these questions, you are ready to start jumper-wiring the board. The scrap leads that have been cut from previously-installed resistors are a good source of jumper wire. However, use 22-gauge insulated wire in situations where any jumpers may make contact with each other.

Hold and Wait Options

The SDK-80 is designed to run without Hold or Wait states. However, a jumper-wire option is available to give either capability.

- ☐ To disable the Hold state, wire J5-2 to J5-3.
- ☐ To enable the Hold state, wire J5-1 to J5-2.
- ☐ If READY is to force an 8080 Wait state, wire J5-8 to J5-9. If not, wire J5-8 to J5-7.

System Bus Enable

One jumper is available to make it possible to selectively disable the SDK-80 system bus.

- ☐ If the bus will be selectively disabled, wire J5-5 to J5-6.
- ☐ If the bus should remain enabled at all times, wire J5-4 to J5-5.

Baud Rate Selection

The communications baud rate can be selected in two ways, depending on the application. If only one baud rate will be employed, the rate can be selected by installing a single jumper wire. If two or more baud rates will be employed in the application, however, the Spectrol rotary switch installed in Chapter 1 will be used for this purpose.

- ☐ To select a fixed baud rate, jumper pad 29 to one of the pads 31-37 per Table 4-1.

Table 4-1. Baud Rate Selection Table.

Baud Rate	Wire Pad 29 To
4800	31
2400	32
1200	33
600	34
300	35
150	36
75 or 110	37

- ☐ For 110 baud, the standard Teletype rate, wire pad 4 to pad 5.

Communication Level Selection

Any of three communication levels can be selected: CRT, Teletype, or TTL. All serial data is passed through connector J3.

Table 4-2. Communications Level Jumper Table.

CRT Configuration Jumpers	TTY Configuration Jumpers	TTL Configuration Jumpers
23 to 24	23 to 26	23 to 25
17 to 18	18 to 19	17 to 18
9 to 10	10 to 11	12 to 13
13 to 14	13 to 14	2 to 3
2 to 3	1 to 2	20 to 21
6 to 8	7 to 8	
27 to 28	15 to 16	
21 to 22	21 to 22	

- ☐ Jumper wire pads 1 through 28 per Table 4-2.
- ☐ If your system does not contain a modem, jumper pad A to pad B.

Address Bus Jumpers

If you do not use 8212 Address Buffer chips on your SDK-80, the address bus must be jumpered across locations A11 and A12. In this situation, connect the following jumpers AT BOTH LOCATION A11 AND LOCATION A12. All jumpers should be installed form the circuit side of the board i.e., NOT the silk-screen side.

- ☐ Jumper pad 3 to pad 4.
- ☐ Jumper pad 5 to pad 6.
- ☐ Jumper pad 7 to pad 8.
- ☐ Jumper pad 9 to pad 10.
- ☐ Jumper pad 15 to pad 16.
- ☐ Jumper pad 17 to pad 18.
- ☐ Jumper pad 19 to pad 20.
- ☐ Jumper pad 21 to pad 22.

Output Wiring

Connector J3 is dedicated as a communications interface (see Table 4-3) and is, in fact, the only committed interface in the SDK-80. All other interfacing is at the discretion of the user.

For example, the 8255 Peripheral Interface might be jumpered directly to connector J1 or, alternately, might be jumpered to TTL buffers in the wire-wrap area before being passed to J1. Conversely, you might wish to add a switch array to the 8255 area in order to send data to the CPU.

Your System Design Kit includes male connectors that mate with the female connectors installed at J1 and J3.

A group of control signals are available at the alphabetic-labeled pads in area two of the board. Table 4-4 identifies these pads.

Table 4-3. Pin Assignments for Communications Interface (J3).

J3 Pin	CRT Configuration	TTY Configuration	TTL Configuration
1			
2	CRT REC. DATA		TTL REC. DATA
3	CRT XMIT DATA		TTL XMIT DATA
4			
5	+12 VDC		
6			
7	SIGNAL GND		SIGNAL GND
8	+12 VDC		
9			
10			
11			
12		TTY REC. RETURN	
13		TTY XMIT	
14			
15			
16			
17			
18			
19			
20	+12 VDC		
21			
22			
23			
24		TTY REC.	
25		TTY XMIT RETURN	

Table 4-4. SDK-80 Control Bus Pads.

Pad	Mnemonic	Description
A	$\overline{\text{CTS}}$	Clear To Send
B	$\overline{\text{RTS}}$	Request to Send
C	Ø2 (TTL)	2.048 MHz Clock
D	$\overline{\text{DSR}}$	Data Set Ready
E	$\overline{\text{I/O W}}$	I/O Write
F	$\overline{\text{DTR}}$	Data Terminal Ready
H	$\overline{\text{INT REQ}}$	Interrupt Request
J	$\overline{\text{STATUS STROBE}}$	Status is on Data Bus
K	$\overline{\text{OSC}}$	18.432 MHz Oscillator
L	$\overline{\text{I/O R}}$	I/O Read
M	HLDA	Hold Acknowledge
N	$\overline{\text{INTA}}$	Interrupt Acknowledge
P	READY	Ready
R	HOLD	Hold
S	$\overline{\text{SYSTEM BUS ENABLE}}$	Enables Data Bus and Address Bus
T	$\overline{\text{MEMR}}$	Memory Read
U	$\overline{\text{MEMW}}$	Memory Write
V	RESET	Reset

INSTALLING INTEGRATED CIRCUIT CHIPS

You have now reached the point where you will start installing IC's in the board, but a few words are in order before you begin.

Special Precautions For Handling MOS IC's

The Kit's MOS IC's (8080, 8111, 8251, 8255, and 8708) are particularly susceptible to static electricity. They can be easily damaged if proper care is not taken in handling them. For this reason, the following steps should be adhered to as closely as possible:

1. All equipment (soldering iron, tools, solder, etc.) should be at the same potential as the PW board, the assembler, the work surface and the IC itself along with its container. This can be accomplished by continuous physical contact with the work surface, the components, and everything else involved in the operation.
2. When handling the IC, develop the habit of first touching the conductive container in which it is stored before touching the IC itself.
3. Always touch the SDK-80's PW board before touching the IC to the board. Try to maintain this contact as much as possible while installing the IC.
4. Handle the IC by the edges. Avoid touching the pins as much as possible.
5. In general, never touch anything to the IC that you have not touched first while touching both it and the IC itself.

Aligning the IC Pins

The connector pins of Integrated Circuit chips are very fragile and can be easily pushed out of line. In fact, sometimes IC's will arrive with one or more pins out of line. Trying to install a misaligned IC is a hapless task and, worse, might cause permanent damage to the chip.

Aligning the pins of an IC is an easy job. Simply lay the IC on its side on your work surface, hold the chip by its body and exert enough pressure so that all pins are perpendicular to the body.

Chip Orientation

The IC's must be correctly oriented on the board or they will not operate properly. One end of the chip will carry some sort of identifying mark, typically a notch or a dot or a + sign. The chip must be installed so that this identifier corresponds to the silkscreened "1" on the board.

Installing IC Chips

After orienting the IC, follow these steps to install it in the board:

1. Start the pins on one side of the IC into their respective holes on the silk-screened side of the PW board. DO NOT PUSH THE PINS IN ALL THE WAY. If you have difficulty getting the pins into the holes, use the tip of a small screwdriver to guide them.
2. Start the pins on the other side of the IC into their holes in the same manner. When all of the pins have been started, set the IC in place by gently rocking it back and forth until it rests as close as possible to the board or socket.
3. If the IC is not installed in a socket, turn the board over and solder each pin to the foil pattern on the back side of the board. Be sure to solder each pin and be careful not to leave any solder bridges.

Removing IC Chips

If required, an IC chip can be removed from a socket by gently rocking it back and forth to start its release. When a gap exists between the chip and socket, pry it gently at alternate ends until the pins start to come loose. A popsicle stick or small screwdriver works well here. Then hold the chip by the ends and pull it free. Try to keep the chip fairly parallel to the socket throughout this operation.

CLOCK GENERATOR

Besides the 8080, the most critical chip in the SDK-80 circuit is the 8224 Clock Generator.

☐ Insert the 8224 Clock Chip into the socket at location A8.

POWER, CLOCK AND RESET VERIFICATION

With this single chip installed, we can check the power and clock inputs and the operation of the Reset Switch. The procedure is as follows:

☐ Connect your power supply to terminal lugs E1-E6 on the SDK-80 board.

NOTE

The SDK-80 edge connector is power-compatible with Intel's MDS (Microcomputer Development System). If you have MDS, the SDK-80 can derive its power through installation in the MDS chassis.

- ☐ Turn power on.
- ☐ Using a voltmeter, verify +5 VDC at the pad provided.
- ☐ Verify +12 VDC at the "+12" pad.
- ☐ Verify your supply's negative voltage at the "-10" pad.
- ☐ Verify -5 VDC at the "-5" pad, near location A17.
- ☐ Press the Reset Switch a few times and check for +4 VDC at A8, pin 1(RESET).

NOTE

Develop the habit of pressing the Reset Switch each time you power-up the system.

- ☐ If you have an oscilloscope, verify that A9 pins 10 and 11 each show 2.048 MHz clocks (Ø2 and Ø1, respectively).
- ☐ Using an oscilloscope, verify that A8 pin 12 shows an 18.432 MHz clock (OSC).
- ☐ Turn the power off.

REMAINDER OF SDK-80 CHIPS

After having verified that the SDK-80 logic is correctly receiving power, the system clocks and the RESET signal, you can finish installing the chip complement. Some of the IC's will plug into sockets, others will have to be soldered onto the board.

The procedure is as follows:

- ☐ Solder the 93S16 chip into location A1.
- ☐ Solder a 74161 chip into locations A2 and A5.
- ☐ Solder the 7406 chip into location A6.
- ☐ If applicable, solder 8212 chips into locations A11 and A12.
- ☐ Solder 8205 chips into locations A13 and A18.
- ☐ Solder 8111 chips into locations A25 and A26.
- ☐ Insert the 8228 chip into the socket at location A9.
- ☐ Insert the 8080A chip into the socket at location A10.
- ☐ Insert the 8251 chip into the socket at location A7.
- ☐ Insert the 8255 chip into the socket at location A3.
- ☐ Insert the pre-programmed 8708 PROM chip into the socket at location A14.

Table 4-5. Power Requirements.

Symbol	Voltage	Minimum System	Maximum System	Unit
V_{CC}	+5V ±5%	1.3	2.1	Amps
V_{DD}	+12V ±5%	.35	.45	Amps
V_{BB}	-10V ±5% -12V ±5%	.20	.30	Amps

START-UP PROCEDURE

You have now completed the SDK-80 assembly and are ready to start up the system. The start-up procedure is as follows:

- ☐ Plug your system communication monitor (CRT, Teletype, etc.) into the SDK-80 connector J3.
- ☐ Turn power on at both the SDK-80 power supply and your communication monitor.
- ☐ Press the Reset Switch.

At this point, your monitor should display the following message:

MCS-80 KIT

If it does, congratulations! You are now ready to start using the system.

TROUBLESHOOTING HINTS

If the SDK-80 system does not work properly, turn the power off and investigate these areas:

1. Verify that all resistors have been properly installed and are correctly color-coded. Appendix C summarizes the color codes.
2. Verify that all capacitors have been properly installed and that all electrolytic capacitors are installed with proper polarity.
3. Verify that both diodes (CR1 and CR2) have been installed with proper polarity.
4. Verify that the metal tabs of all three transistors are properly positioned.
5. Verify that all IC's are installed with their "1"-end identifiers correctly oriented.
6. Verify that all jumpers have been properly installed.

APPENDIX A. SDK-80 MONITOR

INTRODUCTION

The SDK-80 Monitor is an Intel® 8080 program provided in a pre-programmed ROM. The Monitor accepts and acts upon user commands to operate the SDK-80. It also provides input and output facilities in the form of I/O drivers for user console devices. The Monitor provides the following facilities:

- Displaying selected areas of memory.
- Initiating execution of user programs.
- Modifying contents of memory and processor registers.
- Inputting hexadecimal file from the console device to memory.

The Monitor communicates with the user through an interactive console device, normally a Teletype or CRT Terminal. The dialogue between the operator and Monitor consists of user-originated commands in the Monitor's command language, and Monitor responses, either in the form of a printed message or an action being performed. After the cold start procedure (described under the heading, "Cold Start Procedures" in Section III), the Monitor begins the dialogue by typing the sign-on message on the console and requesting a command by presenting a prompt character, "." (period).

MONITOR OPERATIONS

The SDK-80 Monitor is a command controlled operations supervisor for the 8080 Microcomputer System Design Kit. Control commands are discussed in Section II, "Command Structure".

I. FUNCTIONAL SPECIFICATION

A. General Characteristics and Scope of Product

The monitor is a program written in Intel® 8080 macro assembly language. The monitor resides in 1K (K = 1024 bytes) of programmed ROM and is located in the address space of the 8080 microcomputer between 0 and 1K. The non-volatile nature of the program's storage media means that the monitor is available for use immediately after power-on or reset.

B. Description of All Major Functions Performed

1. CONSOLE COMMANDS

The monitor communicates with the operator via an interactive console, normally a teletypewriter. The dialogue between the operator and the monitor consists of commands in the monitor's command language and the monitor's responses. After the cold start procedure, the monitor begins the dialogue by typing a sign-on message on the console and then requests a command by presenting a prompt character, ".". Commands are in the form of a single alphabetic character specifying the command, followed by a list of numeric or alphabetic parameters. Numeric parameters are entered as hexadecimal numbers. The monitor recognizes the characters 0 through 9 and A through F as legal hexadecimal digits. The valid range of numbers is from 1 to 4 hex digits. Longer numbers may be entered, but such numbers will be evaluated modulo 2^{16} so that they will fall into the range specified above.

The only command requiring an alphabetic parameter is the "X" command. The nature of such parameters will be discussed in the section explaining the command.

2. USE OF THE MONITOR FOR PROGRAMMING AND CHECKOUT

The monitor allows the user to enter, check out, and execute small demonstration programs. The monitor contains facilities for memory modification, 8080 CPU register display and modification, program loading from the console device, program initiation, and the recognition of an "RST 7" instruction as an unconditional branch to RAM address 13FDH. By inserting RST 7 instructions in a program under test, or by using the hardware generated RST 7 instruction (if available), the user can cause execution o a program to transfer to a dedicated location, for whatever purposes he desires.

When the user wishes to re-enter the

monitor, he should use an RST 1 instruction, either generated by hardware or coded into his program. When entered in this manner, the monitor will automatically save the state of the 8080: specifically, it will save all registers (A, B, C, D, E, H, L), the CPU flags (F), the user's Program Counter (PC), and the user's Stack Pointer (SP). These may be examined with the X command. When the operator enters a G command, these values will be restored.

3. I/O SYSTEM

The I/O system provides two routines, console character in and console character out, which the user may call upon to read and write, respectively, characters from and to the console device.

C. Applicable Standards

Throughout this specification, the numbering convention for bits in a word is that bit 0 is the least significant, or rightmost bit.

The internal code set used by the monitor is 7 bit (no parity) ASCII.

II. INTERFACE SPECIFICATIONS

A. Command Structure

In the following paragraphs the monitor command language is discussed. Each command is described, and examples of its use are included for clarity. Error conditions that may be encountered while operating the monitor are described in Section IV.C.

The monitor requires each command to be terminated by a carriage return. With the exception of the "S" and "X" commands, the command is not acted upon until the carriage return is sensed. Therefore, the user can abort any command, before he enters the carriage return, by typing any illegal character (such as RUBOUT).

Except where indicated otherwise, a single space is synonymous with the comma for use as a delimiter. Consecutive spaces or commas, or a space or comma immediately following the command letter, will be interpreted as a null parameter. Null parameters are illegal in all commands except the "X" command (see below).

Items enclosed in square brackets "[" and "]" are optional. The consequences of including or omitting them are discussed in the text.

1 DISPLAY MEMORY COMMAND, D

D <low address>, <high address>

Selected areas of addressable memory may be accessed and displayed by the D command. The D command produces a formatted listing of the memory area between <low address> and <high address>, inclusive, on the console device. Each line of the listing begins with the address of the first memory location displayed on that line, represented as 4 hexadecimal digits, followed by up to 16 memory locations, each one represented by 2 hexadecimal digits.

The D command may be aborted during execution by typing an Escape (ESC) on the console. The command will be terminated immediately, and a new prompt issued.

Example

```
     D9,2A
0009 00 11 22 33 44 55 66
0010 77 88 99 AA BB CC DD EE FF 10 20 30 40 50 60 70
0020 80 90 A0 B0 C0 D0 E0 F0 01 02 03
```

2. PROGRAM EXECUTE COMMAND, G

G[<entry point>]

Control of the CPU is transferred from the monitor to the user program by means of the program execute command, G. The <entry point> should be an address in RAM which contains an instruction in the user's program. If no entry point is specified, the monitor uses, as an address, the value on top of the stack when the monitor was entered.

Example

```
G1400
```

Control is passed to location 1400H.

3. INSERT INSTRUCTIONS INTO RAM, I

I <address>

Single instructions, or an entire user program, are entered into RAM with the I command. After sensing the carriage return terminating the command line, the monitor waits for the user to enter a string of hexadecimal digits (0 to 9, A to F). Each digit in the string is converted into its binary value, and then loaded into memory, beginning at the starting address specified and continuing into sequential memory locations. Two hexadecimal digits are loaded into each byte of memory.

Separators between digits (spaces, commas, carriage returns) are ignored; illegal characters, however, will terminate the command with an error message (see section IV.C.1). The character ESC or ALTMODE (which is echoed to the console as "$") terminates the digit string. If an odd number of hex digits have been entered, a 0 will be appended to the string.

Example

```
I1410
112233445566778899$
```

This command puts the following pattern into RAM:

```
1410 11 22 33 44 55 66 77 88 99
```

```
I1440
123456789$
```

This command puts the following pattern into RAM:

```
1440 12 34 56 78 90
```

Note that, since an odd number of hexadecimal digits were entered initially, a 0 was appended to the digit string.

4. MOVE MEMORY COMMAND, M

M <low address>, <high address>, <destination>

The M command moves the contents of memory <low address> and <high address>, inclusive, to the area of RAM beginning at <destination>. The contents of the source field remain undisturbed, unless the receiving field overlaps the source field.

The move operation is performed on a byte-by-byte basis, beginning at <low address>. Care should be taken if <destination> is between <low address> and <high address>. For example, if location 1410 contains 1AH, the command

M1410, 141F, 1411

will result in locations 1410 to 1420 containing "1A1A1A..."

The monitor will continue to move data until the source field is exhausted, or until it reaches address OFFFFH. If the monitor reaches address OFFFFH without exhausting the source field, it will move data into this location, then stop.

Example

```
M1410, 150F, 1510
```

256 bytes of memory are moved from 1410-150F to 1510-160F by this command.

5. SUBSTITUTE MEMORY COMMAND, S

S <address>

The S command allows the user to examine and optionally modify memory locations individually. The command functions as follows:

i. Type an S, followed by the hexadecimal address of the first memory location you wish to examine, followed by a space or comma.

ii. The contents of the location is displayed, followed by a dash (-).

iii. To modify the contents of the location displayed, type in the new data, followed by a space, comma, or carriage return. If you do not wish to modify the location, type only the space, comma, or carriage return.

iv. If a space or comma was typed in step (iii), the next memory location will be displayed as in step (ii). If a carriage return was typed, the S command will be terminated.

Example

```
S1450 AA- BB-CC 01-13 23-24
```

Location 1450, which contains AA is unchanged, but location 1451 (which used to contain BB) now contains CC, 1452 (which used to contain 01) now contains 13, and 1453 (which used to contain 23) now contains 24.

6. EXAMINE AND MODIFY CPU REGISTERS COMMAND, X

X [<register identifier>]

Display and modification of the CPU registers is accomplished via the X command. The X command uses <register identifier> to select the particular register to be displayed. A register identifier is a single alphabetic character denoting a register, defined as follows:

A — 8080 CPU register A
B — 8080 CPU register B
C — 8080 CPU register C
D — 8080 CPU register D
E — 8080 CPU register E
F — 8080 CPU flags byte, displayed in the form as it is stored by the "PUSH PSW" (hex code F5) instruction
H — 8080 CPU register H
L — 8080 CPU register L
M — 8080 CPU registers H and L combined
P — 8080 Program Counter
S — 8080 Stack Pointer

The command operates as follows:

i. Type an X, followed by a register identifier or a carriage return.

ii. The contents of the register are displayed (two hexadecimal digits for A, B, C, D, E, F, H, and L, four hexadecimal digits for M and S), followed by a dash (-).

iii. The register may be modified at this time by typing the new value, followed by a space, comma, or carriage return. If no modification is desired, type only the space, comma, or carriage return.

iv. If a space or comma was typed in step (iii), the next register in sequence (alphabetical order) will be displayed as in step ii (unless S was just displayed in which case the command is terminated). If a carriage return was entered in step iii, the X command is terminated.

v. If a carriage return was typed in step (i) above, an annotated list of all registers and their contents are displayed.

Example

```
XA AA- BB- CC- DD- EE- FF- 12- 34- 1234- 0000
XA AA- 23- CC- 01- EE- FF- 12- 34- 1234- 1010
X
A-AA B-23 C-CC D-01 E-EE F-FF H-12 L-34 M-1234 P-01CF S-03C0
```

B. Console Device Drivers

The monitor interfaces to the console device via a universal synchronous/asynchronous receiver/transmitter (USART). The monitor drivers interface with the USART according to the USART specifications. At the time of the assembly of the kit, the USART may be configured for a particular type of console interface. The actual console device must conform to this interface.

C. Using the I/O System

The user may access the two monitor I/O system routines from his program by calling the routine desired. The following paragraphs describe the routines available and their respective functions.

CI — Console Input

This routine returns a character received from the console device to the caller in the A-register. The A register and the CPU condition codes are affected by this operation. The entry point of this routine is 3FDH.

Example

```
CI      EQU  3FDH
        . . .
        CALL CI
        STA DATA
        . . .
```

CO — Console Output

This routine transmits a character, passed from the caller in the C-register, to the console device. The A and C registers, and the CPU condition codes, are affected by this operation. The entry point of this routine is 3FAH.

Example

```
CO      EQU  3FAH
        . . .
        MVI  C, "."
        CALL CO
```

III. OPERATING SPECIFICATIONS

A. Product Activation Instructions

1. COLD START PROCEDURE

   After a power-on or reset, the monitor will begin execution at location 0 in ROM. The monitor will perform an initialization sequence, and then display a sign-on message on the console. When the monitor is ready for a command, it will prompt with a period, ".".

2. USE OF RAM STORAGE IN THE MONITOR

   The monitor dynamically assigns its RAM stack near the top of the first 1K bytes of RAM (address space from 4K to 5K). The top 3 bytes in this block of RAM are reserved for a transfer address, supplied by the user, which is used as a destination location for RST 7 instructions (or the optional hardwired instruction). Several additional bytes are used, below the stack, for temporary storage. Except for RAM addresses 5K-1 to 5K-256, all other RAM is available for the user.

B. Summary of Normal Use Methodology

(This section, which normally consists of a detailed example of the use of the monitor, will be omitted. Examples of all commands may be found in the sections explaining the monitor commands.)

C. Error Conditions

1. INVALID CHARACTERS

The monitor checks the validity of each character as it is entered from the console. As soon as the monitor determines that the last character entered is illegal in its context, the monitor aborts the command and issues an "*" to indicate the error.

Example

```
D1400, 145G*
```

The character G was encountered in a parameter list where only hexadecimal digits and delimiters are valid.

```
Y*
```

Y is not a valid command.

2. ADDRESS VALUE ERRORS

Some commands require an address pair of the form <low address>, <high address>. If, on these commands, the value of <low address> is greater than or equal to the value of <high address>, the action indicated by the command will be performed on the data at <low address> only.

Addresses are evaluated modulo 2^{16}. Thus, if a hexadecimal address greater than FFFF is entered, only the last 4 hex digits will be used.

Another type of address error may occur when the operator specifies a part of memory in a command which does not exist in his particular configuration. In general, if a nonexistent portion of memory is specified as the source field for an instruction, the data fetched will be unpredictable. If a nonexistent portion of memory is given as the destination field in a command, the command has no effect.

APPENDIX B. MONITOR LISTING

```
8080 MACRO ASSEMBLER, VER 2.3  ERRORS = 0 PAGE 1

                  ;************************************************************************
                  ;
                  ;                 PROGRAM: 8080A BOARD MONITOR
                  ;
                  ;                 COPYRIGHT (C) 1975
                  ;                 INTEL CORPORATION
                  ;                 3065 BOWERS AVENUE
                  ;                 SANTA CLARA, CALIFORNIA  95051
                  ;
                  ;************************************************************************
                  ;
                  ;
                  ; ABSTRACT
                  ; ========
                  ;
                  ; THIS PROGRAM RUNS ON THE 8080A BOARD AND IS DESIGNED TO PROVIDE
                  ; THE USER WITH A MINIMAL MONITOR.  BY USING THIS PROGRAM,
                  ; THE USER CAN EXAMINE AND CHANGE MEMORY OR CPU REGISTERS, LOAD
                  ; A PROGRAM (IN ABSOLUTE HEX) INTO RAM, AND EXECUTE INSTRUCTIONS
                  ; ALREADY IN MEMORY.  THE MONITOR ALSO PROVIDES THE USER WITH
                  ; ROUTINES FOR PERFORMING CONSOLE I/O.
                  ;
                  ;
                  ; PROGRAM ORGANIZATION
                  ; ======= ============
                  ;
                  ; THE LISTING IS ORGANIZED IN THE FOLLOWING WAY.  FIRST THE COMMAND
                  ; RECOGNIZER, WHICH IS THE HIGHEST LEVEL ROUTINE IN THE PROGRAM.
                  ; NEXT THE ROUTINES TO IMPLEMENT THE VARIOUS COMMANDS.  FINALLY,
                  ; THE UTILITY ROUTINES WHICH ACTUALLY DO THE DIRTY WORK.  WITHIN
                  ; EACH SECTION, THE ROUTINES ARE ORGANIZED IN ALPHABETICAL
                  ; ORDER, BY ENTRY POINT OF THE ROUTINE.
                  ;
                  ; THIS PROGRAM EXPECTS TO RUN IN THE FIRST 1K OF ADDRESS SPACE.
                  ; IF, FOR SOME REASON, THE PROGRAM IS RE-ORG'ED, CARE SHOULD
                  ; BE TAKEN TO MAKE SURE THAT THE TRANSFER INSTRUCTIONS FOR RST 1
                  ; AND RST 7 ARE ADJUSTED APPROPRIATELY.
                  ;
                  ; THE PROGRAM ALSO EXPECTS THAT RAM LOCATIONS 5K-1 TO 5K-256,
                  ; INCLUSIVE, ARE RESERVED FOR THE PROGRAM'S OWN USE.  THESE
                  ; LOCATIONS MAY BE ALTERED, HOWEVER, BY CHANGING THE EQU'ED
                  ; SYMBOL "DATA" AS DESIRED.
                  ;
                  ; LIST OF FUNCTIONS
                  ; ==== == =========
                  ;
                  ;     GETCM
                  ;     -----
                  ;
                  ;     DCMD
                  ;     GCMD
```

```
                        ;       ICMD
                        ;       MCMD
                        ;       SCMD
                        ;       XCMD
                        ;       -----
                        ;
                        ;       BREAK
                        ;       CI
                        ;       CNVBN
                        ;       CO
                        ;       CROUT
                        ;       ECHO
                        ;       ERROR
                        ;       FRET
                        ;       GETCH
                        ;       GETHX
                        ;       GETNM
                        ;       HILO
                        ;       NMOUT
                        ;       PRVAL
                        ;       REGDS
                        ;       RGADR
                        ;       RSTTF
                        ;       SRET
                        ;       STHF0
                        ;       STHLF
                        ;       VALDG
                        ;       VALDL
                        ;       -----
                        ;
 0000                           ORG     0H
                        ;
                        ;*******************************************************************
                        ;
                        ;
                        ;                               MONITOR EQUATES
                        ;
                        ;
                        ;*******************************************************************
                        ;
                        ;
 001B                   BRCHR  EQU      1BH      ; CODE FOR BREAK CHARACTER (ESCAPE)
 13FD                   BRLOC  EQU      13FDH    ; LOCATION OF USER BRANCH INSTRUCTION IN RAM
 13FA                   BRTAB  EQU      3FAH     ; LOCATION OF START OF BRANCH TABLE IN ROM
 0027                   CMD    EQU      027H     ; COMMAND INSTRUCTION FOR USART INITIALIZATION
 00FB                   CNCTL  EQU      0FBH     ; CONSOLE (USART) CONTROL PORT
 00FA                   CNIN   EQU      0FAH     ; CONSOLE INPUT PORT
 00FA                   CNOUT  EQU      0FAH     ; CONSOLE OUTPUT PORT
 00FB                   CONST  EQU      0FBH     ; CONSOLE STATUS INPUT PORT
 000D                   CR     EQU      0DH      ; CODE FOR CARRIAGE RETURN
 1300                   DATA   EQU      5*1024-256       ; START OF MONITOR RAM USAGE
 001B               ESC   EQU     1BH      ; CODE FOR ESCAPE CHARACTER
 000F               HCHAR EQU     0FH      ; MASK TO SELECT LOWER HEX CHAR FROM BYTE
 00FF               INVRT EQU     0FFH     ; MASK TO INVERT HALF BYTE FLAG
 000A               LF    EQU     0AH      ; CODE FOR LINE FEED
 0000               LOWER EQU     0        ; DENOTES LOWER HALF OF BYTE IN ICMD
                    ;LSGNON        EQU      ---      ; LENGTH OF SIGNON MESSAGE - DEFINED LATER
 00CF               MODE  EQU     0CFH     ; MODE SET FOR USART INITIALIZATION
                    ;MSTAK         EQU      ---      ; START OF MONITOR STACK - DEFINED LATER
                    ;NCMDS         EQU      ---      ; NUMBER OF VALID COMMANDS
 000F               NEWLN EQU     0FH      ; MASK FOR CHECKING MEMORY ADDR DISPLAY
 007F               PRTY0 EQU     07FH     ; MASK TO CLEAR PARITY BIT FROM CONSOLE CHAR
 13ED               REGS  EQU     DATA+255-18      ; START OF REGISTER SAVE AREA
 0002               RBR   EQU     2        ; MASK TO TEST RECEIVER STATUS
 0038               RSTU  EQU     38H      ; TRANSFER LOCATION FOR RST 7 INSTRUCTION
                    ;RTABS         EQU      ---      ; SIZE OF ENTRY IN RTAB TABLE
 001B               TERM  EQU     1BH      ; CODE FOR ICMD TERMINATING CHARACTER (ESCAPE)
 0001               TRDY  EQU     1        ; MASK TO TEST TRANSMITTER STATUS
 00FF               UPPER EQU     0FFH     ; DENOTES UPPER HALF OF BYTE IN ICMD
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ;                        MONITOR MACROS
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
          1         TRUE  MACRO   WHERE    ; BRANCH IF FUNCTION RETURNS TRUE (SUCCESS)
          1               JC      WHERE
                          ENDM
                    ;
          1         FALSE MACRO   WHERE    ; BRANCH IF FUNCTION RETURNS FALSE (FAILURE)
          1               JNC     WHERE
                          ENDM
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ;                        USART INITIALIZATION CODE
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ;   THE USART IS ASSUMED TO COME UP IN THE RESET POSITION (THIS
                    ;   FUNCTION IS TAKEN CARE OF BY THE HARDWARE).  THE USART WILL
                    ;   BE INITIALIZED IN THE SAME WAY FOR EITHER A TTY OR CRT
                    ;   INTERFACE.  THE FOLLOWING PARAMETERS ARE USED:
```

```
                   ;
                   ;       MODE INSTRUCTION
                   ;       ==== ===========
                   ;
                   ;       2 STOP BITS
                   ;       PARITY DISABLED
                   ;       8 BIT CHARACTERS
                   ;       BAUD RATE FACTOR OF 64
                   ;
                   ;       COMMAND INSTRUCTION
                   ;       ======= ===========
                   ;
                   ;       NO HUNT MODE
                   ;       NOT(RTS) FORCED TO 0
                   ;       RECEIVE ENABLED
                   ;       DATA TERMINAL READY
                   ;       TRANSMIT ENABLED
                   ;
0000    3ECF              MVI     A,MODE
0002    D3FB              OUT     CNCTL     ; OUTPUT MODE SET TO USART
0004    3E27              MVI     A,CMD
0006    D3FB              OUT     CNCTL     ; OUTPUT COMMAND WORD TO USART
                   ;
                   ;**************************************************************
                   ;
                   ;
                   ;                       RESTART ENTRY POINT
                   ;
                   ;
                   ;**************************************************************
                   ;
                   ;
0008               GO:
0008    22F313            SHLD    LSAVE     ; SAVE HL REGISTERSS
000B    E1                POP     H         ; GET TOP OF STACK ENTRY
000C    22F513            SHLD    PSAVE     ; ASSUME THIS IS LAST P COUNTER
000F    210000            LXI     H,0       ; CLEAR HL
0012    39                DAD     SP        ; GET STACK POINTER VALUE
0013    22F713            SHLD    SSAVE     ; SAVE USER'S STACK POINTER
0016    21F313            LXI     H,ASAVE+1         ; NEW VALUE FOR STACK POINTER
0019    F9                SPHL              ; SET MONITOR STACK POINTER FOR REG SAVE
001A    F5                PUSH    PSW       ; SAVE A AND FLAGS
001B    C5                PUSH    B         ; SAVE B AND C
001C    D5                PUSH    D         ; SAVE D AND E
                   ;
                   ;**********************************************************
                   ;
                   ;
                   ;                       PRINT SIGNON MESSAGE
                   ;
                   ;
```

```
                      ;***************************************************************
                      ;
                      ;
001D    219D03              LXI     H,SGNON ; GET ADDRESS OF SIGNON MESSAGE
0020    060E                MVI     B,LSGNON        ; COUNTER FOR CHARACTERS IN MESSAGE
0022                  MSGL:
0022    4E                  MOV     C,M     ; FETCH NEXT CHAR TO C REG
0023    CDE301              CALL    CO      ; SEND IT TO THE CONSOLE
0026    23                  INX     H       ; POINT TO NEXT CHARACTER
0027    05                  DCR     B       ; DECREMENT BYTE COUNTER
0028    C22200              JNZ     MSGL    ; RETURN FOR NEXT CHARACTER
                      ;
                      ;
                      ;*****************************************************************
                      ;
                      ;
                      ;                    COMMAND RECOGNIZING ROUTINE
                      ;
                      ;
                      ;*****************************************************************
                      ;
                      ; FUNCTION: GETCM
                      ; INPUTS: NONE
                      ; OUTPUTS: NONE
                      ; CALLS: GETCH,ECHO,ERROR
                      ; DESTROYS: A,B,C,H,L,F/F'S
                      ; DESCRIPTION: GETCM RECEIVES AN INPUT CHARACTER FROM THE USER
                      ;              AND ATTEMPTS TO LOCATE THIS CHARACTER IN ITS COMMAND
                      ;              CHARACTER TABLE.  IF SUCCESSFUL, THE ROUTINE
                      ;              CORRESPONDING TO THIS CHARACTER IS SELECTED FROM
                      ;              A TABLE OF COMMAND ROUTINE ADDRESSES, AND CONTROL
                      ;              IS TRANSFERRED TO THIS ROUTINE.  IF THE CHARACTER
                      ;              DOES NOT MATCH ANY ENTRIES, CONTROL IS PASSED TO
                      ;              THE ERROR HANDLER.
                      ;
002B                  GETCM:
002B    21ED13              LXI     H,MSTAK ; ALWAYS WANT TO RESET STACK PTR TO MONITOR
002E    F9                  SPHL            ; /STARTING VALUE SO ROUTINES NEEDN'T CLEAN UP
002F    0E2E                MVI     C,'.'   ; PROMPT CHARACTER TO C
0031    CDF401              CALL    ECHO    ; SEND PROMPT CHARACTER TO USER TERMINAL
0034    C33B00              JMP     GTC03   ; WANT TO LEAVE ROOM FOR RST BRANCH
                      ;
0038                        ORG     RSTU    ; ORG TO RST TRANSFER LOCATION
0038    C3FD13              JMP     USRBR   ; JUMP TO USER BRANCH LOCATION
                      ;
003B                  GTC03:
003B    CD1B02              CALL    GETCH   ; GET COMMAND CHARACTER TO A
003E    CDF401              CALL    ECHO    ; ECHO CHARACTER TO USER
0041    79                  MOV     A,C     ; PUT COMMAND CHARACTER INTO ACCUMULATOR
0042    010600              LXI     B,NCMDS ; C CONTAINS LOOP AND INDEX COUNT
0045    21B903              LXI     H,CTAB  ; HL POINTS INTO COMMAND TABLE
```

```
0048                GTC05:
0048   BE                 CMP     M         ; COMPARE TABLE ENTRY AND CHARACTER
0049   CA5400             JZ      GTC10     ; BRANCH IF EQUAL - COMMAND RECOGNIZED
004C   23                 INX     H         ; ELSE, INCREMENT TABLE POINTER
004D   0D                 DCR     C         ; DECREMENT LOOP COUNT
004E   C24800             JNZ     GTC05     ; BRANCH IF NOT AT TABLE END
0051   C30D02             JMP     ERROR     ; ELSE, COMMAND CHARACTER IS ILLEGAL
0054                GTC10:
0054   21AB03             LXI     H,CADR    ; IF GOOD COMMAND, LOAD ADDRESS OF TABLE
                                            ; /OF COMMAND ROUTINE ADDRESSES
0057   09                 DAD     B         ; ADD WHAT IS LEFT OF LOOP COUNT
0058   09                 DAD     B         ; ADD AGAIN - EACH ENTRY IN CADR IS 2 BYTES LONG
0059   7E                 MOV     A,M       ; GET LSP OF ADDRESS OF TABLE ENTRY TO A
005A   23                 INX     H         ; POINT TO NEXT BYTE IN TABLE
005B   66                 MOV     H,M       ; GET MSP OF ADDRESS OF TABLE ENTRY TO H
005C   6F                 MOV     L,A       ; PUT LSP OF ADDRESS OF TABLE ENTRY INTO L
005D   E9                 PCHL              ; NEXT INSTRUCTION COMES FROM COMMAND ROUTINE
                    ;
                    ;
                    ;**************************************************************************
                    ;
                    ;
                    ;             COMMAND IMPLEMENTING ROUTINES
                    ;
                    ;
                    ;**************************************************************************
                    ;
                    ;
                    ; FUNCTION: DCMD
                    ; INPUTS: NONE
                    ; OUTPUTS: NONE
                    ; CALLS: ECHO,NMOUT,HILO,GETCM,CROUT,GETNM
                    ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                    ; DESCRIPTION: DCMD IMPLEMENTS THE DISPLAY MEMORY (D) COMMAND
                    ;
005E                DCMD:
005E   0E02               MVI     C,2       ; GET 2 NUMBERS FROM INPUT STREAM
0060   CD5702             CALL    GETNM
0063   D1                 POP     D         ; ENDING ADDRESS TO DE
0064   E1                 POP     H         ; STARTING ADDRESS TO HL
0065                DCM05:
0065   CDEE01             CALL    CROUT     ; ECHO CARRIAGE RETURN/LINE FEED
0068   7C                 MOV     A,H       ; DISPLAY ADDRESS OF FIRST LOCATION IN LINE
0069   CDC302             CALL    NMOUT
006C   7D                 MOV     A,L       ; ADDRESS IS 2 BYTES LONG
006D   CDC302             CALL    NMOUT
0070                DCM10:
0070   0E20               MVI     C,' '
0072   CDF401             CALL    ECHO      ; USE BLANK AS SEPARATOR
0075   7E                 MOV     A,M       ; GET CONTENTS OF NEXT MEMORY LOCATION
0076   CDC302             CALL    NMOUT     ; DISPLAY CONTENTS
```

```
0079    CDBD01          CALL    BREAK   ; SEE IF USER WANTS OUT
     1          +       TRUE    DCM12   ; IF SO, BRANCH
007C 1  DA8500  +       JC      DCM12
007F    CD9C02          CALL    HILO    ; SEE IF ADDRESS OF DISPLAYED LOCATION IS
                                        ; /GREATER THAN OR EQUAL TO ENDING ADDRESS
     1          +       FALSE   DCM15   ; IF NOT, MORE TO DISPLAY
0082 1  D28B00  +       JNC     DCM15
0085                DCM12:
0085    CDEE01          CALL    CROUT   ; CARRIAGE RETURN/LINE FEED TO END LINE
0088    C32B00          JMP     GETCM   ; ALL DONE
008B                DCM15:
008B    23              INX     H       ; IF MORE TO GO, POINT TO NEXT LOC TO DISPLAY
008C    7D              MOV     A,L     ; GET LOW ORDER BITS OF NEW ADDRESS
008D    E60F            ANI     NEWLN   ; SEE IF LAST HEX DIGIT OF ADDRESS DENOTES
                                        ; /START OF NEW LINE
008F    C27000          JNZ     DCM10   ; NO - NOT AT END OF LINE
0092    C36500          JMP     DCM05   ; YES - START NEW LINE WITH ADDRESS
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ; FUNCTION: GCMD
                    ; INPUTS: NONE
                    ; OUTPUTS: NONE
                    ; CALLS: ERROR,GETHX,RSTTF
                    ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                    ; DESCRIPTION: GCMD IMPLEMENTS THE BEGIN EXECUTION (G) COMMAND.
                    ;
0095                GCMD:
0095    CD2202          CALL    GETHX   ; GET ADDRESS (IF PRESENT) FROM INPUT STREAM
     1          +       FALSE   GCM05   ; BRANCH IF NO NUMBER PRESENT
0098 1  D2AA00  +       JNC     GCM05
009B    7A              MOV     A,D     ; ELSE, GET TERMINATOR
009C    FE0D            CPI     CR      ; SEE IF CARRIAGE RETURN
009E    C20D02          JNZ     ERROR   ; ERROR IF NOT PROPERLY TERMINATED
00A1    21F513          LXI     H,PSAVE ; WANT NUMBER TO REPLACE SAVE PGM COUNTER
00A4    71              MOV     M,C
00A5    23              INX     H
00A6    70              MOV     M,B
00A7    C3B000          JMP     GCM10
00AA                GCM05:
00AA    7A              MOV     A,D     ; IF NO STARTING ADDRESS, MAKE SURE THAT
00AB    FE0D            CPI     CR      ; /CARRIAGE RETURN TERMINATED COMMAND
00AD    C20D02          JNZ     ERROR   ; ERROR IF NOT
00B0                GCM10:
00B0    C32E03          JMP     RSTTF   ; RESTORE REGISTERS AND BEGIN EXECUTION

                    ;
                    ;
                    ;*****************************************************************
```

```
                      ;
                      ;
                      ; FUNCTION: ICMD
                      ; INPUTS: NONE
                      ; OUTPUTS: NONE
                      ; CALLS: ERROR,ECHO,GETCH,VALDL,VALDG,CNVBN,STHLF,GETNM,CROUT
                      ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                      ; DESCRIPTION: ICMD IMPLEMENTS THE INSERT CODE INTO MEMORY (I) COMMAND.
                      ;
00B3                  ICMD:
00B3   0E01                MVI     C,1
00B5   CD5702              CALL    GETNM    ; GET SINGLE NUMBER FROM INPUT STREAM
00B8   3EFF                MVI     A,UPPER
00BA   32F913              STA     TEMP     ; TEMP WILL HOLD THE UPPER/LOWER HALF BYTE FLAG
00BD   D1                  POP     D        ; ADDRESS OF START TO DE
00BE                  ICM05:
00BE   CD1B02              CALL    GETCH    ; GET A CHARACTER FROM INPUT STREAM
00C1   4F                  MOV     C,A
00C2   CDF401              CALL    ECHO     ; ECHO IT
00C5   79                  MOV     A,C      ; PUT CHARACTER BACK INTO A
00C6   FE1B                CPI     TERM     ; SEE IF CHARACTER IS A TERMINATING CHARACTER
00C8   CAF400              JZ      ICM25    ; IF SO, ALL DONE ENTERING CHARACTERS
00CB   CD8A03              CALL    VALDL    ; ELSE, SEE IF VALID DELIMITER
     1              +      TRUE    ICM05    ; IF SO SIMPLY IGNORE THIS CHARACTER
00CE 1 DABE00       +      JC      ICM05
00D1   CD6F03              CALL    VALDG    ; ELSE, CHECK TO SEE IF VALID HEX DIGIT
     1              +      FALSE   ICM20    ; IF NOT, BRANCH TO HANDLE ERROR CONDITION
00D4 1 D2EE00       +      JNC     ICM20
00D7   CDDA01              CALL    CNVBN    ; CONVERT DIGIT TO BINARY
00DA   4F                  MOV     C,A      ; MOVE RESULT TO C
00DB   CD5003              CALL    STHLF    ; STORE IN APPROPRIATE HALF WORD
00DE   3AF913              LDA     TEMP     ; GET HALF BYTE FLAG
00E1   B7                  ORA     A        ; SET F/F'S
00E2   C2E600              JNZ     ICM10    ; BRANCH IF FLAG SET FOR UPPER
00E5   13                  INX     D        ; IF LOWER, INC ADDRESS OF BYTE TO STORE IN
00E6                  ICM10:
00E6   EEFF                XRI     INVRT    ; TOGGLE STATE OF FLAG
00E8   32F913              STA     TEMP     ; PUT NEW VALUE OF FLAG BACK
00EB   C3BE00              JMP     ICM05    ; PROCESS NEXT DIGIT
00EE                  ICM20:
00EE   CD4503              CALL    STHF0    ; ILLEGAL CHARACTER
00F1   C30D02              JMP     ERROR    ; MAKE SURE ENTIRE BYTE FILLED THEN ERROR
00F4                  ICM25:
00F4   CD4503              CALL    STHF0    ; HERE FOR ESCAPE CHARACTER - INPUT IS DONE
00F7   CDEE01              CALL    CROUT    ; ADD CARRIAGE RETURN
00FA   C32B00              JMP     GETCH
                      ;
                      ;
                      ;*****************************************************************
                      ;
                      ;
```

```
 02AA   E1               POP     H         ; IF NOT, RESTORE ORIGINAL HL
 02AB   D5               PUSH    D         ; SAVE DE
 02AC   3EFF             MVI     A,0FFH    ; WANT TO TAKE 2'S COMPLEMENT OF DE CONTENTS
 02AE   AA               XRA     D
 02AF   57               MOV     D,A
 02B0   3EFF             MVI     A,0FFH
 02B2   AB               XRA     E
 02B3   5F               MOV     E,A
 02B4   13               INX     D         ; 2'S COMPLEMENT OF DE TO DE
 02B5   7D               MOV     A,L
 02B6   83               ADD     E         ; ADD HL AND DE
 02B7   7C               MOV     A,H
 02B8   8A               ADC     D         ; THIS OPERATION SETS CARRY PROPERLY
 02B9   D1               POP     D         ; RESTORE ORIGINAL DE CONTENTS
 02BA   78               MOV     A,B       ; RESTORE ORIGINAL CONTENTS OF A
 02BB   C1               POP     B         ; RESTORE ORIGINAL CONTENTS OF BC
 02BC   C9               RET               ; RETURN WITH CARRY SET AS REQUIRED
 02BD               HIL05:
 02BD   E1               POP     H         ; IF HL CONTAINS 0FFFFH, THEN CARRY CAN
 02BE   78               MOV     A,B       ;  /ONLY BE SET TO 1
 02BF   C1               POP     B         ; RESTORE ORIGINAL CONTENTS OF REGISTERS
 02C0   C34303           JMP     SRET      ; SET CARRY AND RETURN
                    ;
                    ;
                    ;*********************************************************************
                    ;
                    ;
                    ; FUNCTION: NMOUT
                    ; INPUTS: A - 8 BIT INTEGER
                    ; OUTPUTS: NONE
                    ; CALLS: ECHO,PRVAL
                    ; DESTROYS: A,B,C,F/F'S
                    ; DESCRIPTION: NNMOUT CONVERTS THE 8 BIT, UNSIGNED INTEGER IN THE
                    ;              A REGISTER INTO 2 ASCII CHARACTERS.  THE ASCII CHARACTERS
                    ;              ARE THE ONES REPRESENTING THE 8 BITS.  THESE TWO
                    ;              CHARACTERS ARE SENT TO THE CONSOLE AT THE CURRENT PRINT
                    ;              POSITION OF THE CONSOLE.
                    ;
 02C3               NMOUT:
 02C3   E5               PUSH    H         ; SAVE HL - DESTROYED BY PRVAL
 02C4   F5               PUSH    PSW       ; SAVE ARGUMENT
 02C5   0F               RRC
 02C6   0F               RRC
 02C7   0F               RRC
 02C8   0F               RRC               ; GET UPPER 4 BITS TO LOW 4 BIT POSITIONS
 02C9   E60F             ANI     HCHAR     ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
 02CB   4F               MOV     C,A
 02CC   CDDE02           CALL    PRVAL     ; CONVERT LOWER 4 BITS TO ASCII
 02CF   CDF401           CALL    ECHO      ; SEND TO TERMINAL
 02D2   F1               POP     PSW       ; GET BACK ARGUMENT
 02D3   E60F             ANI     HCHAR     ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
```

```
Ø2AA   E1              POP     H          ; IF NOT, RESTORE ORIGINAL HL
Ø2AB   D5              PUSH    D          ; SAVE DE
Ø2AC   3EFF            MVI     A,ØFFH     ; WANT TO TAKE 2'S COMPLEMENT OF DE CONTENTS
Ø2AE   AA              XRA     D
Ø2AF   57              MOV     D,A
Ø2BØ   3EFF            MVI     A,ØFFH
Ø2B2   AB              XRA     E
Ø2B3   5F              MOV     E,A
Ø2B4   13              INX     D          ; 2'S COMPLEMENT OF DE TO DE
Ø2B5   7D              MOV     A,L
Ø2B6   83              ADD     E          ; ADD HL AND DE
Ø2B7   7C              MOV     A,H
Ø2B8   8A              ADC     D          ; THIS OPERATION SETS CARRY PROPERLY
Ø2B9   D1              POP     D          ; RESTORE ORIGINAL DE CONTENTS
Ø2BA   78              MOV     A,B        ; RESTORE ORIGINAL CONTENTS OF A
Ø2BB   C1              POP     B          ; RESTORE ORIGINAL CONTENTS OF BC
Ø2BC   C9              RET                ; RETURN WITH CARRY SET AS REQUIRED
Ø2BD            HILØ5:
Ø2BD   E1              POP     H          ; IF HL CONTAINS ØFFFFH, THEN CARRY CAN
Ø2BE   78              MOV     A,B        ;  /ONLY BE SET TO 1
Ø2BF   C1              POP     B          ; RESTORE ORIGINAL CONTENTS OF REGISTERS
Ø2CØ   C343Ø3          JMP     SRET       ; SET CARRY AND RETURN
                ;
                ;
                ;*****************************************************************
                ;
                ;
                ; FUNCTION: NMOUT
                ; INPUTS: A - 8 BIT INTEGER
                ; OUTPUTS: NONE
                ; CALLS: ECHO,PRVAL
                ; DESTROYS: A,B,C,F/F'S
                ; DESCRIPTION: NNMOUT CONVERTS THE 8 BIT, UNSIGNED INTEGER IN THE
                ;              A REGISTER INTO 2 ASCII CHARACTERS.  THE ASCII CHARACTERS
                ;              ARE THE ONES REPRESENTING THE 8 BITS.  THESE TWO
                ;              CHARACTERS ARE SENT TO THE CONSOLE AT THE CURRENT PRINT
                ;              POSITION OF THE CONSOLE.
                ;
Ø2C3            NMOUT:
Ø2C3   E5              PUSH    H          ; SAVE HL - DESTROYED BY PRVAL
Ø2C4   F5              PUSH    PSW        ; SAVE ARGUMENT
Ø2C5   ØF              RRC
Ø2C6   ØF              RRC
Ø2C7   ØF              RRC
Ø2C8   ØF              RRC                ; GET UPPER 4 BITS TO LOW 4 BIT POSITIONS
Ø2C9   E6ØF            ANI     HCHAR      ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
Ø2CB   4F              MOV     C,A
Ø2CC   CDDEØ2          CALL    PRVAL      ; CONVERT LOWER 4 BITS TO ASCII
Ø2CF   CDF4Ø1          CALL    ECHO       ; SEND TO TERMINAL
Ø2D2   F1              POP     PSW        ; GET BACK ARGUMENT
Ø2D3   E6ØF            ANI     HCHAR      ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
```

```
 02D5   4F              MOV     C,A
 02D6   CDDE02          CALL    PRVAL
 02D9   CDF401          CALL    ECHO
 02DC   E1              POP     H        ; RESTORE SAVED VALUE OF HL
 02DD   C9              RET
                ;
                ;
                ;*************************************************************************
                ;
                ;
                ; FUNCTION;  PRVAL
                ; INPUTS: C - INTEGER, RANGE 0 TO F
                ; OUTPUTS: C - ASCII CHARACTER
                ; CALLS: NOTHING
                ; DESTROYS: B,C,H,L,F/F'S
                ; DESCRIPTION: PRVAL CONVERTS A NUMBER IN THE RANGE 0 TO F HEX TO
                ;              THE CORRESPONDING ASCII CHARACTER, 0-9,A-F.  PRVAL
                ;              DOES NOT CHECK THE VALIDITY OF ITS INPUT ARGUMENT.
                ;
 02DE           PRVAL:
 02DE   21BF03          LXI     H,DIGTB  ; ADDRESS OF TABLE
 02E1   0600            MVI     B,0      ; CLEAR HIGH ORDER BITS OF BC
 02E3   09              DAD     B        ; ADD DIGIT VALUE TO HL ADDRESS
 02E4   4E              MOV     C,M      ; FETCH CHARACTER FROM MEMORY
 02E5   C9              RET
                ;
                ;************************************************************************
                ;
                ;
                ; FUNCTION: REGDS
                ; INPUTS: NONE
                ; OUTPUTS: NONE
                ; CALLS: ECHO,NMOUT,ERROR,CROUT
                ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                ; DESCRIPTION: REGDS DISPLAYS THE CONTENTS OF THE REGISTER SAVE
                ;              LOCATIONS, IN FORMATTED FORM, ON THE CONSOLE.  THE
                ;              DISPLAY IIS DRIVEN FROM A TABLE, RTAB, WHICH CONTAINS
                ;              THE REGISTER'S PRINT SYMBOL, SAVE LOCATION ADDRESS,
                ;              AND LENGTH (8 OR 16 BITS).
                ;
 02E6           REGDS:
 02E6   21CF03          LXI     H,RTAB   ; LOAD HL WITH ADDRESS OF START OF TABLE
 02E9           REG05:
 02E9   4E              MOV     C,M      ; GET PRINT SYMBOL OF REGISTER
 02EA   79              MOV     A,C
 02EB   B7              ORA     A        ; TEST FOR 0 - END OF TABLE
 02EC   C2F302          JNZ     REG10    ; IF NOT END, BRANCH
 02EF   CDEE01          CALL    CROUT    ; ELSE, CARRIAGE RETURN/LINE FEED TO END
 02F2   C9              RET              ;DISPLAY
 02F3           REG10:
```

```
 02F3   CDF401          CALL    ECHO     ; ECHO CHARACTER
 02F6   0E3D            MVI     C,'='
 02F8   CDF401          CALL    ECHO     ; OUTPUT EQUALS SIGN, I.E. A=
 02FB   23              INX     H        ; POINT TO START OF SAVE LOCATION ADDRESS
 02FC   5E              MOV     E,M      ; GET LSP OF SAVE LOCATION ADDRESS TO E
 02FD   1613            MVI     D,DATA SHR 8    ; PUT MSP OF SAVE LOC ADDRESS INTO D
 02FF   23              INX     H        ; POINT TO LENGTH FLAG
 0300   1A              LDAX    D        ; GET CONTENTS OF SAVE ADDRESS
 0301   CDC302          CALL    NMOUT    ; DISPLAY ON CONSOLE
 0304   7E              MOV     A,M      ; GET LENGTH FLAG
 0305   B7              ORA     A        ; SET SIGN F/F
 0306   CA0E03          JZ      REG15    ; IF 0, REGISTER IS 8 BITS
 0309   1B              DCX     D        ; ELSE, 16 BIT REGISTER SO MORE TO DISPLAY
 030A   1A              LDAX    D        ; GET LOWER 8 BITS
 030B   CDC302          CALL    NMOUT    ; DISPLAY THEM
 030E                REG15:
 030E   0E20            MVI     C,' '
 0310   CDF401          CALL    ECHO
 0313   23              INX     H        ; POINT TO START OF NEXT TABLE ENTRY
 0314   C3E902          JMP     REG05    ; DO NEXT REGISTER
                     ;
                     ;
                     ;****************************************************************
                     ;
                     ;
                     ; FUNCTION: RGADR
                     ; INPUTS: C - CHARACTER DENOTING REGISTER
                     ; OUTPUTS: BC - ADDRESS OF ENTRY IN RTAB CORRESPONDING TO REGISTER
                     ; CALLS: ERROR
                     ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                     ; DESCRIPTION: RGADR TAKES A SINGLE CHARACTER AS INPUT.  THIS CHARACTER
                     ;              DENOTES A REGISTER.  RGADR SEARCHES THE TABLE RTAB
                     ;              FOR A MATCH ON THE INPUT ARGUMENT.  IF ONE OCCURS,
                     ;              RGADR RETURNS THE ADDRESS OF THE ADDRESS OF THE
                     ;              SAVE LOCATION CORRESPONDING TO THE REGISTER.  THIS
                     ;              ADDRESS POINTS INTO RTAB.  IF NO MATCH OCCURS, THEN
                     ;              THE REGISTER IDENTIFIER IS ILLEGAL AND CONTROL IS
                     ;              PASSED TO THE ERROR ROUTINE.
                     ;
 0317                RGADR:
 0317   21CF03          LXI     H,RTAB   ; HL GETS ADDRESS OF TABLE START
 031A   110300          LXI     D,RTABS  ; DE GET SIZE OF A TABLE ENTRY
 031D                RGA05:
 031D   7E              MOV     A,M      ; GET REGISTER IDENTIFIER
 031E   B7              ORA     A        ; CHECK FOR TABLE END (IDENTIFIER IS 0)
 031F   CA0D02          JZ      ERROR    ; IF AT END OF TABLE, ARGUMENT IS ILLEGAL
 0322   B9              CMP     C        ; ELSE, COMPARE TABLE ENTRY AND ARGUMENT
 0323   CA2A03          JZ      RGA10    ; IF EQUAL, WE'VE FOUND WHAT WE'RE LOOKING FOR
 0326   19              DAD     D        ; ELSE, INCREMENT TABLE POINTER TO NEXT ENTRY
 0327   C31D03          JMP     RGA05    ; TRY AGAIN
 032A                RGA10:
```

```
 032A   23              INX     H        ; IF A MATCH, INCREMENT TABLE POINTER TO
 032B   44              MOV     B,H      ; /SAVE LOCATION ADDRESS
 032C   4D              MOV     C,L      ; RETURN THIS VALUE
 032D   C9              RET
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ;
                  ; FUNCTION: RSTTF
                  ; INPUTS: NONE
                  ; OUTPUTS: NONE
                  ; CALLS: NOTHING
                  ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                  ; DESCRIPTION: RSTTF RESTORES ALL CPU REGISTER, FLIP/FLOPS, STACK
                  ;              POINTER AND PROGRAM COUNTER FROM THEIR RESPECTIVE
                  ;              SAVE LOCATIONS IN MEMORY.  THE ROUTINE THEN TRANSFERS
                  ;              CONTROL TO THE LOCATION SPECIFIED BY THE PROGRAM
                  ;              COUNTER (I.E. THE RESTORED VALUE).  THE ROUTINE
                  ;              EXITS WITH THE INTERRUPTS ENABLED.
                  ;
 032E             RSTTF:
 032E   F3              DI               ; DISABLE INTERRUPTS WHILE RESTORING THINGS
 032F   21ED13          LXI     H,MSTAK  ; SET MONITOR STACK POINTER TO START OF STACK
 0332   F9              SPHL
 0333   D1              POP     D        ; START ALSO END OF REGISTER SAVE AREA
 0334   C1              POP     B
 0335   F1              POP     PSW
 0336   2AF713          LHLD    SSAVE    ; RESTORE USER STACK POINTER
 0339   F9              SPHL
 033A   2AF513          LHLD    PSAVE
 033D   E5              PUSH    H        ; PUT USER RETURN ADDRESS ON USER STACK
 033E   2AF313          LHLD    LSAVE    ; RESTORE HL REGISTERS
 0341   FB              EI               ; ENABLE INTERRUPTS NOW
 0342   C9              RET              ; JUMP TO RESTORED PC LOCATION
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ;
                  ; FUNCTION: SRET
                  ; INPUTS: NONE
                  ; OUTPUTS: CARRY = 1
                  ; CALLS: NOTHING
                  ; DESTROYS: CARRY
                  ; DESCRIPTION: SRET IS JUMPED TO BY ROUTINES WISHING TO RETURN SUCCESS.
                  ;              SRET SETS THE CARRY TRUE AND THEN RETURNS TO THE
                  ;              CALLER OF THE ROUTINE INVOKING SRET.
                  ;
 0343             SRET:
 0343   37              STC              ; SET CARRY TRUE
 0344    C9                 RET                ; RETURN APPROPRIATELY
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ; FUNCTION: STHF0
                    ; INPUTS: DE - 16 BIT ADDRESS OF BYTE TO BE STORED INTO
                    ; OUTPUTS: NONE
                    ; CALLS: STHLF
                    ; DESTROYS: A,B,C,H,L,F/F'S
                    ; DESCRIPTION: STHF0 CHECKS THE HALF BYTE FLAG IN TEMP TO SEE IF
                    ;              IT IS SET TO LOWER.  IF SO, STHF0 STORES A 0 TO
                    ;              PAD OUT THE LOWER HALF OF THE ADDRESSED BYTE;
                    ;              OTHERWISE, THE ROUTINE TAKES NO ACTION.
                    ;
 0345               STHF0:
 0345    3AF913             LDA     TEMP       ; GET HALF BYTE FLAG
 0348    B7                 ORA     A          ; SET F/F'S
 0349    C0                 RNZ                ; IF SET TO UPPER, DON'T DO ANYTHING
 034A    0E00               MVI     C,0        ; ELSE, WANT TO STORE THE VALUE 0
 034C    CD5003             CALL    STHLF      ; DO IT
 034F    C9                 RET
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ; FUNCTION: STHLF
                    ; INPUTS: C - 4 BIT VALUE TO BE STORED IN HALF BYTE
                    ;         DE - 16 BIT ADDRESS OF BYTE TO BE STORED INTO
                    ; OUTPUTS: NONE
                    ; CALLS: NOTHING
                    ; DESTROYS: A,B,C,H,L,F/F'S
                    ; DESCRIPTION: STHLF TAKES THE 4 BIT VALUE IN C AND STORES IT IN
                    ;              HALF OF THE BYTE ADDRESSED BY REGISTERS DE.  THE
                    ;              HALF BYTE USED (EITHER UPPER OR LOWER) IS DENOTED
                    ;              BY THE VALUE OF THE FLAG IN TEMP.  STHLF ASSUMES
                    ;              THAT THIS FLAG HAS BEEN PREVIOUSLY SET
                    ;              (NOMINALLY BY ICMD).
                    ;
 0350               STHLF:
 0350    D5                 PUSH    D
 0351    E1                 POP     H          ; MOVE ADDRESS OF BYTE INTO HL
 0352    79                 MOV     A,C        ; GET VALUE
 0353    E60F               ANI     0FH        ; FORCE TO 4 BIT LENGTH
 0355    4F                 MOV     C,A        ; PUT VALUE BACK
 0356    3AF913             LDA     TEMP       ; GET HALF BYTE FLAG
 0359    B7                 ORA     A          ; CHECK FOR LOWER HALF
 035A    C26303             JNZ     STH05      ; BRANCH IF NOT
 035D    7E                 MOV     A,M        ; ELSE, GET BYTE
```

```
035E    E6F0          ANI     0F0H     ; CLEAR LOWER 4 BITS
0360    B1            ORA     C        ; OR IN VALUE
0361    77            MOV     M,A      ; PUT BYTE BACK
0362    C9            RET
0363               STH05:
0363    7E            MOV     A,M      ; IF UPPER HALF, GET BYTE
0364    E60F          ANI     0FH      ; CLEAR UPPER 4 BITS
0366    47            MOV     B,A      ; SAVE BYTE IN B
0367    79            MOV     A,C      ; GET VALUE
0368    0F            RRC
0369    0F            RRC
036A    0F            RRC
036B    0F            RRC              ; ALIGN TO UPPER 4 BITS
036C    B0            ORA     B        ; OR IN ORIGINAL LOWER 4 BITS
036D    77            MOV     M,A      ; PUT NEW CONFIGURATION BACK
036E    C9            RET
                   ;
                   ;
                   ;*****************************************************************
                   ;
                   ;
                   ; FUNCTION: VALDG
                   ; INPUTS: C - ASCII CHARACTER
                   ; OUTPUTS: CARRY - 1 IF CHARACTER REPRESENTS VALID HEX DIGIT
                   ;                - 0 OTHERWISE
                   ; CALLS: NOTHING
                   ; DESTROYS: A,F/F'S
                   ; DESCRIPTION: VALDG RETURNS SUCCESS IF ITS INPUT ARGUMENT IS
                   ;              AN ASCII CHARACTER REPRESENTING A VALID HEX DIGIT
                   ;                    '-F), AND FAILURE OTHERWISE.
                   ;
036F               VALDG:
036F    79            MOV     A,C
0370    FE30          CPI     '0'      ; TEST CHARACTER AGAINST '0'
0372    FA1802        JM      FRET     ; IF ASCII CODE LESS, CANNOT BE VALID DIGIT
0375    FE39          CPI     '9'      ; ELSE, SEE IF IN RANGE '0'-'9'
0377    FA4303        JM      SRET     ; CODE BETWEEN '0' AND '9'
037A    CA4303        JZ      SRET     ; CODE EQUAL '9'
037D    FE41          CPI     'A'      ; NOT A DIGIT - TRY FOR A LETTER
037F    FA1802        JM      FRET     ; NO - CODE BETWEEN '9' AND 'A'
0382    FE47          CPI     'G'
0384    F21802        JP      FRET     ; NO - CODE GREATER THAN 'F'
0387    C34303        JMP     SRET     ; OKAY - CODE IS 'A' TO 'F', INCLUSIVE
                   ;
                   ;
                   ;*****************************************************************
                   ;
                   ;
                   ; FUNCTION: VALDL
                   ; INPUTS: C - CHARACTER
                   ; OUTPUTS: CARRY - 1 IF INPUT ARGUMENT VALID DELIMTER
```

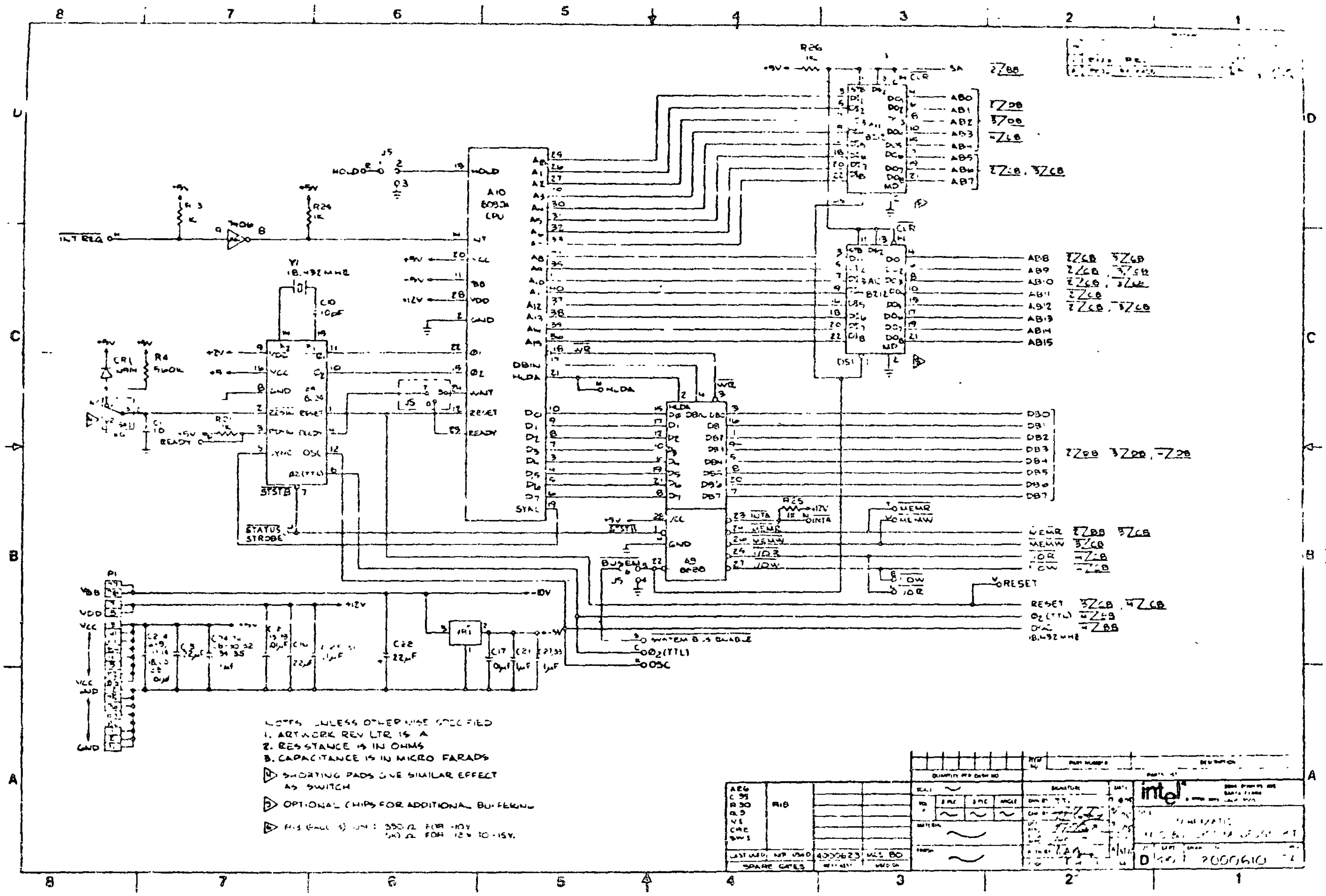
```
                    ;                  - 0 OTHERWISE
                    ; CALLS: NOTHING
                    ; DESTROYS: A,F/F'S
                    ; DESCRIPTION: VALDL RETURNS SUCCESS IF ITS INPUT ARGUMENT IS A VALID
                    ;              DELIMITER CHARACTER (SPACE, COMMA, CARRIAGE RETURN) AND
                    ;              FAILURE OTHERWISE.
                    ;
038A                VALDL:
038A    79                  MOV     A,C
038B    FE2C                CPI     ','         ; CHECK FOR COMMA
038D    CA4303              JZ      SRET
0390    FE0D                CPI     CR          ; CHECK FOR CARRIAGE RETURN
0392    CA4303              JZ      SRET
0395    FE20                CPI     ' '         ; CHECK FOR SPACE
0397    CA4303              JZ      SRET
039A    C31802              JMP     FRET        ; ERROR IF NONE OF THE ABOVE
                    ;
                    ;
                    ;**************************************************************************
                    ;
                    ;
                    ;                               MONITOR TABLES
                    ;
                    ;
                    ;**************************************************************************
                    ;
                    ;
039D                SGNON:                              ; SIGNON MESSAGE
039D    0D0A4D43            DB      CR,LF,'MCS-80 KIT',CR,LF
03A1    532D3830
03A5    204B4954
03A9    0D0A
000E                LSGNON          EQU     $-SGNON ; LENGTH OF SIGNON MESSAGE
                    ;
03AB                CADR:                       ; TABLE OF ADDRESSES OF COMMAND ROUTINES
03AB    0000                DW      0           ; DUMMY
03AD    4101                DW      XCMD
03AF    1D01                DW      SCMD
03B1    FD00                DW      MCMD
03B3    B300                DW      ICMD
03B5    9500                DW      GCMD
03B7    5E00                DW      DCMD
                    ;
03B9                CTAB:                       ; TABLE OF VALID COMMAND CHARACTERS
03B9    44                  DB      'D'
03BA    47                  DB      'G'
03BB    49                  DB      'I'
03BC    4D                  DB      'M'
03BD    53                  DB      'S'
03BE    58                  DB      'X'
0006                NCMDS EQU       $-CTAB      ; NUMBER OF VALID COMMANDS
```

```
                        ;
038F                    DIGTB:                                ; TABLE OF PRINT VALUES OF HEX DIGITS
03BF      30                  DB        '0'
03C0      31                  DB        '1'
03C1      32                  DB        '2'
03C2      33                  DB        '3'
03C3      34                  DB        '4'
03C4      35                  DB        '5'
03C5      36                  DB        '6'
03C6      37                  DB        '7'
03C7      38                  DB        '8'
03C8      39                  DB        '9'
03C9      41                  DB        'A'
03CA      42                  DB        'B'
03CB      43                  DB        'C'
03CC      44                  DB        'D'
03CD      45                  DB        'E'
03CE      46                  DB        'F'
                        ;
03CF                    RTAB:                   ; TABLE OF REGISTER INFORMATION
03CF      41                  DB        'A'     ; REGISTER IDENTIFIER
03D0      F2                  DB        ASAVE AND 0FFH  ; ADDRESS OF REGISTER SAVE LOCATION
03D1      00                  DB        0       ; LENGTH FLAG - 0=8 BITS, 1=16 BITS
0003                    RTABS EQU       $-RTAB  ; SIZE OF AN ENTRY IN THIS TABLE
03D2      42                  DB        'B'
03D3      F0                  DB        BSAVE AND 0FFH
03D4      00                  DB        0
03D5      43                  DB        'C'
03D6      EF                  DB        CSAVE AND 0FFH
03D7      00                  DB        0
03D8      44                  DB        'D'
03D9      EE                  DB        DSAVE AND 0FFH
03DA      00                  DB        0
03DB      45                  DB        'E'
03DC      ED                  DB        ESAVE AND 0FFH
03DD      00                  DB        0
03DE      46                  DB        'F'
03DF      F1                  DB        FSAVE AND 0FFH
03E0      00                  DB        0
03E1      48                  DB        'H'
03E2      F4                  DB        HSAVE AND 0FFH
03E3      00                  DB        0
03E4      4C                  DB        'L'
03E5      F3                  DB        LSAVE AND 0FFH
03E6      00                  DB        0
03E7      4D                  DB        'M'
03E8      F4                  DB        HSAVE AND 0FFH
03E9      01                  DB        1
03EA      50                  DB        'P'
03EB      F6                  DB        PSAVE+1 AND 0FFH
03EC      01                  DB        1
```

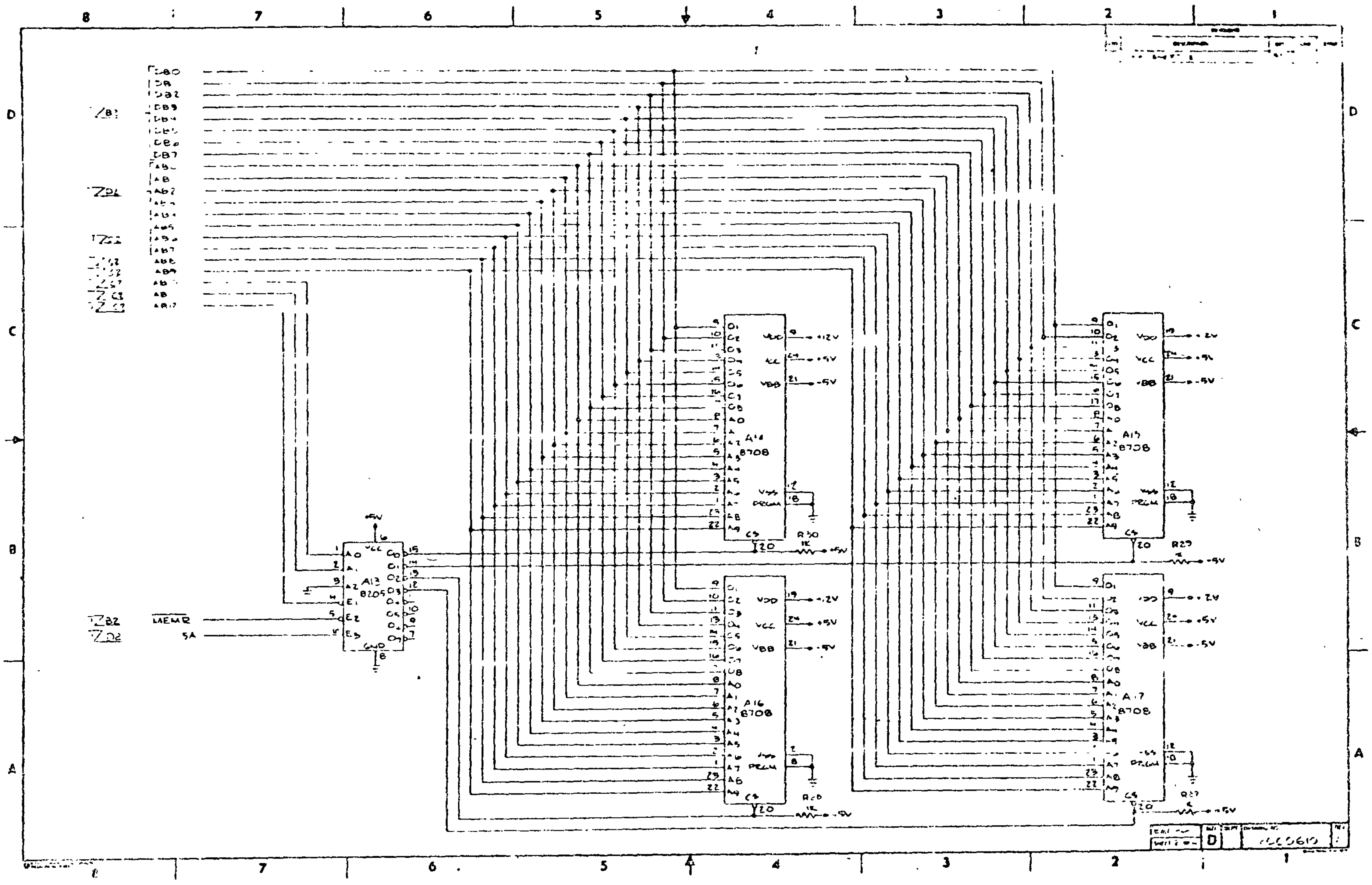
```
 03ED   53               DB      'S'
 03EE   F8               DB      SSAVE+1 AND 0FFH
 03EF   01               DB      1
 03F0   00               DB      0     ; END OF TABLE MARKERS
 03F1   00               DB      0
                 ;
 03FA                    ORG     BRTAB
                 ;
 03FA   C3E301           JMP     CO    ; BRANCH TABLE FOR USER ACCESSIBLE ROUTINES
 03FD   C3D001           JMP     CI
                 ;
                 ;
                 ;**********************************************************************
                 ;
                 ;
 1300                    ORG     DATA
 13ED                    ORG     REGS  ; ORG TO REGISTER SAVE - STACK GOES IN HERE
                 ;
 13ED            MSTAK           EQU     $      ; START OF MONITOR STACK
 13ED   00       ESAVE:          DB      0      ; E REGISTER SAVE LOCATION
 13EE   00       DSAVE:          DB      0      ; D REGISTER SAVE LOCATION
 13EF   00       CSAVE:          DB      0      ; C REGISTER SAVE LOCATION
 13F0   00       BSAVE:          DB      0      ; B REGISTER SAVE LOCATION
 13F1   00       FSAVE:          DB      0      ; FLAGS SAVE LOCATION
 13F2   00       ASAVE:          DB      0      ; A REGISTER SAVE LOCATION
 13F3   00       LSAVE:          DB      0      ; L REGISTER SAVE LOCATION
 13F4   00       HSAVE:          DB      0      ; H REGISTER SAVE LOCATION
 13F5   0000     PSAVE:          DW      0      ; PGM COUNTER SAVE LOCATION
 13F7   0000     SSAVE:          DW      0      ; USER STACK POINTER SAVE LOCATION
 13F9   00       TEMP:           DB      0      ; TEMPORARY MONITOR CELL
                 ;
 13FD                    ORG     BRLOC          ; ORG TO USER BRANCH LOCATION
                 ;
 0003            USRBR:          DS      3      ; BRANCH GOES IN HERE
                 ;
                 ;
                         END
NO PROGRAM ERRORS
```

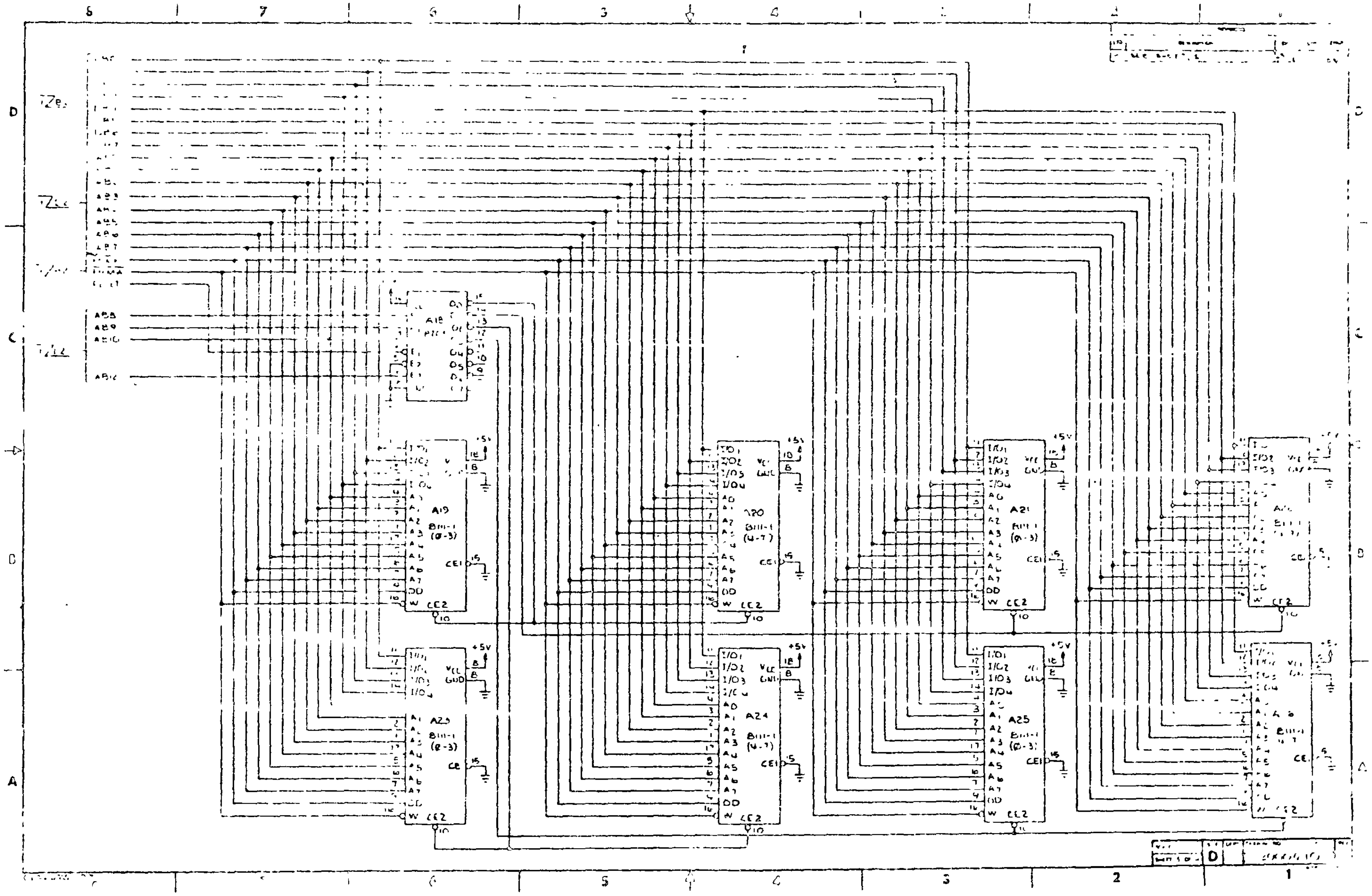
```
8080 MACRO ASSEMBLER, VER 2.3  ERRORS = 0 PAGE 28

                         SYMBOL TABLE

* 01

A       0007        ASAVE   13F2        B       0000        BRCHR   0018
BREAK   01BD        BRLOC   13FD        BRTAB   03FA        BSAVE   13F0
C       0001        CADR    03AB        CI      01D0        CMD     0027
CNCTL   001B        CNIN    00FA        CNOUT   00FA        CNVBN   01DA
CO      0113        CONST   00FB        CR      000D        CROUT   01EE
CSAVE   13EF        CTAB    03B9        D       0002        DATA    1300
DCM05   0065        DCM10   0070        DCM12   0085        DCM15   008B
DCMD    005E        DIGTB   03BF        DSAVE   13EE        E       0003
ECH05   01FD        ECH10   020B        ECHO    01F4        ERROR   020D
ESAVE   13ED        ESC     001B        FALSE   0F9C        FRET    0218
FSAVE   13F1        GCM05   00AA        GCM10   00B0        GCMD    0095
GETCH   021B        GETCM   002B        GETHX   0222        GETNM   0257
GHX05   0228        GHX10   0241        GNM05   025E        GNM10   0273
GNM15   0281        GNM20   0286        GNM25   0291        GNM30   0295
GO      0008   *    GTC03   003B        GTC05   0048        GTC10   0054
H       0004        HCHAR   000F        HIL05   02BD        HILO    029C
HSAVE   13F4        ICM05   00BE        ICM10   00E6        ICM20   00EE
ICM25   00F4        ICMD    00B3        INVRT   00FF        L       0005
LF      000A        LOWER   0000   *    LSAVE   13F3        LSGNO   000E
M       0006        MCM05   0105        MCMD    00FD        MODE    00CF
MSGL    0022        MSTAK   13ED        NCMDS   0006        NEWLN   000F
NMOUT   02C3        PRTY0   007F        PRVAL   02DE        PSAVE   13F5
PSW     0006        RBR     0002        REG05   02E9        REG10   02F3
REG15   030E        REGDS   02E6        REGS    13ED        RGA05   031D
RGA10   032A        RGADR   0317        RSTTF   032E        RSTU    0038
RTAB    03CF        RTABS   0003        SCM05   0122        SCM10   012D
SCM15   013D        SCMD    011D        SGNON   039D        SP      0006
SRET    0343        SSAVE   13F7        STH05   0363        STHF0   0345
STHLF   0350        TEMP    13F9        TERM    001B        TRDY    0001
TRUE    0F9F        UPPER   00FF        USRBR   13FD        VALDG   036F
VALDL   038A        XCM05   0154        XCM10   0163        XCM15   0170
XCM18   017B        XCM20   0194        XCM25   01AB        XCM27   01AC
XCM30   01B4        XCMD    0141

* 02

* 03

* 04

* 05

* 06

* 07

* 08

* 09

* 10

* 11

* 12

* 13

..L
```

APPENDIX C. MCS-80 SCHEMATICS

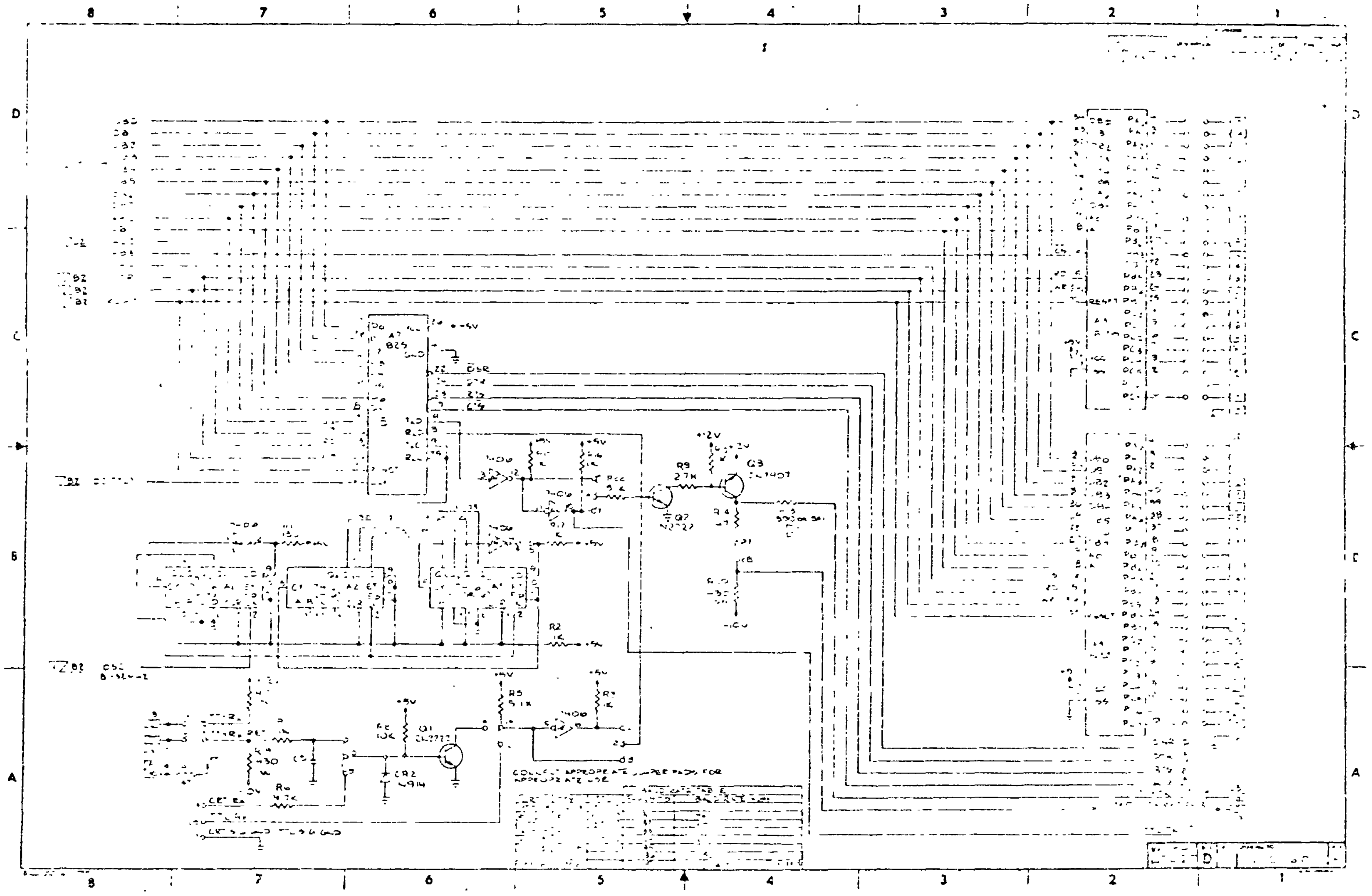


NOTES: UNLESS OTHERWISE SPECIFIED
1. ARTWORK REV LTR IS A
2. RESISTANCE IS IN OHMS
3. CAPACITANCE IS IN MICRO FARADS
4. SHORTING PADS GIVE SIMILAR EFFECT AS SWITCH
5. OPTIONAL CHIPS FOR ADDITIONAL BUFFERING
6. R13 (PAGE 4) ONLY: 390 Ω FOR +10V
510 Ω FOR 12V TO +15V.


DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
AB0
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
AB10
AB11
AB12
MEMR
5A
A13
8205
A14
8708
A15
8708
A16
8708
A17
8708
VDD
VCC
VBB
VSS
PRGM
CS
+12V
+5V
-5V
R30
R29
R28
R27
1K
GND
D



A18
A19
8111-1
(0-3)
A20
(4-7)
A21
A23
A24
A25
AB8
AB9
AB10
AB12
I/O1
I/O2
I/O3
I/O4
VCC
GND
+5V
CE1
CE2
OD
W
D
C
B
A


R9
2.7K
Q3
2N4403
Q2
2N2222
R4
4.7
+12V
R2
1K
+5V
R5
5.1K
R3
1K
R6
10K
Q1
2N2222
CR2
4914
R6
4.7K
430
C5
8251
GND
DSR
A7
A1
A2
RESET
OSC
CONNECT APPROPRIATE JUMPER PADS FOR APPROPRIATE USE
D

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES: TEORIA Y APLICACIONES.

TEMA 6: EL SISTEMA SDK - 80.

PRACTICA.

M. EN C. ANGEL KURI MORALES

ABRIL, 1978

EL SISTEMA SDK-PRACTICA.

El objetivo de esta práctica es que se familiarice usted con el lenguaje y forma de operación del sistema 8080 y, en particular, con el sistema SDK-80.

A continuación hay varios segmentos de código y/o comandos que usted debe de ejecutar. Antes, siga las siguientes instrucciones. Recuerde que el equipo con el que está trabajando es sensible a la estática. Por ello, es recomendable que evite tocarlo antes de "descargarse". (Trate de estar en contacto con la mesa de trabajo cuando manipule los 'kits').

<u>Práctica 1.</u>

1) Utilice el comando X (examinar). Vea el contenido de los registros. En particular vea usted las banderas. Recuerde que los bits del registro de banderas (F) tiene el siguiente formato:

s - Z - 0 - AC - 0 - P-1 - G

Note que la pareja HL es un apuntador natural a memoria. Este apuntador señala a la localidad denotada por "M".

El registro SP (S en el 'kit') indica la localidad de memoria a que se apunta al momento de examinar el CPU.

Finalmente, note que el registro PC (P en el 'kit') no puede accesarse en forma directa; igualmente, el contenido del SP no puede copiarse en forma directa.

El siguiente código almacena el contenido de todos los registros en memoria:

```
GUARDA:  CALL  GETPC; toma el PC y almacena en memoria
         PUSH  H
         LXI  H,0
         DAD  SP
         SHLD GSP;  ahora guarda el 'SP'
         POP  H
         SHLD GHL;  guarda HL
         XCHG
         SHLD GDE;  guarda DE
         MOV  H,B
         MOV  L,C
         SHLD GBC;  guarda BC
         PUSH PSW
         POP  H
         SHLD  GAF; guarda A & Banderas
         HLT
GETPC:   XTHL;   intercambia SP y HL
         SHLD  GPC; guarda PC
         XTHL;  deja como al principio
         RET
```

```
GPC     EQV     1300H
GSP     EQU     1302H
GHL     EQV     1304H
GDE     EQV     1306H
GBC     EQV     1308H
GAF     EQV     130AH
```

2. Pruebe la instrucción I (insert) para el código anterior. Verifique que el siguiente código corresponde al anterior:

```
1300:   CD   50   13        131E:  E3
        E5                         22 A0 13
        21   00   00               E3
        39                         C9
        22   A2   13
        E1
        22   A4   13
        EB
        22   A6   13
        60
        69
        22   A8   13
        F5
        E1
        22   AA   13
        76
```

Una vez que haya insertado el código de (2), use la instrucción D (display). Use el comando D1300, 1323 y verifique que su programa está correcto.

En caso de algún error, modifique la localidad usando el comando S.

S#### - (número correcto)

3) Ahora ejecute el comando

S13A0

e introduzca cualquier valor conocido (por ejemplo 9) tantas veces como sea necesario para asegurar que las localidades de la 13A0 hasta la 13AB (inclusive) tienen un valor conocido.

Note que estas localidades de memoria se usan en su programa para guardar los valores de los registros.

4) Usando el comando X introduzca valores conocidos (los que usted quiera) en los registros del CPU.

5) Utilice el comando G (go) para ejecutar el programa que cargó antes en memoria.

La instrucción G1300 es la adecuada. Note que, puesto que el último byte de código corresponde a HLT (76), el CPU se "muere". Por tanto, ya no le responderá.

Inicialice, pues, el sistema (por medio del botón de Reset).

6) Vuelva a examinar las direcciones de la 13A0 a la 13AB. ¿Los datos que cargó en los registros y los de memoria son iguales? ¿Qué valor tiene el PC?

Si no obtuvo el resultado deseado, cheque su procedimiento y vuelva a empezar.

Note que, en una localidad doble, como las que hemos estado manejando, el byte menos significativo es el primero, y el más significativo es el segundo.

A eso se debe que el código

```
LXI  H,1400
```

corresponda a:

```
21  00  14
```

y no a

```
21  14  00.
```

Igualmente, note que al trabajar con registros de 16 bits (PC, SP,HL,DE,BC,PSW)el registro menos significativo va primero.

Asimismo, en la localidad 13AA se encuentran las banderas. Note que los bits 1, 3 y 5 están fijos a los valores 1,0 y 0 respectivamente.

¿Qué valores tienen las banderas?

Práctica 2.

La siguiente práctica tiene por objeto familiarizarlo con código de ASSEMBLER, para manejo aritmético.

Observe la siguiente rutina.

Una de las ventajas del sistema 8080 es su 'stack'. En este programa hemos implementado una división recursiva, de modo que el resultado se obtiene con 16 bits de precisión en la pareja HL.

Usando el código de la rutina de recursión

a) Cargue el programa en RAM. Recuerde que el 'kit' tiene RAM a partir de la localidad 1300H.

b) Ejecute varias corridas con diferentes datos.

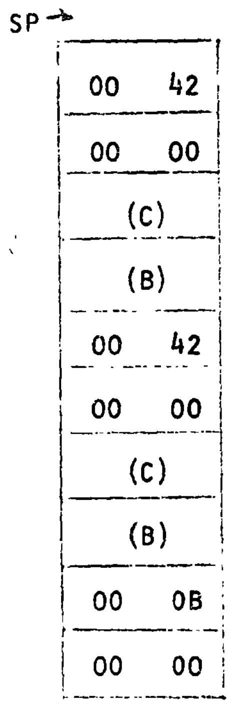
En la siguiente figura se muestra un mapa del 'stack' en su máximo nivel de anidamiento.

Note que el programa está escrito de modo que únicamente una localidad de memoria se requiere utilizar para almacenamiento de variables.

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 1


 00FF                RCRSV EQU OFFH
 0000     31FE00     LXI SP,OFEH
 0003     AF         XRA A
 0004     2F         CMA
 0005     32FF00     STA RCRSV
 0008     CD0C00     CALL DIV
 000B     76         HLT
 000C     B7         DIV: ORA A
 000D     E0         RPO
 000E     F21A00     JP ETC
 0011     60         MOV H,B
 0012     69         MOV L,C
 0013     B7         ORA A
 0014     7A         MOV A,D
 0015     1F         RAR
 0016     57         MOV D,A
 0017     7B         MOV A,E
 0018     1F         RAR
 0019     5F         MOV E,A
 001A     AF         ETC: XRA A
 001B     47         MOV B,A
 001C     0E08       MVI C,8
 001E     7D         COMPARA: MOV A,L
 001F     BB         CMP E
 0020     7C         MOV A,H
 0021     9A         SBB D
 0022     DA2E00     JC SHLFT
 0025     67         MOV H,A
 0026     7D         MOV A,L
 0027     93         SUB E
 0028     6F         MOV L,A
 0029     29         DAD H
 002A     37         STC
 002B     C33000     JMP RESULTADO
 002E     29         SHLFT: DAD H
 002F     B7         ORA A ;RSTC
 0030     78         RESULTADO:MOV A,B
 0031     17         RAL
 0032     47         MOV B,A
 0033     0D         DCR C
 0034     C21E00     JNZ COMPARA
 0037     C5         RECURSION: PUSH B
 0038     3AFF00     LDA RCRSV
 003B     3C         INR A
 003C     32FF00     STA RCRSV
 003F     CD0C00     CALL DIV
 0042     3D         DCR A
 0043     CA4900     JZ L2
 0046     C1         L1: POP B
 0047     60         MOV H,B
 0048     C9         RET
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 2


  0049    C1         L2: POP B
  004A    68         MOV L,B
  004B    C9         RET
                     END
NO PROGRAM ERRORS
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 3


                              SYMBOL TABLE

 * 01

 A        0007       B        0000       C        0001       COMPA   001E
 D        0002       DIV      000C       E        0003       ETC     001A
 H        0004       L        0005       L1       0046 *     L2      0049
 M        0006       PSW      0006       RCRSV    00FF       RECUR   0037 *
 RESUL    0030       SHLFT    002E       SP       0006

$ET=2:02.1 PT=12.2 IO=1.2
```



Note que todos los registros del CPU se utilizan durante la ejecución de la rutina.

<u>Práctica 3.</u>

En esta práctica se desea que noten la diferencia entre el código generado por ASSEMBLER y por compilador (PL/M).

Observe el programa que se presenta a continuación. Aquí se implementa, también, la división, así como la operación "MOD". Esta operación se define como el operador binario que arroja como resultado el residuo de la división entre dos números.

Note usted la falta de economía del compilador, que generó el siguiente código:

```
        LXI   SP,  FAH
        LXI   H, "C"
        MOV   E,M
        INR   L
        MOV   D,M
        MVI   L, FCH; DIR ("B")
        MOV   C,M
        INR   L
        MOV   B,M
        JMP   L1
DIV:    MOV   A,D
        CMA
        MOV   D,A
        MOV   A,E
        CMA
        MOV   E,A
        INX   D
        LXI   H,0
        MVI   A,11H
DØ:     PUSH  H
        DAD   D
        JNC   D1
        XTHL
D1:     POP   H
        PUSH  PSW
        MOV   A,C
        RAL
        MOV   C,A
        MOV   A,B
        RAL
        MOV   B,A
        MOV   A,L
        RAL
        MOV   L,A
        MOV   A,H
        RAL
        MOV   H,A
        POP   PSW
        DCR   A
        JNZ   DØ
        ORA   A
        MOV   A,H
        RAR
        MOV   D,A
        MOV   A,L
        RAR
        MOV   E,A
        RET
L1:     CALL  DIV
        LXI   H,"A"
        MOV   M,C
        INX   H
```

```
MOV  M,B
LXI  H,"B"
MOV  E,M
INR  L
MOV  D,M
INR  L
MOV  C,M
INR  L
MOV  B,M
CALL DIV
LXI  H,"A"
MOV  M,E
INX  H
MOV  M,D
EI
HLT
```

La rutina de división, en este caso, ocupa 49 localidades de memoria, contra 61 localidades en el caso anterior. Sin embargo, por manejo de variables, el programa ocupa 91 localidades contra 72 del programa escrito en ensamblador.

Cargue usted el programa anterior y obtenga los siguientes resultados:

```
8080 PLM1 VERS 4.0

$I=0

00001  1

00002  1   DECLARE (A,B,C) ADDRESS;

00003  1

00004  1   A=B/C;

00005  1

00006  1   A=C MOD B;

00007  1   EOF

NO PROGRAM ERRORS

$ET=33.4 PT=3.9 IO=1.6
E PL/MII/4;FILE FILE22(MAXRECSIZE=22,BLOCKSIZE=22),FILE23(MAXRECSIZE=22,B
$RUNNING 4025
$?

8080 PLM2 VERS 4.0

$I=7 $F=1

STACK SIZE = 4 BYTES
MEMORY...........................0100H
A................................00FAH
B................................00FCH
C................................00FEH
0000H LXI SP FAH    00H    LXI H  FEH    00H    MOV EM INR L  MOV DM
0009H MOV LI FCH    MOV CM INR L  MOV BM JMP    3EH    00H    MOV AD
0012H CMA    MOV DA MOV AE CMA    MOV EA INX D  LXI H  00H    00H
001BH MOV AI 11H  . PUSH H DAD D  JNC    23H    00H    XTHL . POP H
0024H PUSH A MOV AC RAL    MOV CA MOV AB RAL    MOV BA MOV AL RAL
002DH MOV LA MOV AH RAL    MOV HA POP A  DCR A  JNZ    1DH    00H
0036H ORA A  MOV AH RAR    MOV DA MOV AL RAR    MOV EA RET    CALL
003FH 11H    00H    LXI H  FAH    00H    MOV MC INX H  MOV MB LXI H
0048H FCH    00H    MOV EM INR L  MOV DM INR L  MOV CM INR L  MOV BM
0051H CALL   11H    00H    LXI H  FAH    00H    MOV ME INX H  MOV MD
005AH EI     HLT
NO PROGRAM ERRORS

$ET=1:00.9 PT=3.8 IO=1.9
```

125 MOD 13

497 MOD 19

711 / 16

93 / 93

Recuerde que hay que ajustar las localidades de RAM.

Práctica 4.

En esta práctica deberán checar el funcionamiento de un sistema habilitado por interrupciones.

Para la práctica deberán utilizar un 'kit' que contiene un ROM ya programado en las localidades 400H a 7FFH.

1) Examine el listado que se incluye y vea cómo se programó el PPI. Modo 0? Modo 1? Modo 2?.

2) Note que en el SDK-80, las localidades 13FD, 13FE y 13FF deben programarse para que haya un salto (JMP) a la rutina de servicio de interrupciones. ¿En qué localidad está esa rutina? Programe consecuentemente las localidades antes mencionadas.

3) El sistema tiene un "one-shot" para generar el pulso de interrupción. Dispare las interrrupciones.

Note que un sistema como el que está aquí diseñado debe prever que una interrupción llegue en el momento de despliegue. ¿Cómo lo haría Ud.? Note la forma en que se efectuó en este caso.

```
INTERP/80 VERS 1.0
$F-1
LOAD 7 7.
B H.
REFER DIV,

76 LOAD OK

HEX BASE OK

REFER OK
TRACE RESUL T 4BH.

TRACE OK
S B=11,C=23.

SET OK
S D=24,E=17.

SET OK
G.

REFER AT CH=DIV
D CP.

CYZSP  A    B    C    D    E    H    L     HL      SP     PC
*0101*FFH*0BH*17H*18H*11H*00H*00H*0000H*00FCH*000CH
G.

*0011 FFH*00H*08H*0CH*08H*16H*2EH*162EH 00FCH*0030H
MOV AB
 0011*00H 00H 08H 0CH 08H 16H 2EH 162EH 00FCH*0031H
RAL
 0011 00H 00H 08H 0CH 08H 16H 2EH 162EH 00FCH*0032H
MOV BA
 0011 00H 00H 08H 0CH 08H 16H 2EH 162EH 00FCH*0033H
DCR C
*0000 00H 00H*07H 0CH 08H 16H 2EH 162EH 00FCH*0034H
JNZ 1EH
*1000*26H 00H 07H 0CH 08H*14H*4CH*144CH 00FCH*0030H
MOV AB
 1000*00H 00H 07H 0CH 08H 14H 4CH 144CH 00FCH*0031H
RAL
*0000*01H 00H 07H 0CH 08H 14H 4CH 144CH 00FCH*0032H
MOV BA
 0000 01H*01H 07H 0CH 08H 14H 4CH 144CH 00FCH*0033H
DCR C
CYZSP  A    B    C    D    E    H    L     HL      SP     PC
*0001 01H 01H*06H 0CH 08H 14H 4CH 144CH 00FCH*0034H
JNZ 1EH
*1001*44H 01H 06H 0CH 08H*10H*88H*1088H 00FCH*0030H
```

```
MOV AB
 1001*01H 01H 06H 0CH 08H 10H 88H 1088H 00FCH*0031H
RAL
*0001*03H 01H 06H 0CH 08H 10H 88H 1088H 00FCH*0032H
MOV BA
 0001 03H*03H 06H 0CH 08H 10H 88H 1088H 00FCH*0033H
DCR C
 0001 03H 03H*05H 0CH 08H 10H 88H 1088H 00FCH*0034H
JNZ 1EH
*1010*[illegible]0H 03H 05H 0CH 08H*09H*00H*0900H 00FCH*0030H
MOV AB
 1010*03H 03H 05H 0CH 08H 09H 00H 0900H 00FCH*0031H
RAL
*0010*07H 03H 05H 0CH 08H 09H 00H 0900H 00FCH*0032H
MOV BA
 0010 07H*07H 05H 0CH 08H 09H 00H 0900H 00FCH*0033H
DCR C
CYZSP  A   B   C   D   E   H   L    HL     SP     PC
*0000 07H 07H*04H 0CH 08H 09H 00H 0900H 00FCH*0034H
JNZ 1EH
*0011*FCH 07H 04H 0CH 08H*12H 00H*1200H 00FCH*0030H
MOV AB
 0011*07H 07H 04H 0CH 08H 12H 00H 1200H 00FCH*0031H
RAL
 0011*0EH 07H 04H 0CH 08H 12H 00H 1200H 00FCH*0032H
MOV BA
 0011 0EH*0EH 04H 0CH 08H 12H 00H 1200H 00FCH*0033H
DCR C
*0001 0EH 0EH*03H 0CH 08H 12H 00H 1200H 00FCH*0034H
JNZ 1EH
*1010*F8H 0EH 03H 0CH 08H*0BH*F0H*0BF0H 00FCH*0030H
MOV AB
 1010*0EH 0EH 03H 0CH 08H 0BH F0H 0BF0H 00FCH*0031H
RAL
*0010*1DH 0EH 03H 0CH 08H 0BH F0H 0BF0H 00FCH*0032H
MOV BA
 0010 1DH*1DH 03H 0CH 08H 0BH F0H 0BF0H 00FCH*0033H
DCR C
CYZSP  A   B   C   D   E   H   L    HL     SP     PC
*0000 1DH 1DH*02H 0CH 08H 0BH F0H 0BF0H 00FCH*0034H
JNZ 1EH
*0011*FFH 1DH 02H 0CH 08H*17H*E0H*17E0H 00FCH*0030H
MOV AB
 0011*1DH 1DH 02H 0CH 08H 17H E0H 17E0H 00FCH*0031H
RAL
 0011*3AH 1DH 02H 0CH 08H 17H E0H 17E0H 00FCH*0032H
MOV BA
 0011 3AH*3AH 02H 0CH 08H 17H E0H 17E0H 00FCH*0033H
DCR C
*0000 3AH 3AH*01H 0CH 08H 17H E0H 17E0H 00FCH*0034H
JNZ 1EH
*1011*D8H 3AH 01H 0CH 08H 17H*B0H*17B0H 00FCH*0030H
```

```
MOV AB
 1011*3AH 3AH 01H 0CH 08H 17H B0H 17B0H 00FCH*0031H
RAL
*0011*75H 3AH 01H 0CH 08H 17H B0H 17B0H 00FCH*0032H
MOV BA
 0011 75H*75H 01H 0CH 08H 17H B0H 17B0H 00FCH*0033H
DCR C
CYZSP  A   B   C   D   E   H   L    HL     SP    PC
*0101 75H 75H*00H 0CH 08H 17H B0H 17B0H 00FCH*0034H
JNZ 1EH
 0101 75H 75H 00H 0CH 08H 17H B0H 17B0H 00FCH*0037H
PUSH B
 0101 75H 75H 00H 0CH 08H 17H B0H 17B0H*00FAH*0038H
LDA FFH
 0101*FFH 75H 00H 0CH 08H 17H B0H 17B0H 00FAH*003BH
INR A
 0101*00H 75H 00H 0CH 08H 17H B0H 17B0H 00FAH*003CH
STA FFH
 0101 00H 75H 00H 0CH 08H 17H B0H 17B0H 00FAH*003FH
CALL CH
REFER AT CH=DIV


D CP.

CYZSP  A   B   C   D   E   H   L    HL     SP    PC
 0101 00H 75H 00H 0CH 08H 17H B0H 17B0H*00F8H*000CH


G.

*1010*A8H*00H*08H 0CH 08H 17H*50H*1750H 00F8H*0030H
MOV AB
 1010*00H 00H 08H 0CH 08H 17H 50H 1750H 00F8H*0031H
RAL
*0010*01H 00H 08H 0CH 08H 17H 50H 1750H 00F8H*0032H
MOV BA
 0010 01H*01H 08H 0CH 08H 17H 50H 1750H 00F8H*0033H
DCR C
*0000 01H 01H*07H 0CH 08H 17H 50H 1750H 00F8H*0034H
JNZ 1EH
*1001*48H 01H 07H 0CH 08H*16H*90H*1690H 00F8H*0030H
MOV AB
 1001*01H 01H 07H 0CH 08H 16H 90H 1690H 00F8H*0031H
RAL
*0001*03H 01H 07H 0CH 08H 16H 90H 1690H 00F8H*0032H
MOV BA
 0001 03H*03H 07H 0CH 08H 16H 90H 1690H 00F8H*0033H
DCR C
CYZSP  A   B   C   D   E   H   L    HL     SP    PC
 0001 03H 03H*06H 0CH 08H 16H 90H 1690H 00F8H*0034H
JNZ 1EH
*1011*88H 03H 06H 0CH 08H*15H*10H*1510H 00F8H*0030H
```

7

```
MOV AB
 1011*03H 03H 06H 0CH 08H 15H 10H 1510H 00F8H*0031H
RAL
*0011*07H 03H 06H 0CH 08H 15H 10H 1510H 00F8H*0032H
MOV BA
 0011 07H*07H 06H 0CH 08H 15H 10H 1510H 00F8H*0033H
DCR C
*0001 07H 07H*05H 0CH 08H 15H 10H 1510H 00F8H*0034H
JNZ 1EH
*1000*08H 07H 05H 0CH 08H*12H 10H*1210H 00F8H*0030H
MOV AB
 1000*07H 07H 05H 0CH 08H 12H 10H 1210H 00F8H*0031H
RAL
*0000*0FH 07H 05H 0CH 08H 12H 10H 1210H 00F8H*0032H
MOV BA
 0000 0FH*0FH 05H 0CH 08H 12H 10H 1210H 00F8H*0033H
DCR C
CYZSP   A    B    C    D    E    H    L     HL     SP     PC
 0000 0FH 0FH*04H 0CH 08H 12H 10H 1210H 00F8H*0034H
JNZ 1EH
*1000*08H 0FH 04H 0CH 08H*0CH 10H*0C10H 00F8H*0030H
MOV AB
 1000*0FH 0FH 04H 0CH 08H 0CH 10H 0C10H 00F8H*0031H
RAL
*0000*1FH 0FH 04H 0CH 08H 0CH 10H 0C10H 00F8H*0032H
MOV BA
 0000 1FH*1FH 04H 0CH 08H 0CH 10H 0C10H 00F8H*0033H
DCR C
*0001 1FH 1FH*03H 0CH 08H 0CH 10H 0C10H 00F8H*0034H
JNZ 1EH
*1000*08H 1FH 03H 0CH 08H*00H 10H*0010H 00F8H*0030H
MOV AB
 1000*1FH 1FH 03H 0CH 08H 00H 10H 0010H 00F8H*0031H
RAL
*0000*3FH 1FH 03H 0CH 08H 00H 10H 0010H 00F8H*0032H
MOV BA
 0000 3FH*3FH 03H 0CH 08H 00H 10H 0010H 00F8H*0033H
DCR C
CYZSP   A    B    C    D    E    H    L     HL     SP     PC
 0000 3FH 3FH*02H 0CH 08H 00H 10H 0010H 00F8H*0034H
JNZ 1EH
*0010*F4H 3FH 02H 0CH 08H 00H*20H*0020H 00F8H*0030H
MOV AB
 0010*3FH 3FH 02H 0CH 08H 00H 20H 0020H 00F8H*0031H
RAL
 0010*7EH 3FH 02H 0CH 08H 00H 20H 0020H 00F8H*0032H
MOV BA
 0010 7EH*7EH 02H 0CH 08H 00H 20H 0020H 00F8H*0033H
DCR C
*0000 7EH 7EH*01H 0CH 08H 00H 20H 0020H 00F8H*0034H
JNZ 1EH
*0010*F4H 7EH 01H 0CH 08H 00H*40H*0040H 00F8H*0030H
MOV AB
 0010*7EH 7EH 01H 0CH 08H 00H 40H 0040H 00F8H*0031H
```

```
RAL
 0010*FCH 7EH 01H 0CH 08H 00H 40H 0040H 00F8H*0032H
MOV BA
 0010 FCH FCH 01H 0CH 08H 00H 40H 0040H 00F8H*0033H
DCR C
CYZSP  A   B   C   D   E   H   L    HL     SP     PC
*0101 FCH FCH*00H 0CH 08H 00H 40H 0040H 00F8H*0034H
JNZ 1EH
 0101 FCH FCH 00H 0CH 08H 00H 40H 0040H 00F8H*0037H
PUSH B
 0101 FCH FCH 00H 0CH 08H 00H 40H 0040H*00F6H*0038H
LDA FFH
 0101*00H FCH 00H 0CH 08H 00H 40H 0040H 00F6H*003BH
INR A
*0000*01H FCH 00H 0CH 08H 00H 40H 0040H 00F6H*003CH
STA FFH
 0000 01H FCH 00H 0CH 08H 00H 40H 0040H 00F6H*003FH
CALL CH
REFER AT CH=DIV

D CP.

CY7SP  A   B   C   D   E   H   L    HL     SP     PC
 0000 01H FCH 00H 0CH 08H 00H 40H 0040H*00F4H*000CH


G.

 0000 01H FCH 00H 0CH 08H 00H 40H 0040H*00F6H*0042H
DCR A
*0101*00H FCH 00H 0CH 08H 00H 40H 0040H 00F6H*0043H
JZ 49H
 0101 00H FCH 00H 0CH 08H 00H 40H 0040H 00F6H*0049H
POP B
 0101 00H FCH 00H 0CH 08H 00H 40H 0040H*00F8H*004AH
MOV LB
 0101 00H FCH 00H 0CH 08H 00H*FCH*00FCH 00F8H*004BH
RET
 0101 00H FCH 00H 0CH 08H 00H FCH 00FCH*00FAH*0042H
DCR A
*0011*FFH FCH 00H 0CH 08H 00H FCH 00FCH 00FAH*0043H
JZ 49H
 0011 FFH FCH 00H 0CH 08H 00H FCH 00FCH 00FAH*0046H
POP B
 0011 FFH*75H 00H 0CH 08H 00H FCH 00FCH*00FCH*0047H
MOV HB
CYZSP  A   B   C   D   E   H   L    HL     SP     PC
 0011 FFH 75H 00H 0CH 08H*75H FCH*75FCH 00FCH*0048H
RET
HLT CYCLE 1835


D M 0F4H T 0FCH.

00F4H 42H 00H 00H FCH 42H 00H 00H 75H 0BH
```

MICROPROCESADORES: TEORIA Y APLICACIONES

TEMA: 7 y 8 : EL DESARROLLO DE UN SISTEMA.

PROF. ING. MARIO RODRIGUEZ M.

Abril,1978.

APUNTES CURSO MICROPROCESADORES.

Resumen.

Se trata detalladamente algunos de los programas que sirven de respaldo o apoyo en el desarrollo de sistemas integrados con microprocesadores, entre los cuales están los simuladores y los ensambladores.

La utilización de estos programas es cada vez más común lo que implica la necesidad (por parte tanto del usuario del sistema como del diseñador de este) de conocer su funcionamiento y la manera de implementarlo.

I. Simulador.

Un simulador es un "programa" que emula, imita o realiza las actividades de un sistema, tal y como en la práctica serían llevados al cabo por este último.

El programa simulador en lo general corre o es ejecutado en una computadora para emular a otra que puede o no existir físicamente.

Con el fin de facilitar la exposición de este tema partiremos de algunas cuestiones que surgen en la práctica y que tarde o temprano nos haremos como usuarios.

I.1) ¿Cuándo se requiere utilizar un simulador?

En la mayoría de los casos la necesidad de utilizar un simulador surge entre los requerimientos de desarrollo de un proyecto específico, por ejemplo: cuando deba evaluarse la capacidad de un micro-mini o macro sistema para resolver las necesidades de la aplicación, pudiendo una vez realizada la simulación seleccionar una u otra máquina de las propuestas como candidatos o bien verificar los requerimientos de memoria de cada sistema, longitud de código necesario, etc.

La finalidad como salta a la vista es ahorar posibles fracasos en la compra de un computador, asegurando lo más formalmente posible que aquel que resulte seleccionado resolverá la problemática de la aplicación en forma óptima. Este ahorro incluye la parte económica de la realización del proyecto.

I.2) ¿Es conveniente comprarlo o hacerlo?

El costo inherente de estos programas en el mercado es elevado, es claro que algunas veces por razones de tiempo será necesario adquirirlos pero siempre y cuando el sistema a utilizar haya sido "ya" seleccionado, (no debemos olvidar que para cada sistema candidato hay que tener un simulador); por otro lado, el realizar los programas tiene un costo que depende en gran parte de la eficiencia y conocimientos de los programadores, en la misma forma podrá decirse del tiempo de realización. Sin embargo, en muchos casos resultará más práctico.

No hay que olvidar que estos programas necesitarán ser probados, (realizados) y ejecutados en algún sistema; por ejemplo, en una mini o macro-computadora lo cual incrementa el costo de realización.

I.3) ¿Qué conocimientos se requieren para realizarlo?

Hacíamos notar que la necesidad de utilizar un simulador depende en la mayoría de los casos de la aplicación y ésta nos dará un punto de partida para evaluar cuales sistemas podrían satisfacer nuestras necesidades. Por esta razón, se proponen los siguientes pasos a seguir:

I.3.1) Conocimiento de nuestra aplicación.

I.3.2) Conocimiento de la arquitectura de los sistemas propuestos.

En cuanto a:

a) Número de instrucciones.

b) Número de registros.

c) Número de banderas.

d) Número de puertos.

e) Funcionamiento del program counter, unidad aritmética y stack.

I.3.3) Selección del sistema más apropiado.

I.3.4) Selección de la máquina en donde se va a simular el sistema escogido.

I.3.5) Selección del lenguaje de programación.

I.3.6) Estructura del programa simulador.

I.3.7) Determinar las variables y constantes del sistema por simular.

I.3.8) Elaborar un cargador.

I.3.9) Programar.

I.3.10) Probar.

I.3.11) Desarrollar el programa de nuestra aplicación.

I.3.12) Simularlo.

I.3.13) Adquirir o deshechar el sistema propuesto.

I.3.14) Implantar el programa simulado de nuestra aplicación en el sistema físico adquirido.

I.4) Desarrollo.

El conocer la arquitectura de varios sistemas (I.3.2) es cosa sencilla si se tienen los elementos necesarios para poder evaluarlos; ej.: manuales, diagramas, etc. Realmente son pocas las características a comparar entre sistemas; por ejemplo, sus costos, velocidad, número de polarizaciones, sistema mínimo, facilidades de entrada/salida, etc. Sin embargo, para la simulación estos datos no son de interés, bastará con conocer los elementos internos del mismo (I.3.2) pudiendo de aquí seleccionar el más apropiado (I.3.3) en cuanto a facilidades. Por otro lado, será de gran importancia conocer el número de instrucciones y formatos de éstas, la longitud de palabra y el manejo del program counter y el stack pointer si lo tiene. Además de comprender lo más completamente posible la interacción entre registros, banderas y unidades del sistema.

Otro tipo de evaluación (I.3.4) debe realizarse al escoger la máquina en donde se llevará a cabo la programación del programa simulador.

Con el fin de referirnos a un caso real, plantearemos el siguiente ejemplo.

En el Instituto de Investigación en Matemáticas Aplicadas y en Sistemas, nos proponemos a realizar un simulador para un sistema 8080 de Intel. Podemos utilizar el Sistema B6700 o un sistema PDP 11/10. Teniendo ambos sistemas a nuestra disposisición, deberemos escoger aquel en que nos cueste menos trabajo realizar el programa tomando en consideración para esto los lenguajes de programación de cada sistema, los pasos necesarios para escribirlo y verificarlo y el tipo de salidas de este.

El sistema PDP 11/10 nos brinda la posibilidad de (I.3.5) utilizar FORTRAN o ENSAMBLADOR, el sistema B6700 incluye ALGOL, FORTRAN, COBOL, BASIC, etc. dentro de sus lenguajes. La cantidad de memoria es suficiente en ambos sistemas. Entonces, ¿cuál deberemos utilizar? En este caso especial simplemente escogeremos el que mejor nos acomode o conzcamos. Sin embargo, la cantidad de memoria y las facilidades del computador a usar pueden ser determinantes en esta selección. Por ejemplo: El sistema B6700 será el escogido por tener el superlenguaje ALGOL, ya que este me permite realizar la simulación en forma sencilla sobre todo debido a su facilidad de acceso a cada bit de la palabra de B6700 independientemente, cosa que, por otro lado, en

FORTRAN no existe.

Entrando en materia (I.3.6) sabemos que la longitud de palabra de un microsistema es actualmente de 8 ó 16 bits será necesario entonces para facilitar la simulación escoger un sistema que tenga una palabra igual o múltiplo de esta longitud. El sistema de B6700 tiene 48 bits al servicio del usuario; es decir, para un microsistema de 8 bits en tan solo 1 palabra B6700 cabrían 6 instrucciones de formato unitario (instrucciones de 8 bits).

La estructura del Programa Simulador es bastante sencilla si conocemos la forma como cada instrucción es adquirida de memoria y llevada a la unidad de control, decodificada y ejecutada por el sistema a simular.

Esto es el ciclo de fetch y el ciclo de ejecución nos indicarán como debe funcionar el programa simulador y de hecho las rutinas, procedures, o subrutinas de las cuales debe constar. Además el manejo del apuntador de Programa o program counter decidirá paso a paso que instrucción será la siguiente a ejecutar.

Con los conocimientos anteriores será sencillo determinar cuales serán las variables de nuestro programa y cual el tipo de estas. Por ejemplo: En el sistema 8080 se tienen los registros RA, RB, RC, RD, RE, RH, RL, PC y SP, pudiendo referirnos a ellos por pares:

$$\begin{tikzcd} \mathrm{RBX}, \arrow[d] & \mathrm{RDX}, \arrow[d] & \mathrm{RHX}, \arrow[d] & \mathrm{PSW} \arrow[d] \\ (\mathrm{RB}\ \&\ \mathrm{RC}) & (\mathrm{RD}\ \&\ \mathrm{RE}) & (\mathrm{RH}\ \&\ \mathrm{RL}) & (\mathrm{RA}\ \ \mathrm{BAN}) \end{tikzcd}$$

siendo este último la concatenación de RA y las Banderas BZ, BP, BS y BV

OVERFLOW ZERO PARIDAD SIGNO

y una memoria constituída por RAM y ROM de una cierta capacidad; por ejemplo, 16 K.

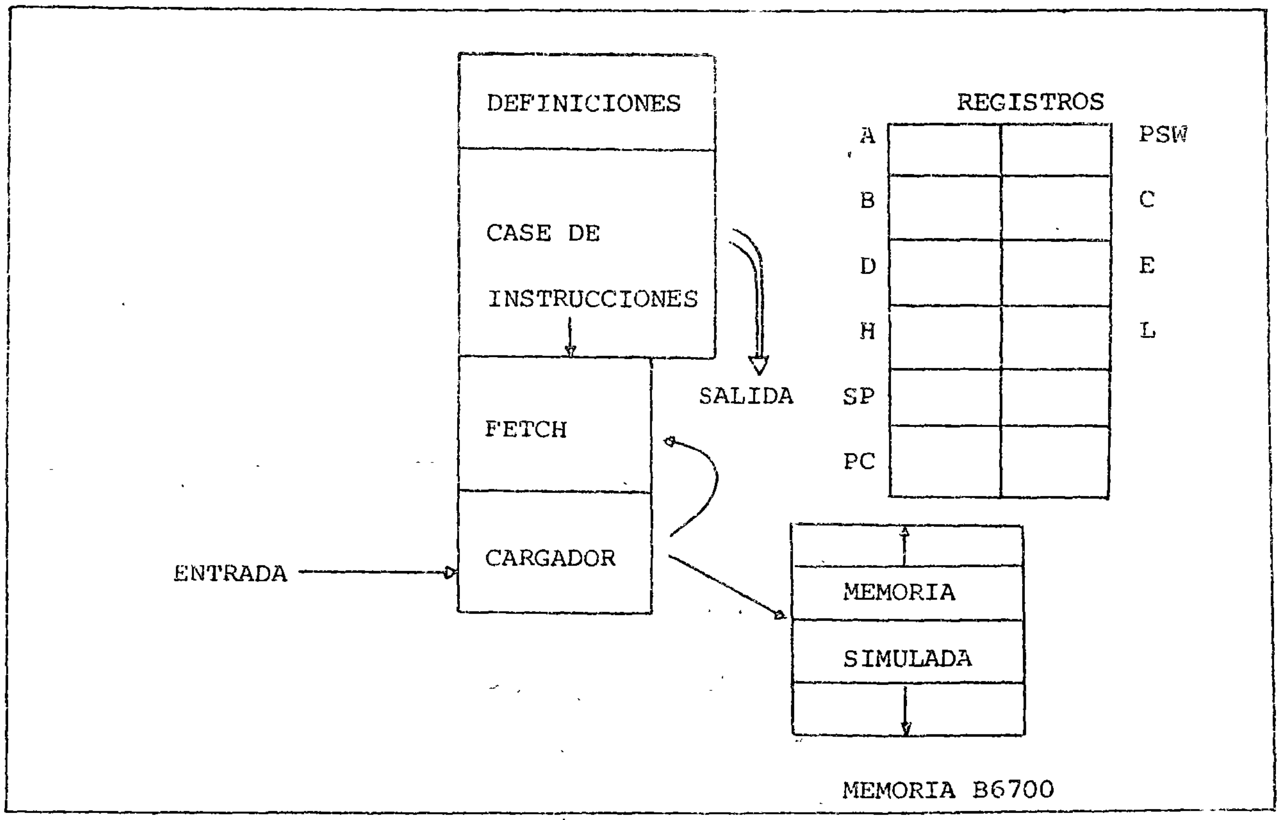
Teniendo en cuenta lo anterior, el programa comenzaría definiendo:

```
BEGIN
ARRAY M [0:2730] ;
ARRAY REG [0:7];
INTEGER BAN, SP, PC, INSTR ; % INSTRUCCION
DEFINE CAMPO = [7:8] #,
       RA = REG [7] . #,
       RB = REG [0] . #,
       RC = REG [1] . #,
       RD = REG [2] . #,
       RE = REG [3] . #,
       RH = REG [4] . #,
       RL = REG [5] . #,
       BZ = BAN . [6:1] #,
       BP = BAN . [5:1]#,
       B5 = BAN . [4:1]#,
       BV = BAN . [3:1]#,
       PCT = PC. [15:16]#,
       SPT = SP. [15:16]#,
RHX = RH & RL#,

RDX = RD & RE#,

RBX = RB & RCH,

PSW = RA & BAN#;
      .
      .
      .
MEM(X) = M [(X) DIV 6].[47-(8 * ((X) MOD 6)): 8] #;
```

En donde esta última definición corresponde a la manera como quedará simulada la memoria, es decir:

M [0]	Byt 0	Byt 1	Byt 2	Byt 3	Byt 4	Byt 5
M [1]	Byt 6	Byt 7				
:						

La estructura del programa es la siguiente:

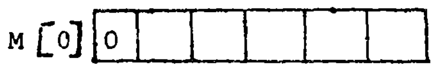


Dada esta estructura, analizaremos detalladamente cada una de sus partes:

El cargador es un procedure o subrutina que permite al usuario introducir en binario las instrucciones del programa a simular para ser colocadas en la memoria simulada.

Cuando el usuario desee introducir la instrucción 00101111 independientemente de lo que ésta represente, él deberá escribir en primer lugar un cero, a continuación otro, luego un uno, luego un cero, etc...hasta concluir.

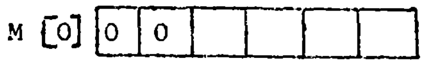
Al introducir el primer cero la terminal envía el código ASCII correspondiente al cero siendo este: 00110000. El cargador recibe este carácter y lo enmascara para leer únicamente el valor del bit menos significativo; es decir, el último de la derecha, el cual corresponde al cero deseado. El cargador ahora toma este valor y lo introduce como uno de los bits de la instrucción como se muestra en la siguiente figura:



bit introducido

Palabra B6700

A continuación el usuario introduce el segundo cero sucediendo algo análogo a lo anterior con la diferencia que el bit ocupado es ahora el siguiente:



En el caso del uno siguiente el código recibido corresponde a 0011 0001.
↑

Realizándose la misma operación finalmente se terminará de introducir la instrucción, quedando:

M [0]										
	0	0	1	0	1	1	1			

En esta forma, el cargador introducirá el programa del usuario a la memoria simulada deteniéndose hasta que el usuario se lo indique con algún crácter de control.

A continuación se forza el 1 ciclo de Fetch apuntando la variable PC a la dirección origen del programa; es decir, a la 1er instrucción del programa del usuario introducida y ésta se introduce en la variable INSTR. 7:8 , es decir:

INSTR.[7:8] = MEM (PC);

Al actualizar la instrucción pararemos a decodificarla; es decir, realizaremos la misma actividad que haría la unidad de control del sistema. Con este fin, se usa un CASE de instrucciones.

El CASE de instrucciones es un decodificador, el cual direcciona a la rutina necesaria dependiendo de los campos de la instrucción, por ejemplo:

Supongamos la instrucción MOV A, B para el contenido del registro B al registro A. Su código sería:

$$\underbrace{0\ 1}_{\text{código de operación}}\ \underbrace{1\ 1\ 1}_{\text{registro A}}\ \underbrace{0\ 0\ 0}_{\text{registro B}}$$

Observando macroscópicamente el programa simulador, se tendría:

```
CASE INST.[7:2] OF
    BEGIN
        00:
        01: R [INST. [5:3]] : = R [INST. [2:3]] ;
        10:
        11:
END;
```

Lo que equivale a RA: = RB;

En forma análoga cada instrucción estará caracterizada por su código de operación y estos códigos formarán una familia fácilmente distinguible.

La simulación en este caso se realiza a nivel instrucción, habrá sin embargo simuladores que sean a nivel circuito o compuertas o a nivel unidades.

I.5. Salidas del Simulador.

Un simulador puede incluir la emulación de puertos de entrada/salida para operaciones como IN ó OUT del sistema 8080; para realizarlas deberán usarse las declaraciones de Read y Write del lenguaje escogido pudiendo de esta manera tener varios puertos simulados.

Habiendo analizado el diseño del simulador, podremos plantear fácilmente cuales serían los requerimientos de salida o bien que tipo de resultados debe entregarnos. Si lo que nos interesa es observar el desarrollo del programa simulado a nivel registros/banderas, el programa a cada cambio de una variable deberá escribir el status del sistema indicando cual o cuales registros/banderas han sido transformados. De ma misma manera, lo anterior será válido, para PC y memoria. Otras de las posibles salidas de interés puede referirse al tiempo gastado por cada instrucción dando la suma en microsegundos, a partir de los ciclos de cada máquina para cada una de ellas.

I.6, Interacción del usuario.

La interacción del usuario con el simulador es de gran importancia, ya que mientras más sencilla sea, el usuario podrá efectuar cualquiera de sus requerimientos. Por ejemplo:

a) Cambiar contenidos de registros/banderas/memoria.

b) Desplegar el status del sistema.

c) Utilizar diversas bases numéricas para marcar sus datos.

d) Producir interrupciones por ciclos.

e) Permitir o inhibir interrupciones.

g) Intercalar instrucciones.

h) Conocer el tiempo de proceso.

i) Desplegar contenidos de memoria programas o datos.

j) Referirse a etiquetas o variables por su nombre simbólico.

Una vez concluído el simulador salta a la vista que el simular un programa utilizando únicamente el cargador presenta los siguientes problemas:

a) El usuario no está libre de equivocar un 0 por un 1 ó un 1 por un 0; si el programa es largo la probabilidad de error es grande.

b) Resulta muy tardada la programación en esta forma.

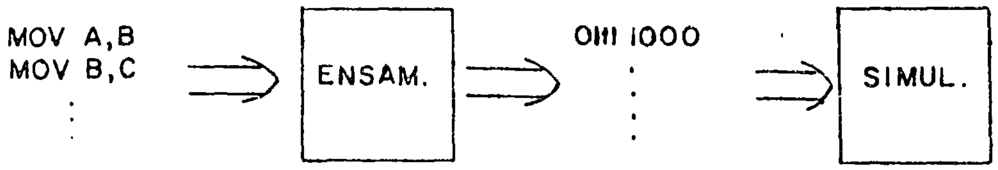
c) Es difícil verificar errores.

Por estas razones surge la necesidad de emplear un programa ensamblador.

II. Ensamblador.

El programa ensamblador correrá a su vez en la misma máquina B6700, teniendo como finalidad principal producir a partir de los mnemónicos de las instrucciones, el código correspondiente a estas, facilitando al mismo tiempo la decoeificación de etiquetas.

En la figura siguiente se muestra la interacción entre simulador y ensamblador:



III. El Simulador INTERP/80.

Dentro de los programas de auxiliares en el desarrollo de programas de aplicación para el sistema 8080 de Intel, se cuenta con el simulador INTERP/80. Durante esta plática se tratará el programa para el Sistema 8080.

El desarrollo de un programa de aplicación se realiza generalmente utilizando un programa ensamblador o un compilador. En el 1er caso, los mnemónicos de cada instrucción son el lenguaje de programación.

Los requerimientos para la elaboración de un programa son los siguientes:

a) Conocer los mnemónicos del sistema/8080.

b) Realizar un diagrama de flujo del programa.

c) Codificarlo en papel.

d) Utilizando el editor del sistema (en nuestro caso cande de B6700). Introducir a un archivo los mnemónicos generados.

e) Ejecutar el ensamblador del lenguaje (en nuestro caso Assembler/8080).

f) Generar un archivo con el código de máquina ensamblado.

g) Ejecutar el simulador.

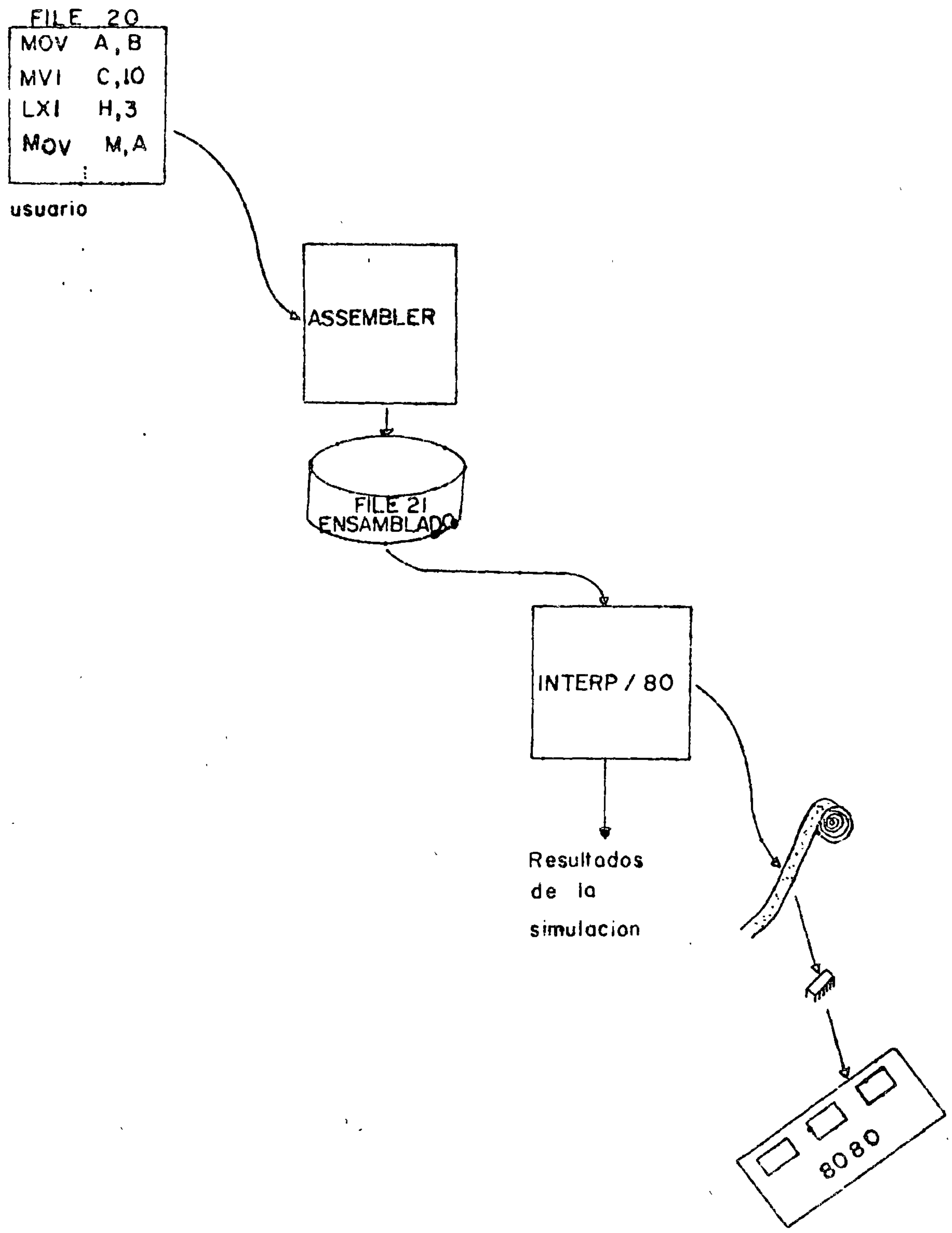
h) Monitorear el programa.

i) Generar del programa correcto una cinta con el código de máquina.

j) Introducir este al ROM en donde quedará residente.

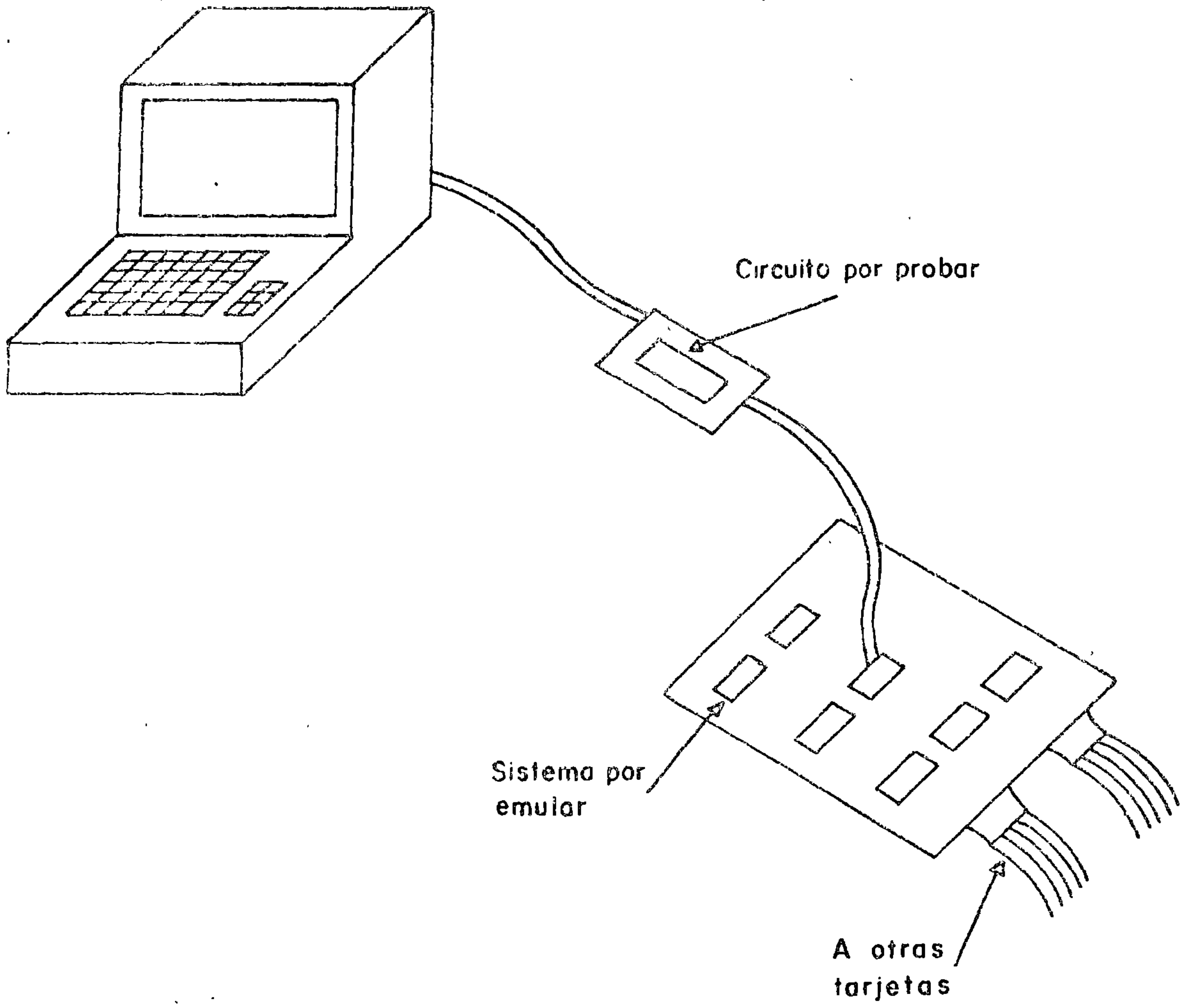
k) Colocarlo en el Sistema /8080.

Descripción del proceso:



En la siguiente figura se contempla una sesión de cande (command & edit) del sistema B6700, especificando paso a paso los mensajes que durante la práctica se deberán generar. Con → se indica lo que el sistema genera. Con * lo que el usuario introduce siempre terminado por ↓ carry return.

A continuación se indican los posibles controles del simulador y del ensamblador:



En la siguiente sección incluiremos una sesión completa de trabajo con el INTERP/80. ←

Paso A.

Coloque su clave en el teletipo en la siguiente forma:

```
                    carry return
* MR85/BOMBON ↲

→ SESSION # 4080:  YOU ARE TTY#

* MAKE MI/PROGRAMA DATA : SEQ ↓

→ WORKFILE MI/PROGRAMA

→ 100 * MOV A, B ↓

→ 200 * MVI A, 3 ↓

→ 300 * LXI H, 10 ↓

→ 400 * END ↓

→ 500 ↓

→ #

* SAVE ↓

→ WORKFILE MI/PROGRAMA SAVED
```

PROGRAMA DE EJEMPLO .... DIVISION ...... RECURSIVIDAD

```
                  RCRSV   EQU 0FFH
                  LXI     SP,0FEH
                  XRA     A
                  CMA
                  STA     RCRSV
                  CALL    DIV
                  HLT
DIV      :        ORA     A
                  RPO
                  JP      ETC
                  MOV     H,B
                  MOV     L,C
                  ORA     A
                  MOV     A,D
                  RAR
                  MOV     D,A
                  MOV     A,E
                  RAR
                  MOV     E,A
ETC      :        XRA     A
                  MOV     B,A
                  MVI     C,8
COMP     :        MOV     A,L
                  CMP     E
                  MOV     A,H
                  SBB     D
                  JC      SHLFT
                  MOV     H,A
                  MOV     A,L
                  SUB     E
                  MOV     L,A
                  DAD     H
                  STC
                  JMP     RESUL
SHLFT    :        DAD     H
                  ORA     A ;RSTC
RESUL    :        MOV     A,B
                  RAL
                  MOV     B,A
                  DCR     C
                  JNZ     COMP
#
```

HOJA 2   CONTINUA .....

```
RECUR :  PUSH   B
         LDA    RCRSV
         INR    A
         STA    RCRSV
         CALL   DIV
         DCR    A
         JZ     L2
L1    :  POP    B
         MOV    H,B
         RET
L2    :  POP    B
         MOV    L,B
         RET
         END
         END
#
```

PARA ENSAMBLAR EL PROGRAMA DE EJEMPLO ES NECESARIO INTRODUCIR LA SIGUIENTE INSTRUCCION :

```
E ASSEMBLER/8080 ; FILE FILE20(MAXRECSIZE=14,BLOCKSIZE=420,TITLE=DIV/NEW),FILE21(TITLE=MIO)
```

% A LO CUAL LA COMPUTADORA CONTESTA :

```
#RUNNING 9481
#?

8080 MACRO ASSEMBLER, VER 2.0
```

% ASI TENEMOS YA CONTROL SOBRE EL SISTEMA Y BASTARA INTRODUCIR LOS SIGUIENTES
% CONTROLES

```
$I=2 $T=1 $M=1
```

% ALO QUE EL SISTEMA RESPONDE

```
BEGIN
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 1

 0OFF                          RCRSV   EQU OFFH
 0000    31FE00                LXI     SP,OFEH
 0003    AF                    XRA     A
 0004    2F                    CMA
 0005    32FF00                STA     RCRSV
 0008    CD0C00                CALL    DIV
 000B    76                    HLT
 000C    B7          DIV    :  ORA     A
 000D    E0                    RPO
 000E    F21A00                JP      ETC
 0011    60                    MOV     H,B
 0012    69                    MOV     L,C
 0013    B7                    ORA     A
 0014    7A                    MOV     A,D
 0015    1F                    RAR
 0016    57                    MOV     D,A
 0017    7B                    MOV     A,E
 0018    1F                    RAR
 0019    5F                    MOV     E,A
 001A    AF          ETC    :  XRA     A
 001B    47                    MOV     B,A
 001C    0E08                  MVI     C,8
 001E    7D          COMP   :  MOV     A,L
 001F    BB                    CMP     E
 0020    7C                    MOV     A,H
 0021    9A                    SBB     D
 0022    DA2E00                JC      SHLFT
 0025    67                    MOV     H,A
 0026    7D                    MOV     A,L
 0027    93                    SUB     E
 0028    6F                    MOV     L,A
 0029    29                    DAD     H
 002A    37                    STC
 002B    C33000                JMP     RESUL
 002E    29          SHLFT  :  DAD     H
 002F    B7                    ORA     A ;RSTC
 0030    78          RESUL  :  MOV     A,B
 0031    17                    RAL
 0032    47                    MOV     B,A
 0033    0D                    DCR     C
 0034    C21E00                JNZ     COMP
 0037    C5          RECUR  :  PUSH    B
 0038    3AFF00                LDA     RCRSV
 003B    3C                    INR     A
 003C    32FF00                STA     RCRSV
 003F    CD0C00                CALL    DIV
 0042    3D                    DCR     A
 0043    CA4900                JZ      L2
 0046    C1          L1     :  POP     B
 0047    60                    MOV     H,B
 0048    C9                    RET
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 2


  0049    C1                   L2     :  POP     B
  004A    68                             MOV     L,B
  004B    C9                             RET
                                         END
NO PROGRAM ERRORS
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 3


                               SYMBOL TABLE

 * 01

 A         0007       B        0000       C        0001       COMP    001E
 D         0002       DIV      000C       E        0003       ETC     001A
 H         0004       L        0005       L1       0046 *     L2      0049
 M         0006       PSW      0006       RCRSV    00FF       RECUR   0037 *
 RESUL     0030       SHIFT    002E       SP       0006


#ET-4:09.0 PT=13.3 IO=1.5
```

```
%A CONTINUACION SIMULAMOS EL PROGRAMA USANDO EL INTERP/80

E INTERP/80 ; FILE FILE21(TITLE=MID)
#RUNNING 9613


$?

INTERP/80 VERS 1.0



%A CONTINUACION INTRODUCIMOS EL CONTROL DE CARGA $F Y LOAD 7 7.


$F=1

LOAD 7 7.


%A LO QUE EL SISTEMA RESPONDE


76 LOAD OK



%A CONTINUACTON  VEAMOS QUE HAY EN MEMORIA DEL SIMULADOR

D M 0 TO 76.

%Y EL SISTEMA NOS MUSTRA  LOS CONTENIDOS



00000 049 254 000 175 047 050 255 000 205 012 000 118 183 224 242 026
00016 000 096 105 183 122 031 087 123 031 095 175 071 014 008 125 187
00032 124 154 218 046 000 103 125 147 111 041 055 195 048 000 041 183
00048 120 023 071 013 194 030 000 197 058 255 000 060 050 255 000 205
00064 012 000 061 202 073 000 193 096 201 193 104 201 000
```

%TAMBIEN SERIA CONVENIENTE MOSTRAR EL MNEMONICO DE CADA INSTRUCCION
%DE NUESTRO PROGRAMA , CON ESTE FIN USAMOS :

```
D M O TO 76 C.

00000 LXI SP    254       000       XRA A     CMA       STA       255
00007 000       CALL      012       000       HLT       ORA A     RPO
00014 JP        026       000       MOV HB    MOV LC    ORA A     MOV AD
00021 RAR       MOV DA    MOV AE    RAR       MOV EA    XRA A     MOV BA
00028 MVI C     008       MOV AL    CMP E     MOV AH    SBB D     JC
00035 046       000       MOV HA    MOV AL    SUB E     MOV LA    DAD H
00042 STC       JMP       048       000       DAD H     ORA A     MOV AB
00049 RAL       MOV BA    DCR C     JNZ       030       000       PUSH B
00056 LDA       255       000       INR A     STA       255       000
00063 CALL      012       000       DCR A     JZ        073       000
00070 POP B     MOV HB    RET       POP B     MOV LB    RET       NOP
```

%O BIEN EN HEXADECIMAL TENEMOS :

```
D M O TO 76 H.

00000 31H FFH 00H AFH 2FH 32H FFH 00H CDH 0CH 00H 76H B7H E0H F2H 1AH
00016 00H 60H 69H B7H 7AH 1FH 57H 7BH 1FH 5FH AFH 47H 0EH 08H 7DH BBH
00032 7CH 9AH DAH 2EH 00H 67H 7DH 93H 6FH 29H 37H C3H 30H 00H 29H B7H
00048 78H 17H 47H 0DH C2H 1EH 00H C5H 3AH FFH 00H 3CH 32H FFH 00H CDH
00064 0CH 00H 3DH CAH 49H 00H C1H 60H C9H C1H 68H C9H 00H
```

%POR OTRO LADO QUE HAY EN LOS REGISTROS DEL SISTEMA ?

```
D CPU.

CYZSP   A    B    C    D    E    H    L     HL     SP     PC
*0000*000*000*000*000*000*000*000*00000*00000*00000
```

%AUN NADA YA QUE TODAVIA NO INICIAMOS LA CORRIDA DEL PROGRAMA, PARA
%VERLO FUNCIONAR PODEMOS PEDIR UN TRACE .

```
TRACE 0 TO 76 .


%A LO QUE EL SISTEMA RESPONDE


TRACE OK


%PIDAMOS 5 CICLOS Y OBSERVEMOS : CUANDO LE INDICAMOS QUE COMIENZE:


CY 5.G.

CYCLE OK
 0000 000 000 000 000 000 000 000 00000 00000 00000
LXI SP 254
 0000 000 000 000 000 000 000 000 00000*00254*00003
XRA A
*0101 000 000 000 000 000 000 000 00000 00254*00004
CMA
 0101*255 000 000 000 000 000 000 00000 00254*00005
STA 255
 0101 255 000 000 000 000 000 000 00000 00254*00008
CALL 12
CYCLE AT 12-DTV


%CON LO ANTERIOR HEMOS COMENZADO HA REALIZAR LA SIMULACION Y
%EN LA MISMA FORMA PODREMOS SEGUIR HASTA EL FINAL .
%MODIFIQUEMOS PRIMERAMENTE EL CONTENIDO DE ALGUNOS REGISTROS
%POR DECIR ALGO QUE  H&L = 258 Y  D&E = 772.


S H=1 , S L = 2. S D = 3. S E = 4.

SET OK
SET OK
SET OK
```

Emulador.

Un emulador a diferencia de un simulador consiste de una pieza de hardware y de un programa de aplicación. Actualmente, el uso de emuladores se ha diversificado y ya se encuentran en el mercado emuladores comerciales.

Funcionamiento.

Como indicamos, una pieza de hardware forma al sistema. Esta por lo regular consiste de un socket que reemplaza la pieza a emular, la finalidad de la emulación puede ser:

a) La verificación del funcionamiento de un sistema ya armado.

b) La verificación o chequeo de uno o más integrados.

c) La verificación de un programa de aplicación.

En nuestro caso el emulador de un programa se conecta de la siguiente manera:

En él podremos en línea observar la actividad del circuito paso a paso o bien contar las instrucciones que han transcurrido o verificar su código.

El modelo con el que se trabajará en clase "no" brinda todas las facilidades que deberían esperarse de estos sistemas, ya que algunos como el "LABISIS" permiten modificar el programa del usuario al encontrarse alguna falla o tambien leer la memoria y los contenidos de registros, banderas, etc.

¿Es conveniente contar con estos sistemas?

Si se tiene un programa simulador puede rastrearse el programa de aplicación; por otro lado, ya en el modelo físico de desarrollo puede habernos fallado alguna conexión no fácilmente detectable, o bien, ya trabajando en el campo del tiempo real puede haber alcances o retardos indeseables que en el simulador son casi imposibles de preveer. Sin embargo, el costo de estos sistemas es elevado y se deberá tener realmente necesidad de ellos para adquirirlos.

La práctica a realizar en el curso consiste en emular el circuito integrado CPU 8080 perteneciente al sistema SDK, con el programa monitor en su memoria ROM.

¿Qué actividad realiza la parte software de un emulador?

Podemos imaginarnos por un momento un sistema emulador, que tipo de circuitería interna tiene, si es capaz de emular a un microprocesador? Pues sencillamente tiene otro microprocesador, el cual interactuando con un programa de aplicación maneja los pins del circuito emulado y comanda sus acciones o actividades paso a paso, teniéndose gracias a esto la posibilidad de detección de fallas, etc.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES : TEORIA Y APLICACIONES

MCS-80 SYSTEM DESING KIT
USER/S GUIDE
APPENDIX A. MONITOR LISTING.

ABRIL DE 1978.

APPENDIX A.   MONITOR LISTING

```
8080 MACRO ASSEMBLER, VER 2.3  ERRORS = 0 PAGE 1

                    ;*****************************************************************
                    ;
                    ;                 PROGRAM: 8080A BOARD MONITOR
                    ;
                    ;                 COPYRIGHT (C) 1975
                    ;                 INTEL CORPORATION
                    ;                 3065 BOWERS AVENUE
                    ;                 SANTA CLARA, CALIFORNIA  95051
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ; ABSTRACT
                    ; ========
                    ;
                    ; THIS PROGRAM RUNS ON THE 8080A BOARD AND IS DESIGNED TO PROVIDE
                    ; THE USER WITH A MINIMAL MONITOR.  BY USING THIS PROGRAM,
                    ; THE USER CAN EXAMINE AND CHANGE MEMORY OR CPU REGISTERS, LOAD
                    ; A PROGRAM (IN ABSOLUTE HEX) INTO RAM, AND EXECUTE INSTRUCTIONS
                    ; ALREADY IN MEMORY.  THE MONITOR ALSO PROVIDES THE USER WITH
                    ; ROUTINES FOR PERFORMING CONSOLE I/O.
                    ;
                    ;
                    ; PROGRAM ORGANIZATION
                    ; ======= ============
                    ;
                    ; THE LISTING IS ORGANIZED IN THE FOLLOWING WAY.  FIRST THE COMMAND
                    ; RECOGNIZER, WHICH IS THE HIGHEST LEVEL ROUTINE IN THE PROGRAM.
                    ; NEXT THE ROUTINES TO IMPLEMENT THE VARIOUS COMMANDS.  FINALLY,
                    ; THE UTILITY ROUTINES WHICH ACTUALLY DO THE DIRTY WORK.  WITHIN
                    ; EACH SECTION, THE ROUTINES ARE ORGANIZED IN ALPHABETICAL
                    ; ORDER, BY ENTRY POINT OF THE ROUTINE.
                    ;
                    ; THIS PROGRAM EXPECTS TO RUN IN THE FIRST 1K OF ADDRESS SPACE.
                    ; IF, FOR SOME REASON, THE PROGRAM IS RE-ORG'ED, CARE SHOULD
                    ; BE TAKEN TO MAKE SURE THAT THE TRANSFER INSTRUCTIONS FOR RST 1
                    ; AND RST 7 ARE ADJUSTED APPROPRIATELY.
                    ;
                    ; THE PROGRAM ALSO EXPECTS THAT RAM LOCATIONS 5K-1 TO 5K-256,
                    ; INCLUSIVE, ARE RESERVED FOR THE PROGRAM'S OWN USE.  THESE
                    ; LOCATIONS MAY BE ALTERED, HOWEVER, BY CHANGING THE EQU'ED
                    ; SYMBOL "DATA" AS DESIRED.
                    ;
                    ; LIST OF FUNCTIONS
                    ; ==== == =========
                    ;
                    ;     GETCM
                    ;     -----
                    ;
                    ;     DCMD
                    ;     GCMD
```

```
                        ;       ICMD
                        ;       MCMD
                        ;       SCMD
                        ;       XCMD
                        ;       -----
                        ;
                        ;       BREAK
                        ;       CI
                        ;       CNVBN
                        ;       CO
                        ;       CROUT
                        ;       ECHO
                        ;       ERROR
                        ;       FRET
                        ;       GETCH
                        ;       GETHX
                        ;       GETNM
                        ;       HILO
                        ;       NMOUT
                        ;       PRVAL
                        ;       REGDS
                        ;       RGADR
                        ;       RSTTF
                        ;       SRET
                        ;       STHF0
                        ;       STHLF
                        ;       VALDG
                        ;       VALDL
                        ;       -----
                        ;
 0000                           ORG     0H
                        ;
                        ;*******************************************************************
                        ;
                        ;
                        ;                               MONITOR EQUATES
                        ;
                        ;
                        ;*******************************************************************
                        ;
                        ;
 001B                   BRCHR EQU       1BH     ; CODE FOR BREAK CHARACTER (ESCAPE)
 13FD                   BRLOC EQU       13FDH   ; LOCATION OF USER BRANCH INSTRUCTION IN RAM
 03FA                   BRTAB EQU       3FAH    ; LOCATION OF START OF BRANCH TABLE IN ROM
 0027                   CMD   EQU       027H    ; COMMAND INSTRUCTION FOR USART INITIALIZATION
 00FB                   CNCTL EQU       0FBH    ; CONSOLE (USART) CONTROL PORT
 00FA                   CNIN  EQU       0FAH    ; CONSOLE INPUT PORT
 00FA                   CNOUT EQU       0FAH    ; CONSOLE OUTPUT PORT
 00FB                   CONST EQU       0FBH    ; CONSOLE STATUS INPUT PORT
 000D                   CR    EQU       0DH     ; CODE FOR CARRIAGE RETURN
 1300                   DATA  EQU       5*1024-256      ; START OF MONITOR RAM USAGE
```

```
001B            ESC   EQU     1BH      ; CODE FOR ESCAPE CHARACTER
000F            HCHAR EQU     0FH      ; MASK TO SELECT LOWER HEX CHAR FROM BYTE
00FF            INVRT EQU     0FFH     ; MASK TO INVERT HALF BYTE FLAG
000A            LF    EQU     0AH      ; CODE FOR LINE FEED
0000            LOWER EQU     0        ; DENOTES LOWER HALF OF BYTE IN ICMD
                ;LSGNON       EQU     ---      ; LENGTH OF SIGNON MESSAGE - DEFINED LATER
00CF            MODE  EQU     0CFH     ; MODE SET FOR USART INITIALIZATION
                ;MSTAK        EQU     ---      ; START OF MONITOR STACK - DEFINED LATER
                ;NCMDS        EQU     ---      ; NUMBER OF VALID COMMANDS
000F            NEWLN EQU     0FH      ; MASK FOR CHECKING MEMORY ADDR DISPLAY
007F            PRTY0 EQU     07FH     ; MASK TO CLEAR PARITY BIT FROM CONSOLE CHAR
13ED            REGS  EQU     DATA+255-18      ; START OF REGISTER SAVE AREA
0002            RBR   EQU     2        ; MASK TO TEST RECEIVER STATUS
0038            RSTU  EQU     38H      ; TRANSFER LOCATION FOR RST 7 INSTRUCTION
                ;RTABS        EQU     ---      ; SIZE OF ENTRY IN RTAB TABLE
001B            TERM  EQU     1BH      ; CODE FOR ICMD TERMINATING CHARACTER (ESCAPE)
0001            TRDY  EQU     1        ; MASK TO TEST TRANSMITTER STATUS
00FF            UPPER EQU     0FFH     ; DENOTES UPPER HALF OF BYTE IN ICMD
                ;
                ;
                ;*****************************************************************
                ;
                ;
                ;                      MONITOR MACROS
                ;
                ;
                ;*****************************************************************
                ;
                ;
     1          TRUE  MACRO   WHERE    ; BRANCH IF FUNCTION RETURNS TRUE (SUCCESS)
     1                JC      WHERE
                      ENDM
                ;
     1          FALSE MACRO   WHERE    ; BRANCH IF FUNCTION RETURNS FALSE (FAILURE)
     1                JNC     WHERE
                      ENDM
                ;
                ;
                ;*****************************************************************
                ;
                ;
                ;                      USART INITIALIZATION CODE
                ;
                ;
                ;*****************************************************************
                ;
                ;
                ;    THE USART IS ASSUMED TO COME UP IN THE RESET POSITION (THIS
                ;    FUNCTION IS TAKEN CARE OF BY THE HARDWARE).  THE USART WILL
                ;    BE INITIALIZED IN THE SAME WAY FOR EITHER A TTY OR CRT
                ;    INTERFACE.  THE FOLLOWING PARAMETERS ARE USED:
                        ;
                        ;       MODE INSTRUCTION
                        ;       ==== ===========
                        ;
                        ;       2 STOP BITS
                        ;       PARITY DISABLED
                        ;       8 BIT CHARACTERS
                        ;       BAUD RATE FACTOR OF 64
                        ;
                        ;       COMMAND INSTRUCTION
                        ;       ======= ===========
                        ;
                        ;       NO HUNT MODE
                        ;       NOT(RTS) FORCED TO 0
                        ;       RECEIVE ENABLED
                        ;       DATA TERMINAL READY
                        ;       TRANSMIT ENABLED
                        ;
0000    3ECF                    MVI     A,MODE
0002    D3FB                    OUT     CNCTL   ; OUTPUT MODE SET TO USART
0004    3E27                    MVI     A,CMD
0006    D3FB                    OUT     CNCTL   ; OUTPUT COMMAND WORD TO USART
                        ;
                        ;*****************************************************************
                        ;
                        ;
                        ;                       RESTART ENTRY POINT
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        ;
0008                    GO:
0008    22F313                  SHLD    LSAVE   ; SAVE HL REGISTERSS
000B    E1                      POP     H       ; GET TOP OF STACK ENTRY
000C    22F513                  SHLD    PSAVE   ; ASSUME THIS IS LAST P COUNTER
000F    210000                  LXI     H,0     ; CLEAR HL
0012    39                      DAD     SP      ; GET STACK POINTER VALUE
0013    22F713                  SHLD    SSAVE   ; SAVE USER'S STACK POINTER
0016    21F313                  LXI     H,ASAVE+1       ; NEW VALUE FOR STACK POINTER
0019    F9                      SPHL            ; SET MONITOR STACK POINTER FOR REG SAVE
001A    F5                      PUSH    PSW     ; SAVE A AND FLAGS
001B    C5                      PUSH    B       ; SAVE B AND C
001C    D5                      PUSH    D       ; SAVE D AND E
                        ;
                        ;************************************************************
                        ;
                        ;
                        ;                       PRINT SIGNON MESSAGE
                        ;
                        ;
```

```
                  ;*****************************************************************
                  ;
                  ;
001D    219D03          LXI     H,SGNON ; GET ADDRESS OF SIGNON MESSAGE
0020    060E            MVI     B,LSGNON        ; COUNTER FOR CHARACTERS IN MESSAGE
0022              MSGL:
0022    4E              MOV     C,M     ; FETCH NEXT CHAR TO C REG
0023    CDE301          CALL    CO      ; SEND IT TO THE CONSOLE
0026    23              INX     H       ; POINT TO NEXT CHARACTER
0027    05              DCR     B       ; DECREMENT BYTE COUNTER
0028    C22200          JNZ     MSGL    ; RETURN FOR NEXT CHARACTER
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ;
                  ;                   COMMAND RECOGNIZING ROUTINE
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ; FUNCTION: GETCM
                  ; INPUTS: NONE
                  ; OUTPUTS: NONE
                  ; CALLS: GETCH,ECHO,ERROR
                  ; DESTROYS: A,B,C,H,L,F/F'S
                  ; DESCRIPTION: GETCM RECEIVES AN INPUT CHARACTER FROM THE USER
                  ;              AND ATTEMPTS TO LOCATE THIS CHARACTER IN ITS COMMAND
                  ;              CHARACTER TABLE.  IF SUCCESSFUL, THE ROUTINE
                  ;              CORRESPONDING TO THIS CHARACTER IS SELECTED FROM
                  ;              A TABLE OF COMMAND ROUTINE ADDRESSES, AND CONTROL
                  ;              IS TRANSFERRED TO THIS ROUTINE.  IF THE CHARACTER
                  ;              DOES NOT MATCH ANY ENTRIES, CONTROL IS PASSED TO
                  ;              THE ERROR HANDLER.
                  ;
002B              GETCM:
002B    21ED13          LXI     H,MSTAK ; ALWAYS WANT TO RESET STACK PTR TO MONITOR
002E    F9              SPHL            ; /STARTING VALUE SO ROUTINES NEEDN'T CLEAN UP
002F    0E2E            MVI     C,'.'   ; PROMPT CHARACTER TO C
0031    CDF401          CALL    ECHO    ; SEND PROMPT CHARACTER TO USER TERMINAL
0034    C33B00          JMP     GTC03   ; WANT TO LEAVE ROOM FOR RST BRANCH
                  ;
0038                    ORG     RSTU    ; ORG TO RST TRANSFER LOCATION
0038    C3FD13          JMP     USRBR   ; JUMP TO USER BRANCH LOCATION
                  ;
003B              GTC03:
003B    CD1B02          CALL    GETCH   ; GET COMMAND CHARACTER TO A
003E    CDF401          CALL    ECHO    ; ECHO CHARACTER TO USER
0041    79              MOV     A,C     ; PUT COMMAND CHARACTER INTO ACCUMULATOR
0042    010600          LXI     B,NCMDS ; C CONTAINS LOOP AND INDEX COUNT
0045    21B903          LXI     H,CTAB  ; HL POINTS INTO COMMAND TABLE
```

```
 0046              GTC05:
 0048    BE              CMP      M         ; COMPARE TABLE ENTRY AND CHARACTER
 0049    CA5400          JZ       GTC10     ; BRANCH IF EQUAL - COMMAND RECOGNIZED
 004C    23              INX      H         ; ELSE, INCREMENT TABLE POINTER
 004D    0D              DCR      C         ; DECREMENT LOOP COUNT
 004E    C24800          JNZ      GTC05     ; BRANCH IF NOT AT TABLE END
 0051    C30D02          JMP      ERROR     ; ELSE, COMMAND CHARACTER IS ILLEGAL
 0054              GTC10:
 0054    21AB03          LXI      H,CADR    ; IF GOOD COMMAND, LOAD ADDRESS OF TABLE
                                            ; /OF COMMAND ROUTINE ADDRESSES
 0057    09              DAD      B         ; ADD WHAT IS LEFT OF LOOP COUNT
 0058    09              DAD      B         ; ADD AGAIN - EACH ENTRY IN CADR IS 2 BYTES LONG
 0059    7E              MOV      A,M       ; GET LSP OF ADDRESS OF TABLE ENTRY TO A
 005A    23              INX      H         ; POINT TO NEXT BYTE IN TABLE
 005B    66              MOV      H,M       ; GET MSP OF ADDRESS OF TABLE ENTRY TO H
 005C    6F              MOV      L,A       ; PUT LSP OF ADDRESS OF TABLE ENTRY INTO L
 005D    E9              PCHL               ; NEXT INSTRUCTION COMES FROM COMMAND ROUTINE
                   ;
                   ;
                   ;*******************************************************************
                   ;
                   ;
                   ;              COMMAND IMPLEMENTING ROUTINES
                   ;
                   ;
                   ;*******************************************************************
                   ;
                   ;
                   ; FUNCTION: DCMD
                   ; INPUTS: NONE
                   ; OUTPUTS: NONE
                   ; CALLS: ECHO,NMOUT,HILO,GETCM,CROUT,GETNM
                   ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                   ; DESCRIPTION: DCMD IMPLEMENTS THE DISPLAY MEMORY (D) COMMAND
                   ;
 005E              DCMD:
 005E    0E02            MVI      C,2       ; GET 2 NUMBERS FROM INPUT STREAM
 0060    CD5702          CALL     GETNM
 0063    D1              POP      D         ; ENDING ADDRESS TO DE
 0064    E1              POP      H         ; STARTING ADDRESS TO HL
 0065              DCM05:
 0065    CDEE01          CALL     CROUT     ; ECHO CARRIAGE RETURN/LINE FEED
 0068    7C              MOV      A,H       ; DISPLAY ADDRESS OF FIRST LOCATION IN LINE
 0069    CDC302          CALL     NMOUT
 006C    7D              MOV      A,L       ; ADDRESS IS 2 BYTES LONG
 006D    CDC302          CALL     NMOUT
 0070              DCM10:
 0070    0E20            MVI      C,' '
 0072    CDF401          CALL     ECHO      ; USE BLANK AS SEPARATOR
 0075    7E              MOV      A,M       ; GET CONTENTS OF NEXT MEMORY LOCATION
 0076    CDC302          CALL     NMOUT     ; DISPLAY CONTENTS
 0079    CDBD01          CALL     BREAK    ; SEE IF USER WANTS OUT
     1           +       TRUE     DCM12    ; IF SO, BRANCH
 007C 1 DA8500   +       JC       DCM12
 007F    CD9C02          CALL     HILO     ; SEE IF ADDRESS OF DISPLAYED LOCATION IS
                                           ; /GREATER THAN OR EQUAL TO ENDING ADDRESS
     1           +       FALSE    DCM15    ; IF NOT, MORE TO DISPLAY
 0082 1 D28B00   +       JNC      DCM15
 0085             DCM12:
 0085    CDEE01          CALL     CROUT    ; CARRIAGE RETURN/LINE FEED TO END LINE
 0088    C32B00          JMP      GETCM    ; ALL DONE
 008B             DCM15:
 008B    23              INX      H        ; IF MORE TO GO, POINT TO NEXT LOC TO DISPLAY
 008C    7D              MOV      A,L      ; GET LOW ORDER BITS OF NEW ADDRESS
 008D    E60F            ANI      NEWLN    ; SEE IF LAST HEX DIGIT OF ADDRESS DENOTES
                                           ; /START OF NEW LINE
 008F    C27000          JNZ      DCM10    ; NO - NOT AT END OF LINE
 0092    C36500          JMP      DCM05    ; YES - START NEW LINE WITH ADDRESS
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ;
                  ; FUNCTION: GCMD
                  ; INPUTS: NONE
                  ; OUTPUTS: NONE
                  ; CALLS: ERROR,GETHX,RSTTF
                  ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                  ; DESCRIPTION: GCMD IMPLEMENTS THE BEGIN EXECUTION (G) COMMAND.
                  ;
 0095             GCMD:
 0095    CD2202          CALL     GETHX    ; GET ADDRESS (IF PRESENT) FROM INPUT STREAM
     1           +       FALSE    GCM05    ; BRANCH IF NO NUMBER PRESENT
 0098 1 D2AA00   +       JNC      GCM05
 009B    7A              MOV      A,D      ; ELSE, GET TERMINATOR
 009C    FE0D            CPI      CR       ; SEE IF CARRIAGE RETURN
 009E    C20D02          JNZ      ERROR    ; ERROR IF NOT PROPERLY TERMINATED
 00A1    21F513          LXI      H,PSAVE  ; WANT NUMBER TO REPLACE SAVE PGM COUNTER
 00A4    71              MOV      M,C
 00A5    23              INX      H
 00A6    70              MOV      M,B
 00A7    C3B000          JMP      GCM10
 00AA             GCM05:
 00AA    7A              MOV      A,D      ; IF NO STARTING ADDRESS, MAKE SURE THAT
 00AB    FE0D            CPI      CR       ; /CARRIAGE RETURN TERMINATED COMMAND
 00AD    C20D02          JNZ      ERROR    ; ERROR IF NOT
 00B0             GCM10:
 00B0    C32E03          JMP      RSTTF    ; RESTORE REGISTERS AND BEGIN EXECUTION


                   *****************************************************************
```

```
                        ;
                        ;
                        ; FUNCTION: ICMD
                        ; INPUTS: NONE
                        ; OUTPUTS: NONE
                        ; CALLS: ERROR,ECHO,GETCH,VALDL,VALDG,CNVBN,STHLF,GETNM,CROUT
                        ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                        ; DESCRIPTION: ICMD IMPLEMENTS THE INSERT CODE INTO MEMORY (I) COMMAND.
                        ;
00B3                    ICMD:
00B3    0E01                    MVI     C,1
00B5    CD5702                  CALL    GETNM   ; GET SINGLE NUMBER FROM INPUT STREAM
00B8    3EFF                    MVI     A,UPPER
00BA    32F913                  STA     TEMP    ; TEMP WILL HOLD THE UPPER/LOWER HALF BYTE FLAG
00BD    D1                      POP     D       ; ADDRESS OF START TO DE
00BE                    ICM05:
00BE    CD1B02                  CALL    GETCH   ; GET A CHARACTER FROM INPUT STREAM
00C1    4F                      MOV     C,A
00C2    CDF401                  CALL    ECHO    ; ECHO IT
00C5    79                      MOV     A,C     ; PUT CHARACTER BACK INTO A
00C6    FE1B                    CPI     TERM    ; SEE IF CHARACTER IS A TERMINATING CHARACTER
00C8    CAF400                  JZ      ICM25   ; IF SO, ALL DONE ENTERING CHARACTERS
00CB    CD8A03                  CALL    VALDL   ; ELSE, SEE IF VALID DELIMITER
      1                 +       TRUE    ICM05   ; IF SO SIMPLY IGNORE THIS CHARACTER
00CE  1 DABE00          +       JC      ICM05
00D1    CD6F03                  CALL    VALDG   ; ELSE, CHECK TO SEE IF VALID HEX DIGIT
      1                 +       FALSE   ICM20   ; IF NOT, BRANCH TO HANDLE ERROR CONDITION
00D4  1 D2EE00          +       JNC     ICM20
00D7    CDDA01                  CALL    CNVBN   ; CONVERT DIGIT TO BINARY
00DA    4F                      MOV     C,A     ; MOVE RESULT TO C
00DB    CD5003                  CALL    STHLF   ; STORE IN APPROPRIATE HALF WORD
00DE    3AF913                  LDA     TEMP    ; GET HALF BYTE FLAG
00E1    B7                      ORA     A       ; SET F/F'S
00E2    C2E600                  JNZ     ICM10   ; BRANCH IF FLAG SET FOR UPPER
00E5    13                      INX     D       ; IF LOWER, INC ADDRESS OF BYTE TO STORE IN
00E6                    ICM10:
00E6    EEFF                    XRI     INVRT   ; TOGGLE STATE OF FLAG
00E8    32F913                  STA     TEMP    ; PUT NEW VALUE OF FLAG BACK
00EB    C3BE00                  JMP     ICM05   ; PROCESS NEXT DIGIT
00EE                    ICM20:
00EE    CD4503                  CALL    STHF0   ; ILLEGAL CHARACTER
00F1    C30D02                  JMP     ERROR   ; MAKE SURE ENTIRE BYTE FILLED THEN ERROR
00F4                    ICM25:
00F4    CD4503                  CALL    STHF0   ; HERE FOR ESCAPE CHARACTER - INPUT IS DONE
00F7    CDEE01                  CALL    CROUT   ; ADD CARRIAGE RETURN
00FA    C32B00                  JMP     GETCM
                        ;
                        ;
                        ;*************************************************************************
                        ;
                        ;
```

```
                       ; FUNCTION: MCMD
                       ; INPUTS: NONE
                       ; OUTPUTS: NONE
                       ; CALLS: GETCM,HILO,GETNM
                       ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                       ; DESCRIPTION: MCMD IMPLEMENTS THE MOVE DATA IN MEMORY (M) COMMAND.
                       ;
00FD                   MCMD:
00FD    0E03                 MVI     C,3
00FF    CD5702               CALL    GETNM    ; GET 3 NUMBERS FROM INPUT STREAM
0102    C1                   POP     B        ; DESTINATION ADDRESS TO BC
0103    E1                   POP     H        ; ENDING ADDRESS TO HL
0104    D1                   POP     D        ; STARTING ADDRESS TO DE
0105                   MCM05:
0105    E5                   PUSH    H        ; SAVE ENDING ADDRESS
0106    62                   MOV     H,D
0107    6B                   MOV     L,E      ; SOURCE ADDRESS TO HL
0108    7E                   MOV     A,M      ; GET SOURCE BYTE
0109    60                   MOV     H,B
010A    69                   MOV     L,C      ; DESTINATION ADDRESS TO HL
010B    77                   MOV     M,A      ; MOVE BYTE TO DESTINATION
010C    03                   INX     B        ; INCREMENT DESTINATION ADDRESS
010D    78                   MOV     A,B
010E    B1                   ORA     C        ; TEST FOR DESTINATION ADDRESS OVERFLOW
010F    CA2B00               JZ      GETCM    ; IF SO, CAN TERMINATE COMMAND
0112    13                   INX     D        ; INCREMENT SOURCE ADDRESS
0113    E1                   POP     H        ; ELSE, GET BACK ENDING ADDRESS
0114    CD9C02               CALL    HILO     ; SEE IF ENDING ADDR>=SOURCE ADDR
     1               +       FALSE   GETCM    ; IF NOT, COMMAND IS DONE
0117 1  D22B00       +       JNC     GETCM
011A    C30501               JMP     MCM05    ; MOVE ANOTHER BYTE
                       ;
                       ;
                       ;*****************************************************************
                       ;
                       ;
                       ; FUNCTION: SCMD
                       ; INPUTS: NONE
                       ; OUTPUTS: NONE
                       ; CALLS: GETHX,GETCM,NMOUT,ECHO
                       ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                       ; DESCRIPTION: SCMD IMPLEMENTS THE SUBSTITUTE INTO MEMORY (S) COMMAND.
                       ;
011D                   SCMD:
011D    CD2202               CALL    GETHX    ; GET A NUMBER, IF PRESENT, FROM INPUT
0120    C5                   PUSH    B
0121    E1                   POP     H        ; GET NUMBER TO HL - DENOTES MEMORY LOCATION
0122                   SCM05:
0122    7A                   MOV     A,D      ; GET TERMINATOR
0123    FE20                 CPI     ' '      ; SEE IF SPACE
0125    CA2D01               JZ      SCM10    ; YES - CONTINUE PROCESSING
```

```
0128    FE2C           CPI     ','       ; ELSE, SEE IF COMMA
012A    C22B00         JNZ     GETCM     ; NO - TERMINATE COMMAND
012D              SCM10:
012D    7E             MOV     A,M       ; GET CONTENTS OF SPECIFIED LOCATION TO A
012E    CDC302         CALL    NMOUT     ; DISPLAY CONTENTS ON CONSOLE
0131    0E2D           MVI     C,'-'
0133    CDF401         CALL    ECHO      ; USE DASH FOR SEPARATOR
0136    CD2202         CALL    GETHX     ; GET NEW VALUE FOR MEMORY LOCATION, IF ANY
      1          +     FALSE   SCM15     ; IF NO VALUE PRESENT, BRANCH
0139 1  D23D01   +     JNC     SCM15
013C    71             MOV     M,C       ; ELSE, STORE LOWER 8 BITS OF NUMBER ENTERED
013D              SCM15:
013D    23             INX     H         ; INCREMENT ADDRESS OF MEMORY LOCATION TO VIEW
013E    C32201         JMP     SCM05
                  ;
                  ;
                  ;*****************************************************************
                  ;
                  ;
                  ; FUNCTION: XCMD
                  ; INPUTS: NONE
                  ; OUTPUTS: NONE
                  ; CALLS: GETCH,ECHO,REGDS,GETCM,ERROR,RGADR,NMOUT,CROUT,GETHX
                  ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                  ; DESCRIPTION: XCMD IMPLEMENTS THE REGISTER EXAMINE AND CHANGE (X)
                  ;              COMMAND.
                  ;
0141              XCMD:
0141    CD1B02         CALL    GETCH     ; GET REGISTER IDENTIFIER
0144    4F             MOV     C,A
0145    CDF401         CALL    ECHO      ; ECHO IT
0148    79             MOV     A,C
0149    FE0D           CPI     CR
014B    C25401         JNZ     XCM05     ; BRANCH IF NOT CARRIAGE RETURN
014E    CDE602         CALL    REGDS     ; ELSE, DISPLAY REGISTER CONTENTS
0151    C32B00         JMP     GETCM     ; THEN TERMINATE COMMAND
0154              XCM05:
0154    4F             MOV     C,A       ; GET REGISTER IDENTIFIER TO C
0155    CD1703         CALL    RGADR     ; CONVERT IDENTIFIER INTO RTAB TABLE ADDR
0158    C5             PUSH    B
0159    E1             POP     H         ; PUT POINTER TO REGISTER ENTRY INTO HL
015A    0E20           MVI     C,' '
015C    CDF401         CALL    ECHO      ; ECHO SPACE TO USER
015F    79             MOV     A,C
0160    32F913         STA     TEMP      ; PUT SPACE INTO TEMP AS DELIMITER
0163              XCM10:
0163    3AF913         LDA     TEMP      ; GET TERMINATOR
0166    FE20           CPI     ' '       ; SEE IF A BLANK
0168    CA7001         JZ      XCM15     ; YES - GO CHECK POINTER INTO TABLE
016B    FE2C           CPI     ','       ; NO - SEE IF COMMA
016D    C22B00         JNZ     GETCM     ; NO - MUST BE CARRIAGE RETURN TO END COMMAND
```

```
 0170             XCM15:
 0170   7E                 MOV    A,M
 0171   B7                 ORA    A        ; SET F/F'S
 0172   C27B01             JNZ    XCM18    ; BRANCH IF NOT AT END OF TABLE
 0175   CDEE01             CALL   CROUT    ; ELSE, OUTPUT CARRIAGE RETURN LINE FEED
 0178   C32800             JMP    GETCM    ; AND EXIT
 017B             XCM18:
 017B   E5                 PUSH   H        ; PUT POINTER ON STACK
 017C   5E                 MOV    E,M
 017D   1613               MVI    D,DATA SHR 8    ; FETCH ADDRESS OF SAVE LOCATION FROM TABLE
 017F   23                 INX    H
 0180   46                 MOV    B,M      ; FETCH LENGTH FLAG FROM TABLE
 0181   D5                 PUSH   D        ; SAVE ADDRESS OF SAVE LOCATION
 0182   D5                 PUSH   D
 0183   E1                 POP    H        ; MOVE ADDRESS TO HL
 0184   C5                 PUSH   B        ; SAVE LENGTH FLAG
 0185   7E                 MOV    A,M      ; GET 8 BITS OF REGISTER FROM SAVE LOCATION
 0186   CDC302             CALL   NMOUT    ; DISPLAY IT
 0189   F1                 POP    PSW      ; GET BACK LENGTH FLAG
 018A   F5                 PUSH   PSW      ; SAVE IT AGAIN
 018B   B7                 ORA    A        ; SET F/F'S
 018C   CA9401             JZ     XCM20    ; IF 8 BIT REGISTER, NOTHING MORE TO DISPLAY
 018F   2B                 DCX    H        ; ELSE, FOR 16 BIT REGISTER, GET LOWER 8 BITS
 0190   7E                 MOV    A,M
 0191   CDC302             CALL   NMOUT    ; DISPLAY THEM
 0194             XCM20:
 0194   0E2D               MVI    C,'-'
 0196   CDF401             CALL   ECHO     ; USE DASH AS SEPARATOR
 0199   CD2202             CALL   GETHX    ; SEE IF THERE IS A VALUE TO PUT INTO REGISTER
      1           +        FALSE  XCM30    ; NO - GO CHECK FOR NEXT REGISTER
 019C 1 D2B401    +        JNC    XCM30
 019F   7A                 MOV    A,D
 01A0   32F913             STA    TEMP     ; ELSE, SAVE THE TERMINATOR FOR NOW
 01A3   F1                 POP    PSW      ; GET BACK LENGTH FLAG
 01A4   E1                 POP    H        ; PUT ADDRESS OF SAVE LOCATION INTO HL
 01A5   B7                 ORA    A        ; SET F/F'S
 01A6   CAAB01             JZ     XCM25    ; IF 8 BIT REGISTER, BRANCH
 01A9   70                 MOV    M,B      ; SAVE UPPER 8 BITS
 01AA   2B                 DCX    H        ; POINT TO SAVE LOCATION FOR LOWER 8 BITS
 01AB             XCM25:
 01AB   71                 MOV    M,C      ; STORE ALL OF 8 BIT OR LOWER 1/2 OF 16 BIT REG
 01AC             XCM27:
 01AC   110300             LXI    D,RTABS  ; SIZE OF ENTRY IN RTAB TABLE
 01AF   E1                 POP    H        ; POINTER INTO REGISTER TABLE RTAB
 01B0   19                 DAD    D        ; ADD ENTRY SIZE TO POINTER
 01B1   C36301             JMP    XCM10    ; DO NEXT REGISTER
 01B4             XCM30
 01B4   7A                 MOV    A,D      ; GET TERMINATOR
 01B5   32F913             STA    TEMP     ; SAVE IN MEMORY
 01B8   D1                 POP    D        ; CLEAR STACK OF LENGTH FLAG AND ADDRESS
 01B9   D1                 POP    D        ; /OF SAVE LOCATION
 01BA   C3AC01          JMP     XCM27   ; GO INCREMENT REGISTER TABLE POINTER
                ;
                ;
                ;*************************************************************
                ;
                ;
                ;                       UTILITY ROUTINES
                ;
                ;
                ;*************************************************************
                ;
                ;
                ; FUNCTION: BREAK
                ; INPUTS: NONE
CTER INPUT      ; OUTPUTS: CARRY - 1 IF ESCAPE CHARA
                ;                - 0 IF ANY OTHER CHARACTER OR NO CHARACTER PENDING
                ; CALLS: NOTHING
                ; DESTROYS: A,F/F'S
                ; DESCRIPTION: BREAK IS USED TO SENSE AN ESCAPE CHARACTER FROM
                ;              THE USER.  IF NO CHARACTER IS PENDING, OR IF THE
                ;              PENDING CHARACTER IS NOT THE ESCAPE, THEN A FAILURE
                ;              RETURN (CARRY=0) IS TAKEN.  IN THIS CASE,
                                                                       THE
                ;              PENDING CHARACTER (IF ANY) IS LOST.  IF THE PENDING
                ;              CHARACTER IS AN ESCAPE CHARACTER, BREAK TAKES A SUCCESS
                ;              RETURN (CARRY=1).
                ;
 01BD           BREAK:
 01BD   DBFB            IN      CONST   ; GET CONSOLE STATUS
 01BF   E602            ANI     RBR     ; SEE IF CHARACTER PENDING
 01C1   CA1802          JZ      FRET    ; NO - TAKE FAILURE RETURN
 01C4   DBFA            IN      CNIN    ; YES - PICK UP CHARACTER
 01C6   E67F            ANI     PRTY0   ; STRIP OFF PARITY BIT
 01C8   FE1B            CPI     BRCHR   ; SEE IF BREAK CHARACTER
 01CA   CA4303          JZ      SRET    ; YES - SUCCESS RETURN
 01CD   C31802          JMP     FRET    ; NO - FAILURE RETURN - CHARACTER LOST
                ;
                ;
                ;*************************************************************
                ;
                ;
                ; FUNCTION: CI
                ; INPUTS: NONE
                ; OUTPUTS: A - CHARACTER FROM CONSOLE
                ; CALLS: NOTHING
                ; DESTROYS: A,F/F'S
                ; DESCRIPTION: CI WAITS UNTIL A CHARACTER HAS BEEN ENTERED AT THE
                ;              CONSOLE AND THEN RETURNS THE CHARACTER, VIA THE A
                ;              REGISSTER, TO THE CALLING ROUTINE.  THIS ROUTINE
                ;              IS CALLED BY THE USER VIA A JUMP TABLE IN RAM.
                ;
 01D0           CI:
```

```
 01D0    DBFB            IN      CONST    ; GET STATUS OF CONSOLE
 01D2    E602            ANI     RBR      ; CHECK FOR RECEIVER BUFFER READY
 01D4    CAD001          JZ      CI       ; NOT YET - WAIT
 01D7    DBFA            IN      CNIN     ; READY SO GET CHARACTER
 01D9    C9              RET
                 ;
                 ;
                 ;*******************************************************************
                 ;
                 ;
                 ; FUNCTION: CNVBN
                 ; INPUTS: C - ASCII CHARACTER '0'-'9' OR 'A'-'F'
                 ; OUTPUTS: A - 0 TO F HEX
                 ; CALLS: NOTHING
                 ; DESTROYS: A,F/F'S
                 ; DESCRIPTION: CNVBN CONVERTS THE ASCII REPRESENTATION OF A HEX
                 ;              CNVBN INTO ITS CORRESPONDING BINARY VALUE.  CNVBN
                 ;              DOES NOT CHECK THE VALIDITY OF ITS INPUT.
                 ;
 01DA            CNVBN:
 01DA    79              MOV     A,C
 01DB    D630            SUI     '0'      ; SUBTRACT CODE FOR '0' FROM ARGUMENT
 01DD    FE0A            CPI     10       ; WANT TO TEST FOR RESULT OF 0 TO 9
 01DF    F8              RM               ; IF SO, THEN ALL DONE
 01E0    D607            SUI     7        ; ELSE, RESULT BETWEEN 17 AND 23 DECIMAL
 01E2    C9              RET              ; SO RETURN AFTER SUBTRACTING BIAS OF 7
                 ;
                 ;
                 ;*********************************************************************
                 ;
                 ;
                 ; FUNCTION: CO
                 ; INPUTS: C - CHARACTER TO OUTPUT TO CONSOLE
                 ; OUTPUTS: C - CHARACTER OUTPUT TO CONSOLE
                 ; CALLS: NOTHING
                 ; DESTROYS: A,F/F'S
                 ; DESCRIPTION: CO WAITS UNTIL THE CONSOLE IS READY TO ACCEPT A CHARACTER
                 ;              AND THEN SENDS THE INPUT ARGUMENT TO THE CONSOLE.
                 ;
 01E3            CO:
 01E3    DBFB            IN      CONST    ; GET STATUS OF CONSOLE
 01E5    E601            ANI     TRDY     ; SEE IF TRANSMITTER READY
 01E7    CAE301          JZ      CO       ; NO - WAIT
 01EA    79              MOV     A,C      ; ELSE, MOVE CHARACTER TO A REGISTER FOR OUTPUT
 01EB    D3FA            OUT     CNOUT    ; SEND TO CONSOLE
 01ED    C9              RET
                 ;
                 ;
                 ;*******************************************************************
                 ;
                 ;
                 ; FUNCTION CROUT
                 ; INPUTS: NONE
                 ; OUTPUTS: NONE
                 ; CALLS: ECHO
                 ; DESTROYS: A,B,C,F/F'S
                 ; DESCRIPTION: CROUT SENDS A CARRIAGE RETURN (AND HENCE A LINE
                 ;              FEED) TO THE CONSOLE.
                 ;
01EE             CROUT:
01EE     0E0D          MVI      C,CR
01F0     CDF401        CALL     ECHO
01F3     C9            RET
                 ;
                 ;
                 ;*******************************************************************
                 ;
                 ;
                 ; FUNCTION: ECHO
                 ; INPUTS: C - CHARACTER TO ECHO TO TERMINAL
                 ; OUTPUTS: C - CHARACTER ECHOED TO TERMINAL
                 ; CALLS: CO
                 ; DESTROYS: A,B,F/F'S
                 ; DESCRIPTION: ECHO TAKES A SINGLE CHARACTER AS INPUT AND, VIA
                 ;              THE MONITOR, SENDS THAT CHARACTER TO THE USER
                 ;              TERMINAL.  A CARRIAGE RETURN IS ECHOED AS A CARRIAGE
                 ;              RETURN LINE FEED, AND AN ESCAPE CHARACTER IS ECHOED AS $.
                 ;
01F4             ECHO:
01F4     41            MOV      B,C      ; SAVE ARGUMENT
01F5     3E1B          MVI      A,ESC
01F7     B8            CMP      B        ; SEE IF ECHOING AN ESCAPE CHARACTER
01F8     C2FD01        JNZ      ECH05    ; NO - BRANCH
01FB     0E24          MVI      C,'$'    ; YES - ECHO AS $
01FD             ECH05:
01FD     CDE301        CALL     CO       ; DO OUTPUT THROUGH MONITOR
0200     3E0D          MVI      A,CR
0202     B8            CMP      B        ; SEE IF CHARACTER ECHOED WAS A CARRIAGE RETURN
0203     C20B02        JNZ      ECH10    ; NO - NO NEED TO TAKE SPECIAL ACTION
0206     0E0A          MVI      C,LF     ; YES - WANT TO ECHO LINE FEED, TOO
0208     CDE301        CALL     CO
020B             ECH10:
020B     48            MOV      C,B      ; RESTORE ARGUMENT
020C     C9            RET
                 ;
                 ;
                 ;*******************************************************************
                 ;
                 ;
                 ; FUNCTION: ERROR
                 ; INPUTS: NONE
                 ; OUTPUTS: NONE
```

```
                ; CALLS: ECHO,CROUT,GETCM
                ; DESTROYS: A,B,C,F/F'S
                ; DESCRIPTION: ERROR PRINTS THE ERROR CHARACTER (CURRENTLY AN ASTERISK)
                ;              ON THE CONSOLE, FOLLOWED BY A CARRIAGE RETURN-LINE FEED,
                ;              AND THEN RETURNS CONTROL TO THE COMMAND RECOGNIZER.
                ;
020D            ERROR:
020D  0E2A             MVI     C '*'
020F  CDF401           CALL    ECHO     ; SEND * TO CONSOLE
0212  CDEE01           CALL    CROUT    ; SKIP TO BEGINNING OF NEXT LINE
0215  C32B00           JMP     GETCM    ; TRY AGAIN FOR ANOTHER COMMAND
                ;
                ;
                ;**********************************************************************
                ;
                ;
                ; FUNCTION: FRET
                ; INPUTS: NONE
                ; OUTPUTS: CARRY - ALWAYS 0
                ; CALLS: NOTHING
                ; DESTROYS: CARRY
                ; DESCRIPTION: FRET IS JUMPED TO BY ANY ROUTINE THAT WISHES TO
                ;              INDICATE FAILURE ON RETURN.  FRET SETS THE CARRY
                ;              FALSE, DENOTING FAILURE, AND THEN RETURNS TO THE
                ;              CALLER OF THE ROUTINE INVOKING FRET.
                ;
0218            FRET:
0218  37               STC              ; FIRST SET CARRY TRUE
0219  3F               CMC              ; THEN COMPLEMENT IT TO MAKE IT FALSE
021A  C9               RET              ; RETURN APPROPRIATELY
                ;
                ;
                ;**********************************************************************
                ;
                ;
                ; FUNCTION: GETCH
                ; INPUTS: NONE
                ; OUTPUTS: C - NEXT CHARACTER IN INPUT STREAM
                ; CALLS: CI
                ; DESTROYS: A,C,F/F'S
                ; DESCRIPTION: GETCH RETURNS THE NEXT CHARACTER IN THE INPUT STREAM
                ;              TO THE CALLING PROGRAM.
                ;
021B            GETCH:
021B  CDD001           CALL    CI       ; GET CHARACTER FROM TERMINAL
021E  E67F             ANI     PRTY0    ; TURN OFF PARITY BIT IN CASE SET BY CONSOLE
0220  4F               MOV     C,A      ; PUT VALUE IN C REGISTER FOR RETURN
0221  C9               RET
                ;
                ;
                ;**********************************************************************
```

```
                   ;
                   ;
                   ; FUNCTION: GETHX
                   ; INPUTS: NONE
                   ; OUTPUTS: BC - 16 BIT INTEGER
                   ;          D - CHARACTER WHICH TERMINATED THE INTEGER
                   ;          CARRY - 1 IF FIRST CHARACTER NOT DELIMITER
                   ;                - 0 IF FIRST CHARACTER IS DELIMITER
                   ; CALLS: GETCH,ECHO,VALDL,VALDG,CNVBN,ERROR
                   ; DESTROYS: A,B,C,D,E,F/F'S
                   ; DESCRIPTION: GETHX ACCEPTS A STRING OF HEX DIGITS FROM THE INPUT
                   ;              STREAM AND RETURNS THEIR VALUE AS A 16 BIT BINARY
                   ;              INTEGER.  IF MORE THAN 4 HEX DIGITS ARE ENTERED,
                   ;              ONLY THE LAST 4 ARE USED.  THE NUMBER TERMINATES WHEN
                   ;              A VALID DELIMITER IS ENCOUNTERED.  THE DELIMITER IS
                   ;              ALSO RETURNED AS AN OUTPUT OF THE FUNCTION.  ILLEGAL
                   ;              CHARACTERS (NOT HEX DIGITS OR DELIMITERS) CAUSE AN
                   ;              ERROR INDICATION.  IF THE FIRST (VALID) CHARACTER
                   ;              ENCOUNTERED IN THE INPUT STREAM IS NOT A DELIMITER,
                   ;              GETHX WILL RETURN WITH THE CARRY BIT SET TO 1;
                   ;              OTHERWISE, THE CARRY BIT IS SET TO 0 AND THE CONTENTS
                   ;              OF BC ARE UNDEFINED.
                   ;
0222               GETHX:
0222     E5              PUSH    H          ; SAVE HL
0223     210000          LXI     H,0        ; INITIALIZE RESULT
0226     1E00            MVI     E,0        ; INITIALIZE DIGIT FLAG TO FALSE
0228               GHX05:
0228     CD1B02          CALL    GETCH      ; GET A CHARACTER
022B     4F              MOV     C,A
022C     CDF401          CALL    ECHO       ; ECHO THE CHARACTER
022F     CD0A03          CALL    VALDL      ; SEE IF DELIMITER
       1         +       FALSE   GHX10      ; NO - BRANCH
0232 1 D24102    +       JNC     GHX10
0235     51              MOV     D,C        ; YES - ALL DONE, BUT WANT TO RETURN DELIMITER
0236     E5              PUSH    H
0237     C1              POP     B          ; MOVE RESULT TO BC
0238     E1              POP     H          ; RESTORE HL
0239     7B              MOV     A,E        ; GET FLAG
023A     B7              ORA     A          ; SET F/F'S
023B     C24303          JNZ     SRET       ; IF FLAG NON-0, A NUMBER HAS BEEN FOUND
023E     CA1802          JZ      FRET       ; ELSE, DELIMITER WAS FIRST CHARACTER
0241               GHX10:
0241     CD6F03          CALL    VALDG      ; IF NOT DELIMITER, SEE IF DIGIT
       1         +       FALSE   ERROR      ; ERROR IF NOT A VALID DIGIT, EITHER
0244 1 D20D02    +       JNC     ERROR
0247     CDDA01          CALL    CNVBN      ; CONVERT DIGIT TO ITS BINARY VALUE
024A     1EFF            MVI     E,0FFH     ; SET DIGIT FLAG NON-0
024C     29              DAD     H          ; *2
024D     29              DAD     H          ; *4
024E     29              DAD     H          ; *8
```

```
 024F    29                 DAD     H         ; *16
 0250    0600               MVI     B,0       ; CLEAR UPPER 8 BITS OF BC PAIR
 0252    4F                 MOV     C,A       ; BINARY VALUE OF CHARACTER INTO C
 0253    09                 DAD     B         ; ADD THIS VALUE TO PARTIAL RESULT
 0254    C32802             JMP     GHX05     ; GET NEXT CHARACTER
                      ;
                      ;
                      ;*****************************************************************
                      ;
                      ;
                      ; FUNCTION: GETNM
                      ; INPUTS: C - COUNT OF NUMBERS TO FIND IN INPUT STREAM
                      ; OUTPUTS: TOP OF STACK - NUMBERS FOUND IN REVERSE ORDER (LAST ON TOP
                      ;                         OF STACK)
                      ; CALLS: GETHX,HILO,ERROR
                      ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                      ; DESCRIPTION: GETNM FINDS A SPECIFIED COUNT OF NUMBERS, BETWEEN 1
                      ;              AND 3, INCLUSIVE,  IN THE INPUT
                      ;              STREAM AND RETURNS THEIR VALUES ON THE STACK.  IF 2
                      ;              OR MORE NUMBERS ARE REQUESTED, THEN THE FIRST MUST BE
                      ;              LESS THAN OR EQUAL TO THE SECOND, OR THE FIRST AND
                      ;              SECOND NUMBERS WILL BE SET EQUAL.  THE LAST NUMBER
                      ;              REQUESTED MUST BE TERMINATED BY A CARRIAGE RETURN
                      ;              OR AN ERROR INDICATION WILL RESULT.
                      ;
 0257                 GETNM:
 0257    2E03               MVI     L,3       ; PUT MAXIMUM ARGUMENT COUNT INTO L
 0259    79                 MOV     A,C       ; GET THE ACTUAL ARGUMENT COUNT
 025A    E603               ANI     3         ; FORCE TO MAXIMUM OF 3
 025C    C8                 RZ                ; IF 0, DON'T BOTHER TO DO ANYTHIING
 025D    67                 MOV     H,A       ; ELSE, PUT ACTUAL COUNT INTO H
 025E                 GNM05:
 025E    CD2202             CALL    GETHX     ; GET A NUMBER FROM INPUT STREAM
       1              +     FALSE   ERROR     ; ERROR IF NOT THERE - TOO FEW NUMBERS
 0261  1 D20D02       +     JNC     ERROR
 0264    C5                 PUSH    B         ; ELSE, SAVE NUMBER ON STACK
 0265    2D                 DCR     L         ; DECREMENT MAXIMUM ARGUMENT COUNT
 0266    25                 DCR     H         ; DECREMENT ACTUAL ARGUMENT COUNT
 0267    CA7302             JZ      GNM10     ; BRANCH IF NO MORE NUMBERS WANTED
 026A    7A                 MOV     A,D       ; ELSE, GET NUMBER TERMINATOR TO A
 026B    FE0D               CPI     CR        ; SEE IF CARRIAGE RETURN
 026D    CA0D02             JZ      ERROR     ; ERROR IF SO - TOO FEW NUMBERS
 0270    C35E02             JMP     GNM05     ; ELSE, PROCESS NEXT NUMBER
 0273                 GNM10:
 0273    7A                 MOV     A,D       ; WHEN COUNT 0, CHECK LAST TERMINATOR
 0274    FE0D               CPI     CR
 0276    C20D02             JNZ     ERROR     ; ERROR IF NOT CARRIAGE RETURN
 0279    01FFFF             LXI     B,0FFFFH          ; HL GETS LARGEST NUMBER
 027C    7D                 MOV     A,L       ; GET WHAT'S LEFT OF MAXIMUM ARG COUNT
 027D    B7                 ORA     A         ; CHECK FOR 0
 027E    CA8602             JZ      GNM20     ; IF YES, 3 NUMBERS WERE INPUT
 0281               GNM15:
 0281      C5               PUSH     B         ; IF NOT, FILL REMAINING ARGUMENTS WITH 0FFFFH.
 0282      2D               DCR      L
 0283      C28102           JNZ      GNM15
 0286               GNM20:
 0286      C1               POP      B         ; GET THE 3 ARGUMENTS OUT
 0287      D1               POP      D
 0288      E1               POP      H
 0289      CD9C02           CALL     HILO      ; SEE IF FIRST >= SECOND
         1         +        FALSE    GNM25     ; NO - BRANCH
 028C  1   D29102  +        JNC      GNM25
 028F      54               MOV      D,H
 0290      5D               MOV      E,L       ; YES - MAKE SECOND EQUAL TO THE FIRST
 0291               GNM25:
 0291      E3               XTHL               ; PUT FIRST ON STACK - GET RETURN ADDR
 0292      D5               PUSH     D         ; PUT SECOND ON STACK
 0293      C5               PUSH     B         ; PUT THIRD ON STACK
 0294      E5               PUSH     H         ; PUT RETURN ADDRESS ON STACK
 0295               GNM30:
 0295      3D               DCR      A         ; DECREMENT RESIDUAL COUNT
 0296      F8               RM                 ; IF NEGATIVE, PROPER RESULTS ON STACK
 0297      E1               POP      H         ; ELSE, GET RETURN ADDR
 0298      E3               XTHL               ; REPLACE TOP RESULT WITH RETURN ADDR
 0299      C39502           JMP      GNM30     ; TRY AGAIN
                    ;
                    ;
                    ;*********************************************************************
                    ;
                    ;
                    ; FUNCTION: HILO
                    ; INPUTS: DE - 16 BIT INTEGER
                    ;         HL - 16 BIT INTEGER
                    ; OUTPUTS: CARRY - 0 IF HL<DE
                    ;                - 1 IF HL>=DE
                    ; CALLS: NOTHING
                    ; DESTROYS: F/F'S
                    ; DESCRIPTION: HILO COMPARES THE 2 16 BIT INTEGERS IN HL AND DE.  THE
                    ;              INTEGERS ARE TREATED AS UNSIGNED NUMBERS.  THE CARRY
                    ;              BIT IS SET ACCORDING TO THE RESULT OF THE COMPARISON.
                    ;
 029C               HILO:
 029C      C5               PUSH     B         ; SAVE BC
 029D      47               MOV      B,A       ; SAVE A IN B REGISTER
 029E      E5               PUSH     H         ; SAVE HL PAIR
 029F      7A               MOV      A,D       ; CHECK FOR DE = 0000H
 02A0      B3               ORA      E
 02A1      CABD02           JZ       HIL05     ; WE'RE AUTOMATICALLY DONE IF IT IS
 02A4      23               INX      H         ; INCREMENT HL BY 1
 02A5      7C               MOV      A,H       ; WANT TO TEST FOR 0 RESULT AFTER
 02A6      B5               ORA      L         ; /INCREMENTING
 02A7      CABD02           JZ       HIL05     ; IF SO, HL MUST HAVE CONTAINED 0FFFFH
 02AA   E1                POP     H         ; IF NOT, RESTORE ORIGINAL HL
 02AB   D5                PUSH    D         ; SAVE DE
 02AC   3EFF              MVI     A,0FFH    ; WANT TO TAKE 2'S COMPLEMENT OF DE CONTENTS
 02AE   AA                XRA     D
 02AF   57                MOV     D,A
 02B0   3EFF              MVI     A,0FFH
 02B2   AB                XRA     E
 02B3   5F                MOV     E,A
 02B4   13                INX     D         ; 2'S COMPLEMENT OF DE TO DE
 02B5   7D                MOV     A,L
 02B6   83                ADD     E         ; ADD HL AND DE
 02B7   7C                MOV     A,H
 02B8   8A                ADC     D         ; THIS OPERATION SETS CARRY PROPERLY
 02B9   D1                POP     D         ; RESTORE ORIGINAL DE CONTENTS
 02BA   78                MOV     A,B       ; RESTORE ORIGINAL CONTENTS OF A
 02BB   C1                POP     B         ; RESTORE ORIGINAL CONTENTS OF BC
 02BC   C9                RET               ; RETURN WITH CARRY SET AS REQUIRED
 02BD             HIL05:
 02BD   E1                POP     H         ; IF HL CONTAINS 0FFFFH, THEN CARRY CAN
 02BE   78                MOV     A,B       ;  /ONLY BE SET TO 1
 02BF   C1                POP     B         ; RESTORE ORIGINAL CONTENTS OF REGISTERS
 02C0   C34303            JMP     SRET      ; SET CARRY AND RETURN
                  ;
                  ;
                  ;**********************************************************************
                  ;
                  ;
                  ; FUNCTION: NMOUT
                  ; INPUTS: A - 8 BIT INTEGER
                  ; OUTPUTS: NONE
                  ; CALLS: ECHO,PRVAL
                  ; DESTROYS: A,B,C,F/F'S
                  ; DESCRIPTION: NNMOUT CONVERTS THE 8 BIT, UNSIGNED INTEGER IN THE
                  ;              A REGISTER INTO 2 ASCII CHARACTERS.  THE ASCII CHARACTERS
                  ;              ARE THE ONES REPRESENTING THE 8 BITS.  THESE TWO
                  ;              CHARACTERS ARE SENT TO THE CONSOLE AT THE CURRENT PRINT
                  ;              POSITION OF THE CONSOLE.
                  ;
02C3              NMOUT:
02C3   E5                 PUSH    H         ; SAVE HL - DESTROYED BY PRVAL
02C4   F5                 PUSH    PSW       ; SAVE ARGUMENT
02C5   0F                 RRC
02C6   0F                 RRC
02C7   0F                 RRC
02C8   0F                 RRC               ; GET UPPER 4 BITS TO LOW 4 BIT POSITIONS
02C9   E60F               ANI     HCHAR     ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
02CB   4F                 MOV     C,A
02CC   CDDE02             CALL    PRVAL     ; CONVERT LOWER 4 BITS TO ASCII
02CF   CDF401             CALL    ECHO      ; SEND TO TERMINAL
02D2   F1                 POP     PSW       ; GET BACK ARGUMENT
02D3   E60F               ANI     HCHAR     ; MASK OUT UPPER 4 BITS - WANT 1 HEX CHAR
```

```
02D5    4F              MOV     C,A
02D6    CDDE02          CALL    PRVAL
02D9    CDF401          CALL    ECHO
02DC    E1              POP     H           RESTORE SAVED VALUE OF HL
02DD    C9              RET
                ;
                ;
                ;*********************************************************************
                ;
                ;
                ; FUNCTION;  PRVAL
                ; INPUTS: C - INTEGER, RANGE 0 TO F
                ; OUTPUTS: C - ASCII CHARACTER
                ; CALLS: NOTHING
                ; DESTROYS: B,C,H,L,F/F'S
                ; DESCRIPTION: PRVAL CONVERTS A NUMBER IN THE RANGE 0 TO F HEX TO
                ;              THE CORRESPONDING ASCII CHARACTER, 0-9,A-F.  PRVAL
                ;              DOES NOT CHECK THE VALIDITY OF ITS INPUT ARGUMENT.
                ;
02DE            PRVAL:
02DE    218F03          LXI     H,DIGTB ; ADDRESS OF TABLE
02E1    0600            MVI     B,0     ; CLEAR HIGH ORDER BITS OF BC
02E3    09              DAD     B       ; ADD DIGIT VALUE TO HL ADDRESS
02E4    4E              MOV     C,M     ; FETCH CHARACTER FROM MEMORY
02E5    C9              RET

                ;*********************************************************************
                ;
                ;
                ; FUNCTION: REGDS
                ; INPUTS: NONE
                ; OUTPUTS: NONE
                ; CALLS: ECHO,NMOUT,ERROR,CROUT
                ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                ; DESCRIPTION: REGDS DISPLAYS THE CONTENTS OF THE REGISTER SAVE
                ;              LOCATIONS, IN FORMATTED FORM, ON THE CONSOLE.  THE
                ;              DISPLAY IIS DRIVEN FROM A TABLE, RTAB, WHICH CONTAINS
                ;              THE REGISTER'S PRINT SYMBOL, SAVE LOCATION ADDRESS,
                ;              AND LENGTH (8 OR 16 BITS).
                ;
02E6            REGDS:
02E6    21CF03          LXI     H,RTAB  ; LOAD HL WITH ADDRESS OF START OF TABLE
02E9            REG05:
02E9    4E              MOV     C,M     ; GET PRINT SYMBOL OF REGISTER
02EA    79              MOV     A,C
02EB    B7              ORA     A       ; TEST FOR 0 - END OF TABLE
02EC    C2F302          JNZ     REG10   ; IF NOT END, BRANCH
02EF    CDEE01          CALL    CROUT   ; ELSE, CARRIAGE RETURN/LINE FEED TO END
02F2    C9              RET             ; /DISPLAY
02F3            REG10:
```

```
02F3    CDF401          CALL    ECHO      ; ECHO CHARACTER
02F6    0E3D            MVI     C,'='
02F8    CDF401          CALL    ECHO      ; OUTPUT EQUALS SIGN, I.E. A=
02FB    23              INX     H         ; POINT TO START OF SAVE LOCATION ADDRESS
02FC    5E              MOV     E,M       ; GET LSP OF SAVE LOCATION ADDRESS TO E
02FD    1613            MVI     D,DATA SHR 8      ; PUT MSP OF SAVE LOC ADDRESS INTO D
02FF    23              INX     H         ; POINT TO LENGTH FLAG
0300    1A              LDAX    D         ; GET CONTENTS OF SAVE ADDRESS
0301    CDC302          CALL    NMOUT     ; DISPLAY ON CONSOLE
0304    7E              MOV     A,M       ; GET LENGTH FLAG
0305    B7              ORA     A         ; SET SIGN F/F
0306    CA0E03          JZ      REG15     ; IF 0, REGISTER IS 8 BITS
0309    1B              DCX     D         ; ELSE, 16 BIT REGISTER SO MORE TO DISPLAY
030A    1A              LDAX    D         ; GET LOWER 8 BITS
030B    CDC302          CALL    NMOUT     ; DISPLAY THEM
030E                REG15:
030E    0E20            MVI     C,' '
0310    CDF401          CALL    ECHO
0313    23              INX     H         ; POINT TO START OF NEXT TABLE ENTRY
0314    C3E902          JMP     REG05     ; DO NEXT REGISTER
                    ;
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ; FUNCTION: RGADR
                    ; INPUTS: C - CHARACTER DENOTING REGISTER
                    ; OUTPUTS: BC - ADDRESS OF ENTRY IN RTAB CORRESPONDING TO REGISTER
                    ; CALLS: ERROR
                    ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                    ; DESCRIPTION: RGADR TAKES A SINGLE CHARACTER AS INPUT.  THIS CHARACTER
                    ;              DENOTES A REGISTER.  RGADR SEARCHES THE TABLE RTAB
                    ;              FOR A MATCH ON THE INPUT ARGUMENT.  IF ONE OCCURS,
                    ;              RGADR RETURNS THE ADDRESS OF THE ADDRESS OF THE
                    ;              SAVE LOCATION CORRESPONDING TO THE REGISTER.  THIS
                    ;              ADDRESS POINTS INTO RTAB.  IF NO MATCH OCCURS, THEN
                    ;              THE REGISTER IDENTIFIER IS ILLEGAL AND CONTROL IS
                    ;              PASSED TO THE ERROR ROUTINE.
                    ;
0317                RGADR:
0317    21CF03          LXI     H,RTAB    ; HL GETS ADDRESS OF TABLE START
031A    110300          LXI     D,RTABS   ; DE GET SIZE OF A TABLE ENTRY
031D                RGA05:
031D    7E              MOV     A,M       ; GET REGISTER IDENTIFIER
031E    B7              ORA     A         ; CHECK FOR TABLE END (IDENTIFIER IS 0)
031F    CA0D02          JZ      ERROR     ; IF AT END OF TABLE, ARGUMENT IS ILLEGAL
0322    B9              CMP     C         ; ELSE, COMPARE TABLE ENTRY AND ARGUMENT
0323    CA2A03          JZ      RGA10     ; IF EQUAL, WE'VE FOUND WHAT WE'RE LOOKING FOR
0326    19              DAD     D         ; ELSE, INCREMENT TABLE POINTER TO NEXT ENTRY
0327    C31D03          JMP     RGA05     ; TRY AGAIN
032A                RGA10:
 032A    23                INX     H         ; IF A MATCH, INCREMENT TABLE POINTER TO
 032B    44                MOV     B,H       ; /SAVE LOCATION ADDRESS
 032C    4D                MOV     C,L       ; RETURN THIS VALUE
 032D    C9                RET
                  ;
                  ;
                  ;**********************************************************************
                  ;
                  ;
                  ; FUNCTION: RSTTF
                  ; INPUTS: NONE
                  ; OUTPUTS: NONE
                  ; CALLS: NOTHING
                  ; DESTROYS: A,B,C,D,E,H,L,F/F'S
                  ; DESCRIPTION: RSTTF RESTORES ALL CPU REGISTER, FLIP/FLOPS, STACK
                  ;              POINTER AND PROGRAM COUNTER FROM THEIR RESPECTIVE
                  ;              SAVE LOCATIONS IN MEMORY.  THE ROUTINE THEN TRANSFERS
                  ;              CONTROL TO THE LOCATION SPECIFIED BY THE PROGRAM
                  ;              COUNTER (I.E. THE RESTORED VALUE).  THE ROUTINE
                  ;              EXITS WITH THE INTERRUPTS ENABLED.
                  ;
 032E             RSTTF:
 032E    F3                DI                ; DISABLE INTERRUPTS WHILE RESTORING THINGS
 032F    21ED13            LXI     H,MSTAK   ; SET MONITOR STACK POINTER TO START OF STACK
 0332    F9                SPHL
 0333    D1                POP     D         ; START ALSO END OF REGISTER SAVE AREA
 0334    C1                POP     B
 0335    F1                POP     PSW
 0336    2AF713            LHLD    SSAVE     ; RESTORE USER STACK POINTER
 0339    F9                SPHL
 033A    2AF513            LHLD    PSAVE
 033D    E5                PUSH    H         ; PUT USER RETURN ADDRESS ON USER STACK
 033E    2AF313            LHLD    LSAVE     ; RESTORE HL REGISTERS
 0341    FB                EI                ; ENABLE INTERRUPTS NOW
 0342    C9                RET               ; JUMP TO RESTORED PC LOCATION
                  ;
                  ;
                  ;**********************************************************************
                  ;
                  ;
                  ; FUNCTION: SRET
                  ; INPUTS: NONE
                  ; OUTPUTS: CARRY = 1
                  ; CALLS: NOTHING
                  ; DESTROYS: CARRY
                  ; DESCRIPTION: SRET IS JUMPED TO BY ROUTINES WISHING TO RETURN SUCCESS.
                  ;              SRET SETS THE CARRY TRUE AND THEN RETURNS TO THE
                  ;              CALLER OF THE ROUTINE INVOKING SRET.
                  ;
 0343             SRET:
 0343    37                STC               ; SET CARRY TRUE
 0344   C9              RET                ; RETURN APPROPRIATELY
                 ;
                 ;
                 ;*****************************************************************
                 ;
                 ;
                 ; FUNCTION: STHF0
                 ; INPUTS: DE - 16 BIT ADDRESS OF BYTE TO BE STORED INTO
                 ; OUTPUTS: NONE
                 ; CALLS: STHLF
                 ; DESTROYS: A,B,C,H,L,F/F'S
                 ; DESCRIPTION: STHF0 CHECKS THE HALF BYTE FLAG IN TEMP TO SEE IF
                 ;              IT IS SET TO LOWER.  IF SO, STHF0 STORES A 0 TO
                 ;              PAD OUT THE LOWER HALF OF THE ADDRESSED BYTE;
                 ;              OTHERWISE, THE ROUTINE TAKES NO ACTION.
                 ;
 0345            STHF0:
 0345   3AF913          LDA     TEMP       ; GET HALF BYTE FLAG
 0348   B7              ORA     A          ; SET F/F'S
 0349   C0              RNZ                ; IF SET TO UPPER, DON'T DO ANYTHING
 034A   0E00            MVI     C,0        ; ELSE, WANT TO STORE THE VALUE 0
 034C   CD5003          CALL    STHLF      ; DO IT
 034F   C9              RET
                 ;
                 ;
                 ;*****************************************************************
                 ;
                 ;
                 ; FUNCTION: STHLF
                 ; INPUTS: C - 4 BIT VALUE TO BE STORED IN HALF BYTE
                 ;         DE - 16 BIT ADDRESS OF BYTE TO BE STORED INTO
                 ; OUTPUTS: NONE
                 ; CALLS: NOTHING
                 ; DESTROYS: A,B,C,H,L,F/F'S
                 ; DESCRIPTION: STHLF TAKES THE 4 BIT VALUE IN C AND STORES IT IN
                 ;              HALF OF THE BYTE ADDRESSED BY REGISTERS DE.  THE
                 ;              HALF BYTE USED (EITHER UPPER OR LOWER) IS DENOTED
                 ;              BY THE VALUE OF THE FLAG IN TEMP.  STHLF ASSUMES
                 ;              THAT THIS FLAG HAS BEEN PREVIOUSLY SET
                 ;              (NOMINALLY BY ICMD).
                 ;
 0350            STHLF:
 0350   D5              PUSH    D
 0351   E1              POP     H          ; MOVE ADDRESS OF BYTE INTO HL
 0352   79              MOV     A,C        ; GET VALUE
 0353   E60F            ANI     0FH        ; FORCE TO 4 BIT LENGTH
 0355   4F              MOV     C,A        ; PUT VALUE BACK
 0356   3AF913          LDA     TEMP       ; GET HALF BYTE FLAG
 0359   B7              ORA     A          ; CHECK FOR LOWER HALF
 035A   C26303          JNZ     STH05      ; BRANCH IF NOT
 035D   7E              MOV     A,M        ; ELSE, GET BYTE
 035E   E6F0          ANI     0F0H     ; CLEAR LOWER 4 BITS
 0360   B1            ORA     C        ; OR IN VALUE
 0361   77            MOV     M,A      ; PUT BYTE BACK
 0362   C9            RET
 0363           STH05:
 0363   7E            MOV     A,M      ; IF UPPER HALF, GET BYTE
 0364   E60F          ANI     0FH      ; CLEAR UPPER 4 BITS
 0366   47            MOV     B,A      ; SAVE BYTE IN B
 0367   79            MOV     A,C      ; GET VALUE
 0368   0F            RRC
 0369   0F            RRC
 036A   0F            RRC
 036B   0F            RRC              ; ALIGN TO UPPER 4 BITS
 036C   B0            ORA     B        ; OR IN ORIGINAL LOWER 4 BITS
 036D   77            MOV     M,A      ; PUT NEW CONFIGURATION BACK
 036E   C9            RET
                ;
                ;
                ;**************************************************************
                ;
                ;
                ; FUNCTION: VALDG
                ; INPUTS: C - ASCII CHARACTER
                ; OUTPUTS: CARRY - 1 IF CHARACTER REPRESENTS VALID HEX DIGIT
                ;                - 0 OTHERWISE
                ; CALLS: NOTHING
                ; DESTROYS: A,F/F'S
                ; DESCRIPTION: VALDG RETURNS SUCCESS IF ITS INPUT ARGUMENT IS
                ;              AN ASCII CHARACTER REPRESENTING A VALID HEX DIGIT
                ;                   `-F), AND FAILURE OTHERWISE.
                ;
 036F           VALDG:
 036F   79            MOV     A,C
 0370   FE30          CPI     '0'      ; TEST CHARACTER AGAINST '0'
 0372   FA1802        JM      FRET     ; IF ASCII CODE LESS, CANNOT BE VALID DIGIT
 0375   FE39          CPI     '9'      ; ELSE, SEE IF IN RANGE '0'-'9'
 0377   FA4303        JM      SRET     ; CODE BETWEEN '0' AND '9'
 037A   CA4303        JZ      SRET     ; CODE EQUAL '9'
 037D   FE41          CPI     'A'      ; NOT A DIGIT - TRY FOR A LETTER
 037F   FA1802        JM      FRET     ; NO - CODE BETWEEN '9' AND 'A'
 0382   FE47          CPI     'G'
 0384   F21802        JP      FRET     ; NO - CODE GREATER THAN 'F'
 0387   C34303        JMP     SRET     ; OKAY - CODE IS 'A' TO 'F', INCLUSIVE
                ;
                ;
                ;**************************************************************
                ;
                ;
                ; FUNCTION: VALDL
                ; INPUTS: C - CHARACTER
                ; OUTPUTS: CARRY - 1 IF INPUT ARGUMENT VALID DELIMTER
```

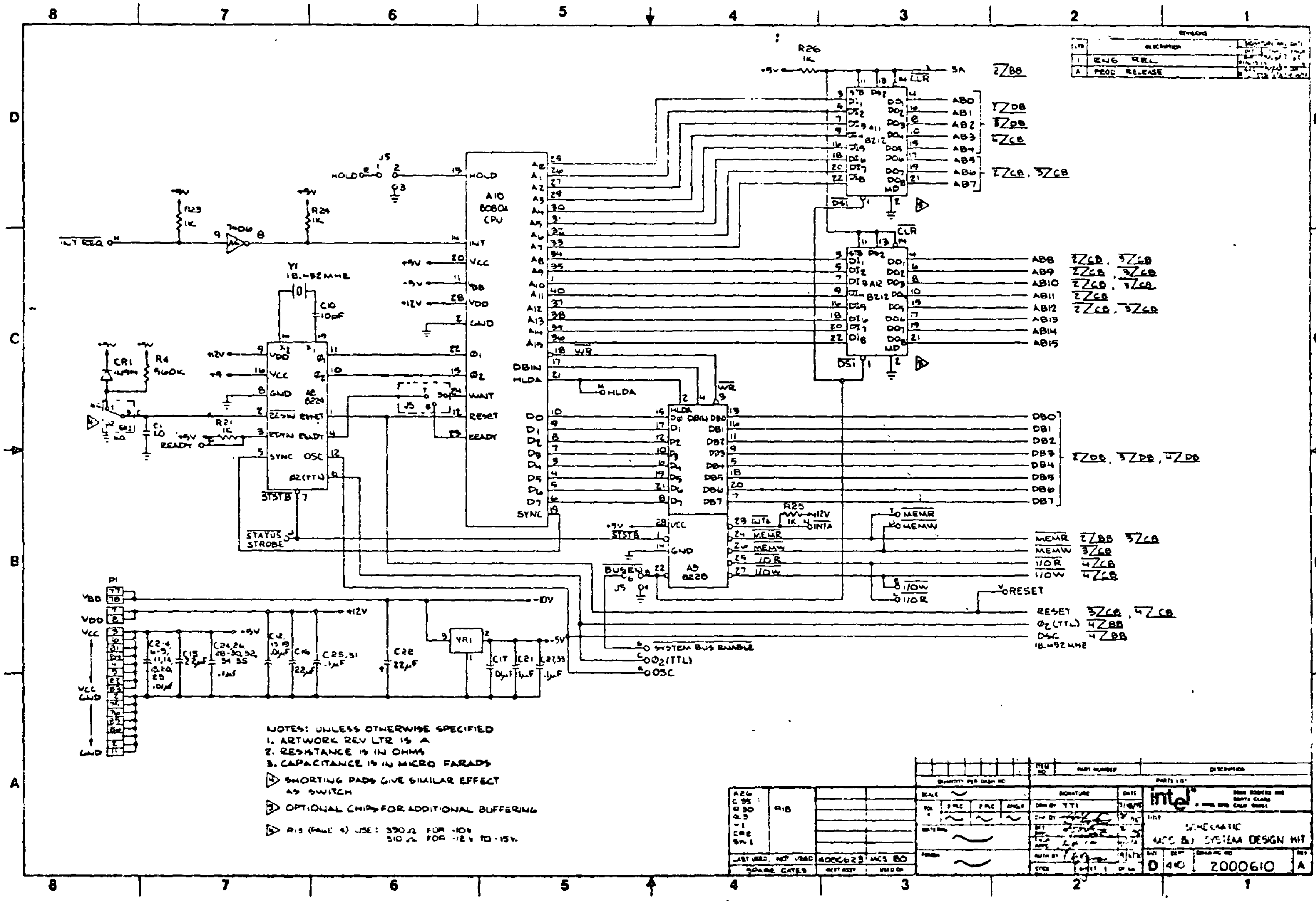
```
                        ;                    - 0 OTHERWISE
                        ; CALLS: NOTHING
                        ; DESTROYS: A,F/F'S
                        ; DESCRIPTION: VALDL RETURNS SUCCESS IF ITS INPUT ARGUMENT IS A VALID
                        ;              DELIMITER CHARACTER (SPACE, COMMA, CARRIAGE RETURN) AND
                        ;              FAILURE OTHERWISE.
                        ;
038A                    VALDL:
038A    79                      MOV     A,C
038B    FE2C                    CPI     ','         ; CHECK FOR COMMA
038D    CA4303                  JZ      SRET
0390    FE0D                    CPI     CR          ; CHECK FOR CARRIAGE RETURN
0392    CA4303                  JZ      SRET
0395    FE20                    CPI     ' '         ; CHECK FOR SPACE
0397    CA4303                  JZ      SRET
039A    C31802                  JMP     FRET        ; ERROR IF NONE OF THE ABOVE
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        ;
                        ;                         MONITOR TABLES
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        ;
039D                    SGNON:                              ; SIGNON MESSAGE
039D    0D0A4D43                DB      CR,LF,'MCS-80 KIT',CR,LF
03A1    532D3830
03A5    204B4954
03A9    0D0A
000E                    LSGNON          EQU     $-SGNON ; LENGTH OF SIGNON MESSAGE
                        ;
03AB                    CADR:                       ; TABLE OF ADDRESSES OF COMMAND ROUTINES
03AB    0000                    DW      0           ; DUMMY
03AD    4101                    DW      XCMD
03AF    1D01                    DW      SCMD
03B1    FD00                    DW      MCMD
03B3    B300                    DW      ICMD
03B5    9500                    DW      GCMD
03B7    5E00                    DW      DCMD
                        ;
03B9                    CTAB:                       ; TABLE OF VALID COMMAND CHARACTERS
03B9    44                      DB      'D'
03BA    47                      DB      'G'
03BB    49                      DB      'I'
03BC    4D                      DB      'M'
03BD    53                      DB      'S'
03BE    58                      DB      'X'
0006                    NCMDS   EQU     $-CTAB      ; NUMBER OF VALID COMMANDS
```

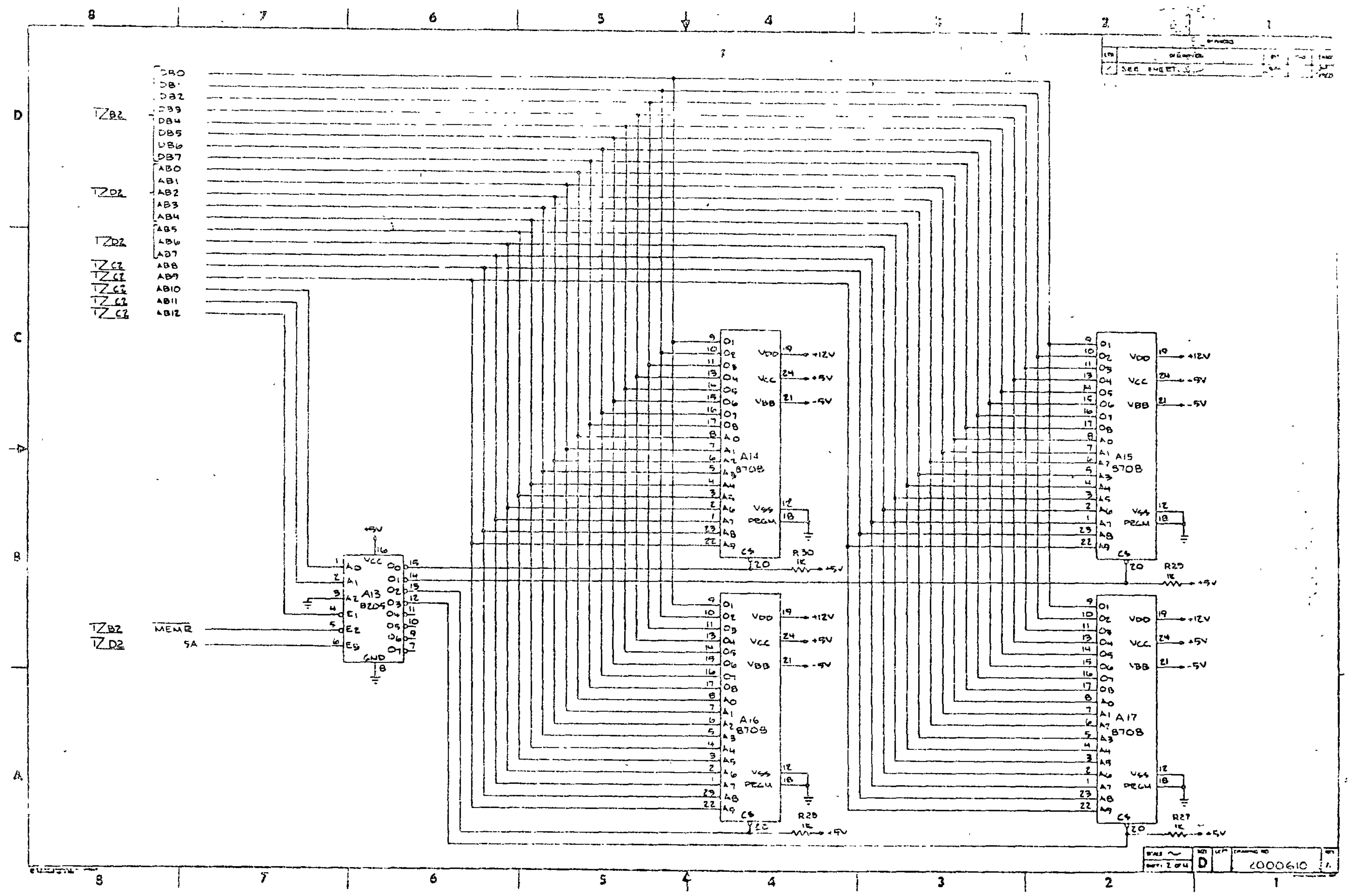
```
                        ;
03BF                    DIGTB:                          ; TABLE OF PRINT VALUES OF HEX DIGITS
03BF     30                   DB        '0'
03C0     31                   DB        '1'
03C1     32                   DB        '2'
03C2     33                   DB        '3'
03C3     34                   DB        '4'
03C4     35                   DB        '5'
03C5     36                   DB        '6'
03C6     37                   DB        '7'
03C7     38                   DB        '8'
03C8     39                   DB        '9'
03C9     41                   DB        'A'
03CA     42                   DB        'B'
03CB     43                   DB        'C'
03CC     44                   DB        'D'
03CD     45                   DB        'E'
03CE     46                   DB        'F'
                        ;
03CF                    RTAB:                   ; TABLE OF REGISTER INFORMATION
03CF     41                   DB        'A'     ; REGISTER IDENTIFIER
03D0     F2                   DB        ASAVE AND 0FFH  ; ADDRESS OF REGISTER SAVE LOCATION
03D1     00                   DB        0       ; LENGTH FLAG - 0=8 BITS, 1=16 BITS
0003                    RTABS EQU       $-RTAB  ; SIZE OF AN ENTRY IN THIS TABLE
03D2     42                   DB        'B'
03D3     F0                   DB        BSAVE AND 0FFH
03D4     00                   DB        0
03D5     43                   DB        'C'
03D6     EF                   DB        CSAVE AND 0FFH
03D7     00                   DB        0
03D8     44                   DB        'D'
03D9     EE                   DB        DSAVE AND 0FFH
03DA     00                   DB        0
03DB     45                   DB        'E'
03DC     ED                   DB        ESAVE AND 0FFH
03DD     00                   DB        0
03DE     46                   DB        'F'
03DF     F1                   DB        FSAVE AND 0FFH
03E0     00                   DB        0
03E1     48                   DB        'H'
03E2     F4                   DB        HSAVE AND 0FFH
03E3     00                   DB        0
03E4     4C                   DB        'L'
03E5     F3                   DB        LSAVE AND 0FFH
03E6     00                   DB        0
03E7     4D                   DB        'M'
03E8     F4                   DB        HSAVE AND 0FFH
03E9     01                   DB        1
03EA     50                   DB        'P'
03EB     F6                   DB        PSAVE+1 AND 0FFH
03EC     01                   DB        1
```

```
 03ED   53              DB      S
 03EE   F8              DB      SSAVE+1 AND 0FFH
 03EF   01              DB      1
 03F0   00              DB      0       END OF TABLE MARKERS
 03F1   00              DB      0
                  ;
 03FA                   ORG     BRTAB
                  ;
 03FA   C3E301          JMP     CO      ; BRANCH TABLE FOR USER ACCESSIBLE ROUTINES
 03FD   C3D001          JMP     CI
                  ;
                  ;
                  ;**********************************************************************
                  ;
                  ;
 1300                   ORG     DATA
 13ED                   ORG     REGS    ; ORG TO REGISTER SAVE - STACK GOES IN HERE
                  ;
 13ED             MSTAK         EQU     $       ; START OF MONITOR STACK
 13ED   00        ESAVE:        DB      0       ; E REGISTER SAVE LOCATION
 13EE   00        DSAVE:        DB      0       ; D REGISTER SAVE LOCATION
 13EF   00        CSAVE:        DB      0       ; C REGISTER SAVE LOCATION
 13F0   00        BSAVE:        DB      0       ; B REGISTER SAVE LOCATION
 13F1   00        FSAVE:        DB      0       ; FLAGS SAVE LOCATION
 13F2   00        ASAVE:        DB      0       ; A REGISTER SAVE LOCATION
 13F3   00        LSAVE:        DB      0       ; L REGISTER SAVE LOCATION
 13F4   00        HSAVE:        DB      0       ; H REGISTER SAVE LOCATION
 13F5   0000      PSAVE:        DW      0       ; PGM COUNTER SAVE LOCATION
 13F7   0000      SSAVE:        DW      0       ; USER STACK POINTER SAVE LOCATION
 13F9   00        TEMP:         DB      0       ; TEMPORARY MONITOR CELL
                  ;
 13FD                   ORG     BRLOC           ; ORG TO USER BRANCH LOCATION
                  ;
 0003             USRBR:        DS      3       ; BRANCH GOES IN HERE
                  ;
                  ;
                        END
NO PROGRAM ERRORS
```

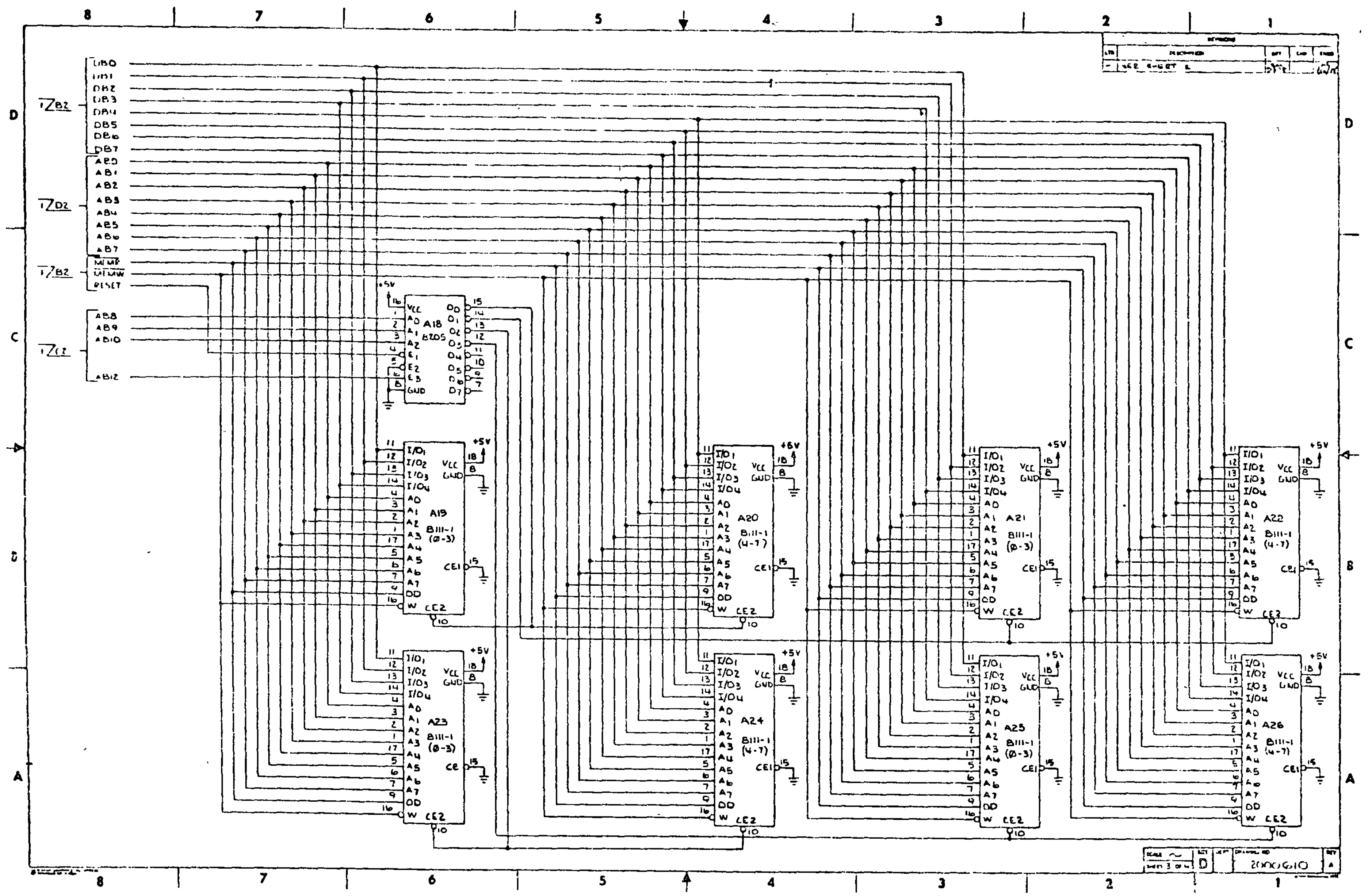
```
                          SYMBOL TABLE

  Ø1

 A       ØØØ7         ASAVE  13F2        B      ØØØØ        BRCHR  ØØ1B
 BREAK   Ø1BD         BRLOC  13FD        BRTAB  Ø3FA        BSAVE  13FØ
 C       ØØØ1         CADR   Ø3AB        CI     Ø1DØ        CMD    ØØ27
 CNCTL   ØØFB         CNIN   ØØFA        CNOUT  ØØFA        CNVBN  Ø1DA
 CO      Ø1E3         CONST  ØØFB        CR     ØØØD        CROUT  Ø1EE
 CSAVE   13EF         CTAB   Ø3B9        D      ØØØ2        DATA   13ØØ
 DCMØ5   ØØ65         DCM1Ø  ØØ7Ø        DCM12  ØØ85        DCM15  ØØ8B
 DCMD    ØØ5E         DIGTB  Ø3BF        DSAVE  13EE        E      ØØØ3
 ECHØ5   Ø1FD         ECH1Ø  Ø2ØB        ECHO   Ø1F4        ERROR  Ø2ØD
 ESAVE   13ED         ESC    ØØ1B        FALSE  ØF9C        FRET   Ø218
 FSAVE   13F1         GCMØ5  ØØAA        GCM1Ø  ØØBØ        GCMD   ØØ95
 GETCH   Ø21B         GETCM  ØØ2B        GETHX  Ø222        GETNM  Ø257
 GHXØ5   Ø228         GHX1Ø  Ø241        GNMØ5  Ø25E        GNM1Ø  Ø273
 GNM15   Ø281         GNM2Ø  Ø286        GNM25  Ø291        GNM3Ø  Ø295
 GO      Ø2Ø8 *       GTCØ3  ØØ3B        GTCØ5  ØØ48        GTC1Ø  ØØ54
 H       ØØØ4         HCHAR  ØØØF        HILØ5  Ø2BD        HILO   Ø29C
 HSAVE   13F4         ICMØ5  ØØBE        ICM1Ø  ØØE6        ICM2Ø  ØØEE
 ICM25   ØØF4         ICMD   ØØB3        INVRT  ØØFF        L      ØØØ5
 LF      ØØØA         LOWER  ØØØØ *      LSAVE  13F3        LSGNO  ØØØE
 M       ØØØ6         MCMØ5  Ø1Ø5        MCMD   ØØFD        MODE   ØØCF
 MSGL    ØØ22         MSTAK  13ED        NCMDS  ØØØ6        NEWLN  ØØØF
 NMOUT   Ø2C3         PRTYØ  ØØ7F        PRVAL  Ø2DE        PSAVE  13F5
 PSW     ØØØ6         RBR    ØØØ2        REGØ5  Ø2E9        REG1Ø  Ø2F3
 REG15   Ø3ØE         REGDS  Ø2E6        REGS   13ED        RGAØ5  Ø31D
 RGA1Ø   Ø32A         RGADR  Ø317        RSTTF  Ø32E        RSTU   ØØ38
 RTAB    Ø3CF         RTABS  ØØØ3        SCMØ5  Ø122        SCM1Ø  Ø12D
 SCM15   Ø13D         SCMD   Ø11D        SGNON  Ø39D        SP     ØØØ6
 SRET    Ø343         SSAVE  13F7        STHØ5  Ø363        STHFØ  Ø345
 STHLF   Ø35Ø         TEMP   13F9        TERM   ØØ1B        TRDY   ØØØ1
 TRUE    ØF9F         UPPER  ØØFF        USRBR  13FD        VALDG  Ø36F
 VALDL   Ø38A         XCMØ5  Ø154        XCM1Ø  Ø163        XCM15  Ø17Ø
 XCM18   Ø17B         XCM2Ø  Ø194        XCM25  Ø1AB        XCM27  Ø1AC
 XCM3Ø   Ø1B4         XCMD   Ø141

* Ø2

* Ø3

* Ø4

* Ø5

* Ø6

* Ø7

* Ø8

* Ø9

* 1Ø

* 11

* 12

* 13

..L
```

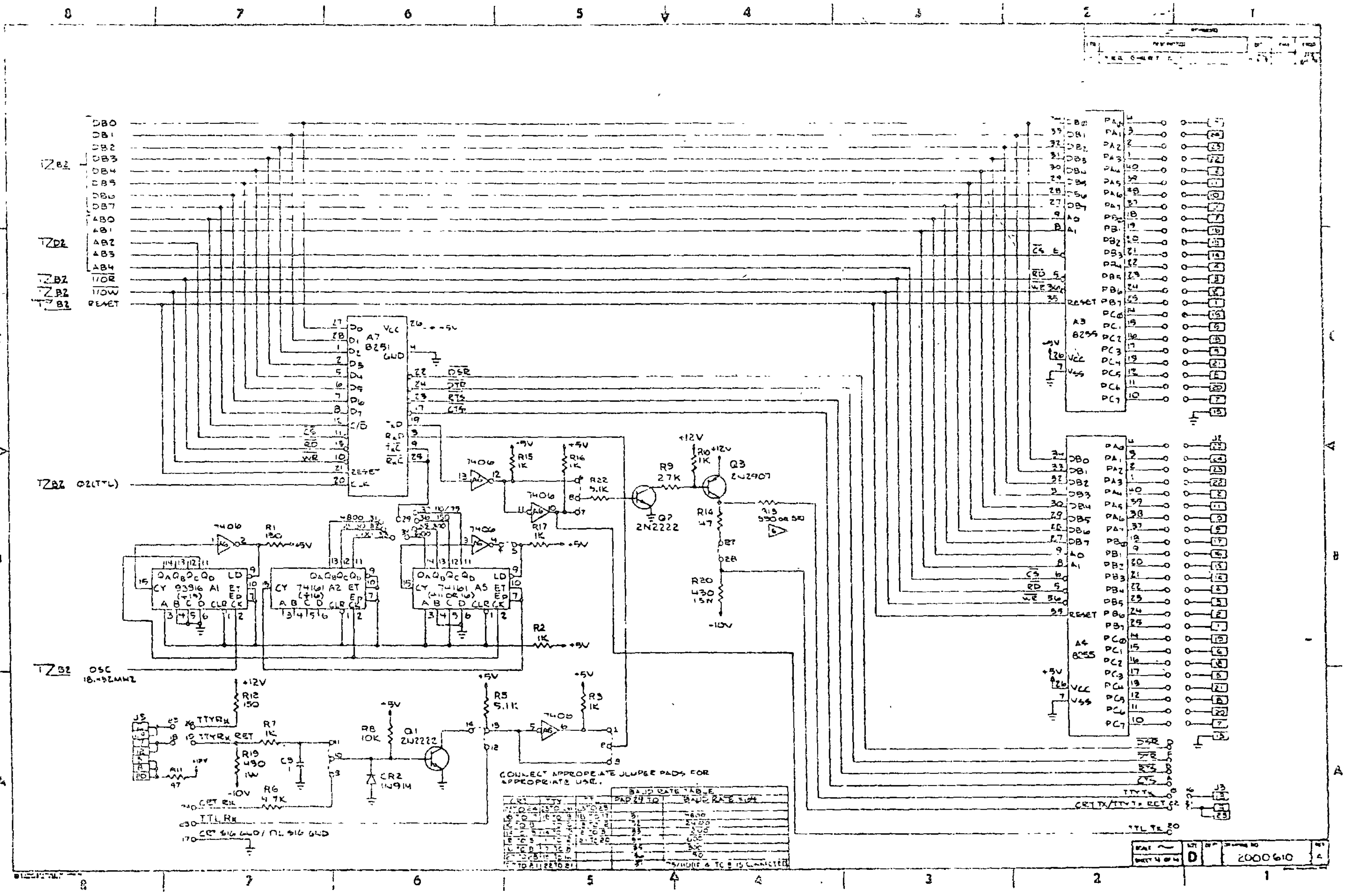
APPENDIX B. MCS-80™ SCHEMATICS





8 7 6 5 4 3 2 1
D C B A
TZB2
TZD2
TZD2
TZC2
TZC2
TZC2
TZC2
TZC2
TZB2
TZD2
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
AB0
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
AB10
AB11
AB12
MEMR
SA
A13
8205
VCC
GND
+5V
A14
8708
A15
8708
A16
8708
A17
8708
VDD
VCC
VBB
VSS
PRGM
CS
+12V
+5V
-5V
R30
R25
R28
R27
1K
SEE SHEET 1
D
2000610


DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
AB0
AB1
AB2
AB3
AB4
AB5
AB6
AB7
MEMR
MEMW
RESET
AB8
AB9
AB10
AB12
A18
8205
A19
8111-1
(Ø-3)
A20
8111-1
(4-7)
A21
8111-1
(Ø-3)
A22
8111-1
(4-7)
A23
8111-1
(Ø-3)
A24
8111-1
(4-7)
A25
8111-1
(Ø-3)
A26
8111-1
(4-7)
+5V
VCC
GND
I/O1
I/O2
I/O3
I/O4
A0
A1
A2
A3
A4
A5
A6
A7
OD
W
CE1
CE2
E1
E2
E3
D0
D1
D2
D3
D4
D5
D6
D7
2000610



CONNECT APPROPRIATE JUMPER PADS FOR APPROPRIATE USE.

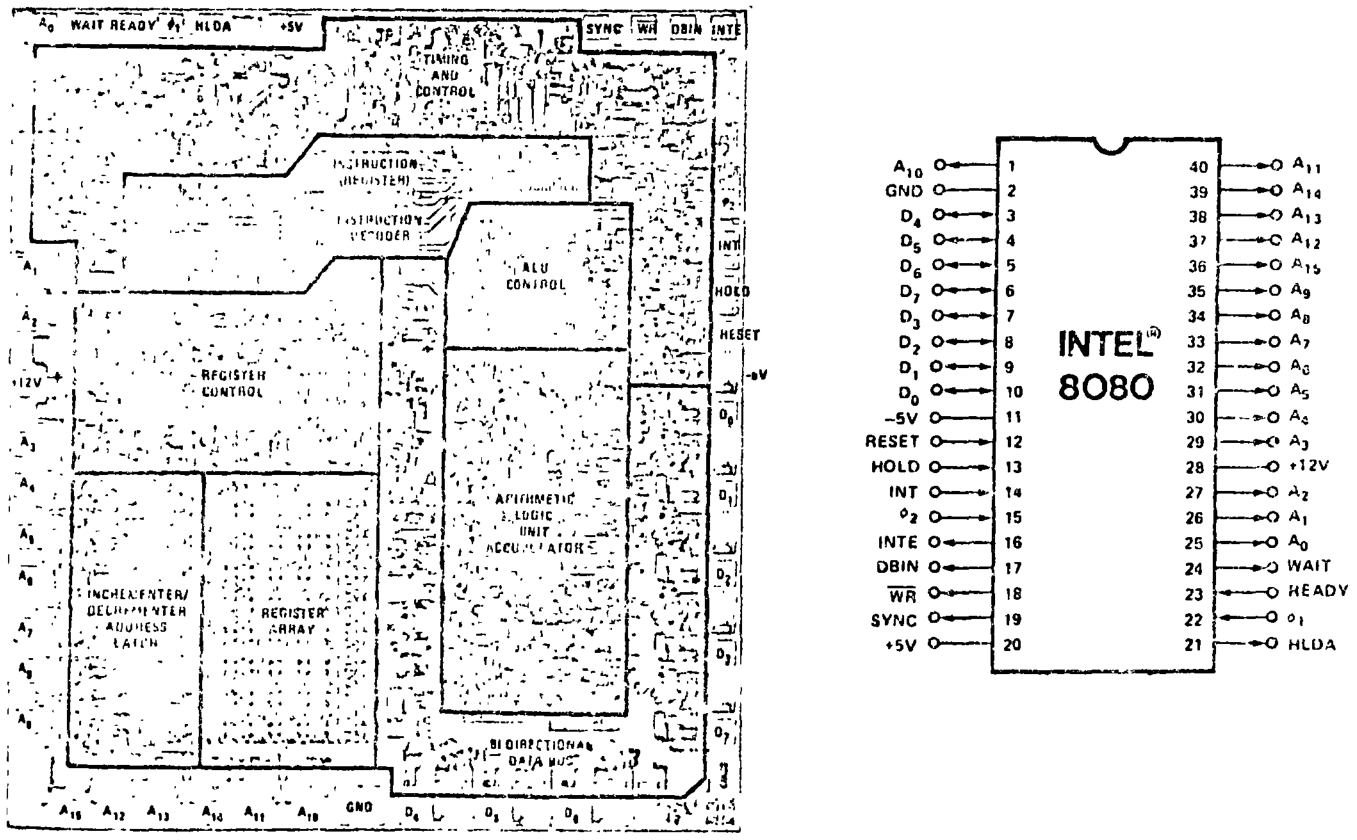
CRT	TTY	TTL
[illegible] TO 24	25 TO [illegible]	13 TO 25
6 TO 7	16 TO 9	18 TO 17
5 TO 8	[illegible]	8 TO 2
14 TO 3	4 TO [illegible]	2 TO 5
8 TO 5	1 TO [illegible]	21 TO 20
6 TO 8	7 TO 5	
27 TO 28	15 TO 14	
[illegible] TO 21	22 TO 21	

BAUD RATE TABLE

PAD 29 TO	BAUD RATE [illegible]
31	4800
32	2400
33	1200
34	600
35	300
36	110
37	75 (NOTE 4 TO 8 IS CONNECTED)

CHAPTER 2 THE 8080 CENTRAL PROCESSOR UNIT

The 8080 is a complete 8-bit parallel, central processor unit (CPU) for use in general purpose digital computer systems. It is fabricated on a single LSI chip (see Figure 3-1), using Intel's n-channel silicon gate MOS process. The 8080 transfers data and internal state information via an 8-bit, bidirectional 3-state Data Bus (D_0-D_7). Memory and peripheral device addresses are transmitted over a separate 16-bit 3-state Address Bus (A_0-A_{15}). Six timing and control outputs (SYNC, DBIN, WAIT, $\overline{WR}$, HLDA and INTE) emanate from the 8080, while four control inputs (READY, HOLD, INT and RESET), four power inputs (+12v, +5v, -5v, and GND) and two clock inputs (ϕ_1 and ϕ_2) are accepted by the 8080.



TIMING AND CONTROL
INSTRUCTION REGISTER
INSTRUCTION DECODER
ALU CONTROL
REGISTER CONTROL
ARITHMETIC LOGIC UNIT ACCUMULATOR
INCREMENTER/DECREMENTER ADDRESS LATCH
REGISTER ARRAY
BI DIRECTIONAL DATA BUS
INTEL® 8080
A10 1, GND 2, D4 3, D5 4, D6 5, D7 6, D3 7, D2 8, D1 9, D0 10, -5V 11, RESET 12, HOLD 13, INT 14, φ2 15, INTE 16, DBIN 17, WR 18, SYNC 19, +5V 20
40 A11, 39 A14, 38 A13, 37 A12, 36 A15, 35 A9, 34 A8, 33 A7, 32 A6, 31 A5, 30 A4, 29 A3, 28 +12V, 27 A2, 26 A1, 25 A0, 24 WAIT, 23 READY, 22 φ1, 21 HLDA

Figure 2-1. 8080 Photomicrograph With Pin Designations

ARCHITECTURE OF THE 8080 CPU

The 8080 CPU consists of the following functional units:

- Register array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bi-directional, 3-state data bus buffer

Figure 2-2 illustrates the functional blocks within the 8080 CPU.

Registers:

The register section consists of a static RAM array organized into six 16-bit registers:

- Program counter (PC)
- Stack pointer (SP)
- Six 8 bit general purpose registers arranged in pairs, referred to as B,C; D,E; and H,L
- A temporary register pair called W,Z

The program counter maintains the memory address of the current program instruction and is incremented automatically during every instruction fetch. The stack pointer maintains the address of the next available stack location in memory. The stack pointer can be initialized to use any portion of read-write memory as a stack. The stack pointer is decremented when data is "pushed" onto the stack and incremented when data is "popped" off the stack (i.e., the stack grows "downward").

The six general purpose registers can be used either as single registers (8-bit) or as register pairs (16-bit). The temporary register pair, W,Z, is not program addressable and is only used for the internal execution of instructions.

Eight-bit data bytes can be transferred between the internal bus and the register array via the register-select multiplexer. Sixteen-bit transfers can proceed between the register array and the address latch or the incrementer/decrementer circuit. The address latch receives data from any of the three register pairs and drives the 16 address output buffers (A_0-A_{15}), as well as the incrementer/decrementer circuit. The incrementer/decrementer circuit receives data from the address latch and sends it to the register array. The 16-bit data can be incremented or decremented or simply transferred between registers.

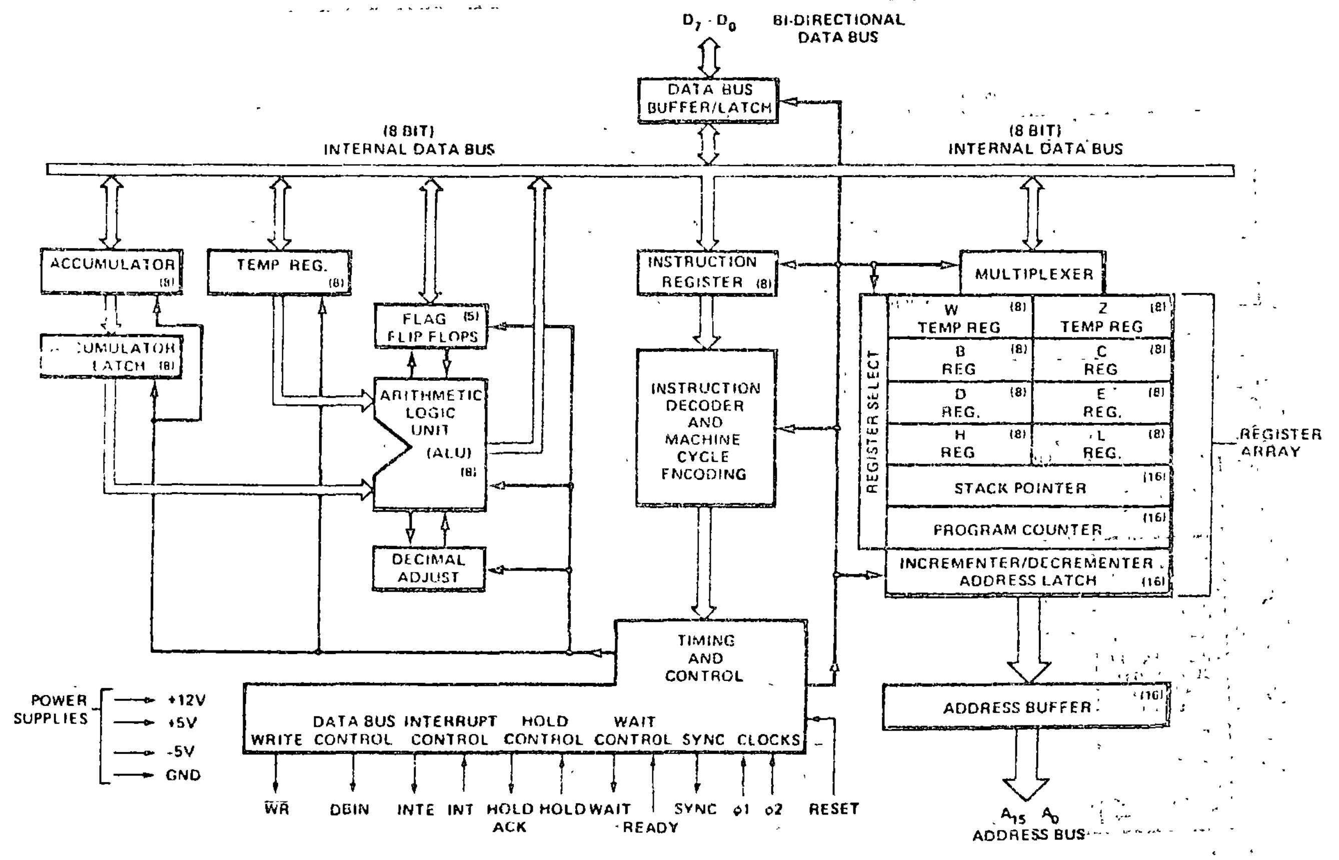


Figure 2-2. 8080 CPU Functional Block Diagram

Arithmetic and Logic Unit (ALU):

The ALU contains the following registers:

- An 8-bit accumulator
- An 8-bit temporary accumulator (ACT)
- A 5-bit flag register: zero, carry, sign, parity and auxiliary carry
- An 8-bit temporary register (TMP)

Arithmetic, logical and rotate operations are performed in the ALU. The ALU is fed by the temporary register (TMP) and the temporary accumulator (ACT) and carry flip-flop. The result of the operation can be transferred to the internal bus or to the accumulator; the ALU also feeds the flag register.

The temporary register (TMP) receives information from the internal bus and can send all or portions of it to the ALU, the flag register and the internal bus.

The accumulator (ACC) can be loaded from the ALU and the internal bus and can transfer data to the temporary accumulator (ACT) and the internal bus. The contents of the accumulator (ACC) and the auxiliary carry flip-flop can be tested for decimal correction during the execution of the DAA instruction (see Chapter 4).

Instruction Register and Control:

During an instruction fetch, the first byte of an instruction (containing the OP code) is transferred from the internal bus to the 8-bit instruction register.

The contents of the instruction register are, in turn, available to the instruction decoder. The output of the decoder, combined with various timing signals, provides the control signals for the register array, ALU and data buffer blocks. In addition, the outputs from the instruction decoder and external control signals feed the timing and state control section which generates the state and cycle timing signals.

Data Bus Buffer:

This 8-bit bidirectional 3-state buffer is used to isolate the CPU's internal bus from the external data bus (D_0 through D_7). In the output mode, the internal bus content is loaded into an 8-bit latch that, in turn, drives the data bus output buffers. The output buffers are switched off during input or non-transfer operations.

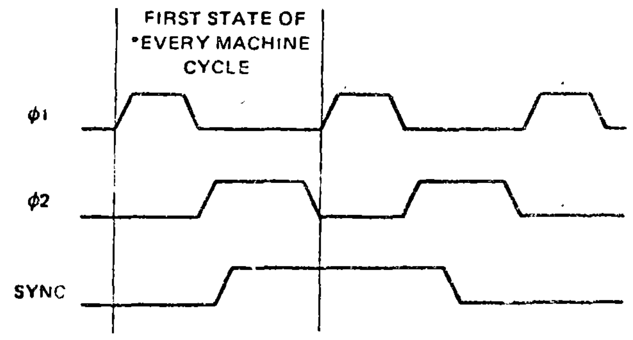
During the input mode, data from the external data bus is transferred to the internal bus. The internal bus is precharged at the beginning of each internal state, except for the transfer state (T_3—described later in this chapter).

THE PROCESSOR CYCLE

An instruction cycle is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction (one, two or three bytes) is extracted from memory and deposited in the CPU's instruction register. During the execution phase, the instruction is decoded and translated into specific processing activities.

Every instruction cycle consists of one, two, three, four or five machine cycles. A machine cycle is required each time the CPU accesses memory or an I/O port. The fetch portion of an instruction cycle requires one machine cycle for each byte to be fetched. The duration of the execution portion of the instruction cycle depends on the kind of instruction that has been fetched. Some instructions do not require any machine cycles other than those necessary to fetch the instruction; other instructions, however, require additional machine cycles to write or read data to/from memory or I/O devices. The DAD instruction is an exception in that it requires two additional machine cycles to complete an internal register-pair add (see Chapter 4).

Each machine cycle consists of three, four or five states. A state is the smallest unit of processing activity and is defined as the interval between two successive positive-going transitions of the ϕ_1 driven clock pulse. The 8080 is driven by a two-phase clock oscillator. All processing activities are referred to the period of this clock. The two non-overlapping clock pulses, labeled ϕ_1 and ϕ_2, are furnished by external circuitry. It is the ϕ_1 clock pulse which divides each machine cycle into states. Timing logic within the 8080 uses the clock inputs to produce a SYNC pulse, which identifies the beginning of every machine cycle. The SYNC pulse is triggered by the low-to-high transition of ϕ_2, as shown in Figure 2-3.



*SYNC DOES NOT OCCUR IN THE SECOND AND THIRD MACHINE CYCLES OF A DAD INSTRUCTION SINCE THESE MACHINE CYCLES ARE USED FOR AN INTERNAL REGISTER-PAIR ADD.

Figure 2-3. ϕ_1, ϕ_2 And SYNC Timing

There are three exceptions to the defined duration of a state. They are the WAIT state, the hold (HLDA) state and the halt (HLTA) state, described later in this chapter. Because the WAIT, the HLDA, and the HLTA states depend upon external events, they are by their nature of indeterminate length. Even these exceptional states, however, must be synchronized with the pulses of the driving clock. Thus, the duration of all states are integral multiples of the clock period.

To summarize then, each clock period marks a state; three to five states constitute a machine cycle; and one to five machine cycles comprise an instruction cycle. A full instruction cycle requires anywhere from four to eighteen states for its completion, depending on the kind of instruction involved.

Machine Cycle Identification:

With the exception of the DAD instruction, there is just one consideration that determines how many machine cycles are required in any given instruction cycle: the number of times that the processor must reference a memory address or an addressable peripheral device, in order to fetch and execute the instruction. Like many processors, the 8080 is so constructed that it can transmit only one address per machine cycle. Thus, if the fetch and execution of an instruction requires two memory references, then the instruction cycle associated with that instruction consists of two machine cycles. If five such references are called for, then the instruction cycle contains five machine cycles.

Every instruction cycle has at least one reference to memory, during which the instruction is fetched. An instruction cycle must always have a fetch, even if the execution of the instruction requires no further references to memory. The first machine cycle in every instruction cycle is therefore a FETCH. Beyond that, there are no fast rules. It depends on the kind of instruction that is fetched.

Consider some examples. The add-register (ADD r) instruction is an instruction that requires only a single machine cycle (FETCH) for its completion. In this one-byte instruction, the contents of one of the CPU's six general purpose registers is added to the existing contents of the accumulator. Since all the information necessary to execute the command is contained in the eight bits of the instruction code, only one memory reference is necessary. Three states are used to extract the instruction from memory, and one additional state is used to accomplish the desired addition. The entire instruction cycle thus requires only one machine cycle that consists of four states, or four periods of the external clock.

Suppose now, however, that we wish to add the contents of a specific memory location to the existing contents of the accumulator (ADD M). Although this is quite similar in principle to the example just cited, several additional steps will be used. An extra machine cycle will be used, in order to address the desired memory location.

The actual sequence is as follows. First the processor extracts from memory the one-byte instruction word addressed by its program counter. This takes three states. The eight-bit instruction word obtained during the FETCH machine cycle is deposited in the CPU's instruction register and used to direct activities during the remainder of the instruction cycle. Next, the processor sends out, as an address, the contents of its H and L registers. The eight-bit data word returned during this MEMORY READ machine cycle is placed in a temporary register inside the 8080 CPU. By now three more clock periods (states) have elapsed. In the seventh and final state, the contents of the temporary register are added to those of the accumulator. Two machine cycles, consisting of seven states in all, complete the "ADD M" instruction cycle.

At the opposite extreme is the save H and L registers (SHLD) instruction, which requires five machine cycles. During an "SHLD" instruction cycle, the contents of the processor's H and L registers are deposited in two sequentially adjacent memory locations; the destination is indicated by two address bytes which are stored in the two memory locations immediately following the operation code byte. The following sequence of events occurs:

(1) A FETCH machine cycle, consisting of four states. During the first three states of this machine cycle, the processor fetches the instruction indicated by its program counter. The program counter is then incremented. The fourth state is used for internal instruction decoding.

(2) A MEMORY READ machine cycle, consisting of three states. During this machine cycle, the byte indicated by the program counter is read from memory and placed in the processor's Z register. The program counter is incremented again.

(3) Another MEMORY READ machine cycle, consisting of three states, in which the byte indicated by the processor's program counter is read from memory and placed in the W register. The program counter is incremented, in anticipation of the next instruction fetch.

(4) A MEMORY WRITE machine cycle, of three states, in which the contents of the L register are transferred to the memory location pointed to by the present contents of the W and Z registers. The state following the transfer is used to increment the W,Z register pair so that it indicates the next memory location to receive data.

(5) A MEMORY WRITE machine cycle, of three states, in which the contents of the H register are transferred to the new memory location pointed to by the W,Z register pair.

In summary, the "SHLD" instruction cycle contains five machine cycles and takes 16 states to execute.

Most instructions fall somewhere between the extremes typified by the "ADD r" and the "SHLD" instructions. The input (INP) and the output (OUT) instructions, for example, require three machine cycles: a FETCH, to obtain the instruction; a MEMORY READ, to obtain the address of the object peripheral; and an INPUT or an OUTPUT machine cycle, to complete the transfer.

While no one instruction cycle will consist of more then five machine cycles, the following ten different types of machine cycles may occur within an instruction cycle:

(1) FETCH (M1)
(2) MEMORY READ
(3) MEMORY WRITE
(4) STACK READ
(5) STACK WRITE
(6) INPUT
(7) OUTPUT
(8) INTERRUPT
(9) HALT
(10) HALT•INTERRUPT

The machine cycles that actually do occur in a particular instruction cycle depend upon the kind of instruction, with the overriding stipulation that the first machine cycle in any instruction cycle is always a FETCH.

The processor identifies the machine cycle in progress by transmitting an eight-bit status word during the first state of every machine cycle. Updated status information is presented on the 8080's data lines (D_0-D_7), during the SYNC interval .This data should be saved in latches, and used to develop control signals for external circuitry. Table 2-1 shows how the positive-true status information is distributed on the processor's data bus.

Status signals are provided principally for the control of external circuitry. Simplicity of interface, rather than machine cycle identification, dictates the logical definition of individual status bits. You will therefore observe that certain processor machine cycles are uniquely identified by a single status bit, but that others are not. The M_1 status bit (D_6), for example, unambiguously identifies a FETCH machine cycle. A STACK READ, on the other hand, is indicated by the coincidence of STACK and MEMR signals. Machine cycle identification data is also valuable in the test and de-bugging phases of system development. Table 2-1 lists the status bit outputs for each type of machine cycle.

State Transition Sequence:

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T_1, T_2, T_3, T_4, T_5 or T_W). The actual number of states depends upon the instruction being executed, and on the particular machine cycle within the greater instruction cycle. The state transition diagram in Figure 2-4 shows how the 8080 proceeds from state to state in the course of a machine cycle. The diagram also shows how the READY, HOLD, and INTERRUPT lines are sampled during the machine cycle, and how the conditions on these lines may modify the basic transition sequence. In the present discussion, we are concerned only with the basic sequence and with the READY function. The HOLD and INTERRUPT functions will be discussed later.

The 8080 CPU does not directly indicate its internal state by transmitting a "state control" output during each state; instead, the 8080 supplies direct control output (INTE, HLDA, DBIN, $\overline{WR}$ and WAIT) for use by external circuitry.

Recall that the 8080 passes through at least three states in every machine cycle, with each state defined by successive low-to-high transitions of the ϕ_1 clock. Figure 2-5 shows the timing relationships in a typical FETCH machine cycle. Events that occur in each state are referenced to transitions of the ϕ_1 and ϕ_2 clock pulses.

The SYNC signal identifies the first state (T_1) in every machine cycle. As shown in Figure 2-5, the SYNC signal is related to the leading edge of the ϕ_2 clock. There is a delay (t_{DC}) between the low-to-high transition of ϕ_2 and the positive-going edge of the SYNC pulse. There also is a corresponding delay (also t_{DC}) between the next ϕ_2 pulse and the falling edge of the SYNC signal. Status information is displayed on D_0-D_7 during the same ϕ_2 to ϕ_2 interval. Switching of the status signals is likewise controlled by ϕ_2.

The rising edge of ϕ_2 during T_1 also loads the processor's address lines (A_0-A15). These lines become stable within a brief delay (t_{DA}) of the ϕ_2 clocking pulse, and they remain stable until the first ϕ_2 pulse after state T_3. This gives the processor ample time to read the data returned from memory..

Once the processor has sent an address to memory, there is an opportunity for the memory to request a WAIT. This it does by pulling the processor's READY line low, prior to the "Ready set-up" interval (t_{RS}) which occurs during the ϕ_2 pulse within state T_2 or T_W. As long as the READY line remains low, the processor will idle, giving the memory time to respond to the addressed data request. Refer to Figure 2-5.

The processor responds to a wait request by entering an alternative state (T_W) at the end of T_2, rather than proceeding directly to the T_3 state. Entry into the T_W state is indicated by a WAIT signal from the processor, acknowledging the memory's request. A low-to-high transition on the WAIT line is triggered by the rising edge of the ϕ_1 clock and occurs within a brief delay (t_{DC}) of the actual entry into the T_W state.

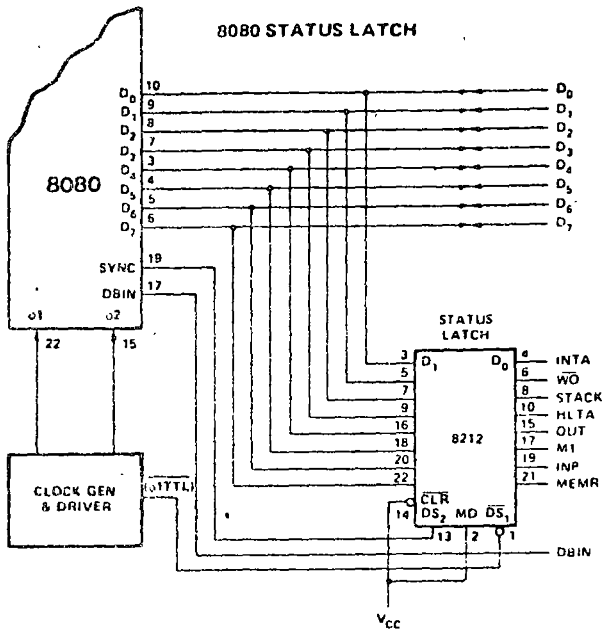
A wait period may be of indefinite duration. The processor remains in the waiting condition until its READY line again goes high. A READY indication must precede the falling edge of the ϕ_2 clock by a specified interval (t_{RS}), in order to guarantee an exit from the T_W state. The cycle may then proceed, beginning with the rising edge of the next ϕ_1 clock. A WAIT interval will therefore consist of an integral number of T_W states and will always be a multiple of the clock period.

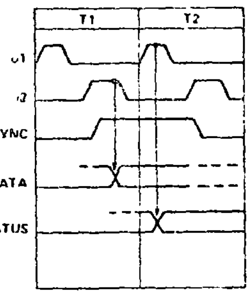
Instructions for the 8080 require from one to five machine cycles for complete execution. The 8080 sends out 8 bit of status information on the data bus at the beginning of each machine cycle (during SYNC time). The following table defines the status information.

STATUS INFORMATION DEFINITION

Symbols	Data Bus Bit	Definition
INTA*	D_0	Acknowledge signal for INTERRUPT request. Signal should be used to gate a restart instruction onto the data bus when DBIN is active.
$\overline{WO}$	D_1	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function ($\overline{WO}$ = 0). Otherwise, a READ memory or INPUT operation will be executed.
STACK	D_2	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
HLTA	D_3	Acknowledge signal for HALT instruction.
OUT	D_4	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when $\overline{WR}$ is active.
M_1	D_5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
INP*	D_6	Indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when DBIN is active.
MEMR*	D_7	Designates that the data bus will be used for memory read data.

*These three status bits can be used to control the flow of data onto the 8080 data bus.





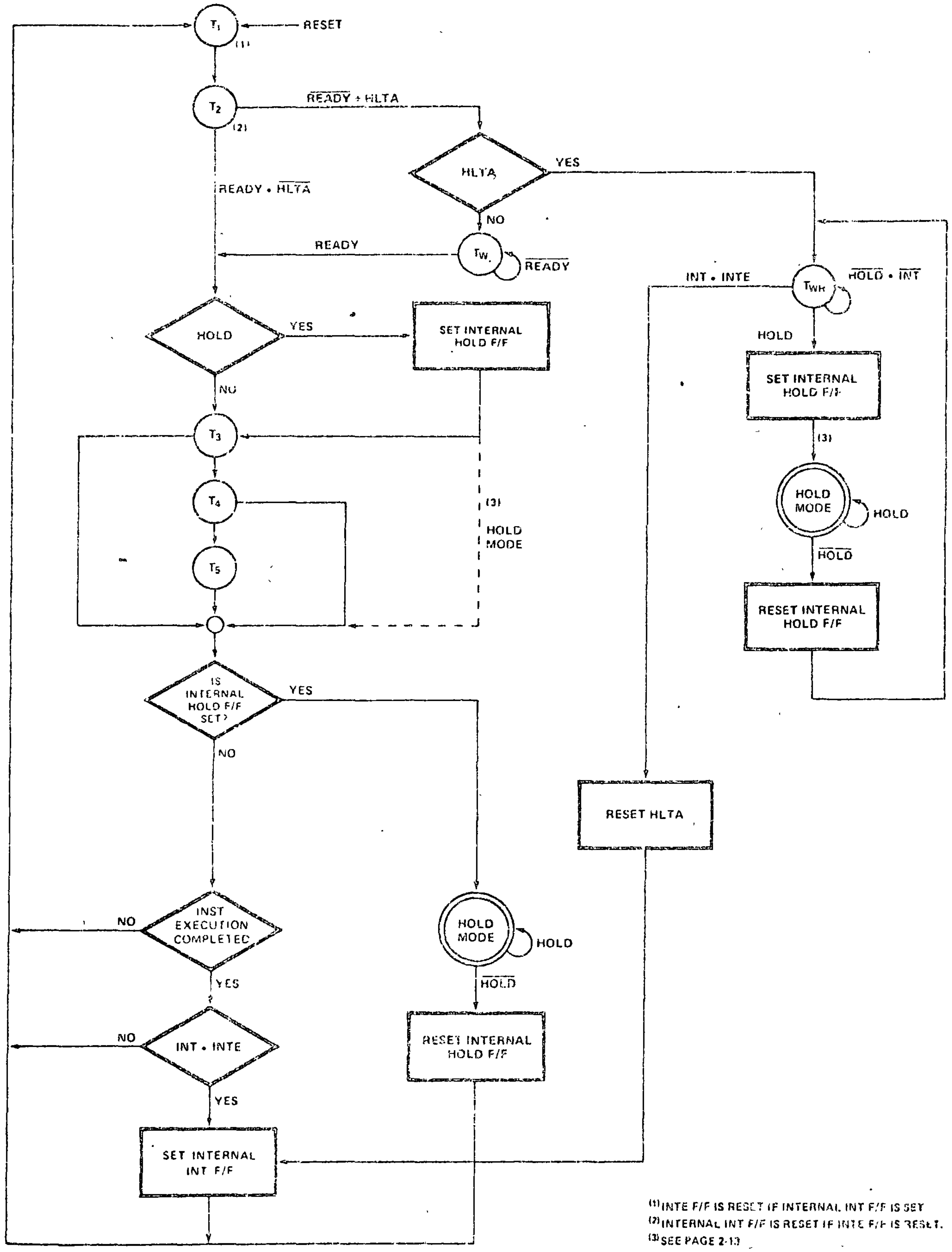
STATUS WORD CHART

TYPE OF MACHINE CYCLE

Ⓝ STATUS WORD

DATA BUS BIT	STATUS INFORMATION	INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Table 2-1. 8080 Status Bit Definitions



(1) INTE F/F IS RESET IF INTERNAL INT F/F IS SET
(2) INTERNAL INT F/F IS RESET IF INTE F/F IS RESET.
(3) SEE PAGE 2-13

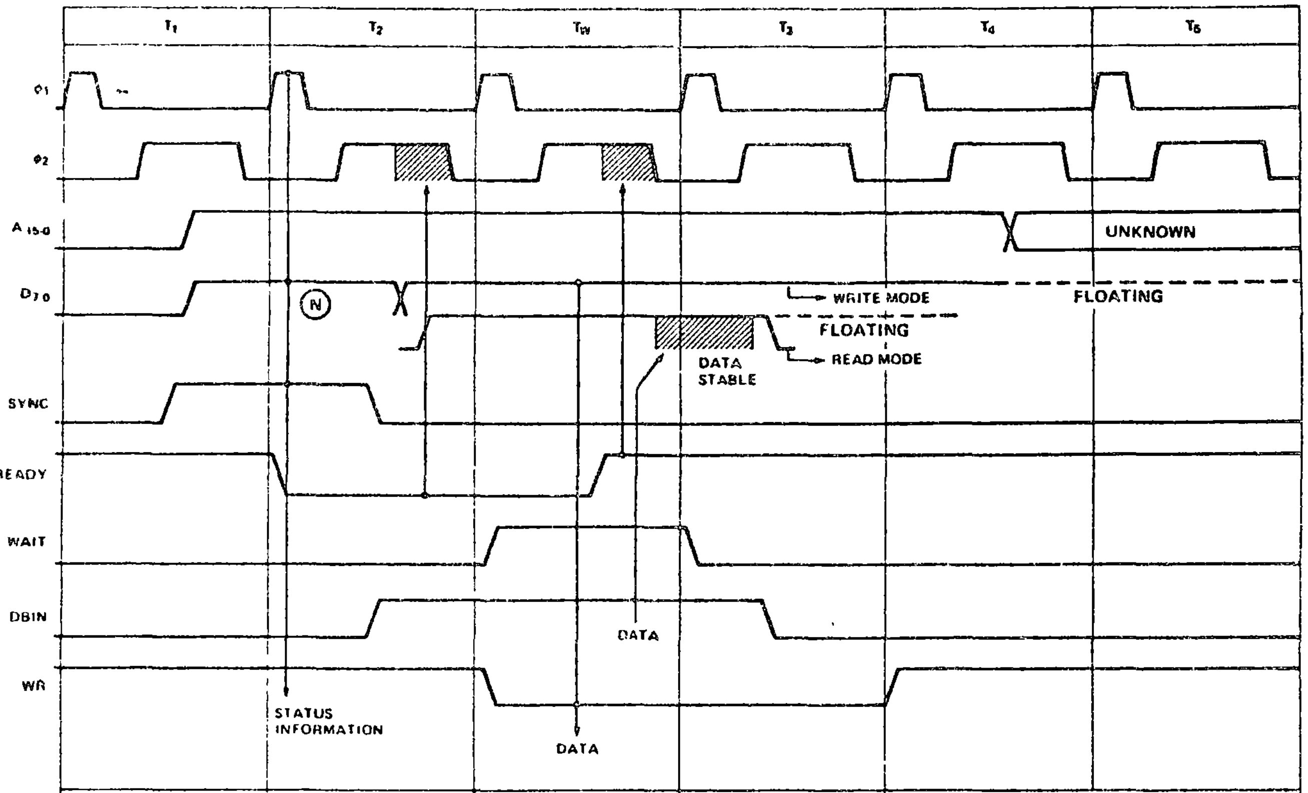
Figure 2-4. CPU State Transition Diagram

The events that take place during the T_3 state are determined by the kind of machine cycle in progress. In a FETCH machine cycle, the processor interprets the data on its data bus as an instruction. During a MEMORY READ or a STACK READ, data on this bus is interpreted as a data word. The processor outputs data on this bus during a MEMORY WRITE machine cycle. During I/O operations, the processor may either transmit or receive data, depending on whether an OUTPUT or an INPUT operation is involved.

Figure 2-6 illustrates the timing that is characteristic of a data input operation. As shown, the low-to-high transition of ϕ_2 during T_2 clears status information from the processor's data lines, preparing these lines for the receipt of incoming data. The data presented to the processor must have stabilized prior to both the "ϕ_1–data set-up" interval (t_{DS1}), that precedes the falling edge of the ϕ_1 pulse defining state T_3, and the "ϕ_2–data set-up" interval (t_{DS2}), that precedes the rising edge of ϕ_2 in state T_3. This same data must remain stable during the "data hold" interval (t_{DH}) that occurs following the rising edge of the ϕ_2 pulse. Data placed on these lines by memory or by other external devices will be sampled during T_3.

During the input of data to the processor, the 8080 generates a DBIN signal which should be used externally to enable the transfer. Machine cycles in which DBIN is available include: FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of ϕ_2 during state T2 and terminated by the corresponding edge of ϕ_2 during T_3. Any T_W phases intervening between T_2 and T_3 will therefore extend DBIN by one or more clock periods.

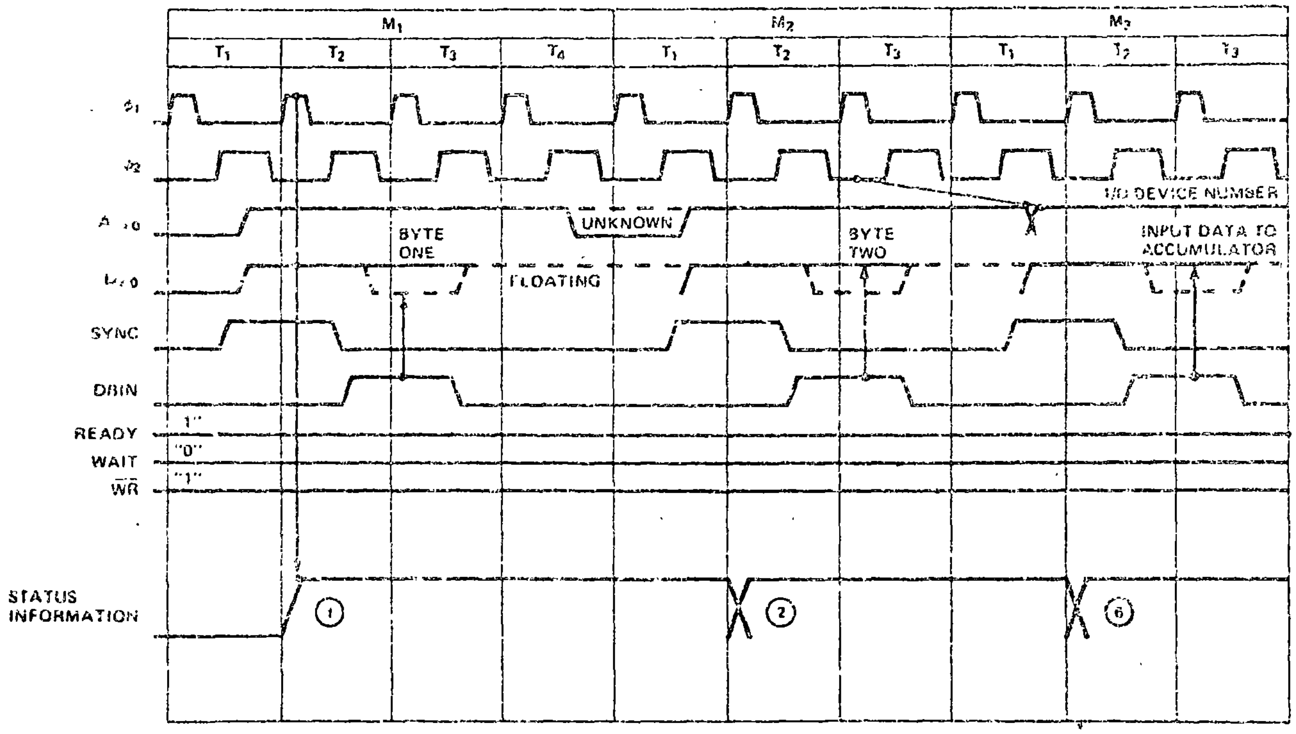
Figure 2-7 shows the timing of a machine cycle in which the processor outputs data. Output data may be destined either for memory or for peripherals. The rising edge of ϕ_2 within state T_2 clears status information from the CPU's data lines, and loads in the data which is to be output to external devices. This substitution takes place within the



A_{15-0} MEMORY ADDRESS OR I/O DEVICE NUMBER D_{7-0} STATUS INFORMATION: INTA, OUT, HLTA, $\overline{\text{WO}}$, MEMR, M_1, INP, STACK	SAMPLE READY HOLD AND HALT	OPTIONAL HALT OR MEMORY ACCESS TIME ADJUST	FETCH DATA OR INSTRUCTION OR WRITE DATA	OPTIONAL INSTRUCTION EXECUTION IF REQUIRED

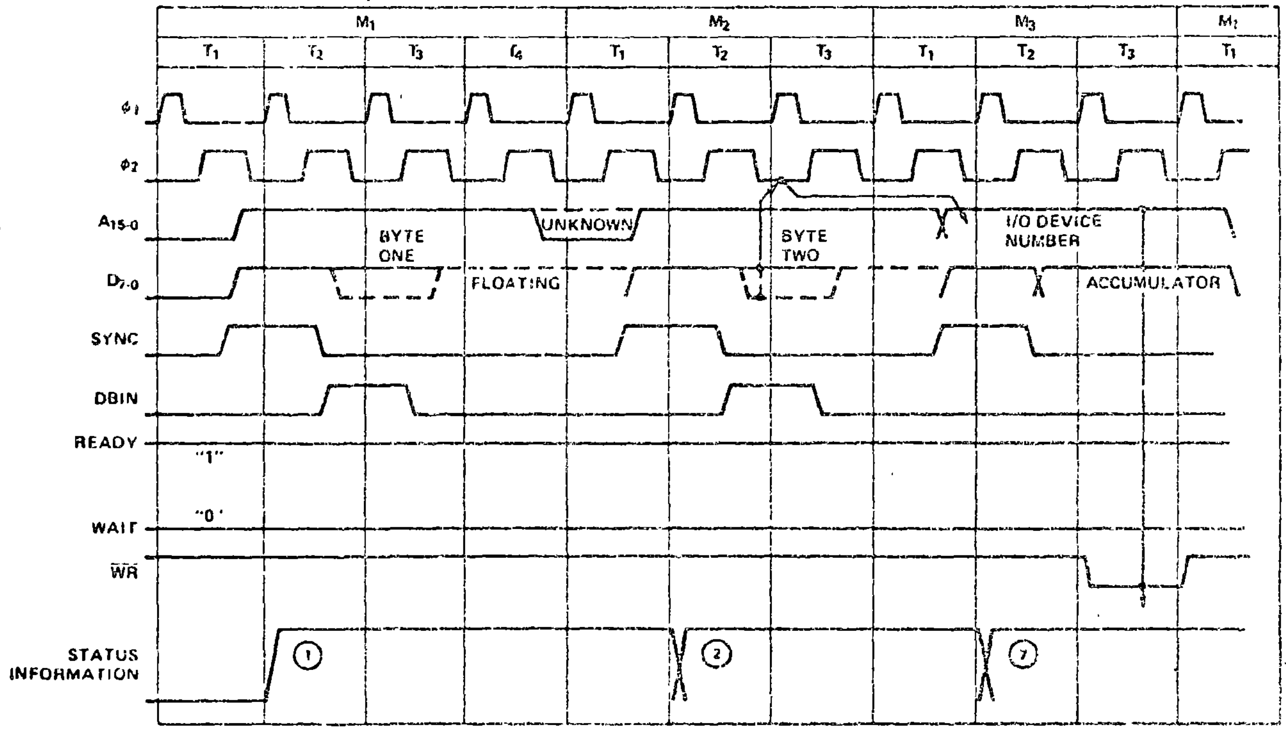
NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-5. Basic 8080 Instruction Cycle



NOTE: Ⓝ Refer to Status Word Chart on Page 2-6.

Figure 2-6. Input Instruction Cycle



NOTE: Ⓝ Refer to Status Word Chart on Page 2-6.

Figure 2-7. Output Instruction Cycle

"data output delay" interval (t_{DD}) following the ϕ_2 clock's leading edge. Data on the bus remains stable throughout the remainder of the machine cycle, until replaced by updated status information in the subsequent T_1 state. Observe that a READY signal is necessary for completion of an OUTPUT machine cycle. Unless such an indication is present, the processor enters the T_W state, following the T_2 state. Data on the output lines remains stable in the interim, and the processing cycle will not proceed until the READY line again goes high.

The 8080 CPU generates a $\overline{WR}$ output for the synchronization of external transfers, during those machine cycles in which the processor outputs data. These include MEMORY WRITE, STACK WRITE, and OUTPUT. The negative-going leading edge of $\overline{WR}$ is referenced to the rising edge of the first ϕ_1 clock pulse following T_2, and occurs within a brief delay (t_{DC}) of that event. $\overline{WR}$ remains low until re-triggered by the leading edge of ϕ_1 during the state following T_3. Note that any T_W states intervening between T_2 and T_3 of the output machine cycle will necessarily extend $\overline{WR}$, in much the same way that DBIN is affected during data input operations.

All processor machine cycles consist of at least three states: T_1, T_2, and T_3 as just described. If the processor has to wait for a response from the peripheral or memory with which it is communicating, then the machine cycle may also contain one or more T_W states. During the three basic states, data is transferred to or from the processor.

After the T_3 state, however, it becomes difficult to generalize. T_4 and T_5 states are available, if the execution of a particular instruction requires them. But not all machine cycles make use of these states. It depends upon the kind of instruction being executed, and on the particular machine cycle within the instruction cycle. The processor will terminate any machine cycle as soon as its processing activities are completed, rather than proceeding through the T_4 and T_5 states every time. Thus the 8080 may exit a machine cycle following the T_3, the T_4, or the T_5 state and proceed directly to the T_1 state of the next machine cycle.

STATE	ASSOCIATED ACTIVITIES
T_1	A memory address or I/O device number is placed on the Address Bus (A_{15-0}); status information is placed on Data Bus (D_{7-0}).
T_2	The CPU samples the READY and HOLD inputs and checks for halt instruction.
TW (optional)	Processor enters wait state if READY is low or if HALT instruction has been executed.
T3	An instruction byte (FETCH machine cycle), data byte (MEMORY READ, STACK READ) or interrupt instruction (INTERRUPT machine cycle) is input to the CPU from the Data Bus; or a data byte (MEMORY WRITE, STACK WRITE or OUTPUT machine cycle) is output onto the data bus.
T4 T5 (optional)	States T_4 and T_5 are available if the execution of a particular instruction requires them; if not, the CPU may skip one or both of them. T_4 and T_5 are only used for internal processor operations.

Table 2-2. State Definitions

INTERRUPT SEQUENCES

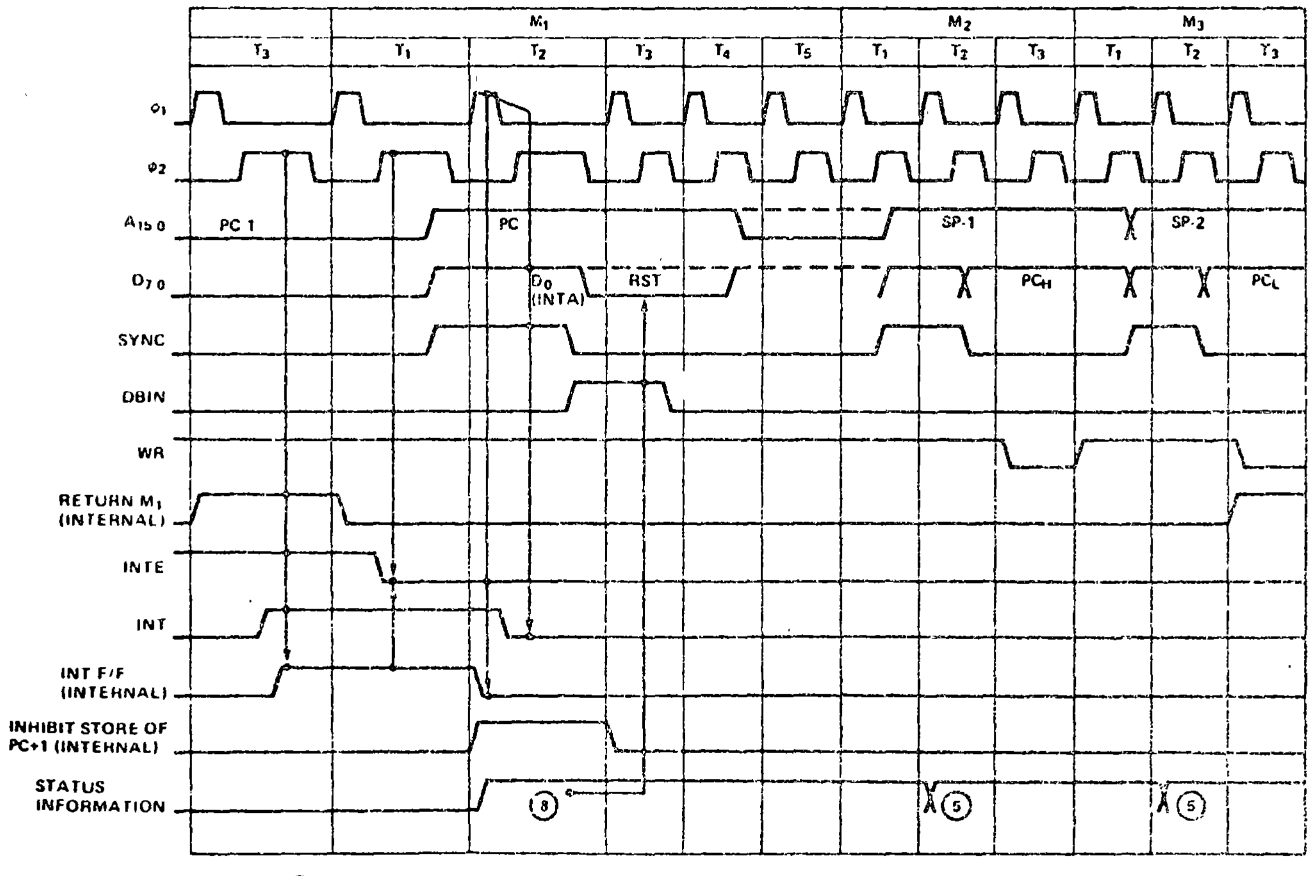
The 8080 has the built-in capacity to handle external interrupt requests. A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high.

The interrupt (INT) input is asynchronous, and a request may therefore originate at any time during any instruction cycle. Internal logic re-clocks the external request; so that a proper correspondence with the driving clock is established. As Figure 2-8 shows, an interrupt request (INT) arriving during the time that the interrupt enable line (INTE) is high, acts in coincidence with the ϕ_2 clock to set the internal interrupt latch. This event takes place during the last state of the instruction cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be processed.

The INTERRUPT machine cycle which follows the arrival of an enabled interrupt request resembles an ordinary FETCH machine cycle in most respects. The M_1 status bit is transmitted as usual during the SYNC interval. It is accompanied, however, by an INTA status bit (D_0) which acknowledges the external request. The contents of the program counter are latched onto the CPU's address lines during T_1, but the counter itself is not incremented during the INTERRUPT machine cycle, as it otherwise would be. In this way, the pre-interrupt status of the program counter is preserved, so that data in the counter may be restored by the interrupted program after the interrupt request has been processed.

The interrupt cycle is otherwise indistinguishable from an ordinary FETCH machine cycle. The processor itself takes no further special action. It is the responsibility of the peripheral logic to see that an eight-bit interrupt instruction is "jammed" onto the processor's data bus during state T_3. In a typical system, this means that the data-in bus from memory must be temporarily disconnected from the processor's main data bus, so that the interrupting device can command the main bus without interference.

The 8080's instruction set provides a special one-byte call which facilitates the processing of interrupts (the ordinary program Call takes three bytes). This is the RESTART instruction (RST). A variable three-bit field embedded in the eight-bit field of the RST enables the interrupting device to direct a Call to one of eight fixed memory locations. The decimal addresses of these dedicated locations are: 0, 8, 16, 24, 32, 40, 48, and 56. Any of these addresses may be used to store the first instruction(s) of a routine designed to service the requirements of an interrupting device. Since the (RST) is a call, completion of the instruction also stores the old program counter contents on the STACK.



NOTE: Ⓝ Refer to Status Word Chart on Page 2-6.

Figure 2-8. Interrupt Timing

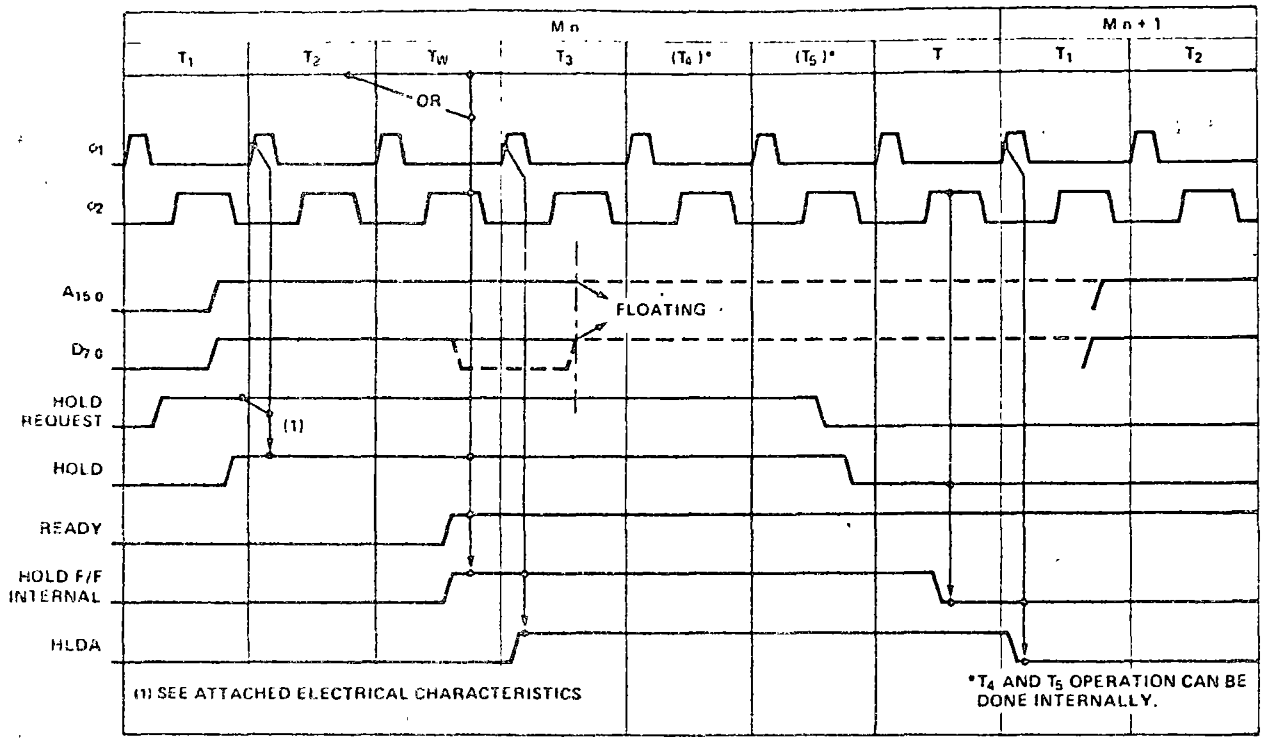


Figure 2-9. HOLD Operation (Read Mode)

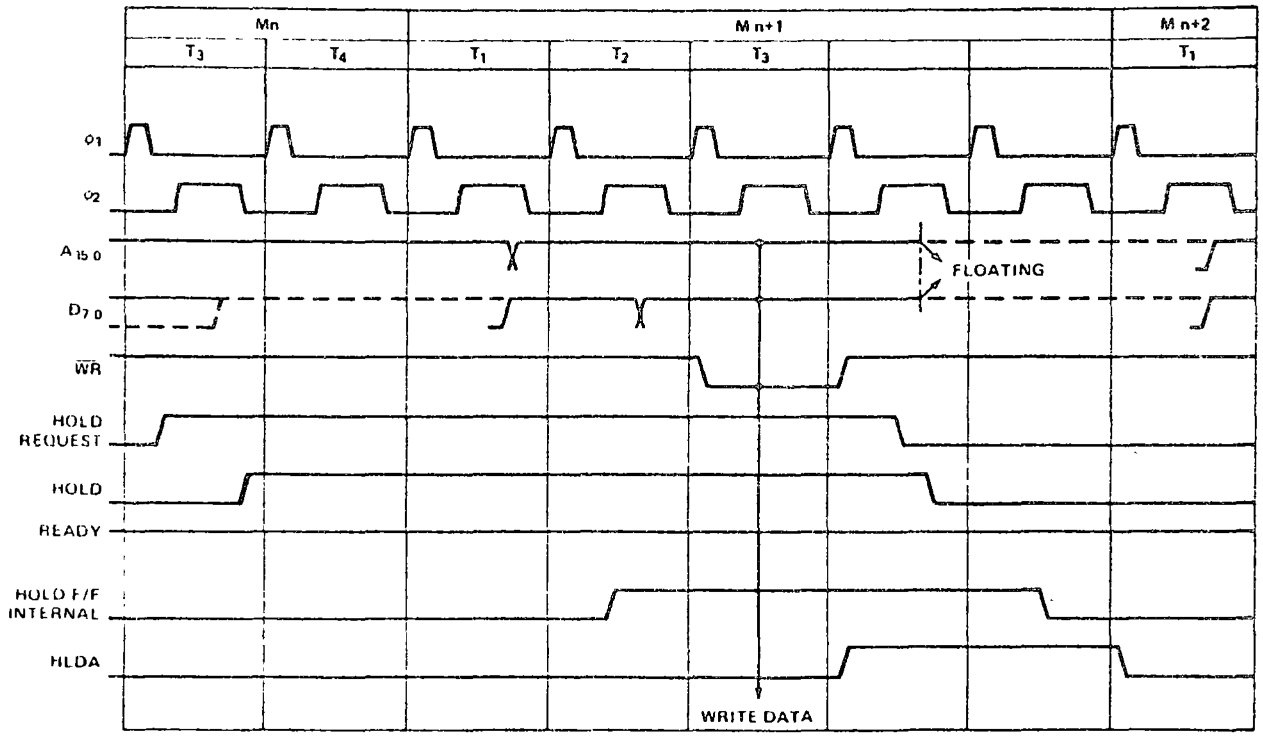


Figure 2-10. HOLD Operation (Write Mode)

HOLD SEQUENCES

The 8080 CPU contains provisions for Direct Memory Access (DMA) operations. By applying a HOLD to the appropriate control pin on the processor, an external device can cause the CPU to suspend its normal operations and relinquish control of the address and data busses. The processor responds to a request of this kind by floating its address to other devices sharing the busses. At the same time, the processor acknowledges the HOLD by placing a high on its HLDA outpin pin. During an acknowledged HOLD, the address and data busses are under control of the peripheral which originated the request, enabling it to conduct memory transfers without processor intervention.

Unlike the interrupt, the HOLD input must be synchronized with the driving clock. A HOLD signal must be stable prior to the "Hold set-up" interval (t_{HS}), that precedes the rising edge of ϕ_2, consequently, external re-clocking logic must be provided to properly synchronize an asynchronous HOLD REQUEST.

Figures 2-9 and 2-10 illustrate the timing involved in HOLD operations. Note the delay between the asynchronous HOLD REQUEST and the re-clocked HOLD. As shown in the diagram, a coincidence of the READY, the HOLD, and the ϕ_2 clocks sets the internal hold latch. Setting the latch enables the subsequent rising edge of the ϕ_1 clock pulse to trigger the HLDA output.

Acknowledgement of the HOLD REQUEST precedes slightly the actual floating of the processor's address and data lines. The processor acknowledges a HOLD at the beginning of T_3, if a read or an input machine cycle is in progress (see Figure 2-9). Otherwise, acknowledgement is deferred until the beginning of the state following T_3 (see Figure 2-10). In both cases, however, the HLDA goes high within a specified delay (t_{DC}) of the rising edge of the selected ϕ_1 clock pulse. Address and data lines are floated within a brief delay after the rising edge of the next ϕ_2 clock pulse. This relationship is also shown in the diagrams.

To all outward appearances, the processor has suspended its operations once the address and data busses are floated. Internally, however, certain functions may continue. If a HOLD REQUEST is acknowledged at T_3, and if the processor is in the middle of a machine cycle which requires four or more states to complete, the CPU proceeds through T_4 and T_5 before coming to a rest. Not until the end of the machine cycle is reached will processing activities cease. Internal processing is thus permitted to overlap the external DMA transfer, improving both the efficiency and the speed of the entire system.

The processor exits the holding state through a sequence similar to that by which it entered. A HOLD REQUEST is terminated asynchronously when the external device has completed its data transfer. External re-clocking logic must ensure that HOLD remains stable through a minimum "hold-time" interval (t_H) following the trailing edge of the subsequent ϕ_2 clock pulse. The HLDA output returns to a low level following the leading edge of the next $\phi 1$ clock pulse. Normal processing resumes with the machine cycle following the last cycle that was executed.

HALT SEQUENCES

When a halt instruction (HLT) is executed, the CPU enters the halt state (T_{WH}) after state T_2 of the next machine cycle, as shown in Figure 2-11. There are only three ways in which the 8080 can exit the halt state:

- A high on the RESET line will always reset the 8080 to state T_1; RESET also clears the program counter.
- A HOLD input will cause the 8080 to enter the hold state, as previously described. When the HOLD line goes low, the 8080 re-enters the halt state on the rising edge of the next ϕ_1 clock pulse.
- An interrupt (i.e., INT goes high while INTE is enabled) will cause the 8080 to exit the Halt state and enter state T_1 on the rising edge of the next ϕ_1 clock pulse. NOTE: The interrupt enable (INTE) flag must be set when the halt state is entered; otherwise, the 8080 will only be able to exit via a RESET signal.

Figure 2-12 illustrates halt sequencing in flow chart form.

START-UP OF THE 8080 CPU

When power is applied initially to the 8080, the processor begins operating immediately. The contents of its program counter, stack pointer, and the other working registers are naturally subject to random factors and cannot be specified. For this reason, it will be necessary to begin the power-up sequence with RESET.

An external RESET signal of three clock period duration (minimum) restores the processor's internal program counter to zero. Program execution thus begins with memory location zero, following a RESET. Systems which require the processor to wait for an explicit start-up signal will store a halt instruction (HLT) in this location. A manual or an automatic INTERRUPT will be used for starting. In other systems, the processor may begin executing its stored program immediately. Note, however, that the RESET has no effect on status flags, or on any of the processor's working registers (accumulator, registers, or stack pointer). The contents of these registers remain indeterminate, until initialized explicitly by the program.

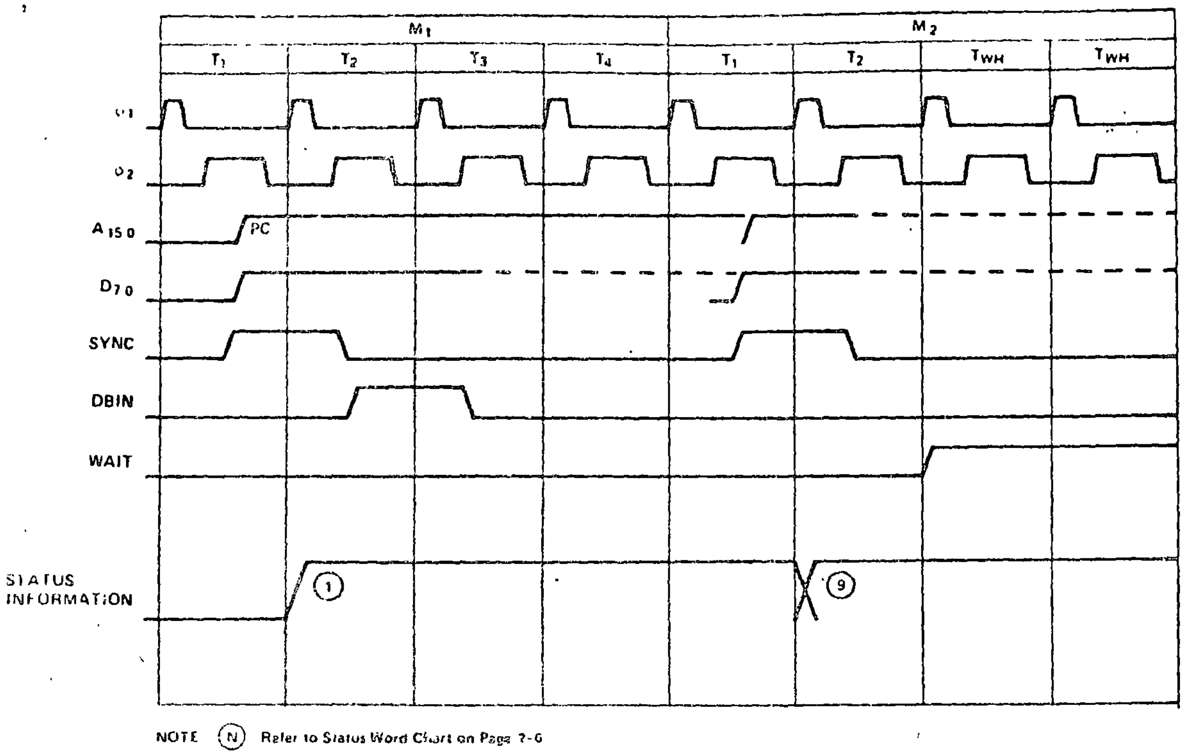


Figure 2-11. HALT Timing

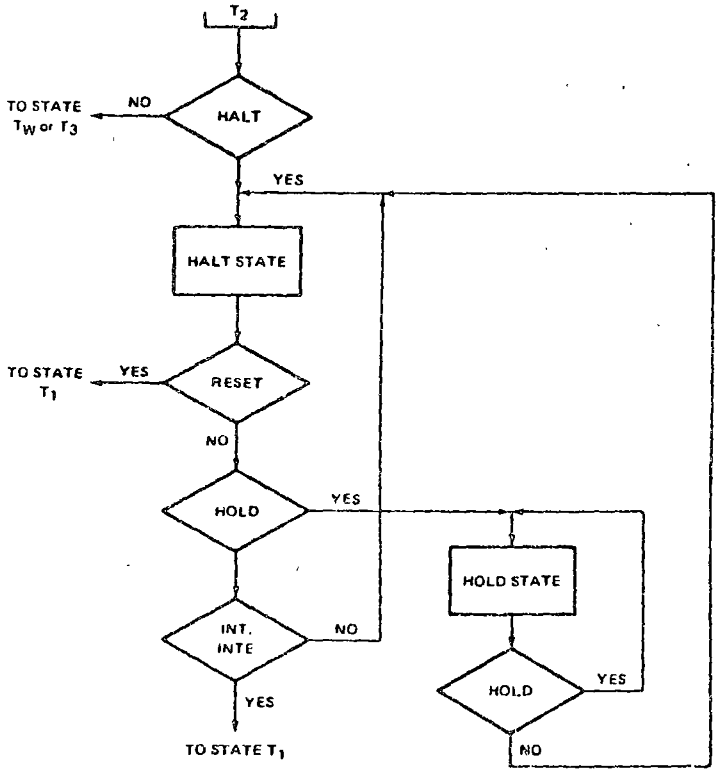


Figure 2-12. HALT Sequence Flow Chart.

CHAPTER 3 INTERFACING THE 8080

This chapter will illustrate, in detail, how to interface the 8080 CPU with Memory and I/O. It will also show the benefits and tradeoffs encountered when using a variety of system architectures to achieve higher throughput, decreased component count or minimization of memory size.

8080 Microcomputer system design lends itself to a simple, modular approach. Such an approach will yield the designer a reliable, high performance system that contains a minimum component count and is easy to manufacture and maintain.

The overall system can be thought of as a simple block diagram. The three (3) blocks in the diagram represent the functions common to any computer system.

CPU Module*	Contains the Central Processing Unit, system timing and interface circuitry to Memory and I/O devices.
Memory	Contains Read Only Memory (ROM) and Read/Write Memory (RAM) for program and data storage.
I/O	Contains circuitry that allows the computer system to communicate with devices or structures existing outside of the CPU or Memory array. for example: Keyboards, Floppy Disks, Paper Tape, etc.

There are three busses that interconnect these blocks:

Data Bus†	A bi-directional path on which data can flow between the CPU and Memory or I/O.
Address Bus	A uni-directional group of lines that identify a particular Memory location or I/O device.
Control Bus	A uni-directional set of signals that indicate the type of activity in current process.

Type of activities: 1. Memory Read
2. Memory Write
3. I/O Read
4. I/O Write
5. Interrupt Acknowledge

*"Module" refers to a functional block, it does not reference a printed circuit board manufactured by INTEL.

†"Bus" refers to a set of signals grouped together because of the similarity of their functions.

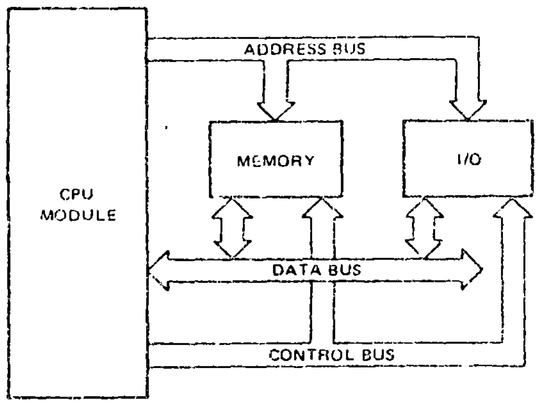


Figure 3-1. Typical Computer System Block Diagram

Basic System Operation

1. The CPU Module issues an activity command on the Control Bus.
2. The CPU Module issues a binary code on the Address Bus to identify which particular Memory location or I/O device will be involved in the current process activity.
3. The CPU Module receives or transmits data with the selected Memory location or I/O device.
4. The CPU Module returns to ① and issues the next activity command.

It is easy to see at this point that the CPU module is the central element in any computer system.

The following pages will cover the detailed design of the CPU Module with the 8080. The three Busses (Data, Address and Control) will be developed and the interconnection to Memory and I/O will be shown.

Design philosophies and system architectures presented in this manual are consistent with product development programs underway at INTEL for the MCS™-80. Thus, the designer who uses this manual as a guide for his total system engineering is assured that all new developments in components and software for MCS-80 from INTEL will be compatible with his design approach.

CPU Module Design

The CPU Module contains three major areas:

1. The 8080 Central Processing Unit
2. A Clock Generator and High Level Driver
3. A bi-directional Data Bus Driver and System Control Logic

The following will discuss the design of the three major areas contained in the CPU Module. This design is presented as an alternative to the Intel® 8224 Clock Generator and Intel 8228 System Controller. By studying the alternative approach, the designer can more clearly see the considerations involved in the specification and engineering of the 8224 and 8228. Standard TTL components and Intel general purpose peripheral devices are used to implement the design and to achieve operational characteristics that are as close as possible to those of the 8224 and 8228. Many auxiliary timing functions and features of the 8224 and 8228 are too complex to practically implement in standard components, so only the basic functions of the 8224 and 8228 are generated. Since significant benefits in system timing and component count reduction can be realized by using the 8224 and 8228, this is the preferred method of implementation.

1. **8080 CPU**

The operation of the 8080 CPU was covered in previous chapters of this manual, so little reference will be made to it in the design of the Module.

2. **Clock Generator and High Level Driver**

The 8080 is a dynamic device, meaning that its internal storage elements and logic circuitry require a timing reference (Clock), supplied by external circuitry, to refresh and provide timing control signals.

The 8080 requires two (2) such Clocks. Their waveforms must be non-overlapping, and comply with the timing and levels specified in the 8080 A.C. and D.C. Characteristics, page 5-15.

Clock Generator Design

The Clock Generator consists of a crystal controlled,

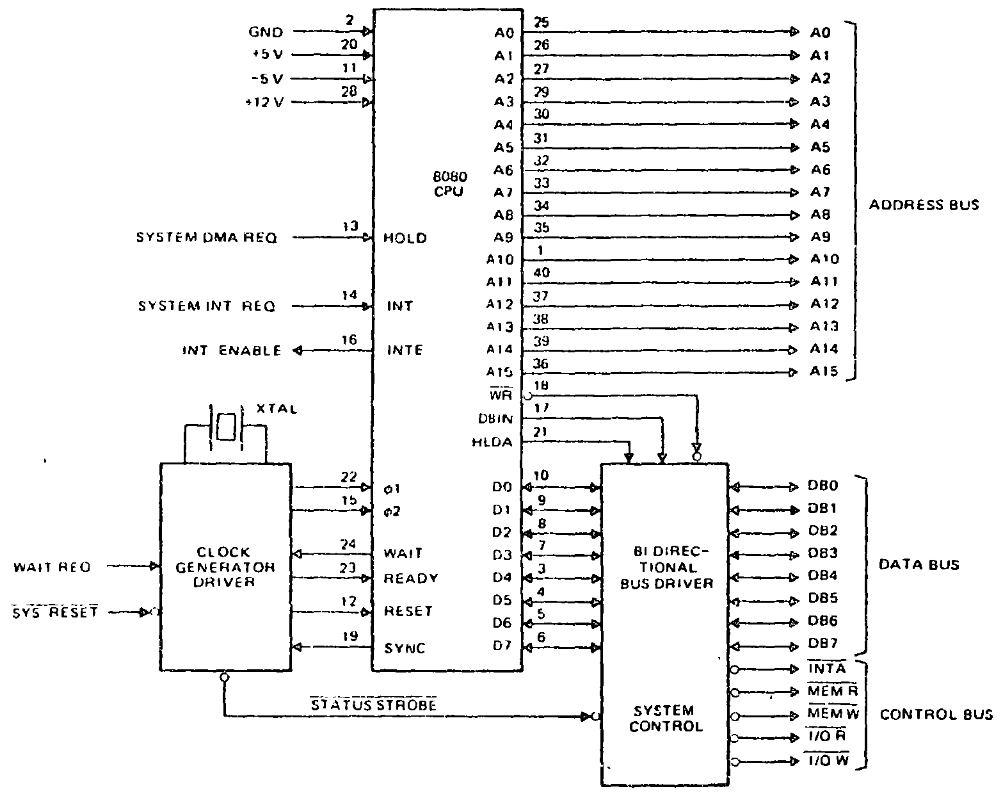


Figure 3-2. 8080 CPU Interface

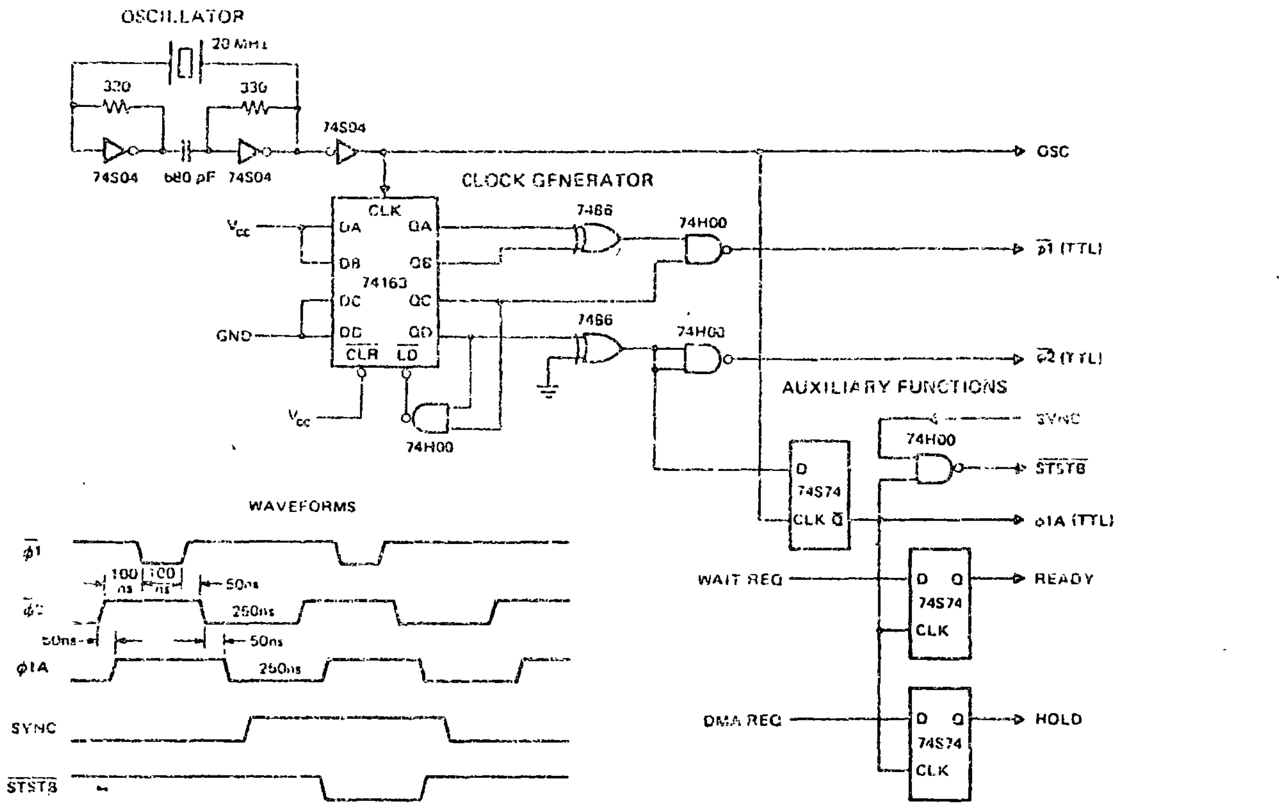


Figure 3-3. 8080 Clock Generator

20 MHZ oscillator, a four bit counter, and gating circuits.

The oscillator provides a 20 MHZ signal to the input of a four (4) bit, presettable, synchronous, binary counter. By presetting the counter as shown in figure 3-3 and clocking it with the 20 MHZ signal, a simple decoding of the counters outputs using standard TTL gates, provides proper timing for the two (2) 8080 clock inputs.

Note that the timing must actually be measured at the output of the High Level Driver to take into account the added delays and waveform distortions within such a device.

High Level Driver Design

The voltage level of the clocks for the 8080 is not TTL compatible like the other signals that input to the 8080. The voltage swing is from .6 volts (V_{ILC}) to 11 volts (V_{IHC}) with risetimes and falltimes under 50 ns. The Capacitive Drive is 20 pf (max.). Thus, a High Level Driver is required to interface the outputs of the Clock Generator (TTL) to the 8080.

The two (2) outputs of the Clock Generator are capacitivity coupled to a dual- High Level clock driver. The driver must be capable of complying with the 8080 clock input specifications, page 5-15. A driver of this type usually has little problem supplying the positive transition when biased from the 8080 V_{DD} supply (12V) but to achieve the low voltage specification (V_{ILC}) .8 volts Max. the driver is biased to the 8080 V_{BB} supply (-5V). This allows the driver to swing from GND to V_{DD} with the aid of a simple resistor divider.

A low resistance series network is added between the driver and the 8080 to eliminate any overshoot of the pulsed waveforms. Now a circuit is apparent that can easily comply with the 8080 specifications. In fact rise and falltimes of this design are typically less than 10 ns.

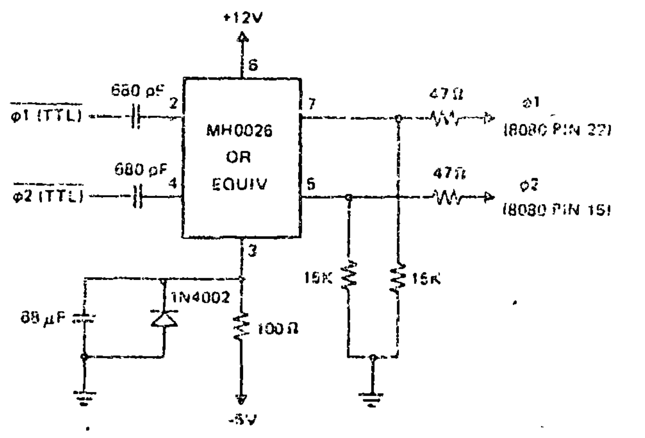


Figure 3-4. High Level Driver

Auxiliary Timing Signals and Functions

The Clock Generator can also be used to provide other signals that the designer can use to simplify large system timing or the interface to dynamic memories.

Functions such as power-on reset, synchronization of external requests (HOLD, READY, etc.) and single step, could easily be added to the Clock Generator to further enhance its capabilities.

For instance, the 20 MHZ signal from the oscillator can be buffered so that it could provide the basis for communication baud rate generation.

The Clock Generator diagram also shows how to generate an advanced timing signal (ϕ1A) that is handy to use in clocking "D" type flipflops to synchronize external requests. It can also be used to generate a strobe ($\overline{STSTB}$) that is the latching signal for the status information which is available on the Data Bus at the beginning of each machine cycle. A simple gating of the SYNC signal from the 8080 and the advanced (ϕ1A) will do the job. See Figure 3-3.

3. **Bi-Directional Bus Driver and System Control Logic**

The system Memory and I/O devices communicate with the CPU over the bi-directional Data Bus. The system Control Bus is used to gate data on and off the Data Bus within the proper timing sequences as dictated by the operation of the 8080 CPU. The data lines of the 8080 CPU, Memory and I/O devices are 3-state in nature, that is, their output drivers have the ability to be forced into a high-impedance mode and are, effectively, removed from the circuit. This 3-state bus technique allows the designer to construct a system around a single, eight (8) bit parallel, bi-directional Data Bus and simply gate the information on or off this bus by selecting or deselecting (3-stating) Memory and I/O devices with signals from the Control Bus.

Bi-Directional Data Bus Driver Design

The 8080 Data Bus (D7-D0) has two (2) major areas of concern for the designer:

1. Input Voltage level (V_{IH}) 3.3 volts minimum.
2. Output Drive Capability (I_{OL}) 1.7 mA maximum.

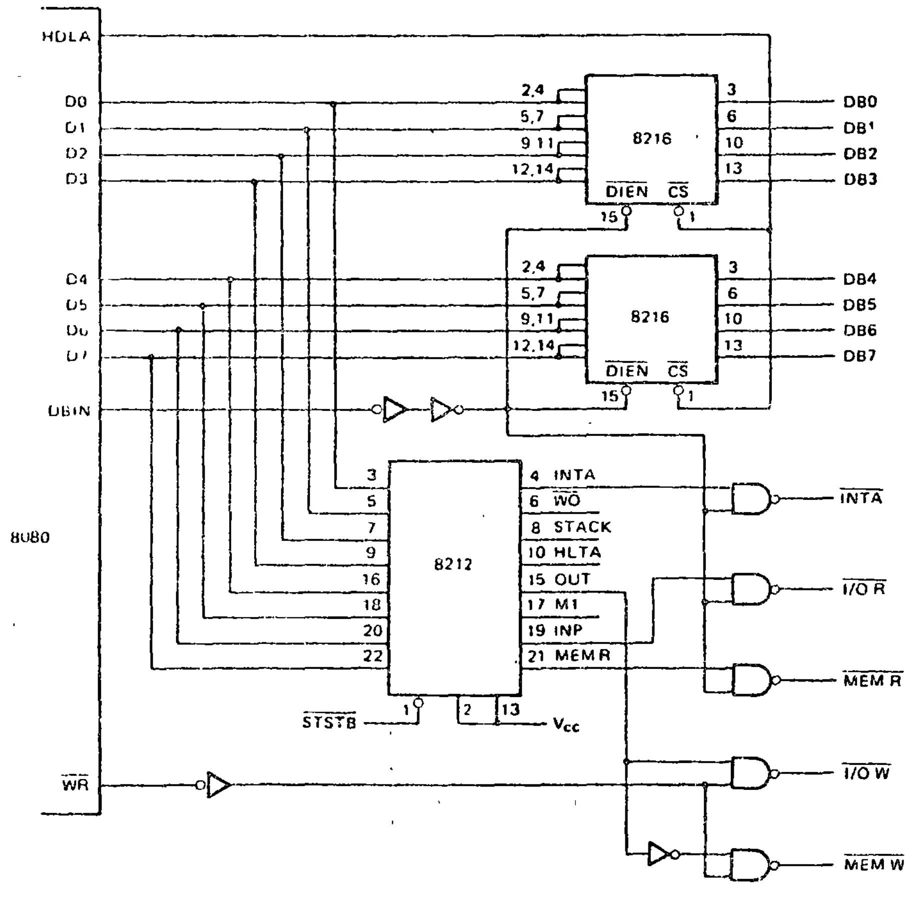


Figure 3-5. 8080 System Control

The input level specification implies that any semiconductor memory or I/O device connected to the 8080 Data Bus must be able to provide a minimum of 3.3 volts in its high state. Most semiconductor memories and standard TTL I/O devices have an output capability of between 2.0 and 2.8 volts, obviously a direct connection onto the 8080 Data Bus would require pullup resistors, whose value should not affect the bus speed or stress the drive capability of the memory or I/O components.

The 8080A output drive capability (I_{OL}) 1.9mA max. is sufficient for small systems where Memory size and I/O requirements are minimal and the entire system is contained on a single printed circuit board. Most systems however, take advantage of the high-performance computing power of the 8080 CPU and thus a more typical system would require some form of buffering on the 8080 Data Bus to support a larger array of Memory and I/O devices which are likely to be on separate boards.

A device specifically designed to do this buffering function is the INTEL® 8216, a (4) four bit bi-directional bus driver whose input voltage level is compatible with standard TTL devices and semiconductor memory components, and has output drive capability of 50 mA. At the 8080 side, the 8216 has a "high" output of 3.65 volts that not only meets the 8080 input spec but provides the designer with a worse case 350 mV noise margin.

A pair of 8216's are connected directly to the 8080 Data Bus (D7-D0) as shown in figure 3-5. Note that the DBIN signal from the 8080 is connected to the direction control input ($\overline{\text{DIEN}}$) so the correct flow of data on the bus is maintained. The chip select ($\overline{\text{CS}}$) of the 8216 is connected to the Hold Acknowledge (HLDA) of the 8080 to allow for DMA activities by deselecting the Data Bus Buffer and forcing the outputs of the 8216's into their high impedance (3-state) mode. This allows other devices to gain access to the data bus (DMA).

System Control Logic Design

The Control Bus maintains discipline of the bi-directional Data Bus, that is, it determines what type of device will have access to the bus (Memory or I/O) and generates signals to assure that these devices transfer Data with the 8080 CPU within the proper timing "windows" as dictated by the CPU operational characteristics.

As described previously, the 8080 issues Status information at the beginning of each Machine Cycle on its Data Bus to indicate what operation will take place during that cycle. A simple (8) bit latch, like an INTEL® 8212, connected directly to the 8080 Data Bus (D7-D0) as shown in figure 3-5 will store the Status information. The signal that loads the data into the Status Latch comes from the Clock Generator, it is Status Strobe ($\overline{\text{STSTB}}$) and occurs at the start of each Machine Cycle.

Note that the Status Latch is connected onto the 8080 Data Bus (D7-D0) before the Bus Buffer. This is to maintain the integrity of the Data Bus and simplify Control Bus timing in DMA dependent environments.

As shown in the diagram, a simple gating of the outputs of the Status Latch with the DBIN and $\overline{\text{WR}}$ signals from the 8080 generate the (4) four Control signals that make up the basic Control Bus.

These four signals: 1. Memory Read ($\overline{\text{MEM R}}$)

2. Memory Write ($\overline{\text{MEM W}}$)

3. I/O Read ($\overline{\text{I/O R}}$)

4. I/O Write ($\overline{\text{I/O W}}$)

connect directly to the MCS-80™ component "family" of ROMs, RAMs and I/O devices.

A fifth signal, Interrupt Acknowledge ($\overline{\text{INTA}}$) is added to the Control Bus by gating data off the Status Latch with the DBIN signal from the 8080 CPU. This signal is used to enable the Interrupt Instruction Port which holds the RST instruction onto the Data Bus.

Other signals that are part of the Control Bus such as $\overline{\text{WO}}$, Stack and M1 are present to aid in the testing of the System and also to simplify interfacing the CPU to dynamic memories or very large systems that require several levels of bus buffering.

Address Buffer Design

The Address Bus (A15-A0) of the 8080, like the Data Bus, is sufficient to support a small system that has a moderate size Memory and I/O structure, confined to a single card. To expand the size of the system that the Address Bus can support a simple buffer can be added, as shown in figure 3-6. The INTEL® 8212 or 8216 is an excellent device for this function. They provide low input loading (.25 mA), high output drive and insert a minimal delay in the System Timing.

Note that the Hold Acknowledge (HLDA) is connected to the buffers so that they are forced into their high-impedance (3-state) mode during DMA activities so that other devices can gain access to the Address Bus.

INTERFACING THE 8080 CPU TO MEMORY AND I/O DEVICES

The 8080 interfaces with standard semiconductor Memory components and I/O devices. In the previous text the proper control signals and buffering were developed which will produce a simple bus system similar to the basic system example shown at the beginning of this chapter.

In Figure 3-6 a simple, but exact 8080 typical system is shown that can be used as a guide for any 8080 system, regardless of size or complexity. It is a "three bus" architecture, using the signals developed in the CPU module.

Note that Memory and I/O devices interface in the same manner and that their isolation is only a function of the definition of the Read-Write signals on the Control Bus. This allows the 8080 system to be configured so that Memory and I/O are treated as a single array (memory mapped I/O) for small systems that require high thruput and have less than 32K memory size. This approach will be brought out later in the chapter.

ROM INTERFACE

A ROM is a device that stores data in the form of Program or other information such as "look-up tables" and is only read from, thus the term Read Only Memory. This type of memory is generally non-volatile, meaning that when the power is removed the information is retained. This feature eliminates the need for extra equipment like tape readers and disks to load programs initially, an important aspect in small system design.

Interfacing standard ROMs, such as the devices shown in the diagram is simple and direct. The output Data lines are connected to the bi-directional Data Bus, the Address inputs tie to the Address bus with possible decoding of the most significant bits as "chip selects" and the $\overline{\text{MEMR}}$ signal from the Control Bus connected to a "chip select" or data buffer. Basically, the CPU issues an address during the first portion of an instruction or data fetch (T1 & T2). This value on the Address Bus selects a specific location within the ROM, then depending on the ROM's delay (access time) the data stored at the addressed location is present at the Data output lines. At this time (T3) the CPU Data Bus is in the "input Mode" and the control logic issues a Memory Read command ($\overline{\text{MEMR}}$) that gates the addressed data on to the Data Bus.

RAM INTERFACE

A RAM is a device that stores data. This data can be program, active "look-up tables," temporary values or external stacks. The difference between RAM and ROM is that data can be written into such devices and are in essence, Read/Write storage elements. RAMs do not hold their data when power is removed so in the case where Program or "look-up tables" data is stored a method to load

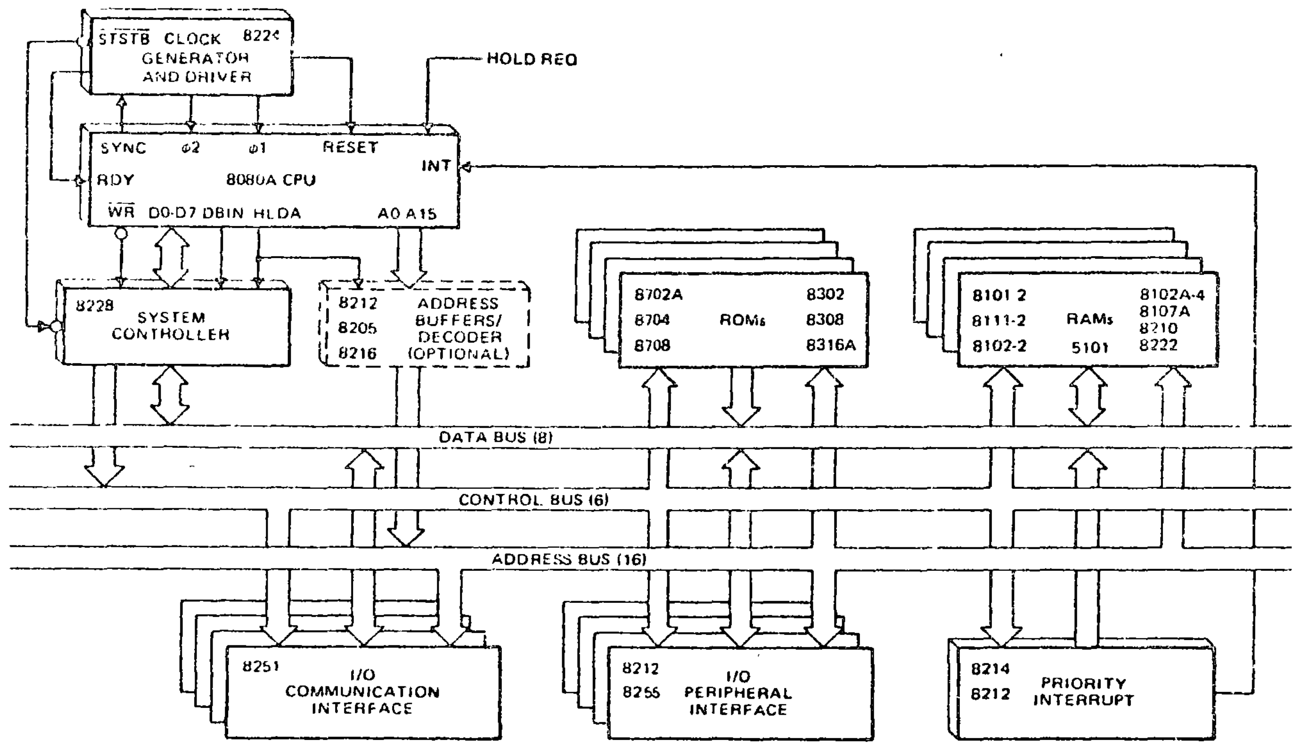

STSTB CLOCK 8224
GENERATOR AND DRIVER
HOLD REQ
SYNC ϕ2 ϕ1 RESET
INT
RDY 8080A CPU
WR D0-D7 DBIN HLDA A0-A15
8228 SYSTEM CONTROLLER
8212 8205 8216 ADDRESS BUFFERS/ DECODER (OPTIONAL)
8702A 8704 8708 ROMs 8302 8308 8316A
8101-2 8111-2 8102-2 RAMs 5101 8102A-4 8107A 8210 8222
DATA BUS (8)
CONTROL BUS (6)
ADDRESS BUS (16)
8251 I/O COMMUNICATION INTERFACE
8212 8255 I/O PERIPHERAL INTERFACE
8214 8212 PRIORITY INTERRUPT

Figure 3-6. Microcomputer System

RAM memory must be provided, such as: Floppy Disk, Paper Tape, etc.

The CPU treats RAM in exactly the same manner as ROM for addressing data to be read. Writing data is very similar; the RAM is issued an address during the first portion of the Memory Write cycle (T1 & T2) in T3 when the data that is to be written is output by the CPU and is stable on the bus an $\overline{\text{MEMW}}$ command is generated. The $\overline{\text{MEMW}}$ signal is connected to the R/W input of the RAM and strobes the data into the addressed location.

In Figure 3-7 a typical Memory system is illustrated to show how standard semiconductor components interface to the 8080 bus. The memory array shown has 8K bytes (8 bits/byte) of ROM storage, using four Intel® 8216As and 512 bytes of RAM storage, using Intel 8111 static RAMs. The basic interface to the bus structure detailed here is common to almost any size memory. The only addition that might have to be made for larger systems is more buffers (8216/8212) and decoders (8205) for generating "chip selects."

The memories chosen for this example have an access time of 850 nS (max) to illustrate that slower, economical devices can be easily interfaced to the 8080 with little effect on performance. When the 8080 is operated from a clock generator with a tCY of 500 nS the required memory access time is Approx. 450-550 nS. See detailed timing specification Pg. 5-16. Using memory devices of this speed such as Intel® 8308, 8102A, 8107A, etc. the READY input to the 8080 CPU can remain "high" because no "wait" states are required. Note that the bus interface to memory shown in Figure 3-7 remains the same. However, if slower memories are to be used, such as the devices illustrated (8316A, 8111) that have access times slower than the minimum requirement a simple logic control of the READY input to the 8080 CPU will insert an extra "wait state" that is equal to one or more clock periods as an access time "adjustment" delay to compensate. The effect of the extra "wait" state is naturally a slower execution time for the instruction. A single "wait" changes the basic instruction cycle to 2.5 microSeconds.

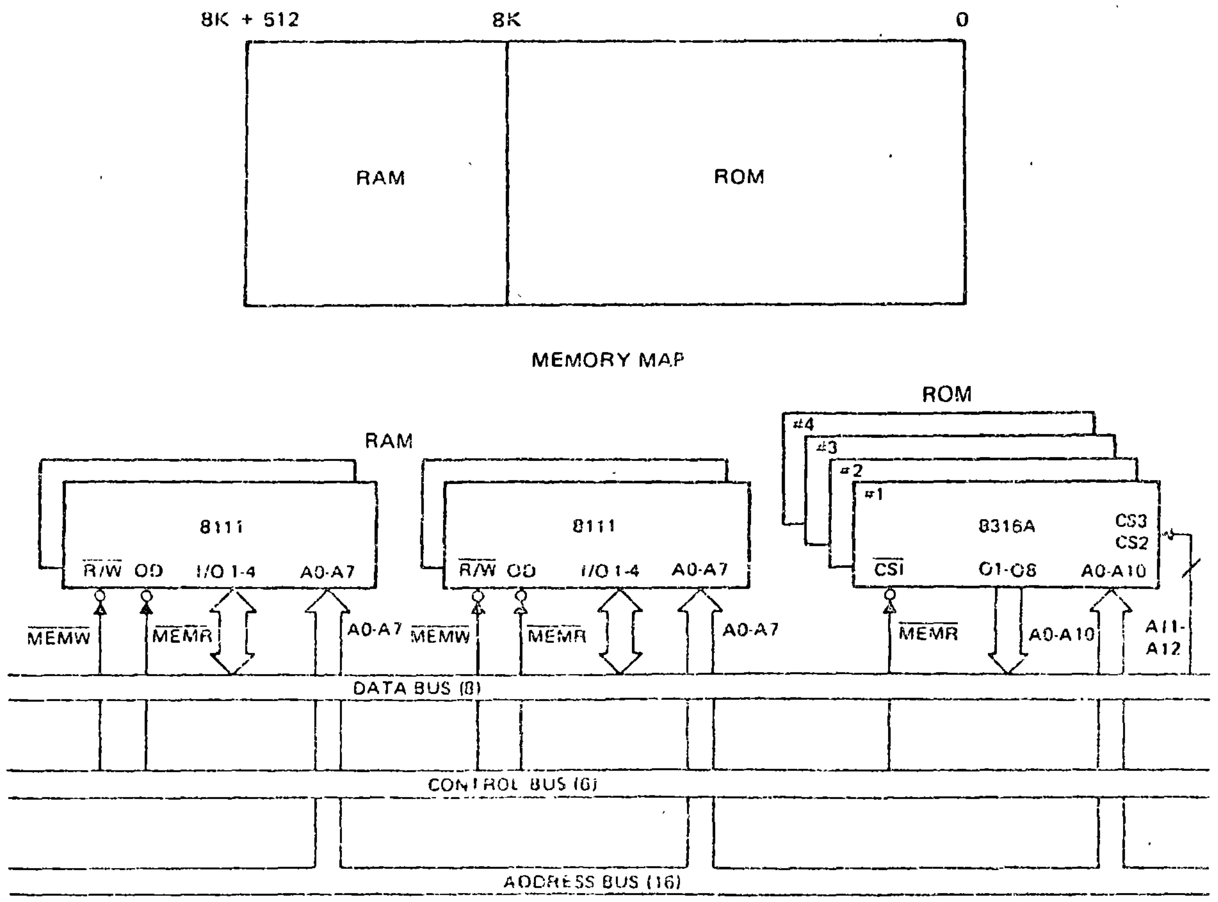


Figure 3-7. Typical Memory Interface

I/O INTERFACE

General Theory

As in any computer based system, the 8080 CPU must be able to communicate with devices or structures that exist outside its normal memory array. Devices like keyboards, paper tape, floppy disks, printers, displays and other control structures are used to input information into the 8080 CPU and display or store the results of the computational activity.

Probably the most important and strongest feature of the 8080 Microcomputer System is the flexibility and power of its I/O structure and the components that support it. There are many ways to structure the I/O array so that it will "fit" the total system environment to maximize efficiency and minimize component count.

The basic operation of the I/O structure can best be viewed as an array of single byte memory locations that can be Read from or Written into. The 8080 CPU has special instructions devoted to managing such transfers (IN, OUT). These instructions generally isolate memory and I/O arrays so that memory address space is not effected by the I/O structure and the general concept is that of a simple transfer to or from the Accumulator with an addressed "PORT". Another method of I/O architecture is to treat the I/O structure as part of the Memory array. This is generally referred to as "Memory Mapped I/O" and provides the designer with a powerful new "instruction set" devoted to I/O manipulation.

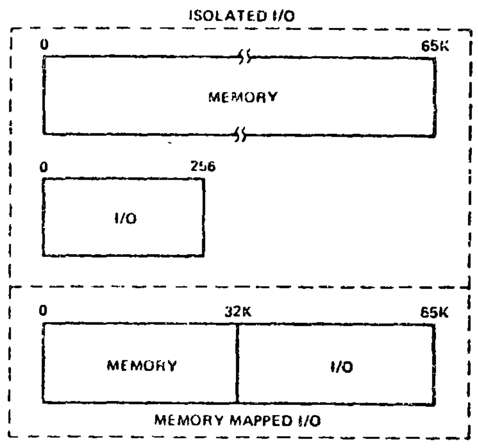


Figure 3-8. Memory/I/O Mapping.

Isolated I/O

In Figure 3-9 the system control signals, previously detailed in this chapter, are shown. This type of I/O architecture separates the memory address space from the I/O address space and uses a conceptually simple transfer to or from Accumulator technique. Such an architecture is easy to understand because I/O communicates only with the Accumulator using the IN or OUT instructions. Also because of the isolation of memory and I/O, the full address space (65K) is uneffected by I/O addressing.

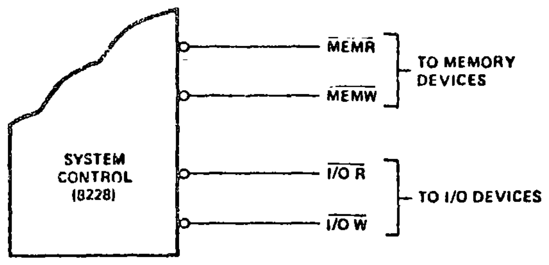


Figure 3-9. Isolated I/O.

Memory Mapped I/O

By assigning an area of memory address space as I/O a powerful architecture can be developed that can manipulate I/O using the same instructions that are used to manipulate memory locations. Thus, a "new" instruction set is created that is devoted to I/O handling.

As shown in Figure 3-10, new control signals are generated by gating the $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals with A_{15}, the most significant address bit. The new I/O control signals connect in exactly the same manner as Isolated I/O, thus the system bus characteristics are unchanged.

By assigning A_{15} as the I/O "flag", a simple method of I/O discipline is maintained:

If A_{15} is a "zero" then Memory is active.
If A_{15} is a "one" then I/O is active.

Other address bits can also be used for this function. A_{15} was chosen because it is the most significant address bit so it is easier to control with software and because it still allows memory addressing of 32K.

I/O devices are still considered addressed "ports" but instead of the Accumulator as the only transfer medium any of the internal registers can be used. All instructions that could be used to operate on memory locations can be used in I/O.

Examples:

MOVr, M	(Input Port to any Register)
MOV M, r	(Output any Register to Port)
MVI M	(Output immediate data to Port)
LDA	(Input to ACC)
STA	(Output from ACC to Port)
LHLD	(16 Bit Input)
SHLD	(16 Bit Output)
ADD M	(Add Port to ACC)
ANA M	("AND" Port with ACC)

It is easy to see that from the list of possible "new" instructions that this type of I/O architecture could have a drastic effect on increased system throughput. It is conceptually more difficult to understand than Isolated I/O and it does limit memory address space, but Memory Mapped I/O can mean a significant increase in overall speed and at the same time reducing required program memory area.

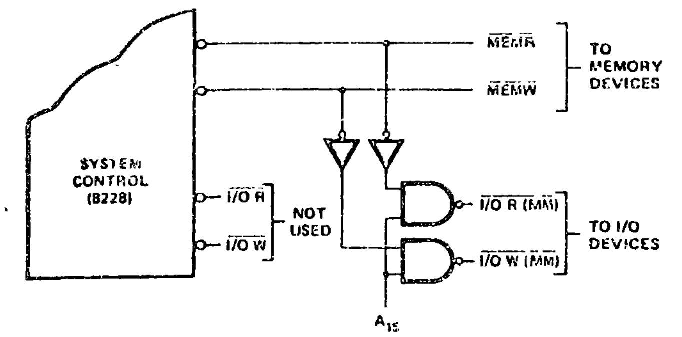


Figure 3-10. Memory Mapped I/O.

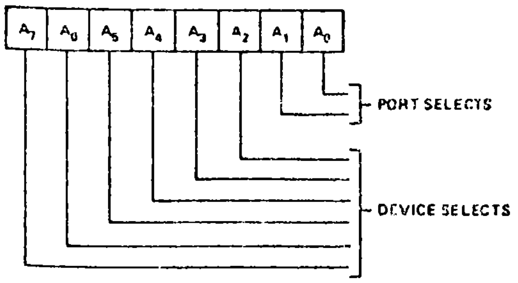
I/O Addressing

With both systems of I/O structure the addressing of each device can be configured to optimize efficiency and reduce component count. One method, the most common, is to decode the address bus into exclusive "chip selects" that enable the addressed I/O device, similar to generating chip-selects in memory arrays.

Another method is called "linear select". In this method, instead of decoding the Address Bus, a singular bit from the bus is assigned as the exclusive enable for a specific I/O device. This method, of course, limits the number of I/O devices that can be addressed but eliminates the need for extra decoders, an important consideration in small system design.

A simple example illustrates the power of such a flexible I/O structure. The first example illustrates the format of the second byte of the IN or OUT instruction using the Isolated I/O technique. The devices used are Intel®8255 Programmable Peripheral Interface units and are linear selected. Each device has three ports and from the format it can be seen that six devices can be addressed without additional decoders.

EXAMPLE #1



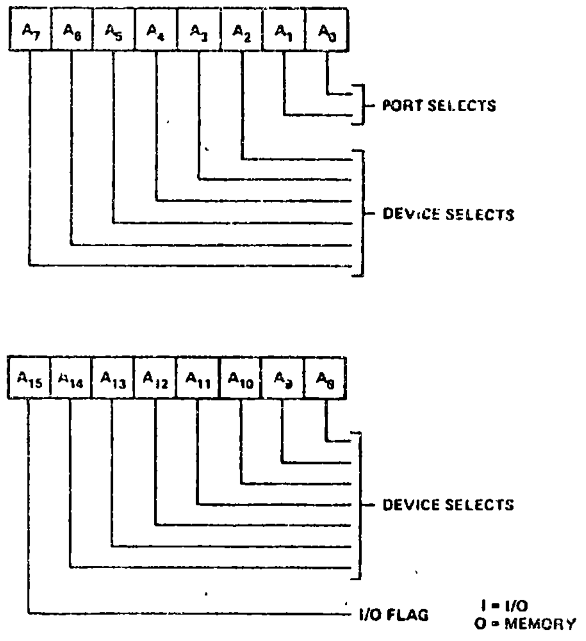
ADDRESSES – 6 – 8255s
(18 PORTS – 144 BITS)

Figure 3-11. Isolated I/O – (Linear Select) (8255)

The second example uses Memory Mapped I/O and linear select to show how thirteen devices (8255) can be addressed without the use of extra decoders. The format shown could be the second and third bytes of the LDA or STA instructions or any other instructions used to manipulate I/O using the Memory Mapped technique.

It is easy to see that such a flexible I/O structure, that can be "tailored" to the overall system environment, provides the designer with a powerful tool to optimize efficiency and minimize component count.

EXAMPLE #2



ADDRESSES – 13 – 8255s
(39 PORTS – 312 BITS)

Figure 3-12. Memory Mapped I/O – (Linear Select (8255)

I/O Interface Example

In Figure 3-16 a typical I/O system is shown that uses a variety of devices (8212, 8251 and 8255). It could be used to interface the peripherals around an intelligent CRT terminals; keyboards, display, and communication interface. Another application could be in a process controller to interface sensors, relays, and motor controls. The limitation of the application area for such a circuit is solely that of the designers imagination.

The I/O structure shown interfaces to the 8080 CPU using the bus architecture developed previously in this chapter. Either Isolated or Memory Mapped techniques can be used, depending on the system I/O environment.

The 8251 provides a serial data communication interface so that the system can transmit and receive data over communication links such as telephone lines.

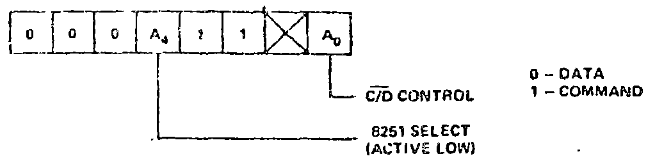


Figure 3-13. 8251 Format.

The two (2) 8255s provide twenty four bits each of programmable I/O data and control so that keyboards, sensors, paper tape, etc., can be interfaced to the system.

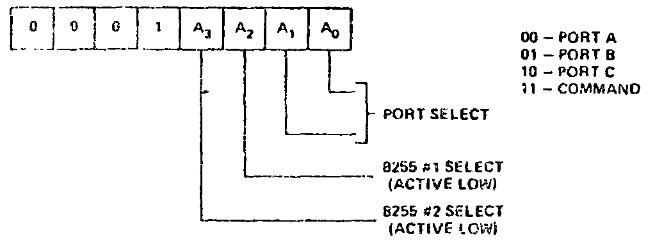
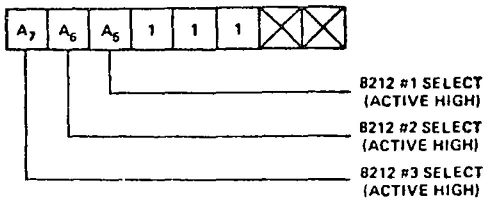


Figure 3-14. 8255 Format.

The three 8212s can be used to drive long lines or LED indicators due to their high drive capability. (15mA)



A7
A6
A5
1
1
1
8212 #1 SELECT (ACTIVE HIGH)
8212 #2 SELECT (ACTIVE HIGH)
8212 #3 SELECT (ACTIVE HIGH)

Figure 3-15. 8212 Format.

Addressing the structure is described in the formats illustrated in Figures 3-13, 3-14, 3-15. Linear Select is used so that no decoders are required thus, each device has an exclusive "enable bit".

The example shows how a powerful yet flexible I/O structure can be created using a minimum component count with devices that are all members of the 8080 Microcomputer System.

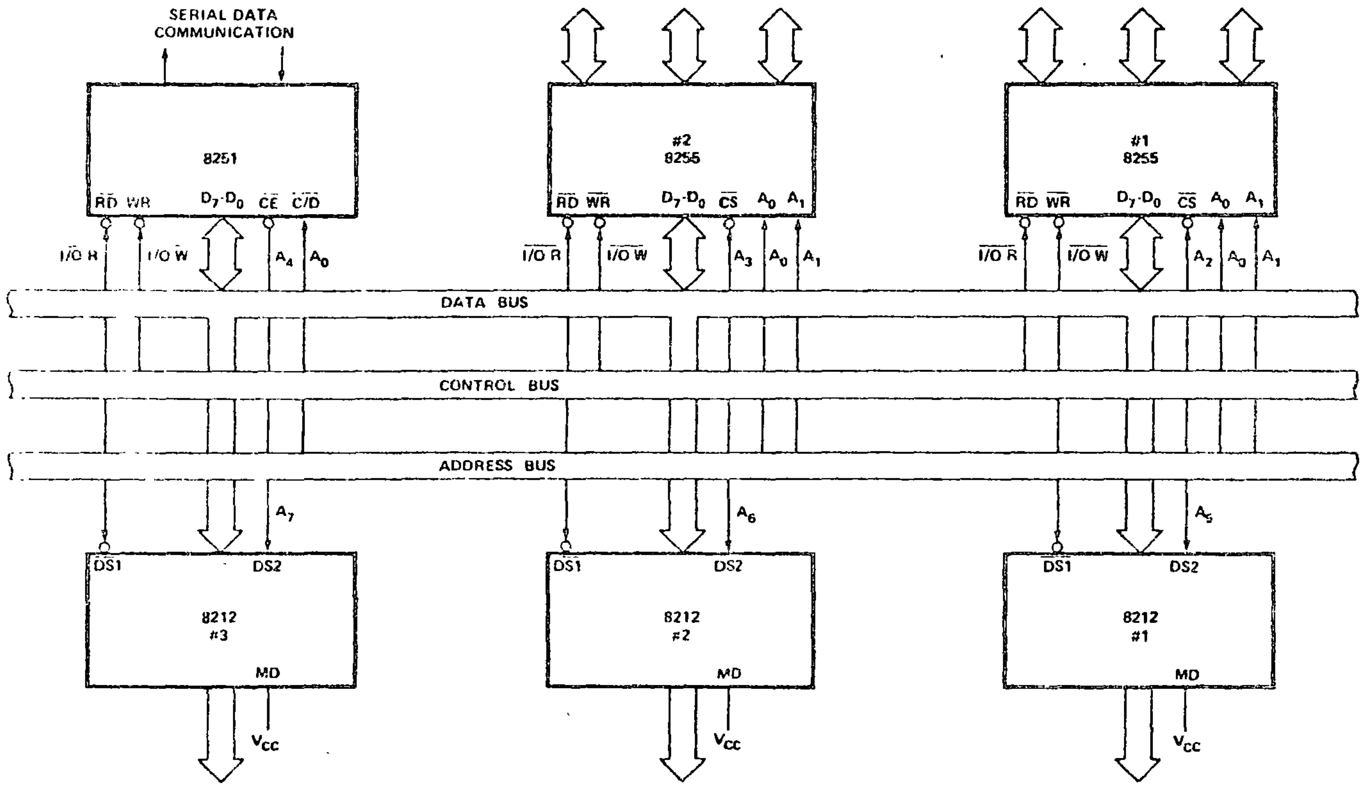


Figure 3-16. Typical I/O Interface.

CHAPTER 4 INSTRUCTION SET

A computer, no matter how sophisticated, can only do what it is "told" to do. One "tells" the computer what to do via a series of coded instructions referred to as a Program. The realm of the programmer is referred to as Software, in contrast to the Hardware that comprises the actual computer equipment. A computer's software refers to all of the programs that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer's Instruction Set.

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logical operations (e.g., OR the contents of two registers) and register operate instructions (e.g., increment a register) are included in the instruction set. A computer's instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide **Conditional Instructions**. A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a program with a decision-making capability.

By logically organizing a sequence of instructions into a coherent program, the programmer can "tell" the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1's and 0's), that is called Machine Code. Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There are programs available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is **Assembly Language**. A unique assembly language mnemonic is assigned to each of the computer's instructions. The programmer can write a program (called the **Source Program**) using these mnemonics and certain operands; the source program is then converted into machine instructions (called the **Object Code**). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an **Assembler** program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

THE 8080 INSTRUCTION SET

The 8080 instruction set includes five different types of instructions:

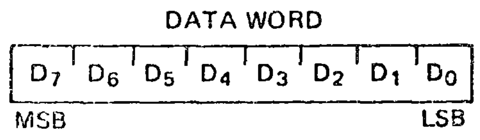
- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group** – add, subtract, increment or decrement data in registers or in memory
- **Logical Group** – AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group** – conditional and unconditional jump instructions, subroutine call instructions and return instructions
- **Stack, I/O and Machine Control Group** – includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

Instruction and Data Formats:

Memory for the 8080 is organized into 8-bit quantities, called Bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

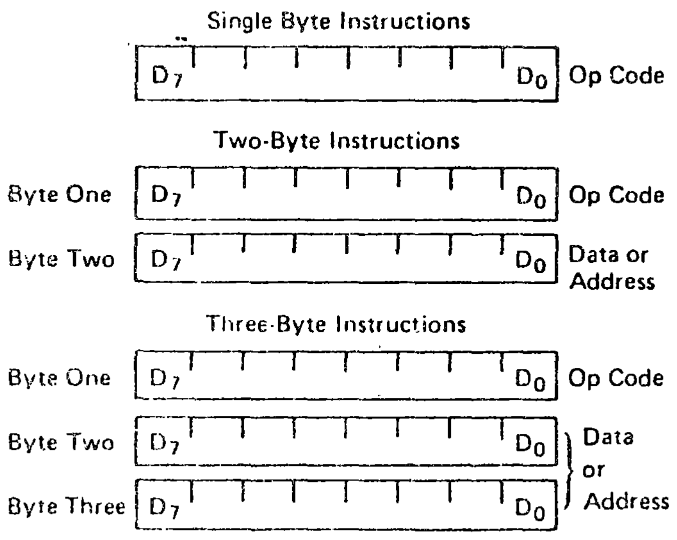
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the Least Significant Bit (LSB), and BIT 7 (of an 8 bit number) is referred to as the Most Significant Bit (MSB).

The 8080 program instructions may be one, two or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- Direct – Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- Register – The instruction specifies the register or register-pair in which the data is located.
- Register Indirect – The instruction specifies a register-pair which contains the memory address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).
- Immediate – The instruction contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- Direct – The branch instruction contains the address of the next instruction to be executed. (Except for the 'RST' instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- Register indirect – The branch instruction indicates a register-pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special one-byte call instruction (usually used during interrupt sequences). RST includes a three-bit field; program control is transferred to the instruction whose address is eight times the contents of this three-bit field.

Condition Flags:

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is "set" by forcing the bit to 1; "reset" by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner:

Zero:	If the result of an instruction has the value 0, this flag is set; otherwise it is reset.
Sign:	If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.
Parity:	If the modulo 2 sum of the bits of the result of the operation is 0, (i.e., if the result has even parity), this flag is set; otherwise it is reset (i.e., if the result has odd parity).
Carry:	If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.

Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r,r1,r2	One of the registers A,B,C,D,E,H,L
DDD,SSS	The bit pattern designating one of the registers A,B,C,D,E,H,L (DDD=destination, SSS=source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp One of the register pairs:

B represents the B,C pair with B as the high-order register and C as the low-order register;

D represents the D,E pair with D as the high-order register and E as the low-order register;

H represents the H,L pair with H as the high-order register and L as the low-order register;

SP represents the 16-bit stack pointer register.

RP The bit pattern designating one of the register pairs B,D,H,SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh	The first (high-order) register of a designated register pair.
rl	The second (low-order) register of a designated register pair.
PC	16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8 bits respectively).
SP	16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8 bits respectively).
r_m	Bit m of the register r (bits are number 7 through 0 from left to right).
Z,S,P,CY,AC	The condition flags: Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.
( )	The contents of the memory location or registers enclosed in the parentheses.
$\leftarrow$	"Is transferred to"
$\wedge$	Logical AND
$\forall$	Exclusive OR
$\vee$	Inclusive OR
$+$	Addition
$-$	Two's complement subtraction
$*$	Multiplication
$\leftrightarrow$	"Is exchanged with"
$\overline{\quad}$	The one's complement (e.g., $\overline{(A)}$)
n	The restart number 0 through 7
NNN	The binary representation 000 through 111 for restart number 0 through 7 respectively.

Description Format:

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The MAC 80 assembler format, consisting of the instruction mnemonic and operand fields, is printed in BOLDFACE on the left side of the first line.
2. The name of the instruction is enclosed in parenthesis on the right side of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.
4. This is followed by a narative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.

6. The last four lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a Conditional Jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see Page 4-2) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group:

This group of instructions transfers data to and from registers and memory. Condition flags are not affected by any instruction in this group.

MOV r1, r2 (Move Register)

(r1) ◄— (r2)

The content of register r2 is moved to register r1.

0	1	D	D	D	S	S	S

Cycles:	1
States:	5
Addressing:	register
Flags:	none

MOV r, M (Move from memory)

(r) ◄— ((H) (L))

The content of the memory location, whose address is in registers H and L, is moved to register r.

0	1	D	D	D	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

MOV M, r (Move to memory)

((H) (L)) ◄— (r)

The content of register r is moved to the memory location whose address is in registers H and L.

0	1	1	1	0	S	S	S

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

MVI r, data (Move Immediate)

(r) ◄— (byte 2)

The content of byte 2 of the instruction is moved to register r.

0	0	D	D	D	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	none

MVI M, data (Move to memory immediate)

((H) (L)) ◄— (byte 2)

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.

0	0	1	1	0	1	1	0
data							

Cycles:	3
States:	10
Addressing:	immed./reg. indirect
Flags:	none

LXI rp, data 16 (Load register pair immediate)

(rh) ◄— (byte 3),

(rl) ◄— (byte 2)

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.

0	0	R	P	0	0	0	1
low-order data							
high-order data							

Cycles:	3
States:	10
Addressing:	immediate
Flags:	none

LDA addr (Load Accumulator direct)

(A) ← ((byte 3)(byte 2))

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.

0	0	1	1	1	0	1	0
low-order addr							
high-order addr							

Cycles:	4
States:	13
Addressing:	direct
Flags:	none

STA addr (Store Accumulator direct)

((byte 3)(byte 2)) ← (A)

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.

0	0	1	1	0	0	1	0
low-order addr							
high order addr							

Cycles:	4
States:	13
Addressing:	direct
Flags:	none

LHLD addr (Load H and L direct)

(L) ← ((byte 3)(byte 2))

(H) ← ((byte 3)(byte 2) + 1)

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register L. The content of the memory location at the succeeding address is moved to register H.

0	0	1	0	1	0	1	0
low-order addr							
high-order addr							

Cycles:	5
States:	16
Addressing:	direct
Flags:	none

SHLD addr (Store H and L direct)

((byte 3)(byte 2)) ← (L)

((byte 3)(byte 2) + 1) ← (H)

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.

0	0	1	0	0	0	1	0
low-order addr							
high-order addr							

Cycles:	5
States:	16
Addressing:	direct
Flags:	none

LDAX rp (Load accumulator indirect)

(A) ← ((rp))

The content of the memory location, whose address is in the register pair rp, is moved to register A. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	1	0	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

STAX rp (Store accumulator indirect)

((rp)) ← (A)

The content of register A is moved to the memory location whose address is in the register pair rp. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	0	0	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

XCHG (Exchange H and L with D and E)

(H) ↔ (D)

(L) ↔ (E)

The contents of registers H and L are exchanged with the contents of registers D and E.

1	1	1	0	1	0	1	1

Cycles:	1
States:	4
Addressing:	register
Flags:	none

Arithmetic Group:

This group of instructions performs arithmetic operations on data in registers and memory.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register)

(A) ← (A) + (r)

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ADD M (Add memory)

(A) ← (A) + ((H) (L))

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.

1	1	0	0	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ADI data (Add immediate)

(A) ← (A) + (byte 2)

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.

1	1	0	0	0	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

ADC r (Add Register with carry)

(A) ← (A) + (r) + (CY)

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ADC M (Add memory with carry)

(A) ← (A) + ((H) (L)) + (CY)

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.

1	0	0	0	1	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ACI data (Add immediate with carry)

(A) ← (A) + (byte 2) + (CY)

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.

1	1	0	0	1	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

SUB r (Subtract Register)

(A) ← (A) − (r)

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

SUB M (Subtract memory)

(A) ← (A) − ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

SUI data (Subtract immediate)

(A) ← (A) − (byte 2)

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	1	0	1	0	1	1	0
data							

Cycles:	2
States.	7
Addressing.	immediate
Flags:	Z,S,P,CY,AC

SBB r (Subtract Register with borrow)

(A) ← (A) − (r) − (CY)

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

(A) ← (A) − ((H) (L)) − (CY)

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

SBI data (Subtract immediate with borrow)

(A) ← (A) − (byte 2) − (CY)

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	1	0	1	1	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

INR r (Increment Register)

(r) ← (r) + 1

The content of register r is incremented by one. Note: All condition flags except CY are affected.

0	0	D	D	D	1	0	0

Cycles:	1
States:	5
Addressing:	register
Flags:	Z,S,P,AC

INR M (Increment memory)

((H) (L)) ← ((H) (L)) + 1

The content of the memory location whose address is contained in the H and L registers is incremented by one. Note: All condition flags except CY are affected.

0	0	1	1	0	1	0	0

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,AC

DCR r (Decrement Register)

(r) ← (r) − 1

The content of register r is decremented by one. Note: All condition flags except CY are affected.

0	0	D	D	D	1	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	Z,S,P,AC

DCR M (Decrement memory)

((H) (L)) ← ((H) (L)) − 1

The content of the memory location whose address is contained in the H and L registers is decremented by one. Note: All condition flags except CY are affected.

0	0	1	1	0	1	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,AC

INX rp (Increment register pair)

(rh) (rl) ← (rh) (rl) + 1

The content of the register pair rp is incremented by one. Note No condition flags are affected.

0	0	R	P	0	0	1	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

DCX rp (Decrement register pair)

(rh) (rl) ← (rh) (rl) − 1

The content of the register pair rp is decremented by one Note: No condition flags are affected.

0	0	R	P	1	0	1	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

DAD rp (Add register pair to H and L)

(H) (L) ← (H) (L) + (rh) (rl)

The content of the register pair rp is added to the content of the register pair H and L. The result is placed in the register pair H and L. Note: Only the CY flag is affected. It is set if there is a carry out of the double precision add; otherwise it is reset.

0	0	R	P	1	0	0	1

Cycles:	3
States:	10
Addressing:	register
Flags:	CY

DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 or if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, or if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.

NOTE: All flags are affected.

0	0	1	0	0	1	1	1

Cycles:	1
States:	4
Flags:	Z,S,P,CY,AC

Logical Group:

This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)

(A) ← (A) ∧ (r)

The content of register r is logically anded with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	0	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ANA M (AND memory)

(A) ← (A) ∧ ((H) (L))

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	0	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ANI data (AND immediate)

$(A) \leftarrow (A) \wedge (\text{byte } 2)$

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	1	1	0	0	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

XRA r (Exclusive OR Register)

$(A) \leftarrow (A) \forall (r)$

The content of register r is exclusive-or'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	0	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

XRA M (Exclusive OR Memory)

$(A) \leftarrow (A) \forall ((H)\ (L))$

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	0	1	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

XRI data (Exclusive OR immediate)

$(A) \leftarrow (A) \forall (\text{byte } 2)$

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	1	1	0	1	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags.	Z,S,P,CY,AC

ORA r (OR Register)

$(A) \leftarrow (A) \vee (r)$

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	1	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ORA M (OR memory)

$(A) \leftarrow (A) \vee ((H)\ (L))$

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	0	1	1	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ORI data (OR Immediate)

$(A) \leftarrow (A) \vee (\text{byte } 2)$

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

1	1	1	1	0	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

CMP r (Compare Register)

$(A) - (r)$

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if $(A) = (r)$. The CY flag is set to 1 if $(A) < (r)$.

1	0	1	1	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

CMP M (Compare memory)

(A) -- ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).

1	0	1	1	1	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

CPI data (Compare immediate)

(A) – (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).

1	1	1	1	1	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$
$(CY) \leftarrow (A_7)$

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. Only the CY flag is affected.

0	0	0	0	0	1	1	1

Cycles: 1
States: 1
Flags: CY

RRC (Rotate right)

$(A_n) \leftarrow (A_{n-1})$; $(A_7) \leftarrow (A_0)$
$(CY) \leftarrow (A_0)$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. Only the CY flag is affected.

0	0	0	0	1	1	1	1

Cycles: 1
States: 4
Flags: CY

RAL (Rotate left through carry)

$(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$
$(A_0) \leftarrow (CY)$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. Only the CY flag is affected.

0	0	0	1	0	1	1	1

Cycles: 1
States: 4
Flags: CY

RAR (Rotate right through carry)

$(A_n) \leftarrow (A_{n+1})$; $(CY) \leftarrow (A_0)$
$(A_7) \leftarrow (CY)$

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. Only the CY flag is affected.

0	0	0	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

CMA (Complement accumulator)

$(A) \leftarrow (\overline{A})$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). No flags are affected.

0	0	1	0	1	1	1	1

Cycles: 1
States: 4
Flags: none

CMC (Complement carry)

(CY) ← $(\overline{CY})$

The CY flag is complemented. No other flags are affected.

0	0	1	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

STC (Set carry)

(CY) ← 1

The CY flag is set to 1. No other flags are affected.

0	0	1	1	0	1	1	1

Cycles: 1
States: 4
Flags: CY

Branch Group:

This group of instructions alter normal sequential program flow.

Condition flags are not affected by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The conditions that may be specified are as follows:

CONDITION		CCC
NZ	– not zero (Z = 0)	000
Z	– zero (Z = 1)	001
NC	– no carry (CY = 0)	010
C	– carry (CY = 1)	011
PO	– parity odd (P = 0)	100
PE	– parity even (P = 1)	101
P	– plus (S = 0)	110
M	– minus (S = 1)	111

JMP addr (Jump)

(PC) ← (byte 3) (byte 2)

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

1	1	0	0	0	0	1	1
low-order addr							
high-order addr							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

Jcondition addr (Conditional jump)

If (CCC),

(PC) ← (byte 3) (byte 2)

If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise, control continues sequentially.

1	1	C	C	C	0	1	0
low-order addr							
high-order addr							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

CALL addr (Call)

((SP) – 1) ← (PCH)
((SP) – 2) ← (PCL)
(SP) ← (SP) – 2
(PC) ← (byte 3) (byte 2)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

1	1	0	0	1	1	0	1
low-order addr							
high-order addr							

Cycles: 5
States: 17
Addressing: immediate/reg. indirect
Flags: none

Ccondition addr (Condition call)

If (CCC),

((SP) – 1) ⟵ (PCH)

((SP) – 2) ⟵ (PCL)

(SP) ⟵ (SP) – 2

(PC) ⟵ (byte 3) (byte 2)

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

Cycles:	3/5
States:	11/17
Addressing:	immediate/reg. indirect
Flags:	none

RET (Return)

(PCL) ⟵ ((SP));

(PCH) ⟵ ((SP) + 1);

(SP) ⟵ (SP) + 2;

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

1	1	0	0	1	0	0	1

Cycles:	3
States:	10
Addressing	reg. indirect
Flags:	none

Rcondition (Conditional return)

If (CCC),

(PCL) ⟵ ((SP))

(PCH) ⟵ ((SP) + 1)

(SP) ⟵ (SP) + 2

If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	0	0	0

Cycles:	1/3
States:	5/11
Addressing:	reg. indirect
Flags:	none

RST n (Restart)

((SP) – 1) ⟵ (PCH)

((SP) – 2) ⟵ (PCL)

(SP) ⟵ (SP) – 2

(PC) ⟵ 8 * (NNN)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.

1	1	N	N	N	1	1	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	N	N	N	0	0	0

Program Counter After Restart

PCHL (Jump H and L indirect – move H and L to PC)

(PCH) ⟵ (H)

(PCL) ⟵ (L)

The content of register H is moved to the high-order eight bits of register PC. The content of register L is moved to the low-order eight bits of register PC.

1	1	1	0	1	0	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

Stack, I/O, and Machine Control Group:

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags.

Unless otherwise specified, condition flags are not affected by any instructions in this group.

PUSH rp (Push)

$((SP) - 1) \leftarrow (rh)$
$((SP) - 2) \leftarrow (rl)$
$(SP) \leftarrow (SP) - 2$

The content of the high-order register of register pair rp is moved to the memory location whose address is one less than the content of register SP. The content of the low-order register of register pair rp is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Note: Register pair rp = SP may not be specified.

1	1	R	P	0	1	0	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
$((SP) - 2)_0 \leftarrow (CY)$, $((SP) - 2)_1 \leftarrow 1$
$((SP) - 2)_2 \leftarrow (P)$, $((SP) - 2)_3 \leftarrow 0$
$((SP) - 2)_4 \leftarrow (AC)$, $((SP) - 2)_5 \leftarrow 0$
$((SP) - 2)_6 \leftarrow (Z)$, $((SP) - 2)_7 \leftarrow (S)$
$(SP) \leftarrow (SP) - 2$

The content of register A is moved to the memory location whose address is one less than register SP. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two.

1	1	1	1	0	1	0	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

FLAG WORD

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

POP rp (Pop)

$(rl) \leftarrow ((SP))$
$(rh) \leftarrow ((SP) + 1)$
$(SP) \leftarrow (SP) + 2$

The content of the memory location, whose address is specified by the content of register SP, is moved to the low-order register of register pair rp. The content of the memory location, whose address is one more than the content of register SP, is moved to the high-order register of register pair rp. The content of register SP is incremented by 2. Note: Register pair rp = SP may not be specified.

1	1	R	P	0	0	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	none

POP PSW (Pop processor status word)

$(CY) \leftarrow ((SP))_0$
$(P) \leftarrow ((SP))_2$
$(AC) \leftarrow ((SP))_4$
$(Z) \leftarrow ((SP))_6$
$(S) \leftarrow ((SP))_7$
$(A) \leftarrow ((SP) + 1)$
$(SP) \leftarrow (SP) + 2$

The content of the memory location whose address is specified by the content of register SP is used to restore the condition flags. The content of the memory location whose address is one more than the content of register SP is moved to register A. The content of register SP is incremented by 2.

1	1	1	1	0	0	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

XTHL (Exchange stack top with H and L)

(L) $\leftrightarrow$ ((SP))
(H) $\leftrightarrow$ ((SP) + 1)

The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.

1	1	1	0	0	0	1	1

Cycles:	5
States:	18
Addressing:	reg. indirect
Flags:	none

SPHL (Move HL to SP)

(SP) $\leftarrow$ (H) (L)

The contents of registers H and L (16 bits) are moved to register SP.

1	1	1	1	1	0	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

IN port (Input)

(A) $\leftarrow$ (data)

The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.

1	1	0	1	1	0	1	1
port							

Cycles:	3
States:	10
Addressing:	direct
Flags:	none

OUT port (Output)

(data) $\leftarrow$ (A)

The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.

1	1	0	1	0	0	1	1
port							

Cycles:	3
States:	10
Addressing:	direct
Flags:	none

EI (Enable interrupts)

The interrupt system is enabled following the execution of the next instruction.

1	1	1	1	1	0	1	1

Cycles:	1
States:	4
Flags:	none

DI (Disable interrupts)

The interrupt system is disabled immediately following the execution of the DI instruction.

1	1	1	1	0	0	1	1

Cycles:	1
States:	4
Flags:	none

HLT (Halt)

The processor is stopped. The registers and flags are unaffected.

0	1	1	1	0	1	1	0

Cycles:	1
States:	7
Flags:	none

NOP (No op)

No operation is performed. The registers and flags are unaffected.

0	0	0	0	0	0	0	0

Cycles:	1
States:	4
Flags:	none

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
MOV r_1, r_2	Move register to register	0	1	D	D	D	S	S	S	5
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4

NOTES. 1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

Schottky Bipolar 8214

PRIORITY INTERRUPT CONTROL UNIT

- Eight Priority Levels
- Current Status Register
- Priority Comparator
- Fully Expandable
- High Performance (50ns)
- 24-Pin Dual In-Line Package

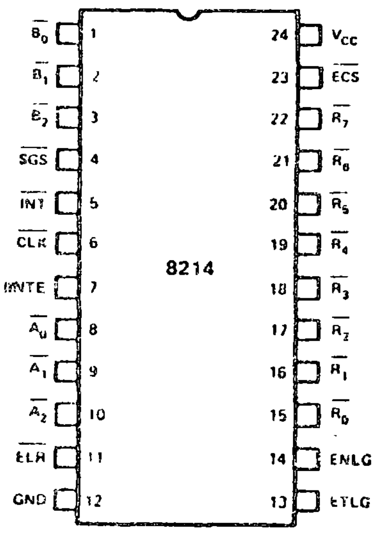
The 8214 is an eight level priority interrupt control unit designed to simplify interrupt driven microcomputer systems.

The PICU can accept eight requesting levels; determine the highest priority, compare this priority to a software controlled current status register and issue an interrupt to the system along with vector information to identify the service routine.

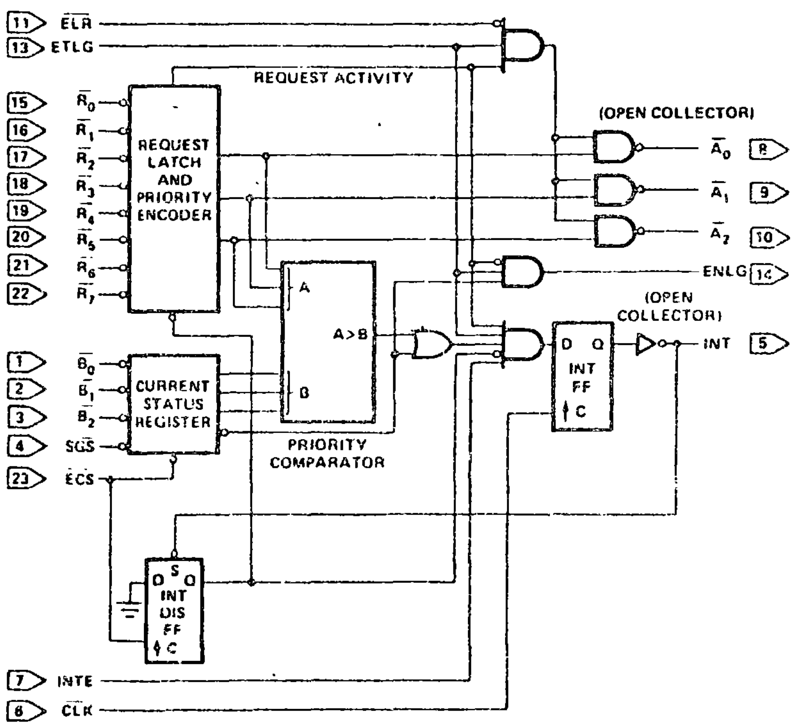
The 8214 is fully expandable by the use of open collector interrupt output and vector information. Control signals are also provided to simplify this function.

The PICU is designed to support a wide variety of vectored interrupt structures and reduce package count in interrupt driven microcomputer systems.

PIN CONFIGURATION



LOGIC DIAGRAM



PIN NAMES

INPUTS	
$\overline{R_0}$–$\overline{R_7}$	REQUEST LEVELS (R_7 HIGHEST PRIORITY)
$\overline{B_0}$–$\overline{B_2}$	CURRENT STATUS
$\overline{SGS}$	STATUS GROUP SELECT
$\overline{ECS}$	ENABLE CURRENT STATUS
INTE	INTERRUPT ENABLE
$\overline{CLK}$	CLOCK (INT F F)
$\overline{ELR}$	ENABLE LEVEL READ
ETLG	ENABLE THIS LEVEL GROUP
OUTPUTS	
$\overline{A_0}$–$\overline{A_2}$	REQUEST LEVELS } OPEN COLLECTOR
$\overline{INT}$	INTERRUPT (ACT LOW) } OPEN COLLECTOR
ENLG	ENABLE NEXT LEVEL GROUP

INTERRUPTS IN MICROCOMPUTER SYSTEMS

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient method so that large amounts of the total systems tasks can be assumed by the microcomputer with little or no effect on throughput.

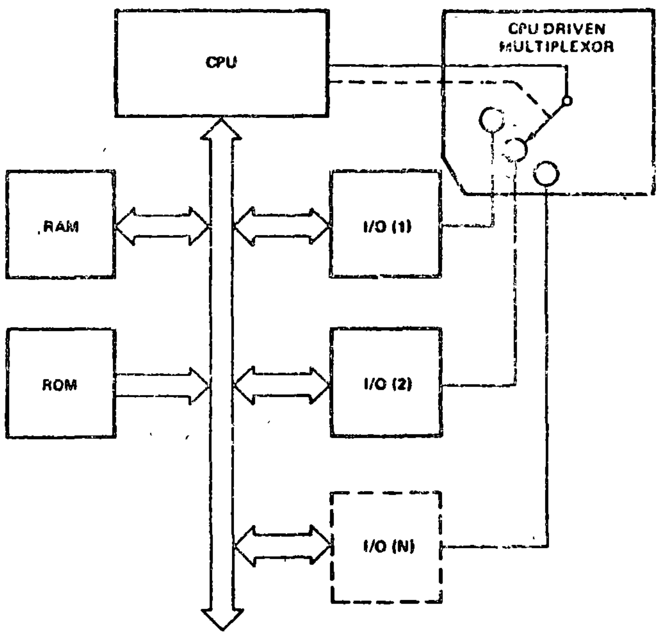
The most common method of servicing such devices is the Polled approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuence polling cycle and that such a method would have a serious, detrimental effect on system throughput thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desireable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete however the processor would resume exactly where it left off.

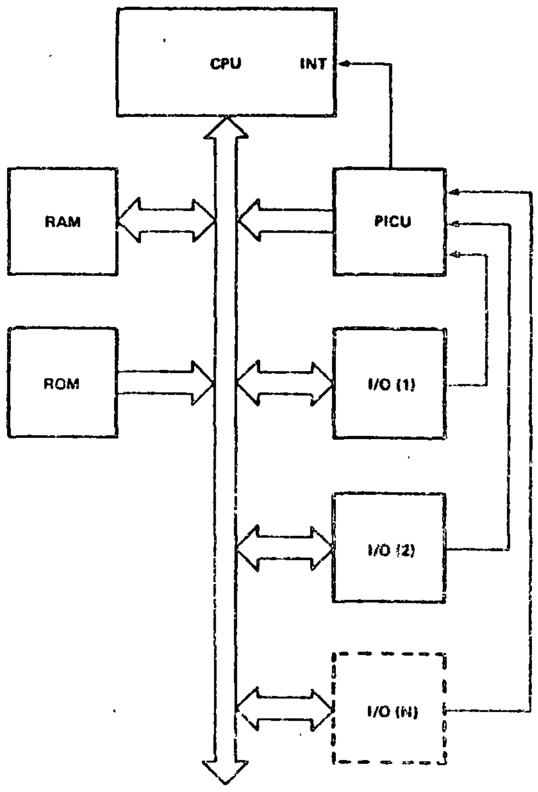
This method is called Interrupt. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Priority Interrupt Control Unit (PICU) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced and issues an Interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PICU, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesing device. The PICU encodes the requesting level into such information for use as a "vector" to the correct Interrupt Service Routine.



Polled Method



Interrupt Method

FUNCTIONAL DESCRIPTION

General

The 8214 is a device specifically designed for use in real time, interrupt driven, microcomputer systems. Basically it is an eight (8) level priority control unit that can accept eight different interrupt requests, determine which has the highest priority, compare that level to a software maintained current status register and issue an interrupt to the system based on this comparison along with vector information to indicate the location of the service routine.

Priority Encoder

The eight requests inputs, which are active low, come into the Priority Encoder. This circuit determines which request input is the most important (highest priority) as preassigned by the designer. ($\overline{R7}$) is the highest priority input to the 8214 and ($\overline{R0}$) is the lowest. The logic of the Priority Encoder is such that if two or more input levels arrive at the same time then the input having the highest priority will take precedence and a three bit output, corresponding to the active level (modulo 8) will be sent out. The Priority Encoder also contains a latch to store the request input. This latch is controlled by the Interrupt Disable Flip-flop so that once an interrupt has been issued by the 8214 the request latch is no longer open. (Note that the latch does not store inactive requests. In order for a request to be monitored by the 8214 it must remain present until it has been serviced.)

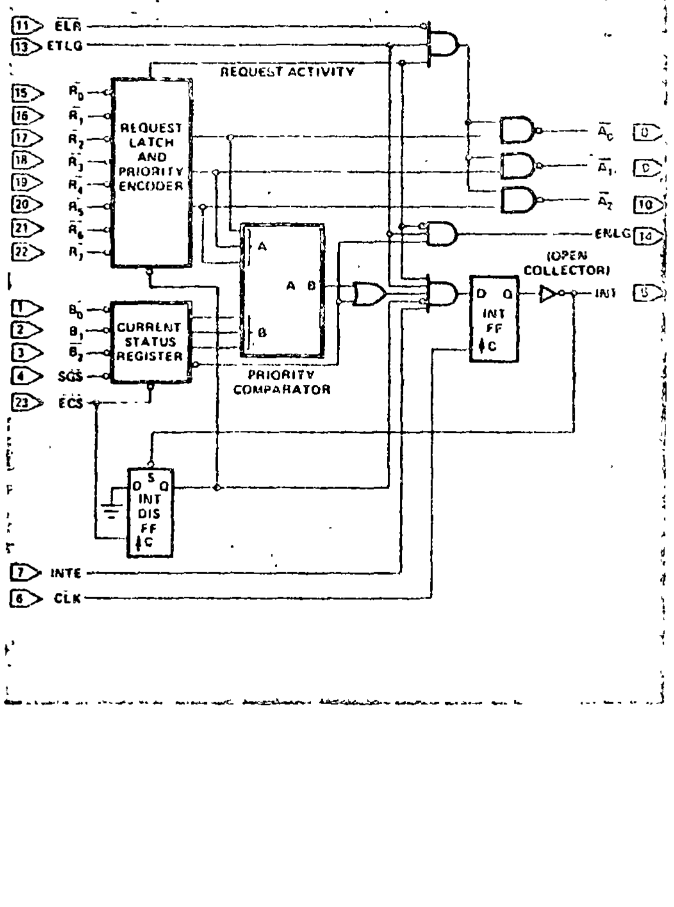
Current Status Register

In an interrupt driven microcomputer system it is important to not only prioritize incoming requests but to ascertain whether such a request is a higher priority than the interrupt currently being serviced.

The Current Status Register is a simple 4-bit latch that is treated as an addressable outport port by the microcomputer system. It is loaded when the $\overline{ECS}$ input goes low. Maintenance of the Current Status Register is performed as a portion of the service routine. Basically, when an interrupt is issued to the system the programmer outputs a binary code (modulo 8) that is the compliment of the interrupt level. This value is stored in the Current Status Register and is compared to all further prioritized incoming requests by the Priority Comparator. In essence, a copy of the current interrupt level is written into the 8214 to be used as a reference for comparison. There is no restriction to this maintenance, other level values can be written into this register as references so that groups of interrupt requests may be disallowed under complete control of the programmer.

Note that the fourth bit in the register is $\overline{SGS}$. This input is part of the value written out by the programmer and performs a special function. The Priority Comparator will only issue an output that indicates the request level is greater than the Current Status Register. If both comparator inputs are equal to zero no output will be present. The $\overline{SGS}$ input allows the programmer to, in effect, disable this comparison and allow the 8214 to issue an interrupt to the system that is based only on the logic of the priority encoder.

LOGIC DIAGRAM



Control Signals

The 8214 also has several inputs that enable the designer to synchronize the interrupt issued to the microprocessor and to allow or disallow such an issuance. Also, signals are provided that permit simple expansion to other 8214s so that more than eight levels can be controlled.

INTE, $\overline{\text{CLK}}$

The INTE (Interrupt Enable) input allows the designer to "shutoff" the interrupt system under control of external logic or possibly under software maintenance. A "zero" on this line will not allow interrupts to be issued to the microcomputer system.

The $\overline{\text{CLK}}$ (Clock) input is actually the trigger that strobes the Interrupt Flip-Flop. It can be connected to one of the clocks of the microprocessor so that the interrupt issued meets the CPU set-up time specification. Note that due to the gating of the input to the Interrupt Flip-Flop the $\overline{\text{INT}}$ output will only be active for the time of a single clock period, so external latching may be required to hold this signal.

$\overline{\text{ELR}}$, ETLG, ENGL

These three signals allow 8214s to be cascaded so that more than eight levels of interrupt requests can be controlled.

Basically, the ENLG output of one 8214 is connected to the ETLG input of the next and so on, with the first 8214 having its ETLG input pulled "high" and assigned the highest priority. When the ENLG output is "high" it indicates that there is no interrupt pending on that device and that interrupts can be monitored on the next lower priority 8214.

This "cascading" can be expanded almost indefinitely to accomodate even the largest of interrupt driven system architectures.

$\overline{\text{A0}}$, $\overline{\text{A1}}$, $\overline{\text{A2}}$

In order to identify which device has interrupted the processor so that the service routine associated with it can be addressed, a pointer or "vector" must accompany the interrupt issued to the microcomputer system.

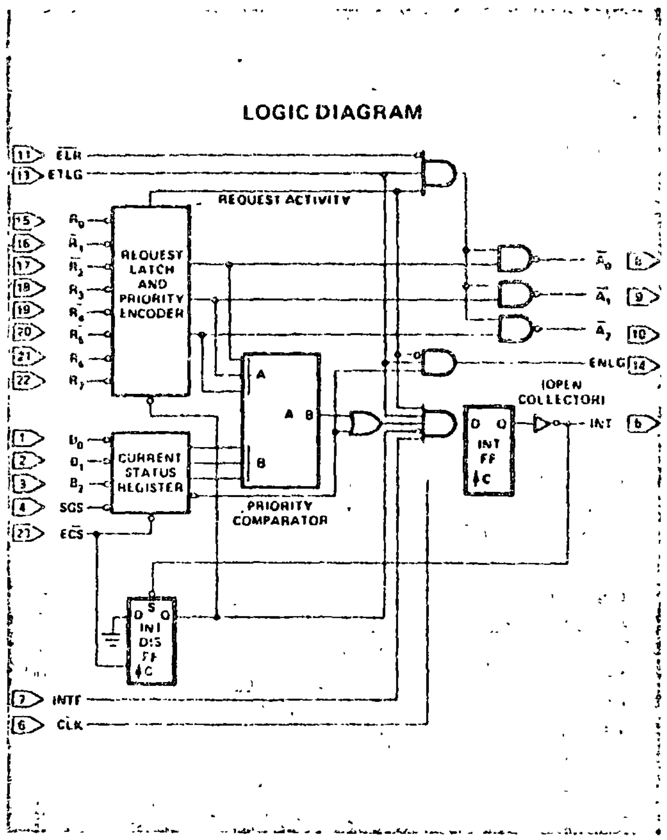
The $\overline{\text{A0}}$, $\overline{\text{A1}}$ and $\overline{\text{A2}}$ outputs represent the complement of the active interrupt level (modulo 8). By using these signals to encode the special instruction, RST, the program counter of the microprocessor, can point to the location of the service routine. Note that these three outputs are gated by the $\overline{\text{ELR}}$ input and are open collector so that expansion is simplified.

$\overline{\text{INT}}$

The $\overline{\text{INT}}$ output of the 8214 is the signal that is issued to the microprocessor to initiate the interrupt sequence. As soon a $\overline{\text{INT}}$ is active the INT DIS FF is set, inhibiting further requests from entering the Request Latch. Only the writing out of the current status information by strobing the $\overline{\text{ECS}}$ input will clear the INT DIS FF and allow requests to enter the latch.

Note that $\overline{\text{INT}}$ is also open collector so that when cascaded to other 8214s an interrupt in any of the active devices will set all INT DIS FFs in the entire array.

LOGIC DIAGRAM



APPLICATIONS OF THE 8214

8 Level Controller (8080)

The most common of applications of the 8214 is that of an eight level priority structure for 8080 or 8008 microcomputer systems.

Shown in the figure below is a detailed logic schematic of a simple circuit that will accept eight input requests, maintain current status, issue the interrupt signal to the 8080 and encode the proper RST instruction to gate onto the data bus.

The eight requests are connected to the 8214 by the designer in whatever order of priority is to be preassigned. For example, eight keyboards could be monitored and each assigned a degree of importance (level of priority) so that faster processor attention or access can be assigned to the critical or time dependent tasks.

The inputs to the Current Status Register are connected to the Data Bus so that data can be written out into this "port".

An 8212 is used to encode the RST instruction and also to act as a 3-state gate to place the proper RST instruction when the 8080 Data Bus is in the input mode. Note that the $\overline{INT}$ signal from the 8214 is latched in the SR flip-flop of the 8212 so that proper timing is maintained. The 8212 is selected (enabled) when the INTA signal from the 8080 status latch and the DBIN from the 8080 are active, this assures that the RST instruction will be placed on the Data Bus at the proper time. Note that the $\overline{INT}$ output from the 8212 is inverted and pulled up before it is connected to the 8080. This is to generate an $\overline{INT}$ signal to the 8080 that has the correct polarity and meets the input voltage requirement (3.3V).

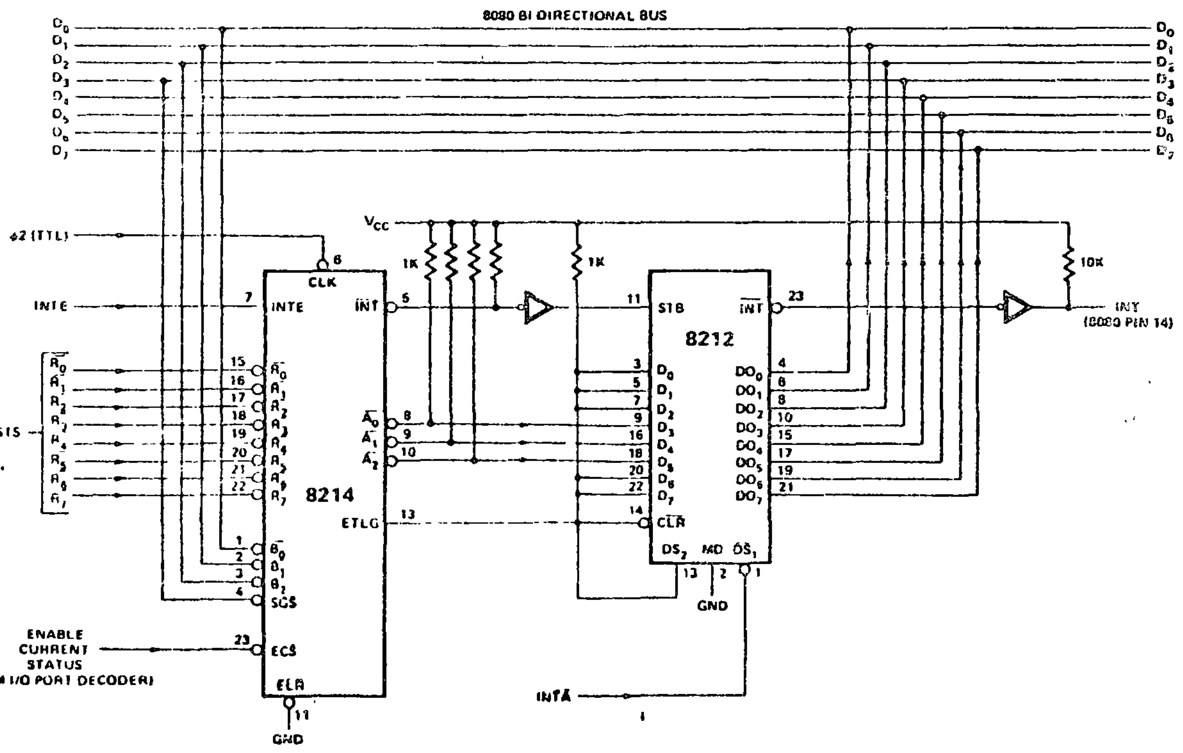
Basic Operation

When the initial interrupt request is presented to the 8214 it will issue an interrupt to the 8080 if the structure is enabled. The 8214 will encode the request into 3 bits (modulo 8) and output them to the 8212. After the acknowledgement of the interrupt has been issued by the 8080 the encoded RST instruction is gated onto the Data Bus by the 8212. The processor executes the instruction and points the program counter to the desired serviced routine. In this routine the programmer will probably save the status of the register array and flags within a series of PUSH instructions (4). Then a copy of the current interrupt level (modulo 8) can be "built" in the Accumulator and output to the Current Status Register of the 8214 for use as a comparison reference for all further incoming requests to the system.

This Vectored Eight Level Priority Interrupt Structure for 8080 microcomputer systems is a powerful yet flexible circuit that is high performance and has a minimal component count.

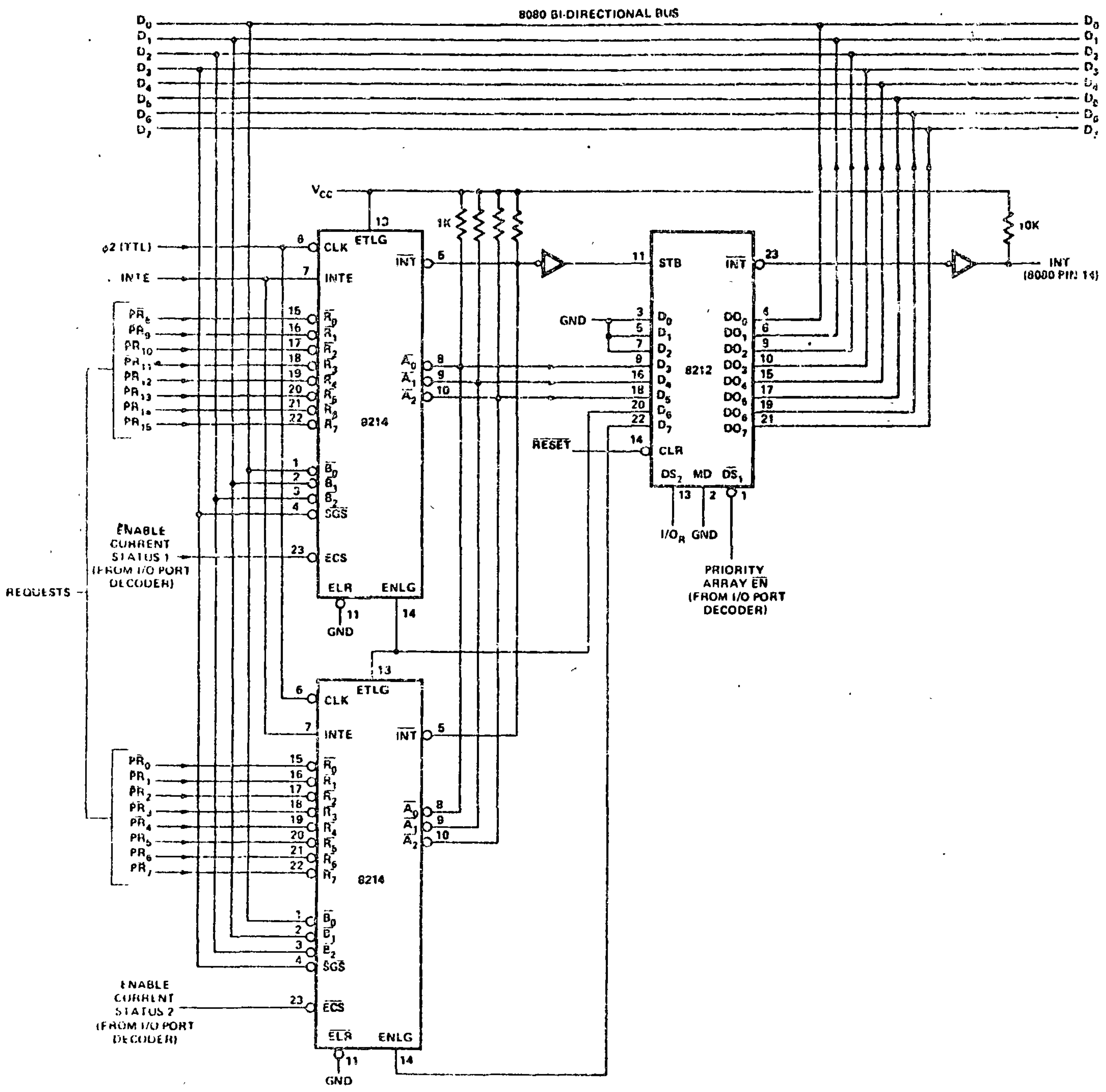
			D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
PRIORITY REQUEST		RST	1	1	A_2	A_1	A_0	1	1	1
LOWEST	0	7	1	1	1	1	1	1	1	1
	1	6	1	1	1	1	0	1	1	1
	2	5	1	1	1	0	1	1	1	1
	3	4	1	1	1	0	0	1	1	1
	4	3	1	1	0	1	1	1	1	1
	5	2	1	1	0	1	0	1	1	1
	6	1	1	1	0	0	1	1	1	1
HIGHEST	7	*0	1	1	0	0	0	1	1	1

*RST 0 WILL VECTOR PROGRAM COUNTER TO LOCATION 0 (ZERO) AND INVOKE THE SAME ROUTINE AS "RESET" INPUT TO 8080
THIS COULD RE INITIALIZE THE SYSTEM BASED ON THE ROUTINE INVOKED
(A CAUTION TO SYSTEM PROGRAMMERS.)



8080 BI DIRECTIONAL BUS
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
φ2 (TTL)
INTE
REQUESTS
$\overline{R_0}$ $\overline{R_1}$ $\overline{R_2}$ $\overline{R_3}$ $\overline{R_4}$ $\overline{R_5}$ $\overline{R_6}$ $\overline{R_7}$
ENABLE CURRENT STATUS (FROM I/O PORT DECODER)
V_{CC}
1K
1K
10K
CLK
INTE
$\overline{INT}$
$\overline{A_0}$ $\overline{A_1}$ $\overline{A_2}$
8214
ETLG
$\overline{B_0}$ B_1 B_2
$\overline{SGS}$
$\overline{ECS}$
ELR
GND
STB
$\overline{INT}$
8212
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
DO_0 DO_1 DO_2 DO_3 DO_4 DO_5 DO_6 DO_7
$\overline{CLR}$
DS_2 MD $\overline{DS_1}$
GND
INTA
INT (8080 PIN 14)

8 Level Controller


8080 BI-DIRECTIONAL BUS
D0 D1 D2 D3 D4 D5 D6 D7
VCC
1K
10K
φ2 (TTL)
INTE
PR8 PR9 PR10 PR11 PR12 PR13 PR14 PR15
PR0 PR1 PR2 PR3 PR4 PR5 PR6 PR7
REQUESTS
ENABLE CURRENT STATUS 1 (FROM I/O PORT DECODER)
ENABLE CURRENT STATUS 2 (FROM I/O PORT DECODER)
8214
CLK
ETLG
INT
ECS
ELR
ENLG
SGS
GND
8212
STB
CLR
RESET
DS2 MD DS1
I/OR
PRIORITY ARRAY EN (FROM I/O PORT DECODER)
INT (8080 PIN 14)

16 Level Controller

APPLICATIONS OF THE 8214

Cascading the 8214

When greater than eight levels of interrupts must be prioritized and serviced, the 8214 can be cascaded with other 8214s to support such an architecture.

On the previous page a simple circuit is shown that can control 16 levels of interrupt and is easily expandable to support up to 40 levels of interrupt by just cascading more 8214s.

As described previously, there are signals provided in the 8214 for cascading ($\overline{ELR}$, ETLG, ENLG) and in effect the ENLG output of the first 8214 "ripples" down to the next and so on. The entire array of 8214s regardless of size, can be thought of as a single priority control unit, with the first having the highest priority and the next 8214 having a lower priority and so on.

In this application, the manner in which software handles the servicing of the interrupt will change. Since more than eight vectors must be generated a method other than the common RST instruction must be implemented. Basically, the priority control array must somehow modify the contents of the 8080 Program Counter so that it can point ("vector") to one of 16 (or how many levels are to be serviced) and fetch the proper service routine. A simple approach is to treat the priority control array as a single input port that can input a value into the Accumulator and use this value as an offset to modify the Program Counter (Indirect Jump).

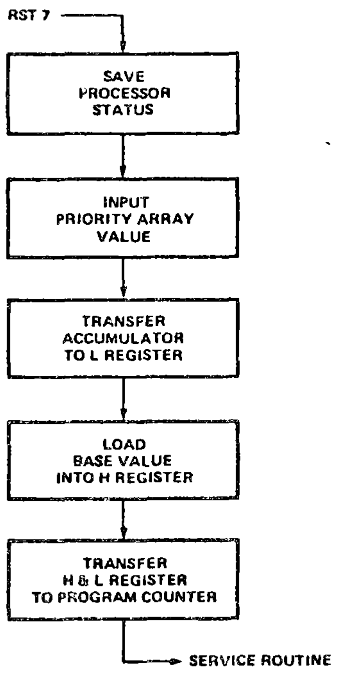
An initial CALL is needed to invoke this Indirect Jump routine so the circuitry is configured to insert an RST 7 (FFh) for all interrupts, thus the Indirect Jump Routine starts at location (56d).

The Assembly Code for the flow chart is as follows:

```
PUSH   PSW  ┐
PUSH   B    ├─ (save processor status)
PUSH   D    │  (22 microseconds)
PUSH   H    ┘
IN     (n)      (input Priority Array Value)
MOV    L,A      (transfer Accumulator to L register)
MVI    H,(n)    (load Base Address into H register)
PCHL            (transfer H&L to Program Counter)
```

(The execution time for the total routine is 35.5 microseconds based on an 8080 clock period of 500ns.)

Following is a basic flowchart of the priority array Indirect Jump routine. Note that the last step in the routine will vector the processor to fetch the proper service routine as dictated by the interrupting level.


RST 7
SAVE PROCESSOR STATUS
INPUT PRIORITY ARRAY VALUE
TRANSFER ACCUMULATOR TO L REGISTER
LOAD BASE VALUE INTO H REGISTER
TRANSFER H & L REGISTER TO PROGRAM COUNTER
SERVICE ROUTINE

		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
REQUEST (PR) PRIORITIES		0-7 $\overline{EN}$	8-15 $\overline{EN}$	$\overline{A_2}$	$\overline{A_1}$	$\overline{A_0}$	0	0	0
LOWEST	0	0	1	1	1	1	0	0	0
	1	0	1	1	1	0	0	0	0
	2	0	1	1	0	1	0	0	0
	3	0	1	1	0	0	0	0	0
	4	0	1	0	1	1	0	0	0
	5	0	1	0	1	0	0	0	0
	6	0	1	0	0	1	0	0	0
	7	0	1	0	0	0	0	0	0
	8	1	0	1	1	1	0	0	0
	9	1	0	1	1	0	0	0	0
	10	1	0	1	0	1	0	0	0
	11	1	0	1	0	0	0	0	0
	12	1	0	0	1	1	0	0	0
	13	1	0	0	1	0	0	0	0
	14	1	0	0	0	1	0	0	0
HIGHEST	15	1	0	0	0	0	0	0	0

Shown in the figure above is a chart of the 16 different array values that are used to offset the Program Counter and vector to the proper service routine. These values are the ones that are loaded into the "L" register; the value loaded into the "H" register with an "immediate instruction" is used to identify the major area of memory where the service routines are stored, similar to a "course setting" and the value in the "L" register is used to identify a specific location, similar to a "fine setting".

Note that D0, D1, and D2 are always set to "zero", this provides the programmer eight (8) memory locations between the start of each service routine so that maintenance of the associated Current Status Register and a JUMP or CALL instruction can be implemented.

This method of interrupt control can be almost indefinitely expanded and provides the system designer with a powerful tool to enhance total system throughput.

MICROPROCESADORES: TEORIA Y APLICACIONES

INTERP/80 USER'S MANUAL

ABRIL, 1978.

8080 SOFTWARE PACKAGE - SIMULATOR

A. INTRODUCTION

This Manual describes the use of a FORTRAN IV program called INTERP/80. This program provides a software simulation of the INTEL 8080 CPU, along with execution monitoring commands to aid program development for the MCS-80.

INTERP/80 accepts machine code produced by the INTEL 8080 Assembler or PL/M 8080 compiler, along with execution commands from a time-sharing terminal, card reader, or disk file. The execution commands allow manipulation of the simulated MCS-80 memory and the 8080 CPU registers. In addition, operand and instruction breakpoints may be set to stop execution at crucial points in the program. Tracing features are also available which allow the CPU operation to be monitored. INTERP/80 provides symbolic reference to storage locations as well as numeric reference in various number bases. The command language is described in the paragraphs which follow.

B. BASIC ELEMENTS

All input to INTERP/80 is "free form". Numbers, symbolic names, and special characters may be placed anywhere within the input line (see margin commands in Section D). Comments may be interspersed in the input, but must be enclosed within the bracketing symbols /* and */.

1. Numbers. Numeric input to INTERP/80 can be expressed in binary, octal, decimal or hexadecimal. The letters B,O,Q,D, and H following the integer number indicates the base, as shown below:

Number	Value
11011B	11011_2
28D	28_{10}
33O	33_8
33Q	33_8
1CH	$1C_{16}$
28	28_{10}

A decimal number is assumed if the base is omitted. Note that although O is allowed to indicate octal integers, Q is also permitted to avoid confusion with the integer O. Note that the leading digit of a hexadecimal number must be one of the digits 0,1,...,9. Thus, $EF2_{16}$ must be expressed as 0EF2H.

On output, INTERP/80 indicates octal integers with Q and omits the D on decimal values. The base used on output defaults to decimal, but may be changed by the user. (See the BASE command in Section C.)

2. Symbolic Names. Symbolic names are strings of contiguous alphabetic and numeric characters not exceeding 32 characters in length. The first character must be alphabetic. Valid symbolic names are:

```
SYMBOLICNAME
X3
G1G2G3
LONGSTRINGOFCHARACTERS
```

3. Special Characters. The special characters recognized by INTERP/80 are: $=./()+-'*,<>:;. All other special characters are replaced by a blank.

C. INTERP/80 Commands

The commands available in INTERP/80 are summarized briefly below. Full details of each command are given in following paragraphs.

Command	Purpose
LOAD	Causes symbol tables and code to be loaded into the simulated MCS-80 memory.
GO	Starts execution of the loaded 8080 code.
INTER	Simulates an 8080 interrupt.
TIME	Sets and displays the simulated 8080 cycle counter.
CYCLE	Allows the simulated CPU to be stopped after a given number of cycles.
TRACE	Enables tracing feature when particular portions of the program are executed.
REFER	Causes the CPU simulation to stop when a particular storage location is referenced.
ALTER	Causes the CPU simulation to stop when the contents of a particular memory location is altered.
CONV	Displays the values of numbers converted to the various number bases.
DISPLAY	Displays memory locations, CPU registers, symbolic locations, and I/O ports.
SET	Allows the values of memory locations, CPU registers, and I/O ports to be altered.
BASE	Allows the default number base used for output to be changed.
INPUT	Controls simulated 8080 input ports.
OUTPUT	Controls simulated 8080 output ports.
PUNCH	Causes output of machine code in BPNF or hexadecimal format.
END	Terminates execution of an 8080 program.

The commands NOINTER, NOTRACE, NOREFER, NOALTER, NOINPUT, and NOOUTPUT are also defined. These commands negate the effects of INTER, TRACE, REFER, ALTER, INPUT and OUTPUT, respectively. In all cases, the commands may be abbreviated. These abbreviations are indicated with the command description.

Commands are typed anywhere on the input line, with as many commands on a line as desired. The symbol "." must follow each command.

The treatment of syntax errors in commands is dependent on the mode of operation. If the command input and output files both correspond to a terminal (interactive mode), and a syntax error occurs in a command which is embedded in a command line containing several commands, the next command is taken from the next command line. If INTERP/80 was initiated in interactive mode, and through a command file chain (see $INPUT and $OUTPUT control switches) is currently accepting commands from a non-interactive (batch) device, then the next command will be accepted from the initial interactive device. Otherwise a command syntax error results in exiting INTERP/80.

The end of data for the execution of INTERP/80 is indicated by a "$EOF" starting in column 1 of the last card.

1. Range-Lists. Many of the INTERP/80 commands accept a "range-list" as an operand. Tracing, for example, can be enabled for a specific range of addresses in the program. The range-list specifies a sequence of contiguous addresses in memory, or a range of numeric values to which the command is applied.

In its simplest form, a range-list is a number (binary, octal, decimal, or hexadecimal), or it may be a pair of numbers separated by the symbol "TO". Thus, valid range-lists are:

```
   10
   63Q
   50 TO 63Q
   OFH TO 11001111B
```

A range-list, however, can also reference a symbolic location, with or without a numeric displacement from the location. Suppose, for example, the symbols START and INCR appear at locations 10 and 32 in the source program. Valid range-lists involving these symbols are:

```
   START        (Same as 10)
   START+6      (Same as 16)
   START-101B   (Same as 5)
   10 TO INCR   (Same as 10 TO 32)
   START+3 TO
     INCR-2     (Same as 13 TO 30)
```

The PL/M language allows duplicate names within the same program which define different storage locations (e.g., the variable I could be defined within both a procedure P1 and a procedure P2). Thus, a symbolic location can be specified as a sequence of symbolic names, separated by the "/" symbol. The symbolic location

P1/I

references the symbol I which follows the symbol P1, while

P2/I

causes INTERP/80 to first locate P1, and the find the symbol I.

In general the reference

$$S_1/S_2/\ldots/S_n \pm d$$

causes a sequential search of the symbol table for S_1, then S_2, and so forth, until S_n is found. The displacement $\pm d$ is then applied to this base location.

The range-list may also contain a reference to the current value of the program counter of the simulated 8080 CPU. The symbol "*" represents this value. If the value of the program counter is 16, for example, the following is a valid range-list:
START TO * (Same as 10 TO 16)
The exact use of the range-list is illustrated with the individual commands.

2. Notation. The following notation is used to describe the INTERP/80 command structure. Elements enclosed within braces {and} are optional, while elements enclosed within the brackets [and] are alternatives, where at least one alternative must be present.

A range-list, for example, can be specified as:
range-element {TO range-element}
where a range-element is defined as:

$$\begin{bmatrix} \text{number} \\ \text{Symbolic-name} \\ * \end{bmatrix} \left\{ \begin{bmatrix} + & \text{number} \\ - & \text{number} \end{bmatrix} \right\}$$

As mentioned previously, command names can always be abbreviated. The required portion of the command is underlined in the command description. The symbol "TO" in the range list can be abbreviated as "T". Thus, the range list above can be redefined as:
range-element {$\underline{\text{T}}$O range-element}.
Finally, the ellipses "..." indicate a list of indefinite length.

The commands are given alphabetically in the following paragraphs starting with a prototype statement using the above notation. A brief description is then given, followed by examples.

3. $\begin{bmatrix} \underline{\text{AL}}\text{TER} \\ \underline{\text{NOA}}\text{LTER} \end{bmatrix}$ range list{,range-list, range-list,..., range-list}.

The ALTER command is an operand breakpoint command which causes the execution of the 8080 CPU to stop whenever an attempt is made by the CPU to store values into a memory location specified in the range-list. When the breakpoint is encountered, INTERP/80 prints ALTER x, where x is the value of the program counter. If a symbol table has been loaded, INTERP/80 will also print the symbolic location closest to location x (plus or minus a displacement value). Execution can be started again with the GO or RUN commands. Examples of the command are:

```
ALTER 0.
ALTER 0 TO 10.
ALTER 10 T INCR.
ALTER START + 2 TO INCR - 0AH.
AL 5, START, X2, 7 T 10, INCR - 3.
```

```
ALT P1/I TO P2/I+3.
```

4. $\underline{B}ASE \left\{ \begin{array}{l} BIN \\ \overline{O}CT \\ \overline{D}EC \\ \overline{H}EX \end{array} \right\}$

This command causes the INTERP/80 system to use the number base specified by the second argument when printing results. This command has no effect on the number bases which are acceptable in the input.

5. CONV range list {, range-list, range-list,..., range-list}.
The conversion command prints the values of the numbers specified in the range-list in binary, octal, decimal and hexadecimal forms. Examples are:

```
CONV 23.
CONV *.
CON 10 TO START +3.
CO 10,30,28Q, 1101B T 33H.
```

6. CYCLE number.
The cycle command causes an automatic break point when the number of machine instructions specified in the command have been executed in the simulation (see the GO command and the $MAXCYCLE parameter, also).

7. DISPLAY display-element {, display-element,..., display-element}.
The display command causes the values of memory locations, symbolic names, CPU registers, and I/O ports to be printed. The output form of these values is determined by the current default base (see the BASE command). The width of the output line determines the output formatting (see the $WIDTH command of Section E).

In its simplest form, a display-element can be one of the 8080 CPU registers:

CY	(carry)	D	
Z	(zero)	E	
S	(sign)	H	
P	(parity)	L	
A		HL	(H&L)
B		SP	(program stack pointer)
C		PC	(program counter)

In this case, valid DISPLAY commands are:

```
DISPLAY CY.
DISP CY, Z, H, HL.
D P, A, PC.
```

A display-element can also be the symbol CPU, in which case all registers are displayed.

The values latched into the I/O ports can be displayed by using a display element of the form:

```
    PORT range-list
```

The ports specified in the range-list (between 0 and 255) are printed. Examples are:

```
    DISPLAY PORT 0.
    DI PO 3, PO 5, PORT 5 TO 8, PO 1001B.
```

The contents of the symbol table can be examined by using a display-element of the form:

$$\text{SYMBOLS} \quad \left\{ \begin{bmatrix} * \\ \text{Symbolic-name} \\ \text{number} \end{bmatrix} \right\}$$

The form
```
    DISPLAY SYMBOLS.
```
prints the entire symbol table, while the form
```
    DISPLAY SYMBOLS number.
```
responds with the symbolic name (± a numeric displacement) which is closest to the address specified by the number. Examples are:
```
    DISP SY.
    DI SY OFFH, SY 32.
```
If the symbol "*" is used in the command, the symbolic location closest to the current program counter is printed. The values contained in memory locations can also be displayed. In this case, the display-element takes the form

$$\text{MEMORY range-list} \left\{ \begin{bmatrix} \text{CODE} \\ \text{BIN} \\ \text{OCT} \\ \text{DEC} \\ \text{HEX} \end{bmatrix} \right\}$$

The range of elements printed is specified in the range-list, while the form of the elements in the display is controlled by the command CODE (decoded instructions) or one of the number bases. If the form is omitted, the default number base is used in the display (see the BASE command). Valid DISPLAY commands are:
```
    DISPLAY MEMORY 20.
    DISP MEM 20 TO 30 CODE.
    D M 0 T 30 D, M 40 TO INCR + 10 OCT.
```

The various display-elements may be mixed in a single DISPLAY command.

Decoded memory dumps give instructions in mnemonic form. These mnemonics differ slightly from those used in the 8080 assembler, for the sake of brevity. The INTERP/80 mnemonics which appear when the code format is specified are listed in Section D.

8. END.
The END command reinitializes the INTERP/80 system. If another

program is subsequently loaded into memory, all break and trace points are reset. Otherwise, the currently loaded program may be rerun with all break and trace points remaining.

```
9.  GO    { [ *      ] }
            [ number ]
```

The GO command causes the execution of the loaded program to begin. In the case that a breakpoint was previously encountered, the execution continues through the breakpoint. If the GO is followed by an *, the breakpoint addresses are printed as they are encountered, but the 8080 CPU does not halt until completion. If the GO is followed by a number, the effect is exactly the same as

```
     CYCLE number. GO.
```

```
10. [ INPUT   ] range-list {,range-list,...,range-list}.
    [ NOINPUT ]
```

The INPUT command specifies a list of input port numbers. Whenever the simulated CPU executes an input instruction (IN) which references a port listed in the range list, the input value is read from the current input file (see the $INPUT parameter in Section E). When operating in interactive mode, INTERP/80 will "prompt" the user by typing

```
     PORT n...
```

at the console, where n is an integer between 0 and 255. The user then types the port input value, and execution continues. Any invalid input, such as the character '*' causes INTERP/80 to return to command mode. In batch mode, INTERP/80 reads data directly from the input file without the prompt, and prints

```
     PORT n...k
```

where n is the port number, and k is the data read from the input. Examples of the INPUT command are

```
     INPUT 5.
     INPUT 0 TO 255.
     INPUT 5, 10, 13 T 18, 66H.
     NOINPUT 10, 70 TO 240.
```

```
11. [ INTER   ]   ( STATE   )
    [ NOINTER ]   ( ENABLE  )
                  ( DISABLE )
                  ( opcode  )
```

The INTER command simulates the 8080 interrupt system. When INTERP/80 is started, the simulated 8080 CPU is initialized with interrupts disabled, and a NOP in the interrupt register. The form

```
                         INTER.
```

causes INTERP/80 to set a pending interrupt (initially a NOP). The interrupt takes place following the next GO or RUN command, one instruction after the interrupts are enabled by the CPU. Interrupts can be enabled manually with the command

```
                         INT ENABLE.
```

or disabled manually with the command

```
                         INT DISABLE.
```

The current state of the interrupts is printed by INTERP/80 when the command

```
                         INT STATE.
```

is issued.
The form   INT opcode.

can be used to place a new instruction into the interrupt register. The opcode in this case is either a number between 0 and 255, or one of the symbolic operation codes listed in Section D. The interrupt is set to pending.
The command

```
                         NOINTER.
```

is used to nullify any pending interrupts.

Examples are

```
     INTER.
     NOINTER.
     INTER ENA.
     INT DIS.
     INTER STATE.
     INT 52.
     INT RST 3.
```

12. LOAD number {number}.
The LOAD command reads the symbol table and 8080 machine code into the simulated memory. The form

```
     LOAD number.
```

reads only the machine code from the file specified by number (see file numbering in Section E). The form

```
     LOAD number number.
```

reads the symbol table from the file specified by the first number and the machine code from the second file. The symbol table is in the form produced by the INTEL 8080 Assembler or 8080 PL/M compiler. The machine code can be either in "BNPF" format (see PROM programming specifications in the INTEL Data Catalog), or in the more compact "Hex" format (see the $FASTLOAD parameter in Section D). The end of machine code is indicated by a "$" occurring in the input. INTERP/80 responds to this command by printing the number of locations used by the program.
Examples are:

```
     LOAD 1.
     LOAD 6 7.
```

Note that if no machine code is to be loaded, the command

```
     LOAD 1. $
```

is sufficient to initialize INTERP/80.

13. $\left[\begin{array}{l}\text{OUTPUT}\\ \text{NOOUTPUT}\end{array}\right]$ range-list {,range-list,...,range-list}.

The OUTPUT command is similar to the INPUT command except that it causes INTERP/80 to print output port values whenever ports referenced in the range-lists are altered with an output instruction (OUT). The output form is

```
     PORT n = k
```

where n is the port number and k is the output value.
Examples are:

```
     OUTPUT 5.
     OUT 5,6,8 TO 20H.
     OU 7 TO 10Q, 13 T 255.
     NOOUT 8,9,10 TO 30.
```

14. $\left[\begin{array}{l}\text{REFER}\\ \text{NOREFER}\end{array}\right]$ range-list {,range-list,...,range-list}.

This command is similar to the ALTER command except that a breakpoint occurs whenever any reference to the memory location takes place. Thus, an instruction fetch, an operand fetch, or an operand store all cause a breakpoint when this command is used. Examples are:

```
     REFER 10.
     RE 10 TO 30Q.
     REF 5,7, START TO START +5, 71Q.
     NOREF 0 TO 10.
```

15. RUN.
The RUN command has exactly the same effect as the command GO *.

16. SET. set-element {,set-element,...,set-element}.
The SET command allows memory locations, CPU registers, and I/O ports to be set to specific values. The register names described under the DISPLAY command can be used in the set-element:

$$\text{register} = \begin{bmatrix}\text{number}\\ *\end{bmatrix}$$

The value of the specified register is set to the number following the "=" or to the value of program counter if "*" is specified. Thus, valid commands are:

```
     SET Z = 0.
     SE A = 3, B = 77Q, PC = OEEH.
     S HL = 28.
```

INTERP/80 allows symbolic references when either the program counter (PC) or the H and L register (HL) are set. Thus, the commands

```
     SET PC = START.
     SET HL = P1.
     SET PC = START +5, HL = P1/I.
```

are all valid SET commands.

A set-element can also be symbol "CPU" in which case all registers are set to zero, including the simulated 8080 cycle count, and any pending interrupt is disabled. Examples are:

```
     SET CPU.
     S CP,PC = 25.
```

The values of I/O ports can also be set by using a set-element of the form

```
     PORT range-list = number  {number number ... number}
```

In this case, the I/O ports specified in the range-list are set to the list of numbers following the "=". If more ports are specified than there are numbers in the list, the numbers are reused starting at the beginning. Examples are:

```
     SET PORT 5 = 10.
     SET PO 6 TO 8 = 1 2 3.
     S PO 10 TO 13 -77Q 2.
     S PO 8 = 10B, PO 12 = 13H, PO 30Q = 16.
```

The values contained in memory locations can be altered directly using a set-element of the form

```
     MEMORY range list = mem-element {mem-element ... mem-element}
```

A mem-element can be a number (between 0 and 255) or a symbolic opcode (see Section D).

As in the case of I/O ports, the memory locations are filled from the list to the right of the equal sign, with numbers and operation codes being reused if the list is exhausted. Examples of this command are:

```
     SET MEMORY 0 = 0.
     S MEM 0 TO 50 = 0.
```

It is sometimes inconvenient to specify both the beginning and final addresses when using the SET MEMORY command. Thus, INTERP/80 allows a special case of the command where only the starting address is specified, and the final address is determined by the number of items following the equal sign. This form is

```
     MEMORY range-element = mem-element {mem-element...mem-element}
```

Examples of this form are given below

```
     SET MEMORY 0 = 0 1 2 3 4 5 6 7.
     SET MEM 5 = OFH 77Q 44H 33, 15H = 5 10 15.
     S M 100 = 5 10 15 20.
```

Whenever a mem-element is a symbolic operation code, the required operands may be either numeric or symbolic. The following SET commands provide examples of the use of these operation codes.

```
     SET MEM 0 TO 100 = NOP.
     SET MEM 50 = MOV AB  MOV CI  OFFH
                  JPO 32FEH JNZ P1+3
                  CALL P1/P2+5  LXI H P1/I.
```

The SET command does not change break or trace points which are in effect.

```
     S M START TO START +5=11111000B 22Q 33H.
```

As in the DISPLAY command, set elements of each type may be intermixed:

```
     SET CP, CY=0, M 5=10, PO 6=12, PC=30.
```

17. TIME {number}.

The TIME command allows display and alteration of the simulated 8080 CPU cycle count. The form

```
     TIME.
```

causes the simulated cycle count and time used by the 8080 CPU to be printed.
The form

```
     TIME number.
```

sets the cycle count to the number given in the command.

18. TRACE / NOTRACE range-list {,range-list,...,range-list}.

The TRACE command causes the INTERP/80 system to print the CPU register contents and the decoded instruction whenever an instruction is fetched from the memory region specified in the range-list. The form of the elements in the trace is defined by the current default base (see BASE command). The trace shows the register contents and operation code <u>before</u> the instruction is executed. The result of the operation <u>is found</u> in the next line of the trace, or through the DISPLAY CPU command. A heading showing the vaious columns in the trace may be printed at various intervals (see the $HEADING control switch).

Examples of the TRACE command are:

```
TRACE 0 TO 100.
TR START TO START +111B.
NOTRACE START, INCR, FOUND TO FOUND +3, 7Q.
```

19. PUNCH range list {number}.
The PUNCH command causes the specified region of the simulated memory to be output either in BPNF or hexadecimal code format depending on the setting of the $FASTLOAD control switch. If the number is present, the code is written into the corresponding output file; otherwise the currently defined file is used. (see the $OUTPUT and $FASTLOAD parameters in Section E). Examples are:

```
PUNCH 0 TO OFFH.
PU START TO FINISH.
```

D. INTERP/80 SYMBOLIC OPERATION CODES

The following symbolic operations codes are defined in INTERP/80 for ease of reference to 8080 CPU instructions. These operations codes are used in the DISPLAY MEMORY command when the CODE option is specified, and are acceptable as input when setting memory locations with the SET MEMORY command. In addition, the INTER command allows symbolic operation codes as input when the interrupt instruction register is set. The user should refer to the 8080 CPU specification for the individual instruction definitions.

1. Data Transfer, Arithmetic, and Logical Operations

The first category of 8080 instructions includes the "move" instructions, along with arithmetic and logical operations. In the description which follows, the symbols i and j can take one of the values A, B, C, D, E, H, L, or M, representing the basic CPU registers and memory. For the immediate instructions, the symbol k may take on any value between 0 and 255, and represents a single byte of immediate data.

Instruction	Examples
MOV ij	MOV AB MOV MH
MVI i k	MVI M 10 MVI A 67Q
INR i	INR A INR H INR M
DCR i	DCR A DCR E DCR M
ADD i	ADD A ADD C
ADI k	ADI 10 ADI 0FAH
ADC i	ADC B ADC D
ACI k	ACI 33Q ACI 176
SUB i	SUB A SUB L
SUI k	SUI 12 SUI 09AH
SBC i	SBC C SBC M
SBI k	SBI 5 SBI 016H
ANA i	ANA A ANA H
ANI k	ANI 17Q ANI 65
XRA i	XRA B XRA M XRA L
XRI k	XRI 1011B XRI 34Q
ORA i	ORA A ORA D
ORI k	ORI 1010$111B ORI 12
CMP i	CMP C CMP M
CPI k	CPI 13 CPI 23Q

2. Jump, Call, and Return Instructions.

In the description of the Jump, Call, and Return operators, the symbol k represents a 16-bit address (on input, k can take the form of a range-element). The unconditional instructions are listed first

Instruction	Examples
JMP k	JMP 2048 JMP 0E8H JMP P1
CALL k	CALL 3333 CALL 4FFH CALL P1+5
RET	RET

The conditional Jump, Call, and Return instructions all are of the form

tc k

where t is one of the symbols J, C, or R, representing a Jump, Call, or Return, respectively. The symbol c is one of the following conditions

C (carry set)	NC (no carry)
Z (zero flag set)	NZ (no zero set)
P (positive)	M (minus)
PO (parity odd)	PE (parity even)

The symbol k is a 16-bit address, as above. Following are a number of examples of conditional transfers.

```
JC      3333
JPO     044FH
CNC     555Q
CM      101010B
RZ      260
RPE     P1/P2+7
```

Note that the above conditions are set only after arithmetic and logical operations have been performed (again, see the 8080 CPU specification).

3. Double Register Instructions

In the description of the double register instructions which follows, the symbol p_1 can take the value B or D representing the register pairs B-C and D-E, respectively. Similarly, the symbol p_2 can take the values B, D, H, or PSW denoting the pairs B-C, D-E, H-L, and the condition flags with the A register, respectively. The symbol p_3 can be one of B, D, H, or SP, representing the pairs B-C, D-E, H-L, and the 16-bit stack pointer. The symbol k is a 16-bit value, as above.

Instruction	Examples
LXI p_3 k	LXI B 0FFFFH LXI SP 2048
PUSH p_2	PUSH PSW PUSH H
POP p_2	POP PSW POP D
DAD p_3	DAD B DAD H DAD SP
STAX p_1	STAX B STAX D
LDAX p_1	LDAX B LDAX D
INX p_3	INX B INX H INX SP
DCX p_3	DCX D DCX H DCX SP
SHLD k	SHLD 0 SHLD 3333H SHLD LAB+5
LHLD k	LHLD 260 LHLD 0EEEH LHLD LAB-3
XCHG	XCHG
XTHL	XTHL
SPHL	SPHL
PCHL	PCHL

4. Miscellaneous Instructions
In the instruction descriptions which follow, the symbol k represents a 16-bit quantity, as above, and the symbol k' denotes an 8-bit value.

Instruction	Meaning and Examples
STA k	STA 255 STA 0EFFH STA GAMMA
LDA k	LDA 44 LDA 3434Q LDA P1/I+5
IN k'	IN 33 IN OFFH IN 10101B
OUT k'	OUT 0 OUT 333Q OUT 25
RST n	RST 0 RST 5 RST 7
CMA	CMA
STC	STC
CMC	CMC
DAA	DAA
RLC	RLC
RRC	RRC
RAL	RAL
RAR	RAR
EI	EI
DI	DI
NOP	NOP
HLT	HLT

E. I/O Formatting Commands
INTERP/80 has a generalized I/O formatting interface which is somewhat dependent upon the installation. In general, a number of files are defined by file numbers (not necessarily corresponding externally to FORTRAN unit numbers). These file numbers correspond to devices as follows:

INPUT

INTERP/80 No.	Device	PDP-10 Device	File Name
1	User's Console	TTY 5	
2	Card Reader	CDR 2	
3	Paper Tape	PAP 6	
4	Magnetic Tape	MAG 16	
5	Magnetic Tape	DEC 9	
6	Disk	DISK 20	FOR20.DAT
7	Disk	DISK 21	FOR21.DAT

OUTPUT

INTERP/80 No.	Device	PDP-10 Device	File Name
1	User's Console	TTY 5	
2	Printer	PTR 3	
3	Paper Tape	PAP 7	
4	Magnetic Tape	MAG 17	
5	Magnetic Tape	DEC 10	
6	Disk	DISK 22	FOR22.DAT
7	Disk	DISK 23	FOR23.DAT

I/O functions are controlled through "$" commands which may be interspersed throughout the input.

Any input line with a "$" in column one, followed by a non-blank character is considered an I/O command. The card is then scanned for an "=" followed by a decimal integer. The character following the "$" and the integer value affect the I/O formatting functions as follows:

Control	Meaning	Initial Value
$BRIEF	Enable brief mode of the instruction trace; only the altered registers are displayed.	0
$COUNT=n	Start the output line count at the value n.	1
$DELETE=n	Delete all characters after column n of the output.	120
$EOF	End-of-file on this device.	0
$FASTLOAD	Load and punch hexadecimal machine code format if 1, otherwise assume BNPF format.	0
$GENLABELS	Print the symbolic label closest to the program counter whenever a break-point occurs.	1
$HEADING=n	Print a trace heading every n lines of the instruction trace. No headings are printed if n=0.	10
$INPUT=n	Read subsequent input from file number n.	1
$LEFT=n	Ignore character positions 1 through n-1 of the input.	1
$OUTPUT=n	Write subsequent output to file number n.	1
$PRINT	Controls listing of the output. If n=0, input lines are not printed; otherwise input is echoed.	0
$RIGHT=n	Ignore all character positions beyond column n of the input.	72
$TERMINAL	INTERP/80 assumes conversational usage if n=1; otherwise batch processing is assumed.	1
$USE	Use numbers on a BPNF code tape as the origin to the words of code to which they are prefixed.	0
$WIDTH=n	This command sets the width of the output line. Note that this affects the format of the DISPLAY MEMORY command.	72

The default values shown above assume conversational use with a teletype or similar device. The defaults can easily be changed by recompiling the INTERP/80 program.

In the case of controls which take on only 0 to 1 values (e.g. $PRINT, $TERMINAL, $FASTLOAD, $GENLABELS, $USE, and $EOF), the equal sign and decimal number may be omitted. The value of the control is complemented in this case.

The value of a control switch may be displayed by prefixing the name of the control with $; for example, $$I will display the value of $INPUT. All control values may be displayed by the use of $$ followed by a space or end-of-line.

Several controls may be referenced in a single command line:

```
                $B $F $P
```

will complement the $BRIEF, $FASTLOAD, and $PRINT controls.

F. INTERP/80 Error Messages

Execution Errors

1. Program counter stack overflow
2. Program counter stack underflow
3. Program counter outside simulated MCS-80 memory
4. Memory reference
5. Invalid machine code operator
6. End of file while reading port input
7. Invalid port input data (not between 0 and 255)

Command Mode Errors

1. Reference outside simulated MCS-80 memory
2. Insufficient space remaining in simulated MCS-80 memory
3. End-of-file encountered before expected
4. Input file number stack overflow (maximum of 7 indirect references)
5. Symbol not found in symbol table
6. Unused

...

10. I/O format command error (toggle has value other than 0 OR 1)
11. Unused
12. Invalid cascaded labels. Must be of form X/Y/Z.
13. Invalid search parameter in display symbol command (must be symbolic name, address, or *)
14. Display symbols command invalid since no symbol table exists
15. Unused
16. Unrecognized command or invalid format in command mode
17. Missing or extra characters following command
18. Lower bound exceeds upper bound or is less than zero in range list
19. The format of the symbol table is invalid (must be a sequence of the form N SY AD, where N is a integer, SY is the symbolic name, and AD is the address
20. Invalid BNPF tape format (character other than N or P was encountered within the B....F field)
21. Invalid hexadecimal code format (bad hex digit or missing :)
22. Unrecognized display element or invalid display format
23. Symbolic name not found in symbol table
24. Invalid address or no symbol table present in display symbol command
25. Output device width too narrow for display memory command (USE/$WIDTH - I/O format command to increase width)

26. Invalid radix in memory display command (must be CODE, BIN, OCT or DEC)
27. Unrecognized set element in set command
28. Missing set list in set command
29. Invalid set list or set value in set command
30. Missing or misplaced = in set command
31. Missing program stack element number in set PS N command
32. Invalid interrupt code specification (either more than one byte, or element exceeds 255, or not a valid 8080 machine instruction).

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES: TEORIA Y APLICACIONES

TEMA : IX : SISTEMAS DE DESARROLLO.

PROF. M. en C. ANGEL KURI M.

Abril, 1978.

```
 COPY EJEM.PLO TO :CO:
   EL SIGUIENTE TEXTO FUE ESCRITO BAJO GOBIERNO DE UN MICROPROCESADOR
8080 DE INTEL. EL TEXTO FUE DEPURADO Y EDITADO SIGUIENDO LOS COMAN-
DOS DEL SISTEMA OPERATIVO RESIDENTE EN MEMORIA.
   LA SEGUNDA PARTE DEL TEXTO (EL PROGRAMA PL/M) FUE GENERADO POR
EL COMPILADOR RESIDENTE EN LA MAQUINA HUESPED (TERMINO QUE MAS ADELAN-
TE SE CLARIFICA).
   ESTE TEXTO TIENE POR OBJETO ILUSTRAR LA VERSATILIDAD QUE PUEDE LLE-
GAR A TENER UN MICROPROCESADOR.
```

EN CUALQUIER SISTEMA DE MICROC[illegible] SE REQUIERE -------
DE UN SISTEMA POR MEDIO DEL CUAL SE PUEDA PROGRAMAR ADECUADA-
MENTE UN MICROSISTEMA; ESTO IMPLICA EL DESARROLLO DE PROGRAMAS
ESCRITOS EN ENSAMBLADOR, COMPILADOR O UNA COMBINACION DE AMBOS;
LA PRUEBA DE LOS PROGRAMAS ASI ESCRITOS; LA GRABACION DE PROMS
Y EPROMS A PARTIR DE DICHOS PROGRAMAS.

EL SISTEMA PRODUCIDO POR INTEL, Y QUE VA A SER DEMOSTRADO,
CONSTA DE LOS SIGUIENTES ELEMENTOS:

1) UN SISTEMA PROCESADOR CENTRAL.
2) MEMORIA RAM (32 K BYTES).
3) UN SISTEMA DE 'BOOT-STRAP' EN ROM (2 K BYTES).
4) UN SISTEMA DE DISKETTES DUAL, CON CAPACIDAD DE ALMACENAMIENTO DE 500,000 BYTES.
5) UNA LECTORA DE CINTA PERFORADA.
6) UN ESCRITORA/PERFORADORA DE CINTA PERFORADA.
7) UN SISTEMA PROGRAMADOR DE PROMS.
8) UN SISTEMA OPERATIVO PARA MENEJO DE DISCOS.
9) UN MACRO-ENSAMBLADOR EN LINEA.
10) UN EMULADOR ('HARDWARE'/'SOFTWARE')

ADICIONALMENTE, SE CUENTA CON UN SISTEMA DE COMPILADOR---
CRUZADO. POR MEDIO DE ESTE COMPILADOR (PL/M) ES POSIBLE ESCRI-
BIR PROGRAMAS PARA EL MICROPROCESADOR EN UN LENGUAJE DE ALTO -
NIVEL, COMPILARLO EN UNA MAQUINA HUESPED (EN ESTE CASO LA ----
B6700) Y PRODUCIR EL CODIGO DEL MICROPROCESADOR PARA SER USADO
EN EL MICROSISTEMA.

PARA EJEMPLIFICAR LA UTILIZACION DE UN SISTEMA COMO EL ANTE-
RIORMENTE MENCIONADO, SEGUIREMOS EL SIGUIENTE PROCESO ILUSTRATIVO:

A) ESCRIBIREMOS UN PROGRAMA EN PL/M, EN UNA TERMINAL B6700.
B) COMPILAREMOS ESTE PROGRAMA EN LA MAQUINA HUESPED.
C) OBTENDREMOS EL CODIGO DE MAQUINA EN FORMATO HEXADECIMAL.
D) PRODUCIREMOS UNA CINTA DE PAPEL, DE MODO QUE EL PROGRAMA PUE-DA SER LEIDO POR EL MDS.
E) LEEREMOS EL CODIGO ASI OBTENIDO PARA PROGRAMARLO EN UN EPROM.
F) PROGRAMAREMOS UN EPROM, DE MODO QUE NUESTRO PROGRAMA SE EN-CUENTRE RESIDENTE EN UN SISTEMA SDK/80.
G) CORREREMOS ESTE PROGRAMA EN EL SDK.

DE ESTA MANERA, TODO EL PROCESO DE DESARROLLO QUEDARA EJEMPLI-
FICADO.

DEBE NOTARSE QUE ADEMAS DE LOS 10 ELEMENTOS DE DESARROLLO ARRI-
BA MENCIONADOS, SE CUENTA ADICIONALMENTE CON LO SIGUIENTE:

11) UN SISTEMA EDITOR PARA ESCRIBIR LOS PROGRAMAS EN ENSAM-BLADOR.
12) UN SISTEMA SIMULADOR, RESIDENTE EN LA MAQUINA HUESPED.
13) UN ENSAMBLADOR CRUZADO RESIDENTE EN LA B6700.
14) UN SISTEMA EMULADOR HP.

EL COSTO APROXIMADO DE TODO ESTE EQUIPO ('HARDWARE' Y 'SOFTWARE')
EXCEDE A 16000 DOLARES. ESTO, POR SUPUESTO, NO INCLUYE TERMINALES,
TIEMPO DE LA MAQUINA HUESPED, SISTEMAS TALES COMO EL SDK/80 Y/O LAS
MEMORIAS DE SOLO LECTURA INVOLUCRADAS.

A CONTINUACION SE INCLUYE EL PROGRAMA EN PL/M QUE TIPIFICARA EL SIS-
TEMA DE DESARROLLLO, CON COMENTARIOS QUE PERMITAN SU FACIL COMPREN-
SION. DEBE NOTARSE QUE ESTE PROGRAMA HACE EXTENSO USO DE UNAS RUTI-
NAS QUE POSIBILITAN LAS OPERACIONES CON DECIMALES.

LAS DECLARACIONES QUE SE INCLUYEN AL PRINCIPIO PERMITEN LA INTER-
FAZ CON EL LENGUAJE ENSAMBLADOR.

EL PROGRAMA USA LAS SIGUIENTES RUTINAS:
1) SUMA (SM). SUMA ALGEBRAICAMENTE DOS NUMEROS EN PUNTO FLOTANTE.
2) DIVISION (DV). DIVIDE DOS NUMEROS EN PUNTO FLOTANTE.
3) MULTIPLICACION (MT). MULTIPLICA DOS NUMEROS EN PUNTO FLOTANTE.
4) ASCII A BCD (AB). EFECTUA LA CONVERSION DE UN NUMERO REPRESENTADO COMO DATOS (DATA) EN ASCII A UNA REPRESENTACION EN PUNTO FLOTANTE. DE ESTA MANERA, ES FACIL ALIMENTAR UN DATO DESDE CONSOLA Y CONVERTIRLO A PUNTO FLOTANTE.
5) BCD A ASCII (BA). EFECTUA LA CONVERSION INVERSA A (AB). ES DECIR, CONVIERTE UN NUMERO FLOTANTE A SU REPRESENTACION EN ASCII, PARA FACILIDAD DE SALIDA POR CONSOLA.

FORMATOS EN LA 8080.
EN LA INTEL/8080 EXISTEN LOS SIGUIENTES MODOS DE ALMACENAMIENTO INTERNO:

1) BYTE. ALMACENAMIENTO EN 8 BITS. CORRESPONDE A UNA PALABRA EN MEMORIA, O A UN REGISTRO EN EL CPU.
2) ADDRESS. ALMACENAMIENTO EN 16 BITS. CORRESPONDE A DOS PALABRAS EN MEMORIA, O A UNA PAREJA DE REGISTROS EN EL CPU.
3) ASCII. EN ESTE CASO, EL ALMACENAMIENTO OCUPA UNA PALABRA EN MEMORIA, PERO CADA PALABRA ASI UTILIZADA CORRESPONDE A UN 'CARACTER'.
ASI, LA LETRA 'A' CORRESPONDE A LA CADENA 01000001; EL NUMERO '3' CORRESPONDE A LA CADENA 00110011; ETC. ESTA REPRESENTACION ES LA QUE GENERALMENTE INTERPRETAN LOS DISPOSITIVOS DE ENTRADA/SALIDA.
4) DECIMAL. ALMACENAMIENTO DE 8 BITS, AGRUPADOS EN 2 CIFRAS 'BCD', TALES QUE SOLO LAS COMBINACIONES BINARIAS DEL 0 AL 9 SON UTILIZADAS.

----5) A ESTE TIPO DE REPRESENTACIONES "NATURALES" DE LA 8080, HEMOS AGREGADO UN TIPO QUE DENOMINAMOS 'PUNTO FLOTANTE'. EN ESTE TIPO DE REPRESENTACION, CADA PALABRA OCUPA 5 LOCALIDADES DE MEMORIA, DE LA SIGUIENTE FORMA:

```
          <EXPONENTE> <BYTE> <BYTE> <BYTE> <BYTE>
```

END DONDE,

```
 <EXPONENTE>=<SIGNO EXPONENTE> <6 BITS DE EXPONENTE> <SIGNO MANTISA>
```

LOS <6 BITS DE EXPONENTE> ESTAN EN COMPLEMENTO A DOS.
ASIMISMO,

```
<BYTE>=<BCD> <BCD>
```

Y CADA <BCD> REPRESENTA UN DIGITO DEL 0 AL 9.
EN ESTA REPRESENTACION, EL NUMERO 3.478 ESTA REPRESENTADO POR:

```
   <0 000001 0> <0011 0100> <0111 1000> <0000 0000> <0000 0000>
```

Y EL NUMERO -0.0076543299, POR:

```
    <1 111110 1> <0111 0110> <0101 0100> <0011 0010> <1001 1001>
```

ANTES DE ENTRAR AL PROGRAMA, ES CONVENIENTE HACER NOTAR LAS SIGUIENTES CARACTERSITICAS DE PL/M:
1) ES UN LENGUAJE ESTRUCTURADO, ES DECIR, PERMITE TRABAJAR EN BLOQUES, TALES COMO PROCEDURES.
2) NO ES UN PROGRAMA RECURSIVO. NO SE PERMITE QUE UNA RUTINA SE LLAME A SI MISMA, O UQE VARIAS RUTINAS SE LLAMEN ENTRE ELLAS CIRCULARMENTE. EN EL EJEMPLO DE DIVISION RECURSIVA QUE SE LES PROPORCIONO, SE ILUSTRA LA FORMA DE LOGRAR RECURSIVIDAD DESDE EL ENSAMBLADOR.
4) PERMITE LA INTERFAZ CON CODIGO GENERADO CON ENSAMBLADOR. ESTO SE ILUSTRA EXTENSAAMENTE EN EL PROGRAMA SIGUIENTE.

EN EL PROGRAMA HAY QUE NOTAR LOS SIGUIENTES PUNTOS:
A) HACEN USO DEL MICROPROCESADOR COMO COMPUTADORA DE PROPOSITO

GENERAL, EN LA CUAL ES POSIBLE LLEVAR A CABO CALCULOS ARITMETICOS COMPLEJOS QUE GENERALMENTE SE ASOCIAN A MAQUINAS DE MUCHO MAYOR COM-PLEJIDAD.
B) UTILIZA CODIGO OPTIMIZADO (ESCRITO EN ENSAMBLADOR) PARA MINI-MIZAR EL TIEMPO DE PROCESO.
C) UTILIZA RUTINAS DE ENTRADA/SALIDA RESIDENTES EN EL MONITOR DEL SDK/80 PARA REDUCIR LA GENERACION DE CODIGO Y FACILITAR LA PROGRA-MACION.
D) PERMITE LA FACIL COMUNICACION MAQUINA-HOMBRE. --- EN ESTE SEN-TIDO CREEMOS NECESARIO HACER ENFASIS, PUES A NIVEL MICROPROCESADORES, ESTE ASPECTO COBRA SINGULAR IMPORTANCIA. ---

-ES DE NOTARSE QUE EN ESTE PROGRAMA HA HABIDO ERRORES DE EDICION,

QUE AHORA PROCEDEREMOS A CORREGIR.

```
EDIT EJEM.PLO

ISIS-II TEXT EDITOR, V1.6
*A$A$A$A$A$A$A$A$A$A$A$A$A$A$A$$
*Z$-2L$2T$$


*B$F UQE$$
*-1L$SUQE$QUE$$
*SEXTENSAAMENTE$EXTENSAMENTE$$
*B$FFORMATOS EN LA 8080.$$
*-1L$65T$$
```

FORMATOS EN LA 8080.
EN LA INTEL/8080 EXISTEN LOS SIGUIENTES MODOS DE ALMACENAMIENTO INTERNO:

1) BYTE. ALMACENAMIENTO EN 8 BITS. CORRESPONDE A UNA PALABRA EN MEMORIA, O A UN REGISTRO EN EL CPU.
2) ADDRESS. ALMACENAMIENTO EN 16 BITS. CORRESPONDE A DOS PALABRAS EN MEMORIA, O A UNA PAREJA DE REGISTROS EN EL CPU.
3) ASCII. EN ESTE CASO, EL ALMACENAMIENTO OCUPA UNA PALABRA EN MEMORIA, PERO CADA PALABRA ASI UTILIZADA CORRESPONDE A UN 'CARACTER'.
ASI, LA LETRA 'A' CORRESPONDE A LA CADENA 01000001; EL NUMERO '3' CORRESPONDE A LA CADENA 00110011; ETC. ESTA REPRESENTACION ES LA QUE GENERALMENTE INTERPRETAN LOS DISPOSITIVOS DE ENTRADA/SALIDA.
4) DECIMAL. ALMACENAMIENTO DE 8 BITS, AGRUPADOS EN 2 CIFRAS 'BCD', TALES QUE SOLO LAS COMBINACIONES BINARIAS DEL 0 AL 9 SON UTILIZADAS.

----5) A ESTE TIPO DE REPRESENTACIONES "NATURALES" DE LA 8080, HEMOS AGREGADO UN TIPO QUE DENOMINAMOS 'PUNTO FLOTANTE'. EN ESTE TIPO DE REPRESENTACION, CADA PALABRA OCUPA 5 LOCALIDADES DE MEMORIA, DE LA SIGUIENTE FORMA:

```
          <EXPONENTE> <BYTE> <BYTE> <BYTE> <BYTE>
    END DONDE,
 <EXPONENTE>=<SIGNO EXPONENTE> <6 BITS DE EXPONENTE> <SIGNO MANTISA>
```

LOS <6 BITS DE EXPONENTE> ESTAN EN COMPLEMENTO A DOS.
ASIMISMO,

```
<BYTE>=<BCD> <BCD>
```

Y CADA <BCD> REPRESENTA UN DIGITO DEL 0 AL 9.
EN ESTA REPRESENTACION, EL NUMERO 3.478 ESTA REPRESENTADO POR:

```
   <0 000001 0> <0011 0100> <0111 1000> <0000 0000> <0000 0000>
```

Y EL NUMERO -0.0076543299, POR:

```
   <1 111110 1> <0111 0110> <0101 0100> <0011 0010> <1001 1001>
```

ANTES DE ENTRAR AL PROGRAMA, ES CONVENIENTE HACER NOTAR LAS SIGUIENTES CARACTERSITICAS DE PL/M:
1) ES UN LENGUAJE ESTRUCTURADO, ES DECIR, PERMITE TRABAJAR EN BLOQUES, TALES COMO PROCEDURES.
2) NO ES UN PROGRAMA RECURSIVO. NO SE PERMITE QUE UNA RUTINA SE LLAME A SI MISMA, O QUE VARIAS RUTINAS SE LLAMEN ENTRE ELLAS CIRCULARMENTE. EN EL EJEMPLO DE DIVISION RECURSIVA QUE SE LES PROPORCIONO, SE ILUSTRA LA FORMA DE LOGRAR RECURSIVIDAD DESDE EL ENSAMBLADOR.
4) PERMITE LA INTERFAZ CON CODIGO GENERADO CON ENSAMBLADOR. ESTO SE ILUSTRA EXTENSAMENTE EN EL PROGRAMA SIGUIENTE.

EN EL PROGRAMA HAY QUE NOTAR LOS SIGUIENTES PUNTOS:
A) HACEN USO DEL MICROPROCESADOR COMO COMPUTADORA DE PROPOSITO GENERAL, EN LA CUAL ES POSIBLE LLEVAR A CABO CALCULOS ARITMETICOS COMPLEJOS QUE GENERALMENTE SE ASOCIAN A MAQUINAS DE MUCHO MAYOR COMPLEJIDAD.
B) UTILIZA CODIGO OPTIMIZADO (ESCRITO EN ENSAMBLADOR) PARA MINIMIZAR EL TIEMPO DE PROCESO.
C) UTILIZA RUTINAS DE ENTRADA/SALIDA RESIDENTES EN EL MONITOR DEL SDK/80 PARA REDUCIR LA GENERACION DE CODIGO Y FACILITAR LA PROGRAMACION.
D) PERMITE LA FACIL COMUNICACION MAQUINA-HOMBRE. --- EN ESTE SENTIDO CREEMOS NECESARIO HACER ENFASIS, PUES A NIVEL MICROPROCESADORES, ESTE ASPECTO COBRA SINGULAR IMPORTANCIA. ---

1

```
1   0C00H;
1
1   DECLARE LIT LITERALLY 'LITERALLY',
1       DCL LIT 'DECLARE';
1
1   DCL (TA,TB,TC,TD) (5) BYTE;
1
1   MT: PROCEDURE(A,B,C);
2   DCL (A,B,C) ADDRESS,
2       RES BASED A ADDRESS,
2       MULT LIT '400H';
2           RES=B;
2           GO TO MULT;
2   END MT;
1
1   DV: PROCEDURE(A,B,C);
2   DCL (A,B,C) ADDRESS,
2       RES BASED A ADDRESS,
2       DIV LIT '53DH';
2           RES=B;
2           GO TO DIV;
2   END DV;
1
1   SM: PROCEDURE(A,B,C);
2   DCL (A,B,C) ADDRESS,
2       RES BASED A ADDRESS,
2       SUM LIT '6BDH';
2           RES=B;
2           GO TO SUM;
2   END SM;
1
1   AB: PROCEDURE(A,B);
```

```
2   DCL (A,B) ADDRESS,
2       ASCBCD LIT '8C8H';
2           GO TO ASCBCD;
2   END AB;
1
1   BA: PROCEDURE(A,B,FF);
2   DCL (A,B) ADDRESS,
2       FF BYTE,
2       PARAM BASED A BYTE,
2       BCDASC LIT '0A41H';
2           PARAM=B;
2           GO TO BCDASC;
2   END BA;
1
1   EQ: PROCEDURE(A,B);
2   DCL (A,B) ADDRESS,
2       EQUAL LIT '08ADH';
2           GO TO EQUAL;
2   END EQ;
1
1   SENO: PROCEDURE(SENOPTR,ARGPTR,N);
2   DCL (SENOPTR,ARGPTR) ADDRESS,
2       SINE BASED SENOPTR ADDRESS,
2       ARG BASED ARGPTR ADDRESS,
2       (I,J,N) BYTE,
2       UNO DATA('1.0D'),
2       (SG,UNOF,NF,S0,S1,S2,FAC,POT)(5) BYTE;
2
2       CALL AB(.UNOF,.UNO);
2       CALL EQ(.NF,.UNOF);
2       CALL EQ(.FAC,.UNOF); /* FAC=1 */
2       CALL EQ(.POT,.ARG); /* POT=ARG */
```

```
2       CALL MT(.SQ,.ARG,.POT); /* SQ=ARG**2 */
2       J=0FFH;
2       DO I=3 TO (2*N-1) BY 2;
2          J=NOT(J);
3          CALL SM(.S1,.NF,.UNOF); /* S1=NF+1 */
3          CALL SM(.S2,.S1,.UNOF); /* S2=NF+2 */
3          CALL EQ(.NF,.S2); /* NF=NF+2 */
3          CALL MT(.S0,.S1,.S2); /* S0=(NF+1)*(NF+2) */
3          CALL MT(.S1,.FAC,.S0);
3          CALL EQ(.FAC,.S1); /*FAC=FAC*(NF+1)*(NF+2) */
3          CALL EQ(.S0,.POT);
3          CALL MT(.POT,.S0,.SQ); /* POT=POT*(ARG**2) */
3          CALL DV(.S0,.POT,.FAC);
3          IF J=0 THEN S0=(NOT(S0) AND 1) OR (S0 AND 0FEH);
3          CALL EQ(.S1,.SINE);
3          CALL SM(.SINE,.S0,.S1); /* SINE=SINE+POT/FAC */
3       END;
2    END SENO;
1
1    COSENO: PROCEDURE(COSPTR,ARGPTR);
2    DCL (COSPTR,ARGPTR) ADDRESS,
2       COS BASED COSPTR BYTE,
2       ARG BASED ARGPTR BYTE,
2       PI$MED DATA('1.5707963D'),
2       TEMP(5) BYTE;
2    CALL AR(.COS,.PI$MED); /* COS=PI/2 */
2    ARG=(NOT(ARG) AND 1) OR (ARG AND 0FEH); /* ARG=-ARG */
2    CALL SM(.TEMP,.COS,.ARG); /* TEMP=PI/2-ARG */
2    CALL SENO(.COS,.TEMP,6); /* COS(ARG)=SENO(PI/2-ARG) */
2    END COSENO;
1
```

```
1    TAN: PROCEDURE(TANPTR,ARGPTR);
2    DCL (TANPTR,ARGPTR) ADDRESS,
2        TAN BASED TANPTR BYTE,
2        ARG BASED ARGPTR BYTE,
2        TEMP(5) BYTE;
2    CALL EQ(.TEMP,.ARG);
2    CALL SENO(.TAN,.ARG,6); /* TAN=SENO(ARG) */
2    CALL COSENO(.ARG,.TEMP); /* ARG= COSENO(TEMP)= COSENO(ARG) */
2    CALL DV(.TEMP,.TAN,.ARG); /*TEMP=TAN/ARG=SENO/COSENO */
2    CALL EQ(.TAN,.TEMP);
2    END TAN;
1
1    READ$BYTE: PROCEDURE BYTE;
2    DCL CHAR BYTE;
2        GETCH: PROCEDURE BYTE;
3            GO TO 21BH;
3        END;
2        ECHO: PROCEDURE(CHAR);
3        DCL CHAR BYTE;
3            GO TO 1F4H;
3        END;
2    CHAR=GETCH;
2    CALL ECHO(CHAR);
2    RETURN CHAR;
2    END READ$BYTE;
1
1    WRITE$CHAR: PROCEDURE(CHAR);
2    DCL CHAR BYTE,
2        CO LIT '1F5H';
2            GO TO CO;
2    END;
```

```
1
1   DO;
1
1   DCL BUFFER(14) BYTE, (ARG,RES)(5) BYTE,
2       (CHAR,I) BYTE,
2       LF LIT '0AH',
2       CR LIT '0DH';
2
2   DO I-0 TO 14;
2       CHAR-READ$BYTE;
3       IF CHAR - CR THEN GO TO EXEC;
3       BUFFER(I)=CHAR;
3   END;
2   EXEC:
2       CALL WRITE$CHAR(LF);
2       CALL AB(.ARG,.BUFFER);
2       I-READ$BYTE;
2       CALL WRITE$CHAR(CR);
2       CALL WRITE$CHAR(LF);
2       I-I-30H;
2       IF I > 2 THEN GO TO FIN;
2       DO CASE I;
2           CALL SENO(.RES,.ARG,6);
3           CALL COSENO(.RES,.ARG);
3           CALL TAN(.RES,.ARG);
3       END;
2
2   CALL BA(.BUFFER,.RES,READ$BYTE);
2   DO I-0 TO 14;
2       CHAR-BUFFER(I);
3       CALL WRITE$CHAR(CHAR);
3   END;
```

6

```
[illegible]

00162  2

00163  2    END /* DO */;

00164  1

00165  1    EOF

NO PROGRAM ERRORS

$ET=4:46.4 PT=51.0 IO=1.9
F FL/MI I/4;FILE FILE23(MAXRECSIZE=22,BLOCKSIZE=22);%
#%
FILE FILE23(MAXRECSIZE=22,BLOCKSIZE=22)
#KEYWORD EXPECTED. SCANNING;
#?

8080 PLM2 VERS 4.0

$T=7 $V=16

STACK SIZE = 10 BYTES
MEMORY...........................1100H
TA...............................106CH
TB...............................1071H
TC...............................1076H
TD...............................107BH
MT...............................0C06H
A................................1080H
B................................1082H
C................................1084H
DV...............................0C1FH
A................................1086H
B................................1088H
C................................108AH
SM...............................0C38H
A................................108CH
B................................108EH
C................................1090H
AR...............................0C51H
A................................1092H
B................................1094H
BO...............................0C5FH
A................................1096H
B................................1098H
IF...............................109BH
FQ...............................0C75H
A................................109CH
B................................109EH
SEND.............................0C83H
SENDPTR..........................10A0H
RCVPTR...........................10A2H
N................................10A5H
I................................10A6H
J................................10A7H
UND..............................0C8FH
SA...............................10A8H
UNDF.............................10ADH
NF...............................10B2H
SO...............................10B7H
S1...............................10BCH
S2...............................10C1H
LAC..............................10C6H
```

```
PUT..............................10C[illegible]
COSENO...........................0DF4H
COSPTR...........................10D0H
ARGPTR...........................10D2H
PIMED............................0E01H
TEMP.............................10D5H
TAN..............................0E4CH
TANPTR...........................10DAH
ARGPTR...........................10DCH
TEMP.............................10DEH
READBYTE.........................0EA1H
CHAR.............................10E3H
GETCH............................0EA4H
ECHO.............................0EA8H
CHAR.............................10E4H
WRITECHAR........................0EC0H
CHAR.............................10E5H
BUFFER...........................10E6H
ARGS.............................10F4H
RES..............................10F9H
CHAR.............................10FEH
I................................10FFH
EXEC.............................0EF4H
FIN..............................0FA4H
NO PROGRAM ERRORS

EFT=2:12.1 PT=17.0 IO=3.5
```

Next-generation development systems are costing less and doing more

To match the expanding range of microprocessor applications, a new family of development tools offers flexible capabilities with a $3,250 bottom price tag

by Paul Rosenfeld, *Intel Corp., Santa Clara, Calif.*

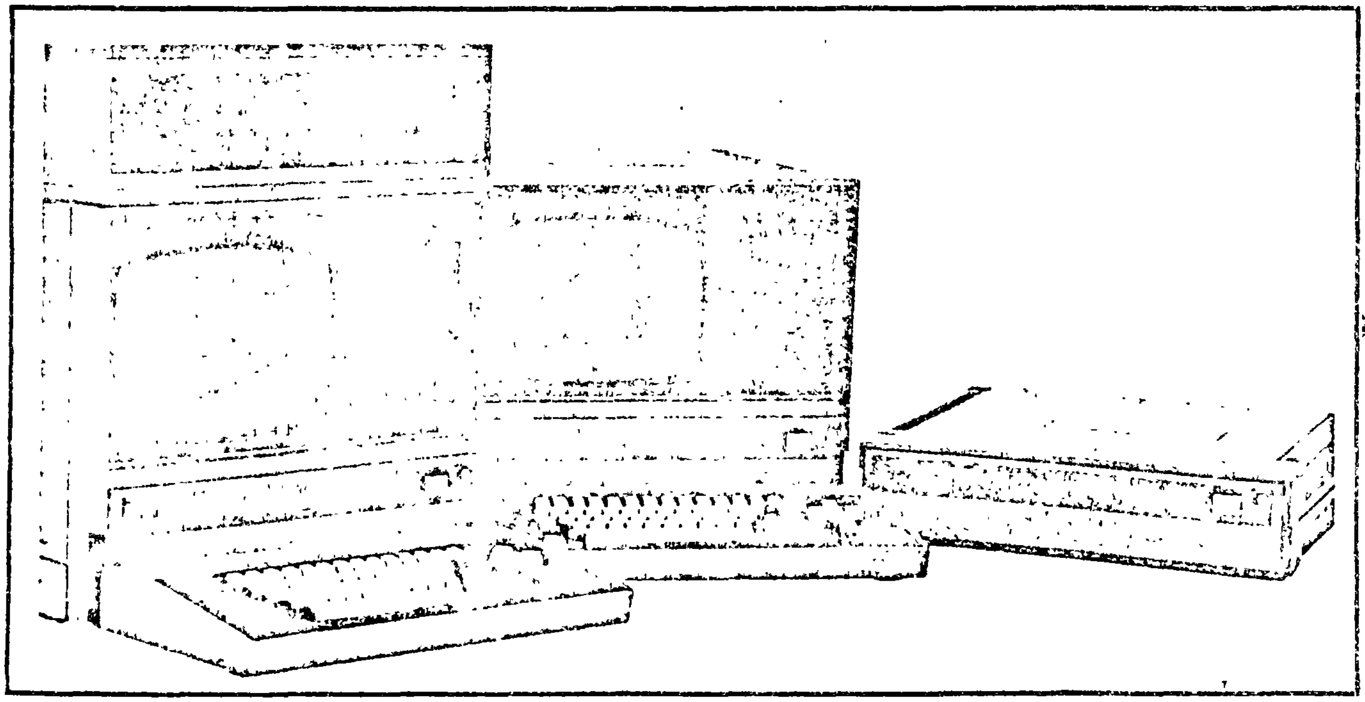
☐ Capitalizing on the latest advances in semiconductor technology, a new generation of development systems helps accomplish the widely varying, detailed design tasks that arise as microprocessors appear in more and more applications. The design process for such products calls for developmental tools that let the designer choose from a variety of solutions, including the appropriate software. As well as filling this need, the new generation of development systems is the next step in the design of basic hardware that can be upgraded quickly to support new, more powerful microprocessors.

During the early years of the microprocessor, development systems were almost unknown. Most users struggled to develop their own special hardware and software, or else they turned to a timesharing service for simulation of their application. The primitive development systems available were useful only for the simplest, least complicated tasks.

However, the technology of the development systems has advanced as rapidly as that of the microprocessor, so that today they provide the tools to solve a wide range of intricate design problems. This advance is due in part to the increased power of microprocessors that are incorporated into the system hardware. It also is due to Intel's development of ICE, the in-circuit emulator [*Electronics*, April 15, 1976, p. 116].

The ICE modules give designers full control over development of a microprocessor-based product, because it permits complete hardware and software debugging in a prototype at real-time speed. Thus it is no longer necessary to build special instruments or use timesharing simulators. Moreover, hardware and software can be debugged together rather than separately.

While the ICE module added significant debugging capability to development systems, new software tools were also needed. Even though most microcomputers are programmed in assembly language, more engineers are turning to high-level languages. The advance of resident compilers in development systems including both PL/M and now Fortran have further increased the need to use a



1. Flexible developer. The Series II microprocessor development system makes extensive use of new peripheral control chips for a compact unit with built-in CRT and floppy-disk drive on two models. There are three models in all, designed to cover a wide range of applications.

SERIES II VS MDS-800 FAMILY

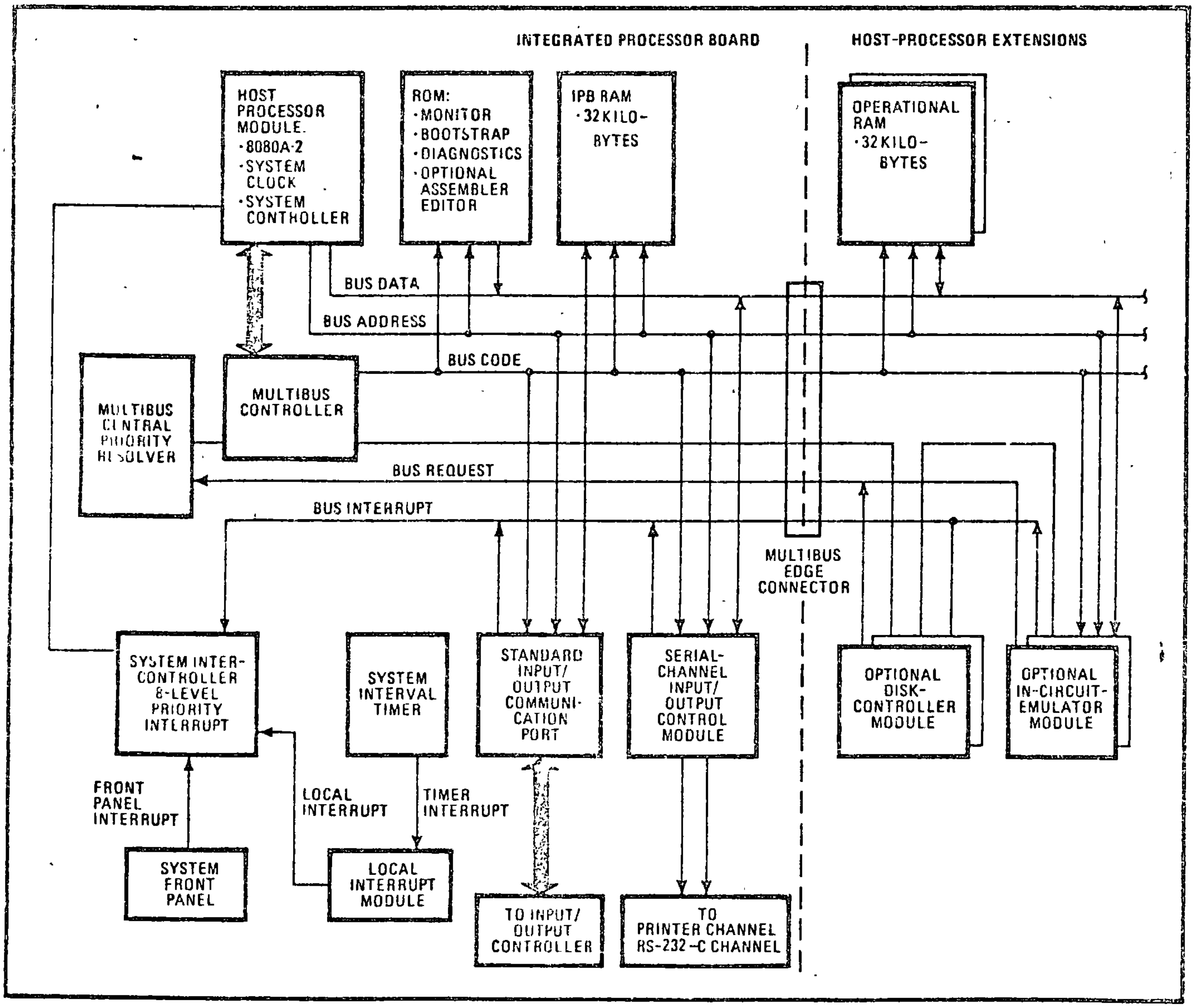
Price		Hardware		
			Series II	MDS-800
Series II, Model 210 MDS 800, plus 16 kilobytes additional RAM	$ 3,250 $ 5,325	Standard boards	integrated processor board input/output controller (outside card cage holding other boards)	CPU front-panel monitor MDS-016 and/or MDS-032
Series II, Model 220 MDS-800, plus 16 kilobytes of RAM, a CRT,display, and a single disk drive	$ 7,450 $10,415	Optional boards for 220/230	32 kilobytes of RAM disk controller (2 boards)	32 kilobytes of RAM disk controller (2 boards)
Series II, Model 230 MDS 800, plus 48 kilobytes of RAM, a CRT, and a dual-density disk drive	$12,900 $13,800	Total slots used	1 (210/220) 4 (230)	
Series II, Model 230 MDS 888	$12,900 $15,000	Slots available for expansion	3 (210/220) 2 (230) (with 4 more as an option)	

microcomputer development system as a design tool.

Now a new series of development systems, designed from the ground up to provide the capabilities needed by the designer, allows all this flexibility in a compact package (Fig. 1). This Intellec Series II family consists of three models with a range of capabilities that make it possible to match the development system to the particular developmental requirement, eliminating the need for a separate keyboard and display, a separate floppy-disk drive, and a controller. What's more, there are significant price reductions over roughly comparable models in the original Intellec series (see table).

Three models for many uses

The simplest Series II unit is the model 210, for the designer who wants to get a small job done fast. Its assembler and test editor, based in read-only memory, and 32,000 bytes of random-access memory provide assembly of source code directly from RAM for small and medium-sized programs. Its $3,250 price tag is a breakthrough in cost.


INTEGRATED PROCESSOR BOARD
HOST-PROCESSOR EXTENSIONS
HOST PROCESSOR MODULE. •8080A-2 •SYSTEM CLOCK •SYSTEM CONTROLLER
ROM: •MONITOR •BOOTSTRAP •DIAGNOSTICS •OPTIONAL ASSEMBLER EDITOR
IPB RAM •32KILO-BYTES
OPERATIONAL RAM •32KILO-BYTES
BUS DATA
BUS ADDRESS
BUS CODE
MULTIBUS CENTRAL PRIORITY RESOLVER
MULTIBUS CONTROLLER
BUS REQUEST
BUS INTERRUPT
MULTIBUS EDGE CONNECTOR
SYSTEM INTER-CONTROLLER 8-LEVEL PRIORITY INTERRUPT
SYSTEM INTERVAL TIMER
STANDARD INPUT/OUTPUT COMMUNI-CATION PORT
SERIAL-CHANNEL INPUT/OUTPUT CONTROL MODULE
OPTIONAL DISK-CONTROLLER MODULE
OPTIONAL IN-CIRCUIT-EMULATOR MODULE
FRONT PANEL INTERRUPT
LOCAL INTERRUPT
TIMER INTERRUPT
SYSTEM FRONT PANEL
LOCAL INTERRUPT MODULE
TO INPUT/OUTPUT CONTROLLER
TO PRINTER CHANNEL RS-232-C CHANNEL

2. Host with the most. The integrated processor board in the Series II has a host processor module designed around the 8080A-2. It features a Multibus controller, which routes communications traffic between the master and slave modules throughout the system.

The second new entry, model 220, is geared toward those designers working on microprocessor-based systems that require more powerful development support capabilities. To the basic series design, it adds a full-sized floppy-disk drive, a cathode-ray-tube display, and a full ASCII keyboard, all in one chassis. It costs $7,450, almost $3,000 less than an earlier Intellec in the same configuration.

The third in the series, model 230, adds 32 kilobytes of RAM and has a million bytes of on-line disk storage with ISIS-II diskette-operating-system software. The price for all this capability is $12,900. To get an earlier Intellec to this level would cost over $13,000.

All three Series II models contain a card cage with an optional expansion module and can support a complete line of peripherals such as printers, paper-tape readers and punches, and programmable-ROM programmers. In addition, the 220 and 230 models can host in-circuit emulators for a variety of microprocessors.

Incorporated into the series are up-to-date microprocessor and memory chips, plus recently developed, highly integrated peripheral controller chips that perform logic functions as well as processing functions. Besides the 8080A-2 central processing unit used in all three models, the 220 and 230 take advantage of the new 8271 floppy-disk controller chip, the 8275 CRT controller chip, and other 8200-family multifunction input/output components. All three models use the UPI-41, a new universal peripheral-interface chip to perform general-purpose parallel peripheral-interface control functions, as well as to control the keyboard on the 220 and 230.

Like the original Intellec, each Series II model has a system monitor resident in ROM. In addition, a ROM-based 8080/8085 assembler and text editor is offered with the model 210. Combined with the capabilities of the monitor, it enables users to edit and assemble their assembly-language programs directly from RAM without the use of paper tape. A ROM editor/assembler from the MCS48 family is optional.

The other two models use the ISIS-II diskette operating system to supplement the system monitor. Through the file-manipulation capabilities of ISIS-II, designers have access to comprehensive disk-based program-development software such as assemblers for a variety of Intel microprocessors, compilers for both PL/M and Fortran, and a sophisticated relocation and linkage system.

Finally, to keep downtime to a minimum, all Series II models provide a ROM-resident diagnostic package. It enables designers to determine whether their Series II systems are working fully and then to locate trouble spots within easily replaced functional blocks.

Architecture promotes expansion

The various components that make up the Series II hardware are in two independent microcomputer subsystems. Each on its own printed-circuit board, these subsystems communicate with each other and maintain a master-slave relationship.

The master is called the integrated processor board (Fig. 2), and essentially it is a complete microcomputer system. In addition to the usual management and control functions of a master subsystem, it serves as the interface with the in-circuit emulator and other optional microcomputer-development modules that may reside in the card cage. The modules that make up the master system are interconnected through Intel's Multibus® concept, which allows expansion in modular fashion. (See "A ride on the bus" p. 117.)

The host CPU is designed with the MCS-80 microcomputer family. The 8080A-2 microprocessor has the power for fast and efficient support of the operating system, several high-level languages, and other microcomputer-related software tasks, as well as interfacing with the bus. The interface between the central processor module and the Multibus is the 8218 Multibus master control chip.

The main memory module accommodates both ROM and RAM chips. The read-only memory is used for fixed system programs such as the system monitor, bootstrap, diagnostics, and in the case of model 210, the editor and assembler. The RAM is used for programs that need not remain in memory: system and user programs, including the ISIS-II, assemblers, and compilers.

The RAM and ROM address spaces partially overlap, which in effect extends the available main memory beyond the 64,000 bytes allowed by the 8080 microprocessor. This partial overlap is possible because the program in the overlapped section of ROM, the system bootstrap and diagnostics, is never executed concurrently with RAM programs in the same address space.

All on one board

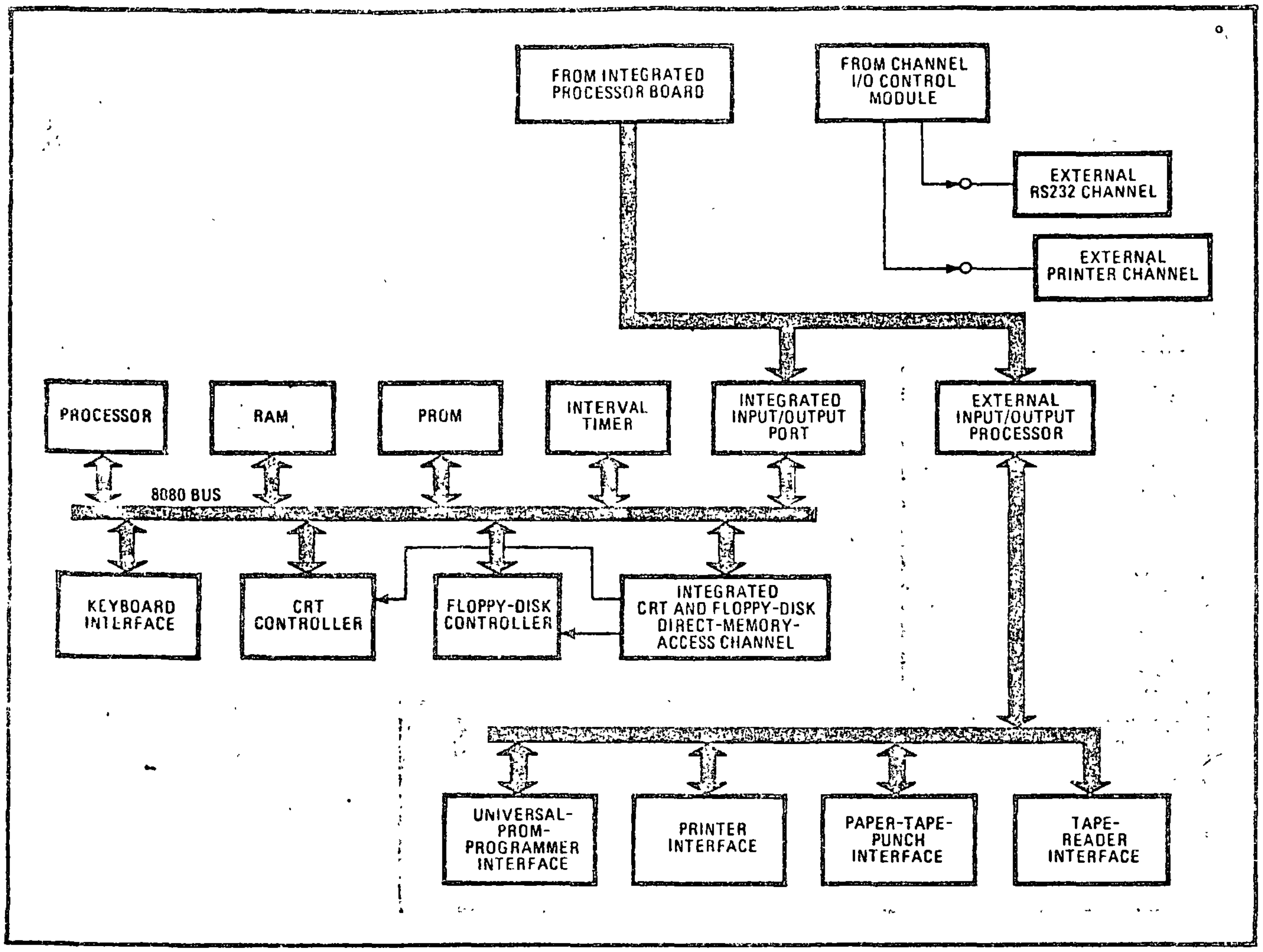
Using the connecting Multibus configuration, the integrated processor board brings the 8080A-2 CPU and the memory module together with two serial channels with both RS-232-C and current-loop provisions, the timer and interrupt modules, and the communication port to the slave I/O control processor. In addition, it has provisions for plugging in a small piggyback pc board that houses 20,000 more kilobytes of ROM. The original Intellec required five boards for the same configuration.

The resident RAM is implemented with 16,384-bit chips, and the resident ROM also uses 16-k chips. The two serial channels are provided via two 8251 universal synchronous/asynchronous receiver/transmitters. An 8253 interval timer generates the serial-channel baud-rate signal, as well as the logical real-time clock interrupt. System and front-panel interrupts are controlled via an 8259 interrupt processor. Interface to the slave I/O is with an 8255 programmable I/O controller.

To facilitate communications with various options that extend the host processor, the integrated processor board initiates and controls the bus by means of an edge connector. This arrangement extends the bus to the pc motherboard and to the rest of the system.

The slave processor is on the I/O controller board and comes in two versions. The one used with the 220 and 230 contains controllers both for the integrated CRT and the floppy-disk drive and for the interface to parallel peripherals. For the 210, a pared-down version contains only the interface controller for parallel peripherals. Both are shown in Fig. 3, where the pared-down version is within the color tint.

Appearing in both versions is the new UPI-41, the


FROM INTEGRATED PROCESSOR BOARD
FROM CHANNEL I/O CONTROL MODULE
EXTERNAL RS232 CHANNEL
EXTERNAL PRINTER CHANNEL
PROCESSOR
RAM
PROM
INTERVAL TIMER
INTEGRATED INPUT/OUTPUT PORT
EXTERNAL INPUT/OUTPUT PROCESSOR
8080 BUS
KEYBOARD INTERFACE
CRT CONTROLLER
FLOPPY-DISK CONTROLLER
INTEGRATED CRT AND FLOPPY-DISK DIRECT-MEMORY-ACCESS CHANNEL
UNIVERSAL-PROM-PROGRAMMER INTERFACE
PRINTER INTERFACE
PAPER-TAPE-PUNCH INTERFACE
TAPE-READER INTERFACE

3. To the outside world. The input/output controller board is the link between the integrated processor board and various I/O units. In the model 210, a UPI-41 I/O processor controls external units (shaded area). The 220 and 230 have control chips for the CRT and disk drive.

single-chip peripheral controller containing an on-chip 1,024-bit ROM, 64 bytes of RAM, and the necessary I/O ports to perform all control and data-transfer functions that are required between the peripherals and the host computer. The 220/230 version, on the other hand, adds its own 8080A-2 processor to set up and direct the concurrent operation of the floppy disk and the CRT display.

The keyboard has its own independent single-chip computer, another UPI-41. It performs all necessary keyboard tasks including creating key codes, controlling key rollover and multiple-key depression, and transmitting key codes to this board's 8080A-2 via an 8255 I/O port. The I/O controller board contains 8,000 bytes of built-in RAM, which functions both as the data-transfer buffer between the host processor and the peripherals that are part of the package and as temporary storage for the slave processor.

Data transfer between the CRT and the disk drive and the host processor takes place in two steps. First, there is transfer of data between the host memory and the slave RAM. This transfer takes place on a byte-at-a-time basis under the direct program control of the processors in both the host and the slave. Next comes data transfer between the slave memory and the target peripheral. This transfer takes place under direct-memory-access control, using an 8257 DMA controller.

The data-transfer operation to the external parallel peripherals, such as a printer or PROM programmer, takes place through the UPI-41 single-chip peripheral controller. For the packaged peripherals—the CRT, the floppy-disk controller, and the DMA controller—the programs reside in 8 kilobytes of ROM on the I/O controller board. They are responsible for setup, direction, and allocation of slave-processor resources, including the various controllers and the memory space. Also, they are responsible for controlling data transfer to and from the host processor. The host-slave transfers are initiated upon request from the host and take place over an 8-bit bidirectional data link between the host and the slave, using a standard protocol that was developed for communication between the 8080 and the UPI-41.

Packaging together the most commonly used elements in a microprocessor development system (the two boards, the floppy disk, and the CRT console) has saved the cost of separate power supplies and separate enclosures, thus reducing overall system cost. In addition, the unit takes up less space on the user's work bench.

Based on studies of the usual configurations of systems under development, six card slots are included in

A ride on the bus

The Multibus configuration is a clocked asynchronous bus allowing multiple independent and concurrently operating processor modules to communicate with each other and to share access to common system resources efficiently and economically. Modules connected with the Multibus scheme are distinguished as either masters or slaves, with the former being able to gain control of the bus and initiate data-transfer operations by requesting a bus cycle from the bus control logic, when one is required.

Bus requests are assigned priorities so that access is granted to the module with the highest priority when two or more simultaneously request access to the bus. The module granted the bus access will maintain control until its transfer cycle is complete.

For example, when the central-processor module obtains a bus access to fetch an instruction byte from memory, it maintains control of the bus through the transfer of address to the memory and receipt of the instructions back from the memory. Therefore, the bus-transfer cycle time and the bus throughput are a function of the type of transfer and the response time of the addressed module to the requested operation. In this case, the memory access time determines the length of the transfer cycle.

A significant gain in performance comes from the Multibus setup's ability to transfer control from one master to the next in parallel with the on-going data transfer operation. This feature eliminates costly overhead time. The bus is accessible to the various modules within the Series II system via a printed-circuit motherboard that has provisions for plugging in various standard as well as optional modules.

all models. The 210 and 220 use only one slot while the 230 uses four slots. The extra slots should satisfy the needs for most development systems; however, four extra card slots can be added to all models by attaching an expansion chassis. The I/O controller board does not take up a slot. It is mounted flush against the back panel to eliminate extra wiring in the chassis.

Matching software to needs

Microcomputer applications span the range from simple controller designs to large multiprocessor designs with hundreds of thousands of bytes of program code. Clearly, microprocessor development software must also span a range of capabilities to match these varied applications. In the case of the Intellec Series II, software is available on several levels consistent with the hardware capabilities of each model.

Two distinct, yet compatible, operating systems are available, as mentioned: a ROM-based system monitor and the ISIS-II diskette operating system. The system monitor provides basic supervisory functions for all Series II systems. These functions include the user interface for systems based on the 210, a rudimentary program debug and checkout facility for 8080 programs, and a generalized I/O system accessible to the designer's program. The 210 uses the monitor as its sole operating system. ISIS-II, which provides a set of file management services to augment the supervisory functions of the monitor, is on a floppy disk with the 220 and 230.

The ISIS-II command language is easy to use and does not force upon the design engineer the types of artificial abstractions present all too often in other command languages. Nonetheless, a simple-to-use command language does not imply a limited operating system. ISIS-II supports a wide selection of system configurations, from the simplest 220 to a 230 system with six floppy-disk drives for a total of 2.5 million bytes of on-line storage, plus a high-speed line printer and various kinds of input devices.

An important feature of the Series II is the availability of higher-level languages and macro-assemblers for coding program reliably. On the high end of the spectrum are both PL/M, invented by Intel in 1973, and a new resident Fortran compiler. The PL/M compiler, available for the 230, generates code for both the 8080 and the 8085 microprocessors.

Since many engineers have some familiarity with Fortran, a new compiler for the Intellec systems supports a language compatible with the 1977 ANSI Fortran specification. Therefore, it offers designers the ability to connect software from one hardware system to other computer systems. The introduction of Fortran 77 in a development system opens new avenues for creativity in product development. This compiler is designed to run on model 230 under the ISIS-II operating system.

The advantages of sophisticated programming languages become apparent when developing software by building up modules. It may be advantageous both for speed of the development process and for reliability and ease of maintenance of the final product to produce modular software. Yet, for many applications, it is not feasible or wise to use a single programming language when developing the software. A powerful feature of ISIS-II is the ability to link programs written in PL/M, Fortran, and assembly language easily without any direct user knowledge of the interface.

A sophisticated relocation and linkage package included with the ISIS-II operating system allows the software developer to write 8080 or 8085 programs in small modules, each module being in any Intel programming language, and then automatically link the modules together. The final module may then be located any place in system memory, taking into account the location of ROM and RAM. Also, a library management program in ISIS-II allows the user to store frequently used routines where they may be easily linked to any referencing program.

In short, the user can choose from a variety of software development tools to build the system that best suits specific needs, yet be assured that in the end software requirements will be met. Moreover, the elements are in place for continually upgrading of development system hardware and software to keep pace with advances in microprocessors. □

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES: TEORIA Y APLICACIONES

LISTADOS

M. EN C. ANGEL KURI MORALES

ABRIL, 1978.

Palacio de Minería Calle de Tacuba 5, primer piso. México 1, D. F.

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 1


 0008                 N EQU 8
 1400                 LAST EQU 1400H
 13FF                 F EQU LAST-1
 13FD                 J EQU LAST-3
 13FF                 VAL SET F
 13FD                 DVSR SET J

 0000     AF          BNFLT: XRA A
 0001     2604        MVI H,N/2
 0003     C5          PUSH B ; DIR(BCD)
 0004     02          BNFT1: STAX B ; INICIALIZA CON CEROS
 0005     03          INX B
 0006     25          DCR H
 0007     C20400      JNZ BNFT1
 000A     C1          POP B
 000B     3E05        MVI A,5 ; A<>I
 000D     211027      LXI H,10000
 0010     22FD13      SHLD DVSR ; VALOR INICIAL DEL DIVISOR.
 0013     210000      LXI H,0 ; H=FF, L=J.
 0016     F5          BNFT2: PUSH PSW ; SP1=I        --- 1)
 0017     E5          PUSH H ; SP2=FF&J              --- 2)
 0018     D5          PUSH D ; SP3=DIR(BIN)          --- 3)
 0019     2AFD13      LHLD DVSR
 001C     EB          XCHG
 001D     C5          PUSH B ; HL=DIR(BIN)
                             ; SP4=DIR(BCD)          --- 4)
 001E     4E          MOV C,M
 001F     23          INX H
 0020     46          MOV B,M ; BC=BIN
                              ; DE=DVSR
 0021     CD8700      CALL FLDIV
 0024     79          MOV A,C
 0025     32FF13      STA VAL ; VAL=BC/DE
 0028     E1          POP H ; --- 3)
 0029     E3          XTHL ; HL=DIR(BIN)
 002A     73          MOV M,E
 002B     23          INX H
 002C     72          MOV M,D
 002D     2B          DCX H
 002E     E5          PUSH H ; SP4=DIR(BIN)
 002F     2AFD13      LHLD DVSR
 0032     44          MOV B,H
 0033     4D          MOV C,L
 0034     110A00      LXI D,10
 0037     CD8700      CALL FLDIV
 003A     60          MOV H,B
 003B     69          MOV L,C
 003C     22FD13      SHLD DVSR
 003F     3AFF13      LDA VAL
 0042     E1          POP H ; HL=DIR(BIN)
 0043     E3          XTHL ; HL=DIR(BCD)
```

```
8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 2


 0044    86         ADD M
 0045    C25200     JNZ BNFT4
 0048    D1         BNFT3: POP D ; DE=DIR(BIN)
 0049    44         MOV B,H      ; BC=DIR(BCD)
 004A    4D         MOV C,L
 004B    E1         BNFT1: POP H ; H=FF. L=J.
 004C    F1         POP PSW ; A-T.
 004D    3D         DCR A
 004E    C8         RZ
 004F    C31600     JMP BNFT2
                    ;
 0052               BNFT4:  ; DE=DIR (BIN)
 0052    D1         POP D ; HL=DIR (BCD)
 0053    34         INR M
 0054    34         INR M ; BCD=BCD+2
 0055    E3         XTHL  ; H=FF. L-J.        --- SP2=DIR(BCD)
 0056    7C         MOV A,H
 0057    2F         CMA
 0058    B7         ORA A
 0059    67         MOV H,A
 005A    F27100     JP BNFT5
 005D    2C         INR L
 005E    7D         MOV A,L
 005F    E3         XTHL
 0060    0600       MVI B,0
 0062    4F         MOV C,A
 0063    09         DAD B
 0064    3A1F13     LDA VAL
 0067    E60F       ANI 0FH
 0069    07         RLC
 006A    07         RLC
 006B    07         RLC
 006C    07         RLC
 006D    77         MOV M,A ; BCD(J)=SHI (VAL,4)
 006E    C37B00     JMP BNFT6
 0071    0600       BNFT5: MVI B,0
 0073    4D         MOV C,L
 0074    E3         XTHL
 0075    09         DAD B
 0076    3A1F13     LDA VAL
 0079    B6         ORA M
 007A    77         MOV M,A ; BCD(J)=BCD(J) OR VAL
 007B    79         BNFT6: MOV A,C
 007C    2F         CMA
 007D    3C         INR A
 007E    4F         MOV C,A
 007F    06FF       MVI B,0FFH ; (BC)
 0081    09         DAD B
 0082    44         MOV B,H
 0083    4D         MOV C,L ; BC-DIR(BCD)
 0084    C34B00     JMP BNFT1
```

```
                              ;
                              ;**** SUBRUTINA DE DIVISION BINARIA
                              ;
                              ; ESTA SUBRUTINA CAEPTA DOS NUMEROS BINARIOS DE 16
                              ; BITS CADA UNO( EN LOS REGISTROS BC Y DE).
                              ; A LA SALIDA, EN EL REGISTRO BC ENTREGA
                              ;         BC DIV DE
                              ; Y EN EL REGISTRO DE, ENTREGA
                              ;         BC MOD DE
                              ;
  0087    7A                  PLDIV: MOV A,D
  0088    2F                  CMA
  0089    57                  MOV D,A
  008A    7B                  MOV A,E
  008B    2F                  CMA
  008C    5F                  MOV E,A
  008D    13                  INX D
  008E    210000              LXI H,0
  0091    3E11                MVI A,11H
  0093    E5                  PLDV2: PUSH H
  0094    19                  DAD D
  0095    D29900              JNC PLDV1
  0098    E3                  XTHL
  0099    E1                  PLDV1: POP H
  009A    F5                  PUSH PSW
  009B    79                  MOV A,C
  009C    17                  RAL
  009D    4F                  MOV C,A
  009E    78                  MOV A,B
  009F    17                  RAL
  00A0    47                  MOV B,A
  00A1    7D                  MOV A,L
  00A2    17                  RAL
  00A3    6F                  MOV L,A
  00A4    7C                  MOV A,H
  00A5    17                  RAL
  00A6    67                  MOV H,A
  00A7    F1                  POP PSW
  00A8    3D                  DCR A
  00A9    C29300              JNZ PLDV2
  00AC    B7                  ORA A
  00AD    7C                  MOV A,H
  00AE    1F                  RAR
  00AF    57                  MOV D,A
  00B0    7D                  MOV A,L
  00B1    1F                  RAR
  00B2    5F                  MOV E,A
  00B3    C9                  RET
                              ;
                              END
NO PROGRAM ERRORS
```

```
'8080 MACRO ASSEMBLER, VER 2.0  ERRORS = 0 PAGE 4


                          SYMBOL TABLE

 * 01

 A       0007      B       0000      BNFLT   0000 *    BNFT1   0004
 BNFT2   001A      BNFT3   004B *    BNFT4   0052      BNFT5   0071
 BNFT6   007B      BNFTI   004B      C       0001      D       0002
 DVSR    13FB      E       0003      F       13FF      H       0004
 J       13FD      L       0005      LAST    1400      M       0006
 N       0008      PLDIV   0087      PLDV1   0099      PLDV2   0093
 PSW     0006      SP      0006      VAL     13FF

$ET=4:51.2 PT=26.2 10=1.2
```

```
$L=6

00001   1

00002   1   DECLARE  LIT LITERALLY 'LITERALLY',

00003   1       DCL LIT 'DECLARE';

00004   1

00005   1   BIB: PROCEDURE(BCD$PTR,BIN);

00006   2       DCL (I,J,FF,VAL) BYTE,

00007   2       (BIN,DVSR) ADDRESS,

00008   2       BCD$PTR ADDRESS,

00009   2       BCD BASED BCD$PTR BYTE;

00010   2   BCD(1),BCD(2),BCD(3),BCD(4)=0;

00011   2   BCD,J,FF=0;  DVSR=10000;

00012   2   DO I=1 TO 5;

00013   2       VAL=BIN/DVSR; BIN=BIN MOD DVSR; DVSR=DVSR/10;

00014   3       IF VAL=0 AND BCD=0 THEN GO TO I$BIB;

00015   3       BCD=BCD+2;

00016   3       FF=NOT(FF);

00017   3       IF FF THEN DO;

00018   3           J=J+1;

00019   4           BCD(J)=SHL(VAL,4);

00020   4       END;

00021   3       ELSE BCD(J)=BCD(J) OR VAL;

00022   3   I$BIB: END;

00023   2   END BIB;

00024   1

00025   1   EOF

NO PROGRAM ERRORS

#ET=1:37.3 PT=13.5 IO=1.6
E PL/MII/4;FILE FILE23(MAXRECSIZE=22,BLOCKSIZE=22);%
#%
FILE FILE23(MAXRECSIZE=22,BLOCKSIZE=22)
#RUNNING 7387
#?

8080 PLM2 VERS 4.0
```

```
$I=7 $F=1

STACK SIZE = 0 BYTES
MEMORY...........................0200H
BIR..............................0006H
BCDPIR...........................01F6H
BIN..............................01F8H
I................................01FAH
J................................01FBH
FF...............................01FCH
VAL..............................01FDH
DVSR.............................01FEH
IBIR.............................0119H
0000H LXI SP F6H    01H    JMP    21H    01H    LXI H  F6H    01H
0009H MOV MC INX H  MOV MB INR L  MOV ME INX H  MOV MD LHLD   F6H
0012H 01H    INX H  PUSH H LXI B  02H    00H    LHLD   F6H    01H
001BH DAD B  PUSH H LXI B  03H    00H    LHLD   F6H    01H    DAD B
0024H PUSH H LXI B  04H    00H    LHLD   F6H    01H    DAD B  MOV MI
002DH 00H    POP H  MOV MI 00H    POP H  MOV MI 00H    POP H  MOV MI
0036H 00H    LHLD   F6H    01H    MOV MI 00H    LXI H  FBH    01H
003FH MOV MI 00H    INR L  MOV MI 00H    MOV LI FEH    MOV MI 10H
0048H INX H  MOV MI 27H    MOV LI FAH    MOV MI 01H    MOV AI 05H
0051H LXI H  FAH    01H    SUB M  JC     20H    01H    LXI H  FEH
005AH 01H    MOV EM INR L  MOV DM MOV LI F8H    MOV CM INR L  MOV BM
0063H JMP    93H    00H    MOV AD CMA    MOV DA MOV AE CMA    MOV EA
006CH INX D  LXI H  00H    00H    MOV AI 11H    PUSH H DAD D  JNC
0075H 78H    00H    XTHL   POP H  PUSH A MOV AC RAL    MOV CA MOV AB
007EH RAL    MOV BA MOV AL RAL    MOV LA MOV AH RAL    MOV HA POP A
0087H DCR A  JNZ    72H    00H    ORA A  MOV AH RAR    MOV DA MOV AL
0090H RAR    MOV EA RET    CALL   66H    00H    LXI H  FDH    01H
0099H MOV MC LXI H  FEH    01H    MOV EM INR L  MOV DM MOV LI F8H
00A2H MOV CM INR L  MOV BM CALL   66H    00H    LXI H  F8H    01H
00ABH MOV ME INX H  MOV MD MOV EI 0AH    MOV DI 00H    LXI H  FEH
00B4H 01H    MOV CM INR L  MOV BM CALL   66H    00H    LXI H  FEH
00BDH 01H    MOV MC INX H  MOV MD MOV LI FDH    MOV AM SUB I  00H
00C6H SUB I  01H    SBC A  LHLD   F6H    01H    MOV CA MOV AM SUB I
00CFH 00H    SUB I  01H    SBC A  ANA C  RRC    JC     19H    01H
00D8H LHLD   F6H    01H    MOV AM ADD I  02H    MOV MA LXI H  FCH
00E1H 01H    MOV AM CMA    MOV MA RRC    JNC    FFH    00H    DCR L
00EAH INR M  MOV CM MOV BI 00H    LHLD   F6H    01H    DAD B  XCHG
00F3H LXI H  FDH    01H    MOV AM ADD A  ADD A  ADD A  ADD A  STAX D
00FCH JMP    19H    01H    DCR L  MOV CM MOV BI 00H    LHLD   F6H
0105H 01H    DAD D  PUSH H LXI H  FBH    01H    MOV CM MOV BI 00H
010EH LHLD   F6H    01H    DAD B  MOV AM LXI H  FDH    01H    ORA M
0117H POP H  MOV MA LXI H  FAH    01H    INR M  JNZ    4FH    00H
0120H RET    EI     HLT
NO PROGRAM ERRORS

$ET=2:13.6 PT=10.9 IO=1.7
```

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

MICROPROCESADORES : TEORIA Y APLICACIONES

PROGRAMAS PARA GRAFICAR HISTOGRAMAS Y CLASIFICAR UN VECTOR

ABRIL, 1978

Palacio de Minería Calle de Tacuba 5, primer piso. México 1, D. F.

```
1 REM ************************************************^P
2 REM *
3 REM *       PROGRAMA PARA CLASIFICAR UN VECTOR
4 REM *
5 REM ************************************************
7 REM *
8 PRINT "PROGRAMA PARA CLASIFICAR UN VECTOR":PRINT
9 REM -   DIMENSIONAMIENTO
10 INPUT"CUANTOS NUMEROS SON ?" ;C
20 DIM A(C)
30 FOR I=1 TO C
40 A(I)=RND
50 NEXT I
54 REM
55 REM   CLASIFICACION
60 PRINT"SE INICIA EL SORT"
70 N=C
75 M=N
80 M=INT(M/2)
90 IF M=0 THEN 200
100 K=N-M
110 FOR J=1 TO K
111 I=J
130 L=I+M
140 IF A(I) <= A(L) THEN 180
150 T=A(I)
151 A(I)=A(L)
152 A(L)=T
160 I=I-M
170 IF I>=1 THEN 130
180 NEXT J
190 GOTO 80
200 PRINT"SE TERMINO EL SORT"
215 REM
217 REM   IMPRESION DEL VECTOR CLASIFICADO
220 FOR I=1 TO C
230 PRINT A(I)
240 NEXT I
250 END
```

```
2 REM ***********************************************P
3 REM *
4 REM *  PROGRAMA PARA GRAFICAR HISTOGRAMAS
5 REM *
6 REM ********************************************'P*
7 REM *
8 PRINT "PROGRAMA PARA GRAFICAR HISTOGRAMAS"
10 REM PROGRAMA QUE GRAFICA HISTOGRAMAS
15 PRINT"IMPRIMA LA FECHA DIA,MES,AÑO"
16 INPUT F1,F2,F3
20 PRINT "IMPRIMA EL TITULO DE LA GRAFICA"
25 INPUT P$
26 PRINT"IMPRIMA EL SUB-TITULO"
27 INPUT Z$
30 PRINT "IMPRIMA EL ANCHO DE LAS BARRAS(1,2,3)"
50 INPUT N1
60 PRINT "IMPRIMA EL MAXIMO LARGO DE LAS BARRAS"
65 PRINT "ENTRE (10 A 60)"
67 INPUT W1
70 PRINT "IMPRIMA EL SIMBOLO A USAR EN LA GRAFICA"
75 INPUT S$
90 PRINT"IMPRIMA EL NUMERO TOTAL DE BARRAS"
100 INPUT L1
105 DIM X(L1),Y(L1),J$(L1)
110 PRINT "IMPRIMA LAS UNIDADES USADAS"
120 INPUT Y$
130 PRINT "IMPRIMA DE UNO EN UNO LOS VALORES"
140 FOR I=1 TO L1
150 PRINT "DATO ";I;" VALOR ";
160 INPUT Y(I):PRINT"IMPRIMA EL NOMBRE DE LA COLUMNA (MENOS DE 5 LETRAS)";
165 INPUT J$(I)
170 NEXT I
180 M1=0
190 REM SE CALCULA EL MAXIMO
200 FOR I=1 TO L1
210 IF M1>Y(I) THEN 230
220 M1=Y(I)
230 NEXT I
240 REM SE ESCALAN LOS VALORES
250 FOR I=1 TO L1
260 X(I)=INT((W1*Y(I)/M1+0.5))
270 NEXT I
275 PRINT
280 PRINT:PRINT:PRINT:PRINT CHR$(14);TAB(15);P$:PRINT:PRINT
290 PRINT:PRINT :PRINT CHR$(14);TAB(5);Z$:PRINT:PRINT
300 PRINT TAB(50);"FECHA ";F1;"-";F2;"-";F3
330 PRINT
340 PRINT "CADA ";S$;" REPRESENTA ";M1/W1;" ";Y$
350 PRINT
360 PRINT TAB(7);"UNIDADES"
370 PRINT TAB(5);Y$;TAB(17);"MAXIMA LONGITUD DE LAS BARRAS ";W1
380 PRINT
382 FOR E=1 TO 80:PRINT"-";:NEXT E:PRINT
385 PRINT TAB(15);"I";TAB(W1+17);"I"
390 FOR I=1 TO L1
400 FOR J=1 TO N1
410 IF J=1 THEN 440
420 PRINT TAB(15);"I";
```

```
430 GOTO 4[illegible]
440 REM
445 PRINT J$(I);TAB(9);Y(I);TAB(15);"I";
450 L2=X(I)
460 FOR K=1 TO L2
470 PRINT S$;
480 NEXT K
490 PRINT TAB(W1+17);"I"
500 NEXT J
510 PRINT TAB(15);"I";TAB(W1+17);"I"
520 NEXT I
521 FOR E=1 TO 80:PRINT"-";:NEXT E:PRINT
522 PRINT CHR$(13):PRINT CHR$(12);
524 PRINT
530 PRINT"DESEA REPETIR ESTA GRAFICA"
540 INPUT I$
550 IF I$="SI" THEN 275
560 PRINT"DESEA OTRA GRAFICA CON NUEVOS VALORES"
570 INPUT I$
580 IF I$="SI" THEN 10
590 PRINT:PRINT:PRINT"HASTA LUEGO"
```

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES DEL 3 AL 14 DE ABRIL DE 1978

	NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
1.	ING. RAMON ACEVEDO S.	ELECTRONICA, S.A.
	Varsovia No. 36-3o. Piso	Varsovia No. 36-3o. Piso
	Col. Juárez	Col. Juárez
	México 6, D.F.	México 6, D.F.
	Tel. 533-11-80	Tel. 533-11-80
2.	FRANCISCO ALMADA VALENZUELA	S.A.R.H.
	P. E. Calles 1362 - 402	Reforma No. 35-11o. Piso
	Col. Prado	Col. Centro
	México 13, D.F.	México 1, D.F.
	Tel. 539-05-65	Tel. 591-03-83
3.	ARTURO ARIAS AGUIRRE	INST. MEXICANO DEL PETROLEO
	Av. de los Reyes No. 126	Av. de los 100 Mts. No. 152
	Lote 3 Mza. 79-B	Col. San Bartolo
	Col. Fracc. J. de los Reyes	México, D.F.
	Ixtacala Tlanepantla, Edo. Méx.	Tel. 567-66-00 Ext. 2177
4.	SERGIO G. BAÑUET MORALES	CENTRO DE SERVICIOS DE COMPUTO
	Batalla de Celaya Edif. 1	Ciudad Universitaria
	Col. Residencial Militar	
	México, D.F.	
	TEL.	
5.	ING. EFREN BAÑUELOS DE SANTIAGO	UNIVERSIDAD AUTONOMA DE
	Almendro No. 112 F	Av. Ramón López Velarde S/N
	Col. Las Arboledas	Zacatecas, Zac.
	Zacatecas, Zac.	
6.	ING. FERNANDO BUSSEY S.	INDUSTRIA Y POTENCIA, S.A.
	Oriente156 No. 40	Río Nazas No. 136-3o. Piso
	Col. Moctezuma	Col. Cuauhtémoc
	México 9, D.F.	México 5, D.F.
	Tel. 522-33-99	Tel. 533-54-25
7.	ING. ERWIN CAMACHO LOPEZ V.	INST. MEXICANO DEL PETROLEO
	Hda. San Nicolás el Grande No. 51	Av. de los 100 Mts. No. 152
	Col. Hda. del Rosario	Col. Industrial Vallejo
	México 16, D.F.	México 14, D.F.
	Tel. 561-41-17	Tel. 567-54-76

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES
DEL 3 AL 14 DE ABRIL DE 1978

	NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
8.	ING. JOSE LUIS CAMPOS MEDINA Callejón 4 de julio No. 105 Zacatecas, Zac. Tel.	UNIVERSIDAD AUTONOMA DE ZACATECAS Av. Ramón López Velarde S/N Zacatecas, Zac.
9.	MAURO CARDENAS HERNANDEZ Vallarta No. 25 Col. Coyoacán México 21, D.F. Tel. 554-85-62	EPSIZON, S.A. Sevilla No. 415 Col. Portales México 13, D.F. Tel.
10.	ING. CARLOS G. CARRILLO ROJAS Calle Fresnos No. 117 Fracc. Los Arboledas Zacatecas, Zac. Tel.	UNIVERSIDAD AUTONOMA DE ZACATECAS Av. Ramón López Velarde S/N Zacatecas, Zac. Tel. 2-08-27
11.	GERARDO EMILIO FERNANDEZ PIÑA Sur 125 "A" Núm 40 Col. Los Cipreses México 13, D.F. Tel. 582-06-95	INST. MEXICANO DEL PETROLEO Av. de los 100 Mts. No. 152 Col. Sn. Bartolo México 14, D.F. Tel. 567-66-00 Ext. 2345
12.	CLAUDIO GONZALEZ HERMOSILLO Retorno 204 No. 2 Col. U. Modelo México 13, D.F. Tel. 582-95-39	HONEYWELL, S.A. Av. Constituyentes No. 900 Col. Lomas Altas México 10, D.F. Tel. 570-20-33
13.	FRACISCO JAVIER GONZALEZ GONZALEZ Nuevo León 223-1 Col. Condesa México 11, D.F. Te . 515-87-18	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria Tel. 550-52-15 Ext. 4536
14.	ING. JORGE GONZALEZ Y GONZALEZ Sullivan No. 117-4 Col. San Rafael México 4, D.F. Tel. 546-46-43	COMISION FEDERAL DE ELECTRICIDAD Rodano No. 14-9o. Piso Col. Cuauhtémoc México 5, D.F. Tel. 553-65-86

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES DEL 3 AL 14 DE ABRIL DE 1978

	NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
15.	VICENTE GUERRERO ROJO Lago Gascasonica No. 255 Col. Hichapan México 17, D.F. Tel. 399-34-70	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria
16.	ING. IGNACIO DE JESUS GUTIERREZ HIDALGO Paseo de la Soledad No. 11 Col. Lomas Verdes Edo. de México Tel. 562-46-37	INDUSTRIA Y POTENCIA, S.A. Río Nazas No. 136-3er. Piso Col. Cuauhtémoc México 5, D.F. Tel. 533-54-25
17.	ING. MIGUEL ANGEL LOMELI ARAGON José A. Torres No. 681-4 Col. Asturias México 8, D.F. Tel. 519-66-35	LABORATORIO EN ELECTRICIDAD Y ELECTRONICA INDUSTRIAL, S.A. Pto. La Paz No. 111 Esq. Mazatlá Col. Casas Alemán Tel. 781-77-92
18.	ING. FAUSTO MANCEBO DEL CASTILLO P. José Vaconselos No. 2 Circuito Pensadores Satélite, Edo. de Méx. Tel. 576-90-55	SEDAS REAL, S.A. San Esteban No. 46 Sn. Esteban Naucalpan Edo. de Méx. Tel. 576-20-55
19.	ING. ROBERTO MACIAS PEREZ Calle 71 No. 19-4 Col. Puebla México 9, D.F. Tel. 558-53-40	S. T. C. Calz. Ignacio Zaragoza No. 239 Col. J. Balbuena México 9, D.F. Tel. 521-86-20 Ext. 568
20.	ING. JAIME RENE MARTINEZ MARTINEZ Dr. Coss No. 542 Norte Monterrey, N.L. Tel. 75-06-75	INTELEX MTY, S.A. Padre Mier y Serafin Peña Monterrey, N.L. Tel. 44-66-16
21.	ING. JORGE S. MARTINEZ SARACHO Deportes No. 76 Col. Las Arboledas Atizapán, Edo. de Méx. Tel. 379-23-65	S. C. T. Lerdo de Tejada No. 6 Col. Sn. Juan Ixhuatepec Edo. de México Tel. 569-37-69

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES
DEL 3 AL 14 DE ABRIL DE 1978

	NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
22.	HECTOR MEDELLIN VARGAS Bráhms No. 109-2 Col. Vallejo México 15, D.F. Tel. 537-41-79	INST. MEXICANO DEL PETROLEO Av. de los 100 Mts. No. 152 Col. Sn. Bartolo T. México 14, D.F. Tel. 567-66-00
23.	ING. EUSEBIO MEJIA MALDONADO Siracusa No. 81 Col. Lomas Estrella México 13, D.F. Tel. 581-86-83	RADIO UNIVERSIDAD Adolfo Prieto No. 133 Col. Del Valle México 12, D.F. Tel. 543-96-17
24.	FCO. DAVID MEJIA RODRIGUEZ Vicente Suárez No. 132-201 Col. Condesa México 11, D.F. Tel. 553-73-84	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel. 550-52-15 Ext. 4542
25.	ING. JOSE LUIS NAREDO V. Victor Hugo No. 56-B Col. Anzures México 5, D.F. Tel. 528-65-00	INSTITUTO DE INVESTIGACIONES ELECTRICAS Shakespeare No. 6 Col. Anzures Tel. 525-64-62
26.	ING. SERGIO OLIVERA O.	CENTRO DE SERVICIOS DE COMPUTO
27.	ING. HOMERO HUGO ORTEGON Cond. Constitución Edif. 44-1 Monterrey, N.L. Tel. 44-02-14	UNION CARIBE MEXICANA, S.A. Carretera Miguel Alemán Km. 16 Apodaca, N.L. Tel. 54-45-40
28.	JUAN ORTIZ ANGUIANO Manchuria No. 11 Col. Romero Rubio México 9, D.F. Tel. 760-34-54	SISTEMAS ELECTRONICOS DE CONTROL INDUSTRIAL Manchuria No. 11 Col. Romero Rubio México 9, D.F. Tel. 760-34-54

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES DEL 3 AL 14 DE ABRIL DE 1978

	NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
29.	GUILLERMO OVIEDO VARGAS Lago Chalco No. 60-10 Col. Anáhuac México 17, D.F. Tel. 531-73-92	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel. 550-52-15 Ext. 4536
30.	JUAN ANTONIO PEREZ CHOWELL Pasadena No. 3 Col. Lomas Capistrano Atizapán de Zaragoza Edo. de México Tel. 398-26-20	SPERRY RAND - DIVISION UNIVAC Presidente Masarik No. 29-1o.P. Col. Anzúres México 5, D.F. Tel. 250-10-66
31.	ING. JOSE LUIS PEREZ GARCIA Guanabana No. 262 Col. Nva. Sta. María México 16, D.F. Tel. 556-15-69	HONEYWELL, S.A. Constituyentes No. 900 México 10, D.F. Tel.
32.	ING. JULIO CESAR QUIÑONES M. Azuela No. 51 Col. Sta. Ma. La Rivera México 4, D.F. Tel. 547-87-38	INDUSTRIA Y POTENCIA, S.A. Río Nazas No. 136-3er. nivel Col Cuauhtémoc México 5, D.F. Tel. 533-54-25
33.	ING. EDUARDO RAMIREZ SANCHEZ Cruz Azul No. 161 Col. Industrial México 14, D.F. Tel. 759-05-50	FACULTAD DE INGENIERIA UNAM. Ciudad Universitaria México 20, D.F. Tel. 550-52-15 Ext. 3755
34.	ING. ANDRES N. REYNOSO GOMEZ Nonoalco No. 29-15-5-304 Col. Tlatelolco México 3, D.F. Tel. 597-52-20	E.S.I.A. I.P.N. Unidad Profesional Zacatenco Col. Lindavista México, D.F. Tel.
35.	VICTOR GERMAN SANCHEZ Tampico No. 11-6 Col. Roma México 7, D.F. Tel.	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel.

DIRECTORIO DE ASISTENTES AL CURSO: MICROPROCESADORES TEORIA Y APLICACIONES
DEL 3 AL 14 DE ABRIL DE 1978

NOMBRE Y DIRECCION	EMPRESA Y DIRECCION
36. FRANCISCO JAVIER SANTOYO Av. Revolución No. 23-602 Co. Tacubaya México 18, D.F. Tel. 516-39-14	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel. 550-52-15 Ext. 4544
37. SERGIO ALFREDO SANTIAGO RIVERA Hidalgo No. 216-201 Col. Ixtapalapa México 13, D.F. Tel.	POLICOLOR - MANTENIMIENTO ELECT. Marsella No. 89 Col. Juárez México 6, D.F. Tel. 533-23-85
38, ARMANDO TELLEZ VELASCO Castilla No. 279 Col. Alamos México 13, D.F. Tel. 590-86-73	CENTRO DE INVESTIAGACION Y DESARROLLO, TELEFONOS DE MEXICO Ernesto Pugibet // 12 Torre Ake 3o México 1, D.F. Tel. 585-34-44 Ext. 9884
39. ING. JOSE A. TOVAR MARTINEZ M. A. de Quevedo No. 218 Col. Coyoacán México 21, D.F. Tel. 554-17-29	INSTITUTO DE INVESTIGACIONES ELECTRICAS Shakespeare No. 6 Col. Anzúres México 5, D.F. Tel. 525-64-52
40. LUIS FERNANDO TURCOTT RIOS José Ma. Correa No. 199 Col. Asturias México 8, D.F. Tel. 519-79-65	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel.
41. HECTOR JULIAN VAZQUEZ RAMIREZ Retorno 13 No. 41 Col. El Centinela México 21, D.F. Tel. 544-61-55	INST. MEXICANO DEL PETROLEO Av. de los 100 Mts. No. 152 Col. San. Bartolo A. México 14, D.F. Tel. 567-66-00 Ext. 2157
42. RICARDO ENRIQUE VITE SAN PEDRO Durango No. 268 Col. Roma México 7, D.F. Tel. 514-52-73	CENTRO DE SERVICIOS DE COMPUTO Ciudad Universitaria México 20, D.F. Tel.