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.

---

*"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.

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
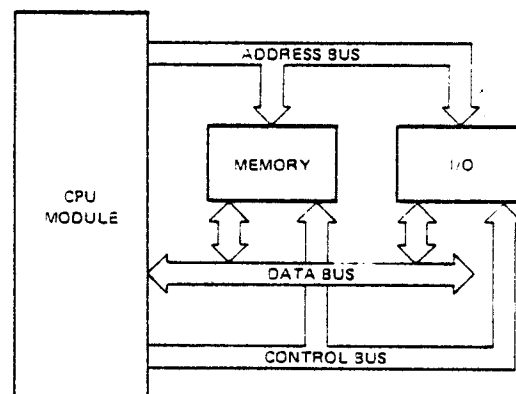   5. Interrupt Acknowledge

---

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.[TM] 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

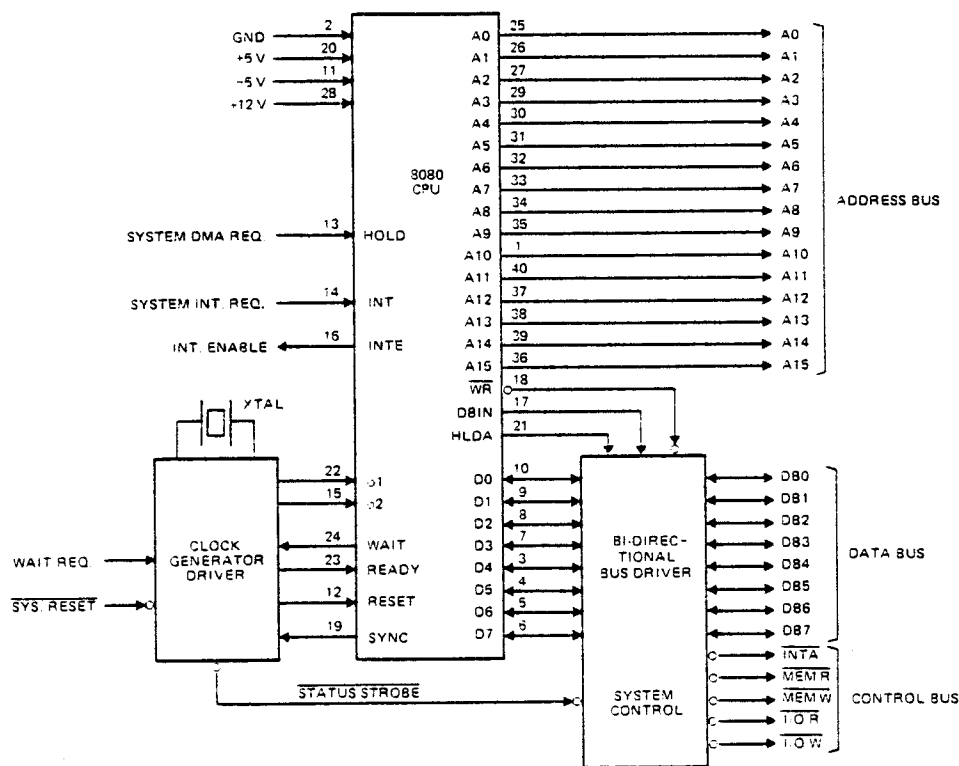The Clock Generator consists of a crystal controlled,
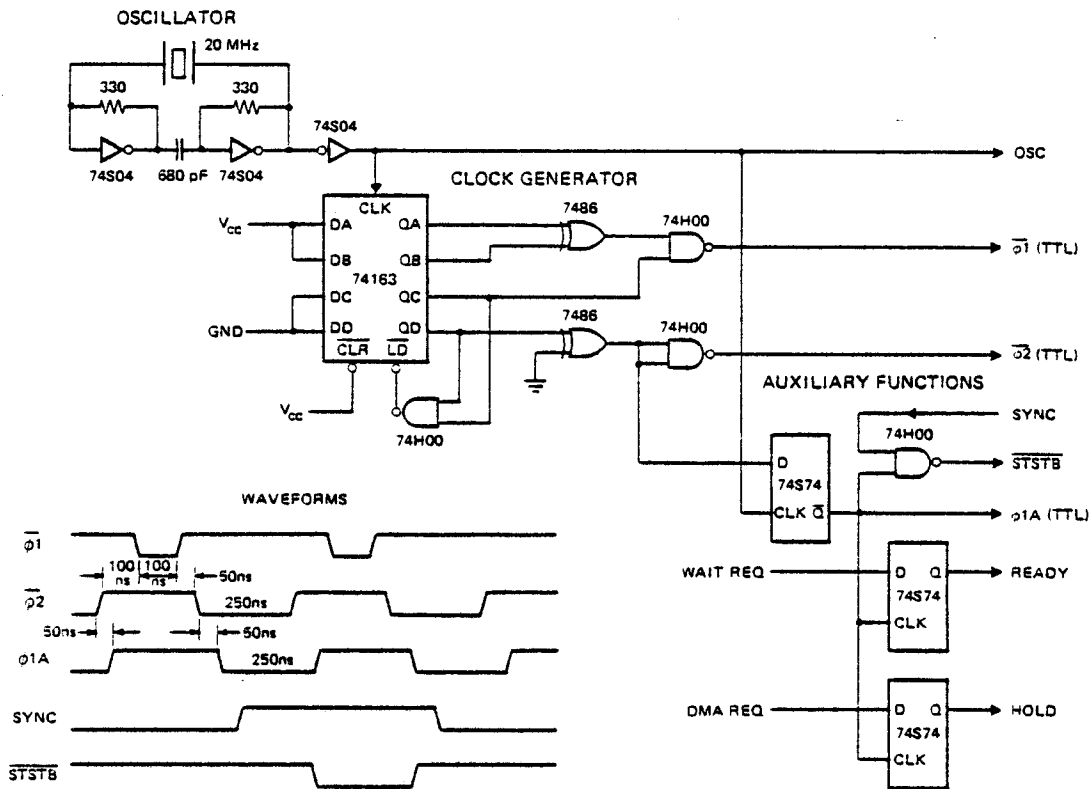

Figure 3-2. 8080 CPU Interface

OSCILLATOR

CLOCK GENERATOR

WAVEFORMS

AUXILIARY FUNCTIONS

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.
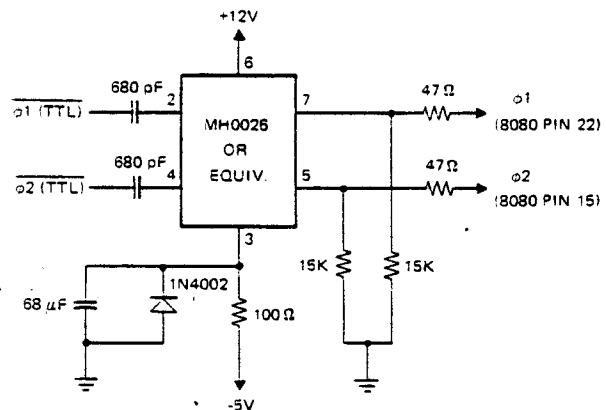

Figure 3-4. High Level Driver

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 ($\overline{\text{STSTB}}$) 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.
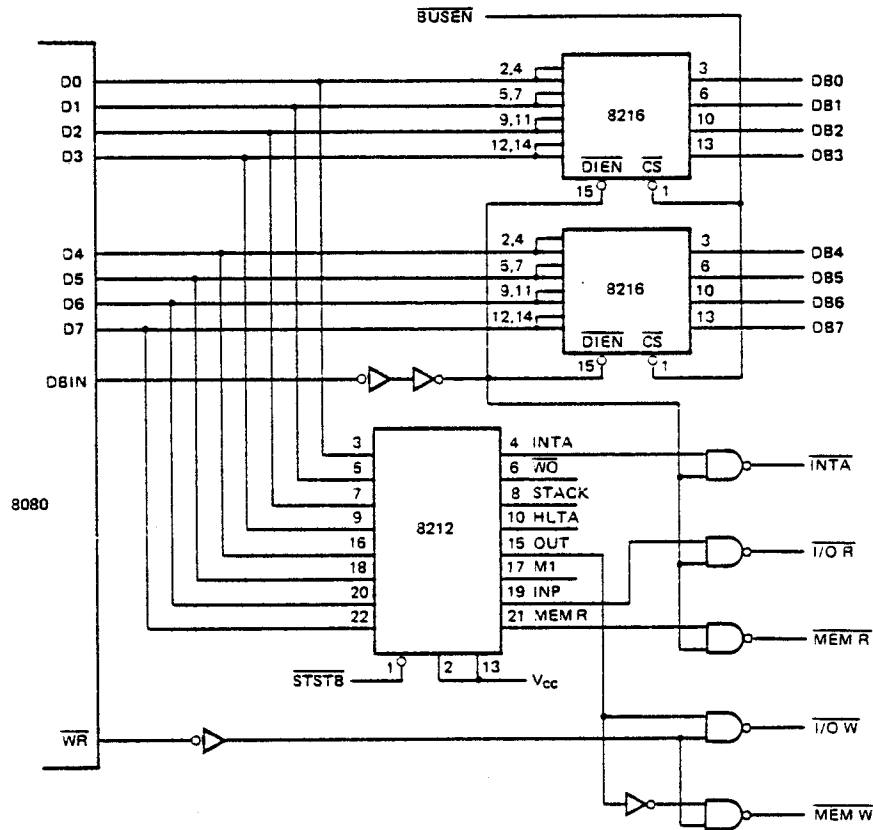2. Output Drive Capability (I_{OL}) 1.7 mA maximum.

Figure 3-5. 8080 System Control

The input level specification implies that any semi-conductor 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 ($\overline{DIEN}$) so the correct flow of data on the bus is maintained. The chip select ($\overline{CS}$) of the 8216 is connected to BUS ENABLE ($\overline{BUSEN}$) 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 ($\overline{STSTB}$) 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 $\overline{WR}$ signals from the 8080 generate the (4) four Control signals that make up the basic Control Bus.

These four signals: 1. Memory Read ($\overline{MEM\ R}$)

2. Memory Write ($\overline{MEM\ W}$)

3. I/O Read ($\overline{I/O\ R}$)

4. I/O Write ($\overline{I/O\ W}$)

connect directly to the MCS-80™ component "family" of ROMs, RAMs and I/O devices.

A fifth signal, Interrupt Acknowledge ($\overline{INTA}$) 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 $\overline{WO}$, 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 BUS ENABLE ($\overline{BUSEN}$) 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 $\overline{MEMR}$ 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 ($\overline{MEMR}$) 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
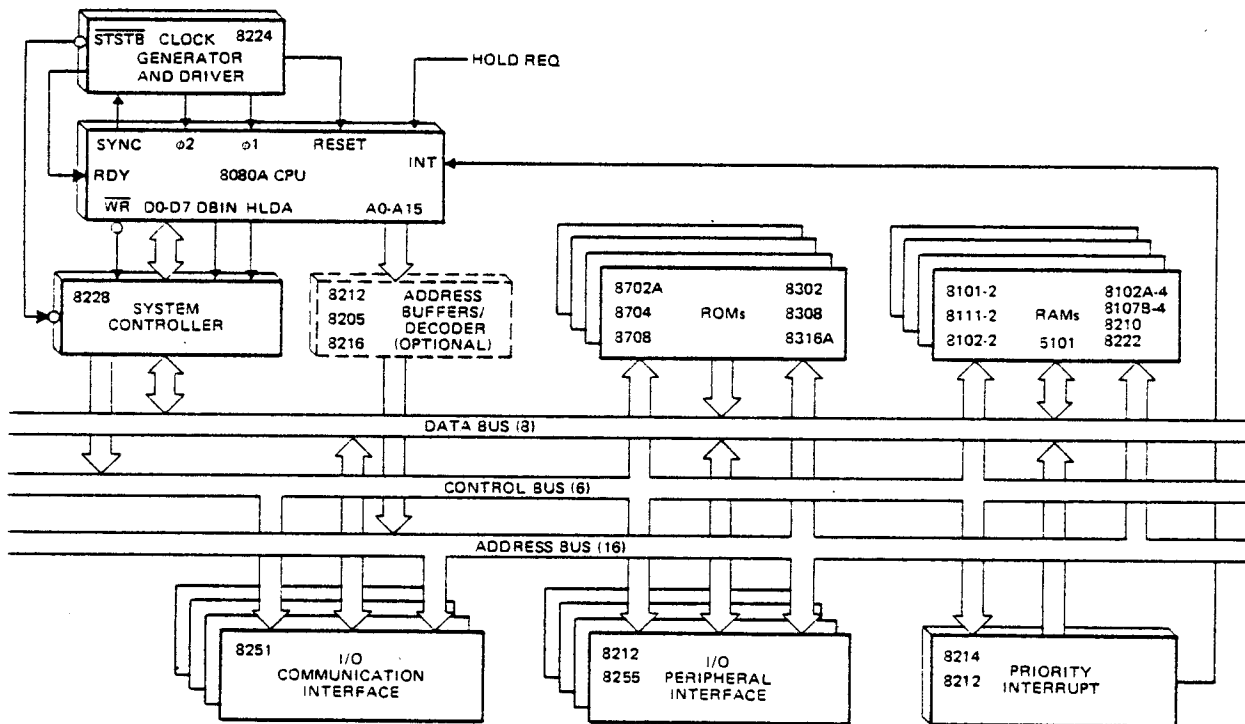

Figure 3-6. Microcomputer System

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 $\overline{MEMW}$ command is generated. The $\overline{MEMW}$ 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.
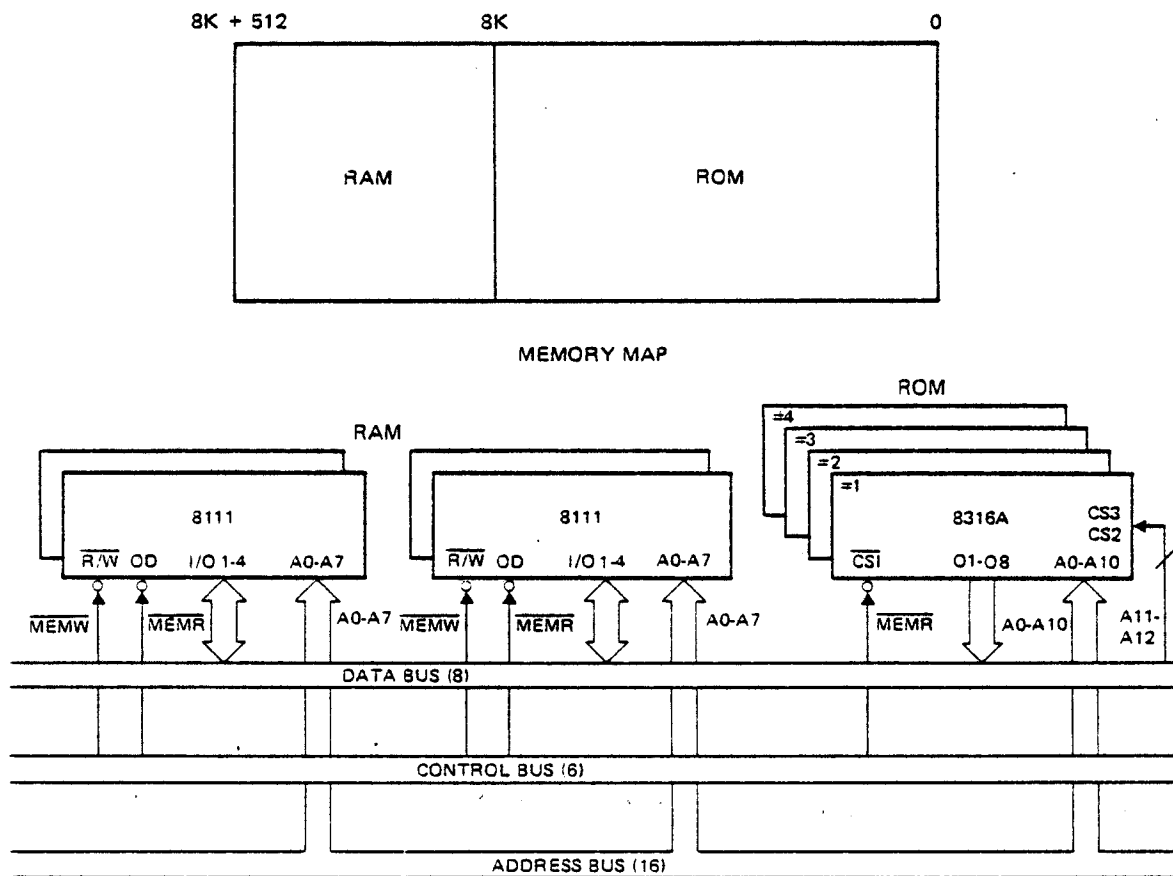

Figure 3-7. Typical Memory Interface

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.
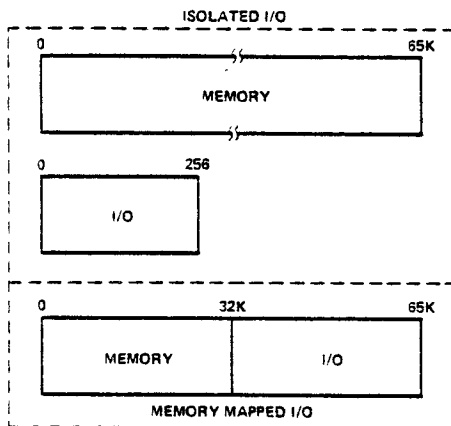

Figure 3-8. Memory/I/O Mapping.

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.
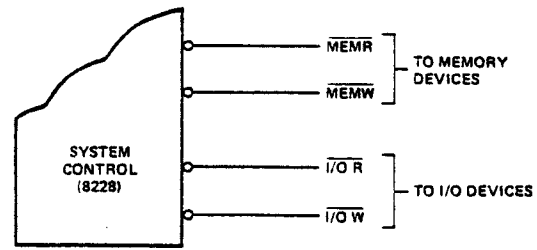

Figure 3-9. Isolated I/O.

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 $\overline{MEMR}$ and $\overline{MEMW}$ 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:

MOV r, 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.
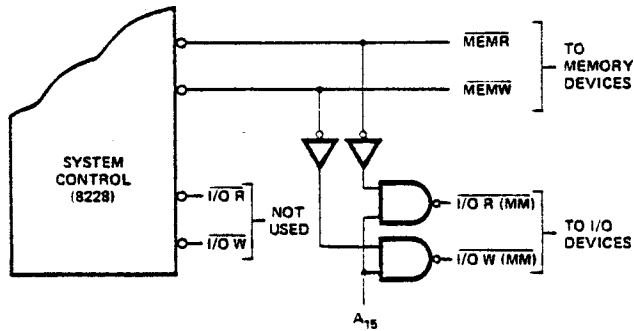
Figure 3-10. Memory Mapped I/O.

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
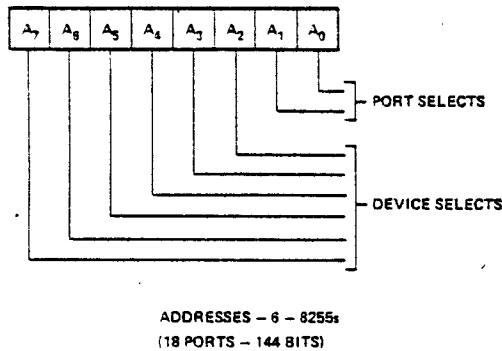

ADDRESSES – 6 – 8255s
(18 PORTS – 144 BITS)

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
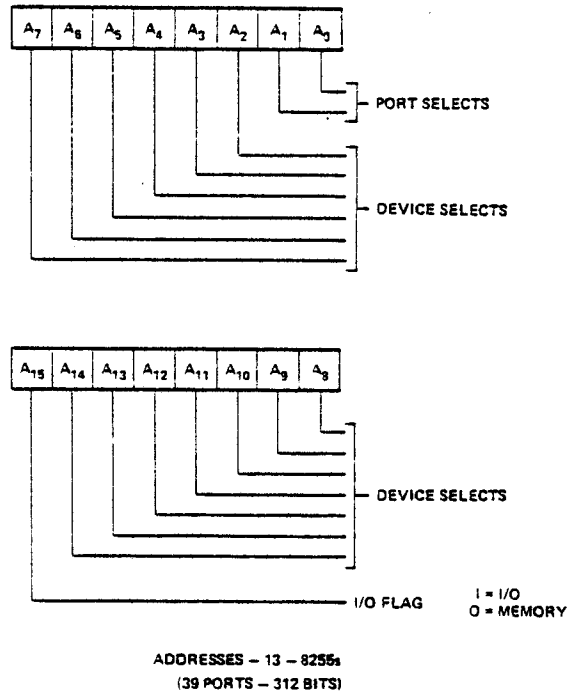(39 PORTS – 312 BITS)

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 (3212, 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.
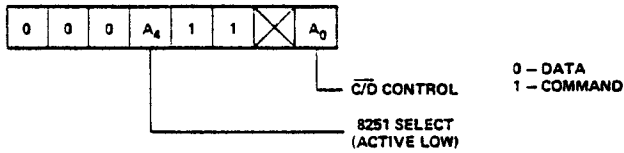
Figure 3-13. 8251 Format.

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.
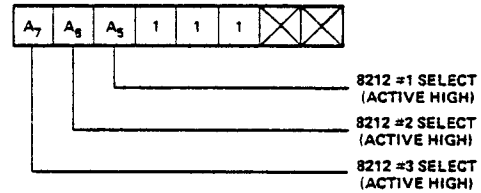

Figure 3-14. 8255 Format.

The three 8212s can be used to drive long lines or LED indicators due to their high drive capability. (15mA)

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.
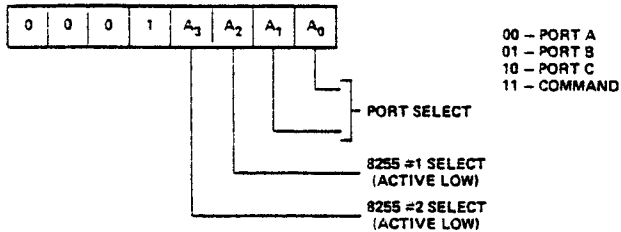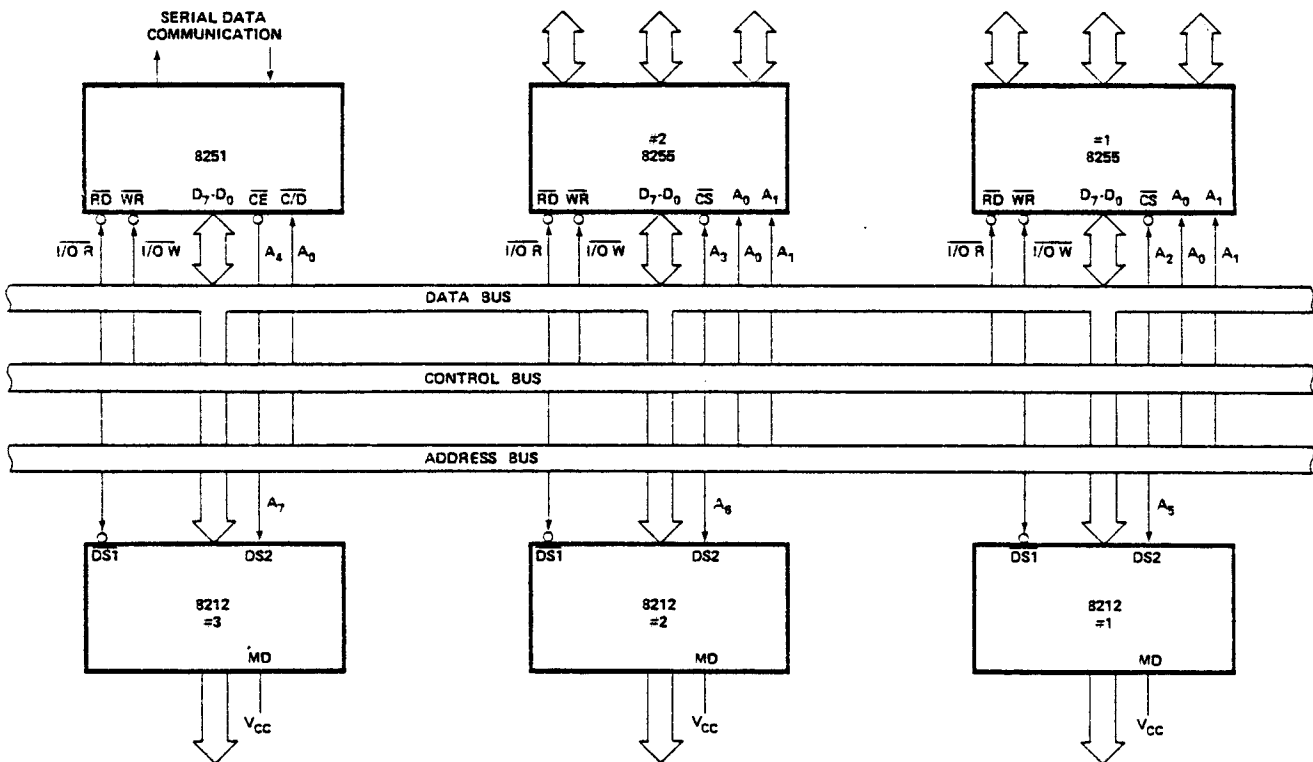

Figure 3-16. Typical I/O Interface.

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.
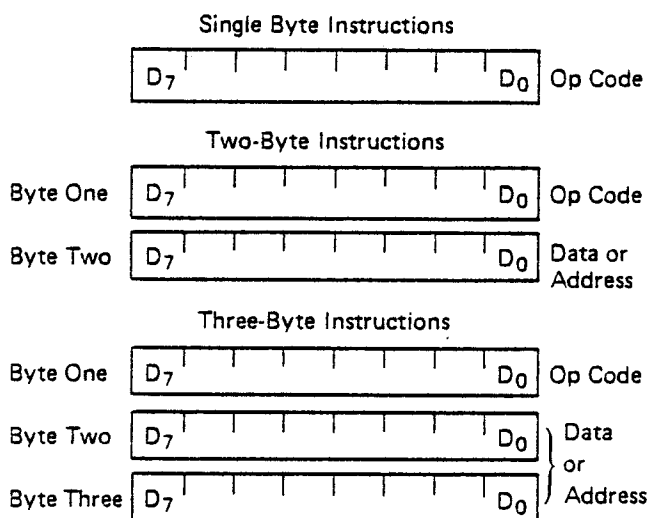
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:

DATA WORD

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

MSB   LSB
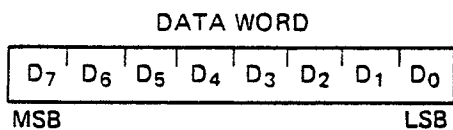
When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the Least Significant Bit (LSB), and BIT 7 (of an 8 bit number) is referred to as the Most Significant Bit (MSB).

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.

Single Byte Instructions

| D_7 | | | | | | D_0 | Op Code

Two-Byte Instructions

Byte One | D_7 | | | | | | | D_0 | Op Code

Byte Two | D_7 | | | | | | | D_0 | Data or Address

Three-Byte Instructions

Byte One | D_7 | | | | | | | D_0 | Op Code

Byte Two | D_7 | | | | | | | D_0 |) Data

Byte Three | D_7 | | | | | | | D_0 |) or Address

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.
←	"Is transferred to"
∧	Logical AND
⊻	Exclusive OR
∨	Inclusive OR
+	Addition
−	Two's complement subtraction
*	Multiplication
↔	"Is exchanged with"
‾	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.
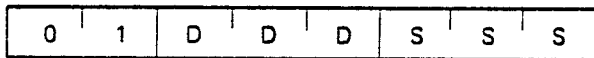
Data Transfer Group:

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.

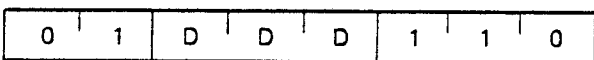| 0	1	D	D	D	S	S	S

　　　　　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.

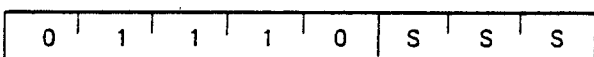| 0	1	D	D	D	1	1	0

　　　　　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.

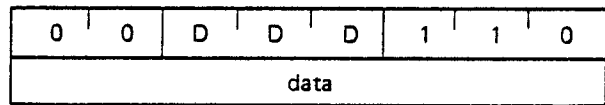| 0	1	1	1	0	S	S	S

　　　　　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.

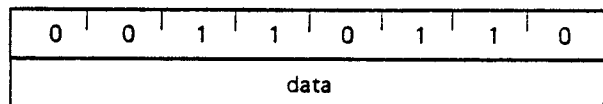| 0	0	D	D	D	1	1	0
data							

　　　　　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

2-54
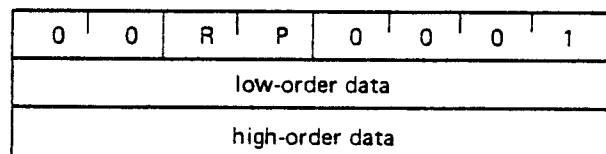
LDA addr   (Load Accumulator direct)

$(A) \leftarrow ((\text{byte 3})(\text{byte 2}))$

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							
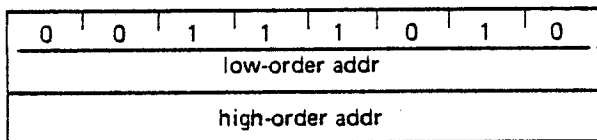
Cycles: 4
States: 13
Addressing: direct
Flags: none

STA addr   (Store Accumulator direct)

$((\text{byte 3})(\text{byte 2})) \leftarrow (A)$

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							
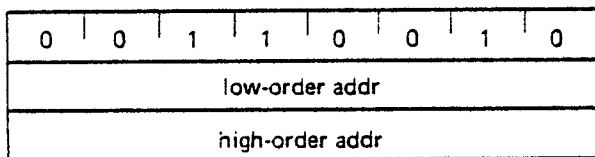
Cycles: 4
States: 13
Addressing: direct
Flags: none

LHLD addr   (Load H and L direct)

$(L) \leftarrow ((\text{byte 3})(\text{byte 2}))$
$(H) \leftarrow ((\text{byte 3})(\text{byte 2}) + 1)$

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							
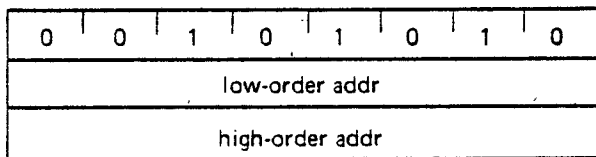
Cycles: 5
States: 16
Addressing: direct
Flags: none

SHLD addr   (Store H and L direct)

$((\text{byte 3})(\text{byte 2})) \leftarrow (L)$
$((\text{byte 3})(\text{byte 2}) + 1) \leftarrow (H)$

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							
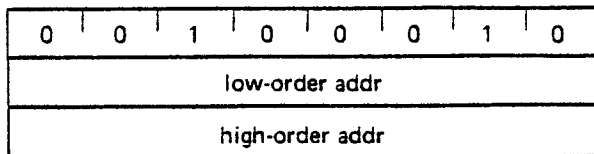
Cycles: 5
States: 16
Addressing: direct
Flags: none

LDAX rp   (Load accumulator indirect)

$(A) \leftarrow ((rp))$

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
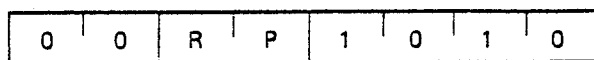
Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

STAX rp   (Store accumulator indirect)

$((rp)) \leftarrow (A)$

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
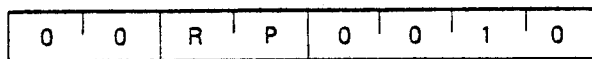
Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

XCHG   (Exchange H and L with D and E)

$(H) \leftrightarrow (D)$
$(L) \leftrightarrow (E)$

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
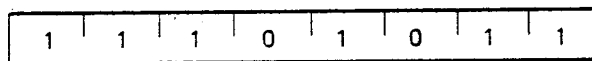
Cycles: 1
States: 4
Addressing: register
Flags: none

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)

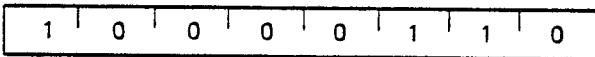The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	S	S	S
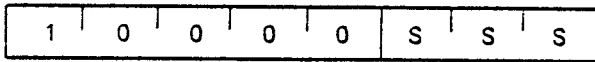
Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

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.
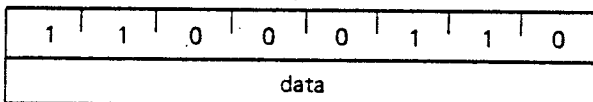
1	0	0	0	0	1	1	0

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.
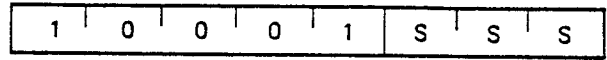
1	1	0	0	0	1	1	0
data							

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.
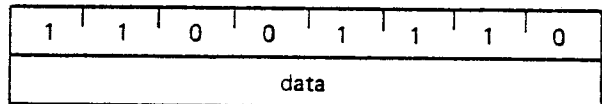
1	0	0	0	1	1	1	0

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

2-56

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.
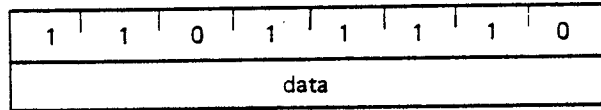
1	0	0	1	0	1	1	0

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.
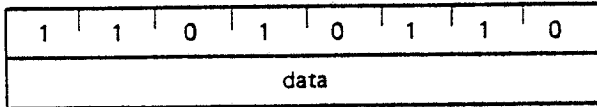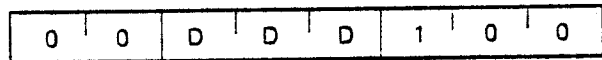
1	1	0	1	1	1	1	0
data								

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

INR r   (Increment Register)

(r) ←— (r) + 1

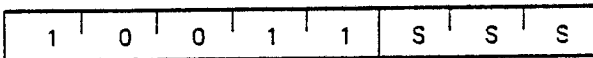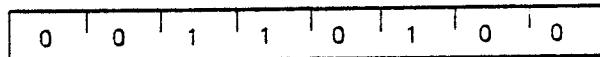The content of register r is incremented by one. Note: All condition flags **except CY** are affected.

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.
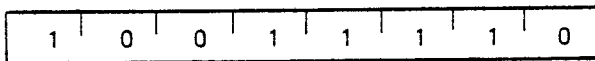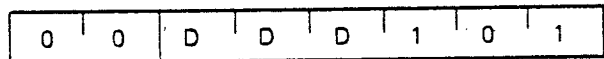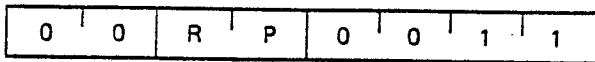
0	0	1	1	0	1	0	1

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.**

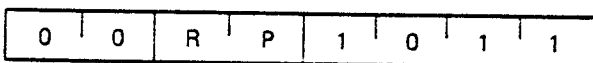| 0	0	R	P	0	0	1	1

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.**

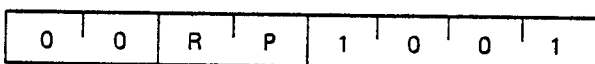| 0	0	R	P	1	0	1	1

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
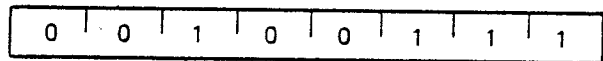
Cycles: 1
States: 4
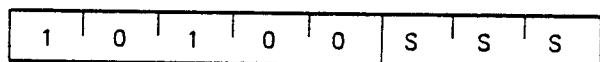Flags: Z,S,P,CY,AC

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 flag is 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 flag is cleared.**
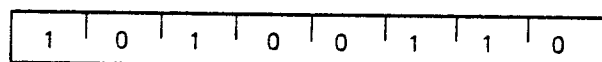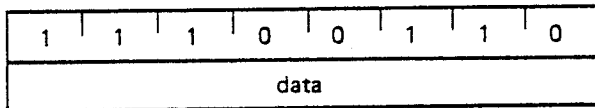
1	0	1	0	0	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

2-58

ANI data   (AND immediate)

(A) ←— (A) ∧ (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.**
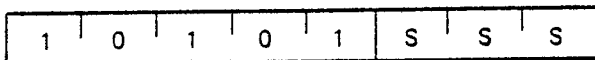
1	1	1	0	0	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

XRA r   (Exclusive OR Register)

(A) ←— (A) ∀ (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) ←— (A) ∀ ((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
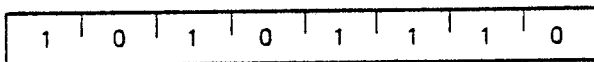
Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

XRI data   (Exclusive OR immediate)

(A) ←— (A) ∀ (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) ←— (A) V (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
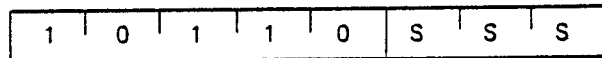
Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ORA M   (OR memory)

(A) ←— (A) V ((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) ←— (A) V (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.**

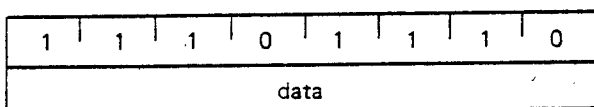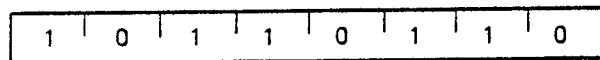| 1	1	1	1	0	1	1	0
data							
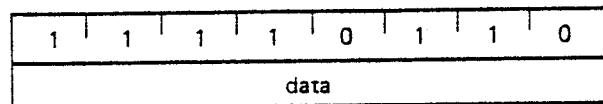
Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

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

2-59
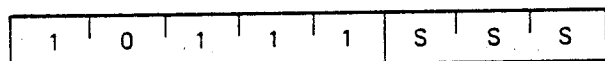
CMP M (Compare memory)

$(A) - ((H) (L))$

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 $(A) < ((H) (L))$.

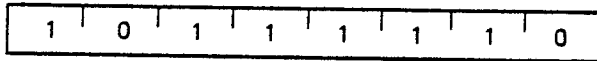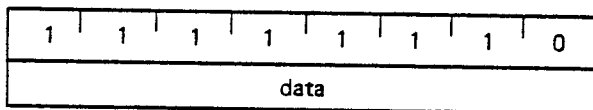| 1	0	1	1	1	1	1	0

    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 $(A) < (byte 2)$.

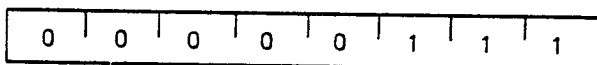| 1	1	1	1	1	1	1	0
data								

    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: 4
    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
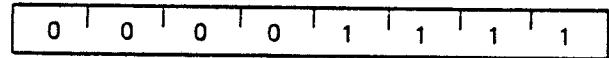
    Cycles: 1
    States: 4
    Flags: CY

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
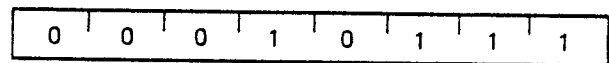
    Cycles: 1
    States: 4
    Flags: CY

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
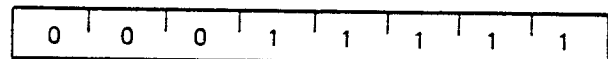
    Cycles: 1
    States: 4
    Flags: CY

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
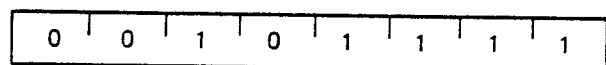
    Cycles: 1
    States: 4
    Flags: none

CMC   (Complement carry)

(CY) ◄─── $\overline{(CY)}$

The CY flag is complemented. **No other flags are affected.**
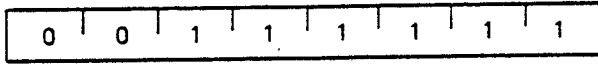
0	0	1	1	1	1	1	1
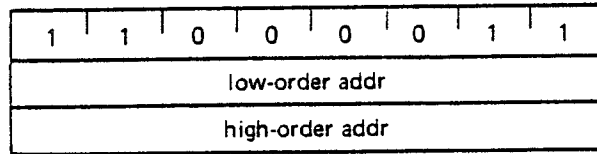
Cycles: 1
States: 4
Flags: CY

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 ad-

dress 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							
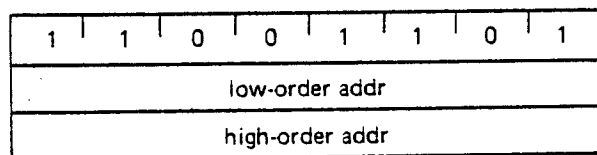
Cycles: 5
States: 17
Addressing: immediate/reg. indirect
Flags: none

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							
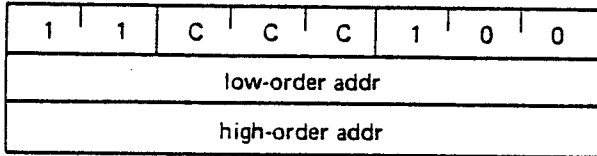
        Cycles: 3/5
        States: 11/17
   Addressing: immediate/reg. indirect
        Flags: none

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
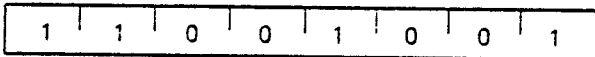
        Cycles: 3
        States: 10
   Addressing: reg. indirect
        Flags: none

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
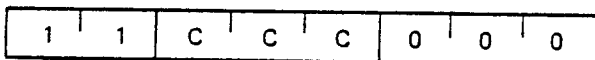
        Cycles: 1/3
        States: 5/11
   Addressing: reg. indirect
        Flags: none

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
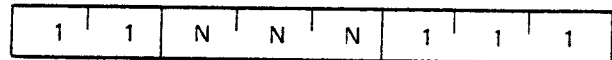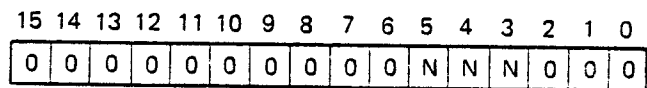
Program Counter After Restart

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

2-62
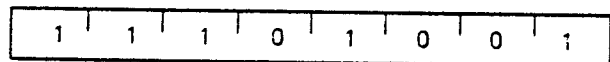
Stack, I/O, and Machine Control Group:

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.**

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

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

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.**
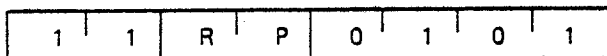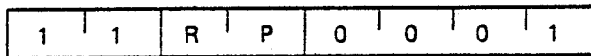
1	1	R	P	0	0	0	1
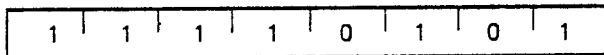
Cycles: 3
States: 10
Addressing: reg. indirect
Flags: none

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
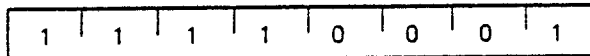
Cycles: 3
States: 10
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

XTHL   (Exchange stack top with H and L)

(L) ⟷ ((SP))

(H) ⟷ ((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
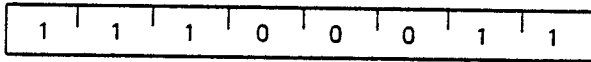
Cycles: 5
States: 18
Addressing: reg. indirect
Flags: none

SPHL   (Move HL to SP)

(SP) ⟵ (H) (L)

The contents of registers H and L (16 bits) are moved to register SP.

1	1	1	1	1	0	0	1
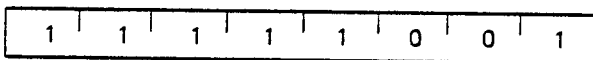
Cycles: 1
States: 5
Addressing: register
Flags: none

IN port   (Input)

(A) ⟵ (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							
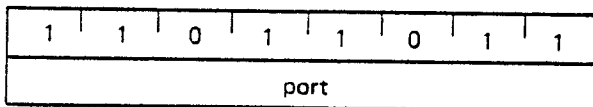
Cycles: 3
States: 10
Addressing: direct
Flags: none

OUT port   (Output)

(data) ⟵ (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							
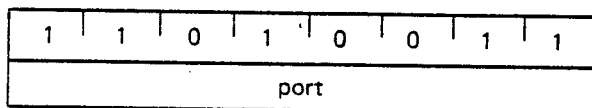
Cycles: 3
States: 10
Addressing: direct
Flags: none

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
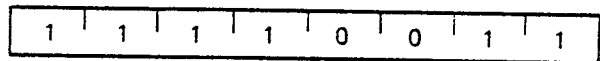
Cycles: 1
States: 4
Flags: none

HLT   (Halt)

The processor is stopped. The registers and flags are unaffected.

0	1	1	1	0	1	1	0
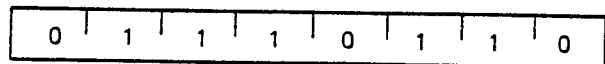
Cycles: 1
States: 7
Flags: none

NOP   (No op)

No operation is performed. The registers and flags are unaffected.

0	0	0	0	0	0	0	0
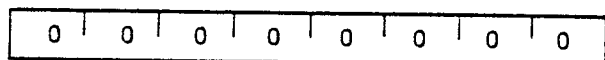
Cycles: 1
States: 4
Flags: none

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	D7	D6	D5	D4	D3	D2	D1	D0	Clock[2] Cycles
MOV r1,r2	Move register to register	0	1	0	0	0	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	0	0	0	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	0	0	0	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	0	0	0	1	0	0	5
DCR r	Decrement register	0	0	0	0	0	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
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.

ALTAIR 8800b
SECTION III
THEORY OF OPERATION