MICROPROCESSOR

A microprocessor (sometimes abbreviated μP) is a digital electronic component with miniaturized transistors on a single semiconductor integrated circuit (IC) .It is a multipurpose, Programmable clock-driven, register based electronic device that read binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as outputs. A Microprocessor is a clock driven semiconductor device consisting of electronic circuits manufactured by using either a LSI or VLSI technique.

Three basic characteristics differentiate microprocessors:

- **Instruction set**: The set of instructions that the microprocessor can execute.
- **Bandwidth**: The number of bits processed in a single instruction.
- **Clock speed**: Given in megahertz (MHz), the clock speed determines how many instructions per second the processor can execute

A typical programmable machine can be represented with three components: MPU, Memory and I/O as shown in Figure

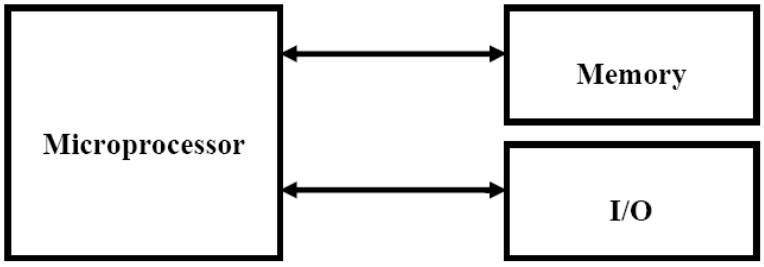


Figure: A Programmable Machine

These three components work together or interact with each other to perform a given task; thus they comprise a system. The machine (system) represented in above figure can be programmed to turn machine on and off, compute mathematical functions, or keep trace of guidance system. This system may be simple or sophisticated, depending on its applications. The MPU applications are classified primarily in two categories: reprogrammable systems and embedded systems- In reprogrammable systems, such as Microcomputers, the MPU is used for computing and data processing. In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to end user.

MICROCOMPUTER

The term *microcomputer* is generally synonymous with personal computer, or a computer that depends on a microprocessor. Microcomputers are designed to be used by individuals, whether in the form of PCs, workstations or notebook computers. A microcomputer contains a CPU on a microchip (the microprocessor), a memory system (typically ROM and RAM), a bus system and I/O ports, typically housed in a motherboard.

Microcomputers are small computers. They range from small controllers that work directly with 4-bit words to larger units that work directly with 32-bit words. Some of the more powerful Microcomputers have all or most of the features of earlier minicomputers. Examples of

Microcomputers are Intel 8051 controller-a single board computer, IBM PC and Apple Macintosh computer.

MICROCONTROLLER

It is a highly integrated chip that contains all the components comprising a controller. Single-chip Microcomputers are also known as Microcontrollers. They are used primarily to perform dedicated functions. They are used primarily to perform dedicated functions or as slaves in distributed processing.

Generally they include all the essential elements of a computer on a single chip: MPU,R/W memory, ROM and I/O lines and timers. Unlike a general-purpose computer, which also includes all of these components, a microcontroller is designed for a very specific task - to control a particular system. A microcontroller differs from a microprocessor, which is a general-purpose chip that is used to create a multi-function computer or device and requires multiple chips to handle various tasks. A microcontroller is meant to be more self-contained and independent, and functions as a tiny, dedicated computer. The great advantage of microcontrollers, as opposed to using larger microprocessors, is that the parts-count and design costs of the item being controlled can be kept to a minimum. They are typically designed using CMOS (complementary metal oxide semiconductor) technology, an efficient fabrication technique that uses less power and is more immune to power spikes than other techniques. Microcontrollers are sometimes called embedded microcontrollers, which just mean that they are part of an embedded system that is, one part of a larger device or system. Typical examples of the single-chip microcomputers are the Intel 8051, AT89C51, AT89C52 and AVR,PIC. Most of the micro controllers have an 8-bit word size, at least 64 bytes of R/W memory, and 1K byte of ROM,I/O lines varies from 16 to 40

GENERAL ARCHITECTURE OF MICROCOMPUTER SYSTEM

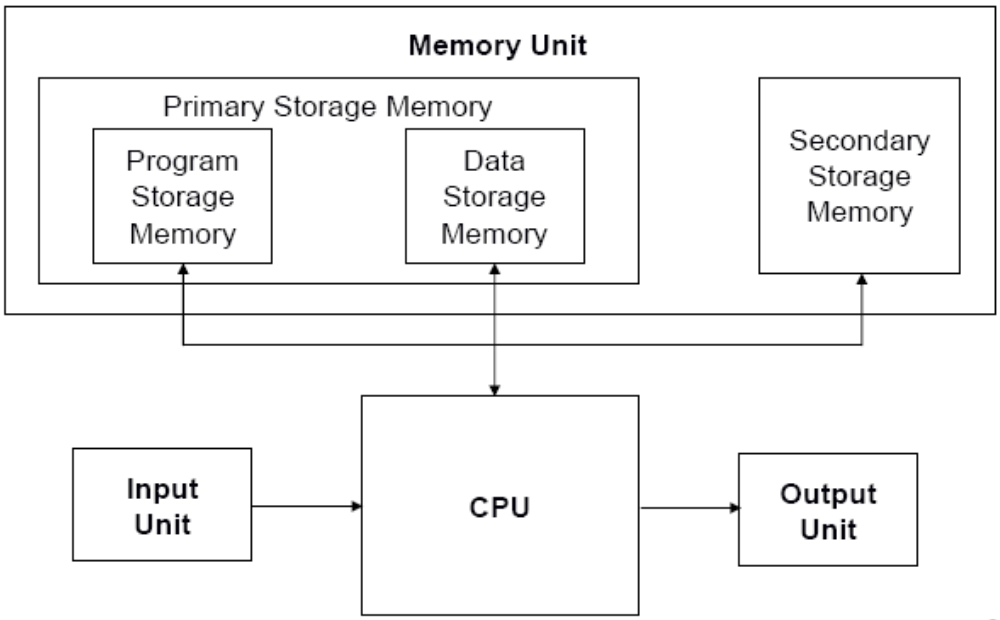


Figure shows a block diagram for a simple Microcomputer. The major parts are the CPU, Memory and I/O. These three parts are connected by three sets of parallel lines called buses. The three buses are address bus, data bus and the control bus.

MEMORY

The First Purpose of memory is to store binary codes for the sequences of instructions you want the computer to carry out. It consists of RAM and ROM.The second purpose of the memory is to store the binary-coded data with which the computer is going to be working.

INPUT/OUTPUT

The input/output or I/O Section allows the computer to take in data from the outside world or send data to the outside world. Peripherals such as keyboards, video display terminals, printers are connected to I/O Port.

CPU

The CPU controls the operation of the computer. In a microcomputer CPU is a microprocessor. it fetches binary coded instructions from memory, decodes the instructions into a series of simple actions and carries out these actions in a sequence of steps. The CPU also contains an address counter or instruction pointer register, which holds the address of the next instruction or data item to be fetched from memory.

ADDRESS BUS

The address bus consists of 16, 20, 24 or 32 parallel signal lines. On these lines the CPU sends out the address of the memory location that is to be written to or read from. The no of memory location that the CPU can address is determined by the number of address lines. If the CPU has N address lines, then it can directly address 2^n memory locations, i.e. CPU with 16 address lines can address 2^{16} or 65536 memory locations.

DATA BUS

The data bus consists of 8, 16 or 32 parallel signal lines. The data bus lines are bi-directional. This means that the CPU can read data in from memory or it can send data out to memory

CONTROL BUS

The control bus consists of 4 to 10 parallel signal lines. The CPU sends out signals on the control bus to enable the output of addressed memory devices or port devices. Typical control bus signals are Memory Read, Memory Write, I/O Read and I/O Write.

COMPONENTS OF CPU

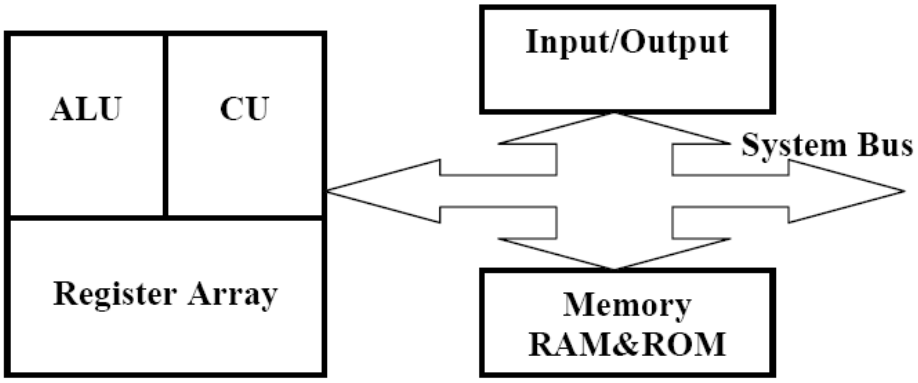


Figure: Microprocessor Based System with Bus Architecture.

The CPU is divided into three segments: ALU, Register array and Control Unit.

ARITHMETIC LOGIC UNIT

This is the area of Microprocessor where various computing functions are performed on data. The ALU performs operations such as addition, subtraction and logic operations such as AND, OR and exclusive OR.

REGISTER ARRAY

These are storage devices to store data temporarily. There are different types of registers depending upon the Microprocessors. These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through the instructions. General purpose Registers of 8086 includes AL, AH, BL, BH, CL, CH, DL, DH. There is also instruction pointer and decoder to decode the instructions fetched from memory.

CONTROL UNIT

The Control Unit Provides the necessary timing and control signals to all the operations in the Microcomputer. It controls the flow of data between the Microprocessor and Memory and Peripherals. The Control unit performs 2 basic tasks-

1. SEQUENCING The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed. It decodes instruction to generate it.

2. EXECUTION The control unit causes each micro operation to be performed.

CONTROL SIGNALS

For the control unit to perform its function it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system.

- **Inputs:**

Clock, Instruction Register, Flags

•Outputs:

Control signals to Memory

Control signals to I/O

Control Signals within the Processor.

EVOLUTION OF INTEL SERIES MICROPROCESSOR

Intel 4004* Microprocessor Chip

In 1969, Intel Corporation began work on a project to develop a set of chips for a series of high-performance programmable calculators for Busicom, a Japanese company. Marcian E. "Ted" Hoff, who had joined Intel in 1968, is assigned to the project. Ted Hoff, along with Federico Faggin, Stan Mazor and others developed a design that included four chips.

The four chip combination included a central processing unit chip (CPU), a read-only memory chip (ROM), and a random access memory chip (RAM), and a shift-register chip for input and output (IO). This design was the first microprocessor chip, which Intel named the 4004.

The Intel 4004 was one-eighth of an inch wide by one-sixteenth of an inch long and contained 2300 metal-oxide semiconductor transistors (MOS). Its computing power was equal to the giant 18,000 vacuum tube ENIAC built in 1946.

The Intel 4004 could execute 60,000 operations per second. Masatoshi Shima, of Busicom, designed the logic for the chip. Shima later joined Intel. Intel sold Busicom the processor design for $60,000, but later bought back the design rights when Stan Mazor and Ted Hoff lobbied for the many other potential uses of the 4004 chip.

Intel 8008* Chip and the Intel 8080* Chip

The 8008 was an 8-bit microprocessor chip and was introduced in April 1972. Designers were Ted Hoff, Federico Faggin, Stan Mazor and Hal Feeney.

An even greater achievement was the 8080 chip, introduced in 1974. The 8080 had 10 times the performance of the 8008 chip and could execute 290,000 instructions per second. It had 64 bytes of addressable memory, and sold for $360 per chip. It quickly became an industry standard. Designers included Mazor, Faggin and Masatoshi Shima.

Intel 8086* Microprocessor

The 8086, announced by Intel Corporation in 1978, had 10 times the performance of the 8080 chip announced in 1974.

The 8086 established a new 16-bit software architecture. The project team included Bill Pohlman, Bob Koehler, John Bayliss, Jim Mckevitt, Chuck Wildman, Steve Morse and others. Motorola introduced the 68000 chip a year later, which directly competed with the 8086. By 1984, however, the 8086 chip was outselling the 68000 by approximately 9 to 1. The 8088 chip was released in 1981.

Intel 80286* Microprocessor

In 1982, Intel Corporation released the 80286 microprocessor chip. At the time of its introduction, the 80286 microprocessor has three times the performance of any other 16-bit chip on the market. The 80286 offered on-chip memory management, making it suitable for multitasking operations. Intel's project leader for the 286 is Gene Hill. Intel also released the 80186 chip, which was an improvement over earlier Intel chips. The 80186 design team was lead by Dave Stamm.

Intel 80386* Microprocessor chip

The Intel 80386 is a 32-bit microprocessor containing over 275,000 transistors on a single chip. The 80386 (commonly known as the "386 chip") could handle four million operations per second and handle memory up to four gigabytes (4,294,967,296). The 386 was also compatible with Intel's earlier processor line for the IBM PC and compatibles and could run software designed for those processors as well. The 386 chip brought desktop personal computing power to a new level. John Crawford was the architecture manager for the Intel 386 and the Intel 486 microprocessors, and co-manager of Pentium microprocessor development.

Intel 80486* Microprocessor chip

In 1989, Intel Corporation announced the 80486 chip, a highly integrated 32-bit microprocessor combining 80386 compatibility, RISC-style CPU, 80387 math co-processor compatibility, 8-Kilobyte on-chip cache and built- in multiprocessing support. The 80486 has a reported capability of holding 1.16 million transistors and is about four times faster than the 80386 processor. Initial uses of the 80486 chip will be for LAN servers and high- end workstations.

John Crawford was the architecture manager for the Intel 386 and the Intel 486 microprocessors, and co-manager of the Pentium microprocessor development. The most common varieties of the 80486 chip are the 486SX (25Mhz), 486DX (33Mhz), and the 486DX2 (66Mhz).

Intel Pentium Microprocessor

In 1993, Intel announced the Pentium chip. The word "Pentium" comes from the Greek root word "pentas" meaning "five." The Pentium is the 80586 chip.

The Pentium is a 32-bit chip with superscalar design, and is estimated to be two times faster than the 486DX2 (66MHz) chip. The Pentium uses dual pipelines to allow it to process two separate instructions in a single cycle. The Pentium has a 64 bit bus interface, an eight bit code cache, an eight bit data cache, and branch prediction memory bank. Don Alpert was the architecture manager of the Pentium, John Crawford was co-manager. The Pentium is a CISC-based (complex instruction set computer) chip containing 3.3 million transistors.

Architecture of 8 bit microprocessor (Intel 8085)

8085

The Intel 8085 microprocessor is an NMOS 8-bit device. Sixteen address bits provide access to 65,536 bytes of 8 bits each. Eight bi-directional data lines provide access to a system data bus. Control is provided by a variety of lines which support memory and I/O interfacing, and a flexible interrupt system. The 8085 provides an upward mobility in design from the 8080 by supporting all of the 8080's instruction set and interrupt capabilities. At the same time, it provides cleaner designs by virtue of a greater on-device component density, and by requiring only a 5 volt supply. In addition, the 8085 is available in two clock speeds.

The 8085 comes in two models, the 8085A and the 8085A-2. The 8085A expects a main clock frequency of 3 MHz, while the 8085A-2 expects a main clock frequency of 5 MHz. In both cases, the clock is a single phase square wave. This single clock is generated within the 8085 itself, requiring only a crystal externally. This eliminates the need for an external clock generator device. In all other respects, the A and A-2 devices are identical.

CPU Architecture

Control Unit

Generates signals within microprocessor to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the microprocessor to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Arithmetic Logic Unit

The ALU performs the actual numerical and logic operation such as 'add', 'subtract', 'AND', 'OR', etc. Uses data from memory and from Accumulator to perform arithmetic. Always stores result of operation in Accumulator.

Registers

The 8085/8080A-programming model includes six registers, one accumulator, and one flag register, as shown in Fig. 2.1 In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

The 8085/8080A has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flags

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions. For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry — called the Carry flag (CY) – is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The Fig. 2.1 shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the micro- processor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register/Decoder

Temporary store for the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and ‘decodes’ or interprets the instruction. Decoded instruction then passed to next stage.

Memory Address Register

Holds address, received from PC, of next program instruction. Feeds the address bus with addresses of location of the program under execution.

Control Generator

Generates signals within microprocessor to carry out the instruction which has been decoded. In reality causes certain connections between blocks of the microprocessor to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack in the example. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

General Purpose Registers

Microprocessor requires extra registers for versatility. Can be used to store additional data during aprogram. More complex processors may have a variety of differently named registers.

System Bus

Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: Address Bus, Data Bus and Control Bus.

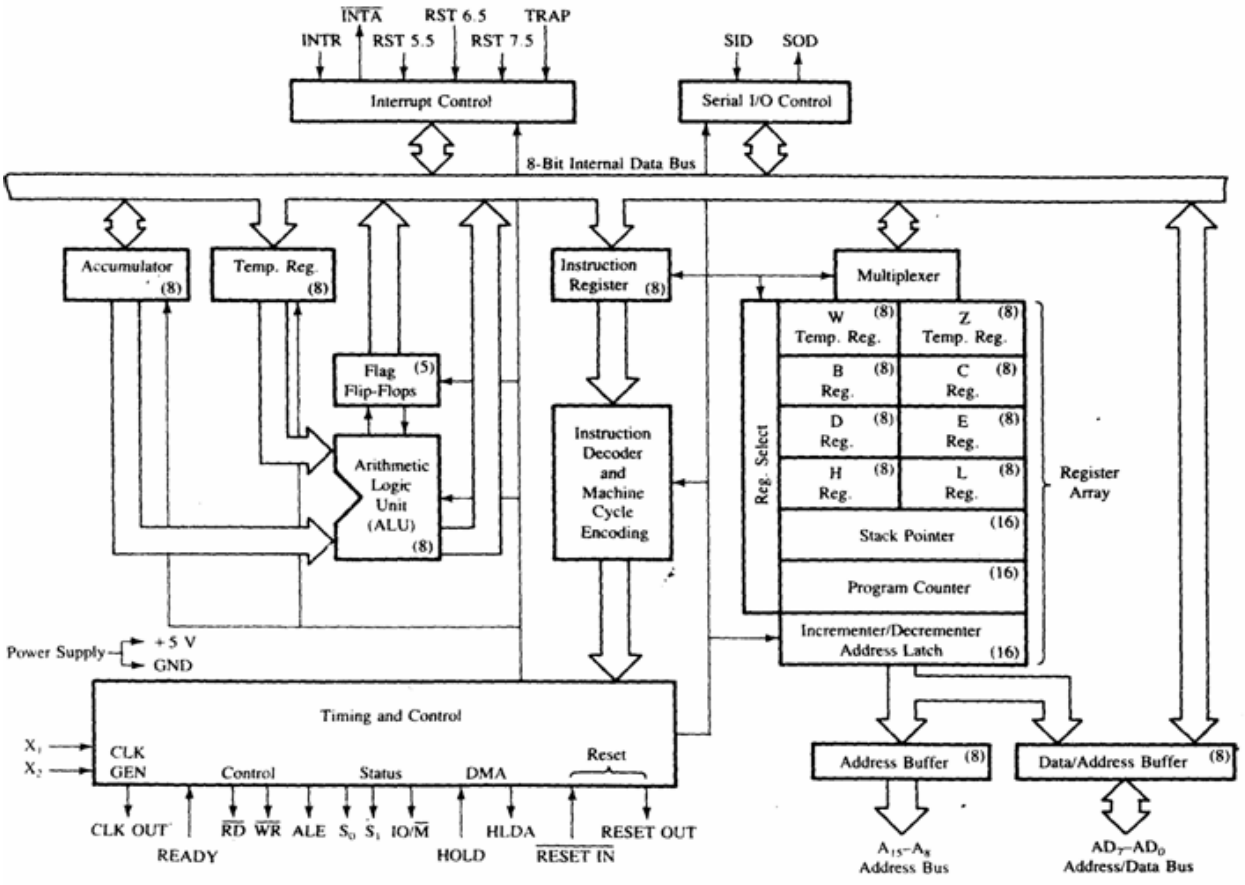


Fig 2.1: Internal Architecture of 8085

Address Bus

One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to 'open' the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its "width" is 16 bits. A 16 bit binary number allows 216 different numbers, or 32000 different numbers, ie 0000000000000000 up to 1111111111111111. Because memory consists of boxes, each with a unique address, the size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus, eg 0000000000000011 (3 in decimal), to the memory. The memory selects box number 3 for reading or writing data. Address bus is unidirectional, i.e numbers only sent from microprocessor to memory, not other way.

Data Bus

Data Bus: carries 'data', in binary form, between microprocessor and other external units, such as memory. Typical size is 8 or 16 bits. The Data Bus typically consists of 8 wires. Therefore, 28 combinations of binary digits. Data bus used to transmit "data", ie information, results of arithmetic, etc, between memory and the microprocessor. Bus is bi-directional. Size of the data bus determines what arithmetic can be done. If only 8 bits wide then largest number is 11111111 (255 in decimal). Therefore, larger number have to be broken down into chunks of 255. This slows microprocessor. Data Bus also carries instructions from memory to the microprocessor. Size of the bus therefore limits the number of possible instructions to 256, each specified by a separate number.

Control Bus

Control Bus are various lines which have specific functions for coordinating and controlling microprocessor operations. Eg: Read/NotWrite line, single binary digit. controls whether memory is being 'written to' (data stored in mem) or 'read from' (data taken out of mem) 1 = Read, 0 = Write. May also include clock line(s) for timing/synchronising, 'interrupts', 'reset' etc. Typically microprocessor has 10 control lines. Cannot function correctly without these vital control signals.

Flag Register

The Status Flags of the 8080 and 8085 are single bits which indicate the logical conditions that existed as a result of the execution of the instruction just completed. This allows instructions following to act accordingly, such as a branch as a result of two values comparing equal. The flags are:

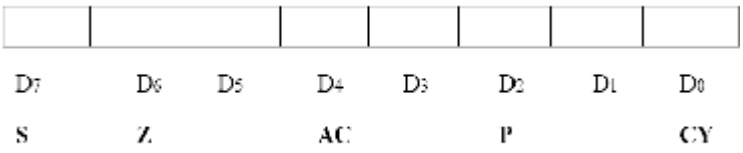
ZERO FLAG: This flag is set to a 1 by the instruction just ending if the A Register contains a result of all 0's. Besides the obvious mathematical applications, this is useful in determining equality in a compare operation or in logical AND or OR operations where the result left the A Register with no bit set to a 1 (the AND was not satisfied). If any bits were left set to a 1 in the A Register, the flag will be reset to a 0 condition.

SIGN FLAG: This flag is set to a 1 by the instruction just ending if the leftmost, or highest order bit of the A Register is set to a 1. The leftmost bit of a byte in signed arithmetic is the sign bit, and will be 0 if the value in the lower seven bits is positive, and 1 if the value is negative.

PARITY FLAG: This flag is set to a 1 by the instruction just ending if the A Register is left with an even number of bits set on, i.e., in even parity. If the number of bits in the A Register is odd, the bit is left off. This may be useful in I/O operations with serial devices, or anyplace that error checking is to be done.

CARRY FLAG: This flag is set to a 1 by the instruction just ending if a carry out of the leftmost bit occurred during the execution of the instruction. An example would be the addition of two 8-bit numbers whose sum was 9 bits long. The 9th bit would be lost, yielding an erroneous answer if the carry bit was not captured and held by this flag. This flag is also set if a borrow occurred during a subtraction or a compare operation.

AUXILIARY CARRY FLAG: This flag is set to 1 by the instruction just ending if a carry occurred from bit 3 to bit 4 of the A Register during the instruction's execution. Because of the relationships of decimal in pure BCD to hexadecimal coding, it is possible to bring BCD values directly into the A Register and perform mathematical operations on them. The result, however, will be as if two hex characters are being processed. If the result must be returned to the program as BCD rather than as hex, the Decimal Adjust Accumulator (DAA) instruction can make that translation; the Auxiliary Carry Flag is provided to assist in this operation.



Instruction Format

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size**:**

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, “byte” and “word” are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction. These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8- bit binary format in memory; each requires one memory location.

MOV rd, rs

rd <– rs copies contents of rs into rd.

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r

A <– A + r

Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand.

MVI r,data

r <– data

Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

ADI data

A <– A + data

OUT port

where port is an 8-bit device address. (Port) <– A. Since the byte is not the data but points directly to where it is located this is called direct addressing.

Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

This instruction would require three memory locations to store in memory. Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp <– data16

Example:

LXI H, 0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A <– (addr) Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing

Instruction Set

Data Transfer Group

This group of instructions provides the 8085 with the ability to move data around inside the RAM, between the RAM and the registers of the micro processor, and between registers within the micro processor. They are important because a good deal of moving must be done to offset the fact that in an 8-bit byte, there is insufficient room to define the operands as specifically as is done, for example, in the PDP11. These instructions do not affect the condition codes. A few comments on the OP code groups follow:

MOV Group: These three instructions (MOV r1,r2, MOV r,M, and MOV M,r) are the general means of moving data between memory and registers. They move one byte with each execution. The second and third examples illustrate the use of the Register Indirect mode of addressing, in which the H&L registers of the Micro Processor contain an address, and the data is moved to or from that address. This saves space, in that the instruction is only one byte long. It requires, however, that the H&L registers be previously set up with the address required. The letter “M”, when appearing as an operand in this description, specifies Register Indirect mode with H&L as the register to contain the address. No other register pair is used as such.

MVI Group: These two instructions (MVI r, data, and MVI M, data) provide a means of loading a byte immediately into a register or a memory address. Note that the Register Indirect mode again is evident. The immediate data is stored as a byte directly below the instruction byte.

LXI Instruction: This single instruction (LXI rp,data16) provides a means of loading any register pair with a two-byte value. The second byte l of the instruction is loaded into the leftmost, or low-order, register of the pair, while the third byte is loaded into the rightmost, or high order, register of the pair.

LDA & STA: The Load Accumulator Direct (LDA) and the Store Accumulator Direct (STA) instructions provide a means of moving a byte between the accumulator and a RAM address. This may also be done with the MOV OP code, but only indirectly, that is, with the help of the H&L registers. The address of the byte to be loaded or stored follows the instruction, again with the inverse addressing.

LHLD & SHLD: The Load H&L Direct (LHLD) and Store H&L Direct (SHLD) instructions provide a means of moving two bytes between the HL register pair and a RAM address. Since the H&L register pair is heavily used in the Register Indirect mode, these instructions provide a quick means of loading the two bytes of an address into the pair in only one instruction. The two bytes following the instruction contain an address in RAM, again low-order in byte 2, and high-order in byte 3. For the LHLD, this address points to a single byte, which is obtained and loaded into the L register. The second byte from RAM is obtained from the address one higher than the RAM byte, and loaded into the H register. The SHLD simply stores as above, instead of load operation.

LDAX & STAX: The Load Accumulator Indirect (LDAX) and Store Accumulator Indirect (STAX) instructions provide a means of moving data between the accumulator and a memory location indirectly, with the RAM address contained in either the BC or DE register pair. This is not the same as the MOV, which uses only the HL register pair. This instruction permits the accumulator to access groups of data bytes, as may be necessary with long precision arithmetic. Obviously, the BC or DE pair must be previously loaded with the address desired.

XCHG Instruction: The Exchange (XCHG) instruction permits the HL register pair's contents to be exchanged with the DE register pair's contents. This allows an address to be built in the DE pair, then, when ready, to be transferred at once to the HL pair. This would be advantageous in complex data handling.

Arithmetic Group

This group provides the 8085 with mathematical ability to manipulate 8-bit data, and, by judicious use of the condition codes, to manipulate larger values. The A register (Accumulator) can perform true adds, subtracts, and compares. The other registers can only increment or decrement by 1. Unless otherwise indicated, all the condition code flags are affected. A few comments follow:

ADD Instructions: The Add Register (ADD r) and Add Memory (ADD M) instructions add the byte specified, either in a register or in the address contained by the H&L registers, into the accumulator. They assume that the accumulator already has in it the other value to participate in the add. The sum will remain in the accumulator. If the answer resulted in a ninth bit, it is stored in the Carry flag of the PSW.

ADD with Carry: The Add Register with Carry (ADC r) and Add Memory with Carry (ADC M) instructions will add the specified byte, either in a register or in the address contained by the H&L registers, AND the value of the Carry bit, into the accumulator. By including the carry bit in the operation, mathematical operations on values longer than 8 bits are possible. As above, the first value must already be loaded in the A register prior to execution of these instructions. The sum remains in the accumulator. If the answer resulted in a ninth bit, it is stored in the Carry flag.

ADD immediates: The Add Immediate (ADI) and Add Immediate with Carry (ACI) instructions provide a means of adding a fixed value into the accumulator. These instructions assume that an initial value has already been loaded into the accumulator. The immediate data is provided by the second byte of the instruction. The ACI instruction adds the immediate value and the value of the Carry flag, while the ADI does not take the Carry flag into account. The sum remains in the accumulator. If the answer resulted in a ninth bit, it is stored in the Carry flag.

Subtract Instructions: The Subtract Register (SUB r) and the Subtract Memory (SUB M) instructions subtract the specified byte, in a register or in the address contained by the H&L registers, from the contents of the accumulator. The accumulator must have the first value already loaded , prior to the execution of the instructions. The subtract is accomplished by the complement-and-add technique, in which the two's complement of the specified value is computed first, and then added to the contents of the A register. The Carry flag will be set to a 1 if a borrow was required during the subtraction.

Subtract with Borrow: The Subtract Register with Borrow (SBB r) and the Subtract Memory with Borrow (SBB M) instructions will subtract the specified byte, either in a register or in the address contained in the H&L registers, and the value of the Carry flag, from the contents of the A register. The first value must be loaded into the A register prior to the execution of the instructions. The subtract is accomplished by the complement-and-add technique. The Carry flag will be set to a 1 if a borrow was required during the subtraction.

Subtract immediate: The Subtract Immediate (SUI data) and Subtract Immediate with Borrow (SBI data) instructions provide a means of subtracting a fixed value from the contents of the accumulator. The immediate value is provided by the second byte of the instruction. The first value must be loaded into the accumulator prior to the execution of the instructions. The subtract is accomplished by the complement-and-add technique. The SBI instruction will subtract both the immediate value and the contents of the Carry flag from the A register, while the SUI does not take the Carry flag into account. The Carry flag will be set at the end of the instruction if , a borrow was required during execution.

Increment Instructions: The Increment Register (INR r) and Increment Memory (INR M) instructions provide a quick means of adding one to the contents of a register or memory location. These instructions allow the programmer to create counting routines and reiterations. Note that the Carry flag is not affected by these instructions.

Decrement Instructions: The Decrement Register (DCR r) and Decrement Memory (DCR M) instructions provide a quick means of subtracting one from the contents of a register or a memory location. These instructions allow the programmer to create counting routines and reiterations. Note that the Carry flag is not affected by these instructions.

Register Pair Instructions: The Increment Register Pair (INX rp) and Decrement Register Pair (DCX rp) instructions provide a means of adding to, or subtracting from, a 16-bit value contained in a register pair. In the INX instruction, this means that the carry from the sum of the low order byte of the pair and the one will be added into the upper byte automatically. In the DCX instruction, this means that a borrow from the high-order byte, if required, will be allowed into the low-order byte, if the subtraction of one from the low-order byte demands it. Note that none of the flags are affected.

D0UBLE ADD: The Add Register Pair to H&L (DAD rp) instruction adds a 16 bit value already existing in the BC or DE register pair into the 16-bit value contained in the H&L registers. The sum remains in the H&L registers. The Carry flag will be set if a carry occurred out of the high order byte; a carry from low- to high-order bytes within the add is taken into account automatically. This instruction allows a fixed index-like value to be added to the H&L registers for Register Indirect mode.

Decimal Adjust: The Decimal Adjust Accumulator (DAA) instruction converts the 8-bit value in the A register, which normally is assumed to be two 4 bit hexadecimal values, into two 4-bit BCD values. This allows the programmer to accept input data as BCD, process it in the accumulator using essentially hexadecimal arithmetic, and then convert the result back into BCD. This may be done by virtue of the fact that the ten numbers of BCD (0 to 9) are coded in binary exactly as are the first ten of the sixteen numbers of binary coded hexadecimal. i.e., adding 38_{10} and 38_{16} are exactly the same. The conversion be may accomplished by the use of the Auxiliary Carry flag. If the contents of the low-order four bits of the A register is >9, or if the AC flag is set, a value of 6 is added to these bits. Then the high-order four bits of the A register are examined; again, if they contain a value >9, or if the Carry flag is on, a 6 is added to them. The Carry flag, of course, indicates that the hexadecimal value of the byte before the instruction, when translated to BCD, is too large to fit in one byte.

Logical Group

This group of instructions provides the decision-making ability of the 8085, and includes some logically oriented utility instructions as well. By using these instructions, the condition flags may be set so that they can be tested by Jump-on-condition instructions. Unless otherwise noted, all the condition codes are affected. A few notes follow:

AND Instructions: The And Register (ANA r) and And Memory (ANA M) instructions perform a logical And function between the specified byte, either in a register or in the address contained in the H&L registers, and the contents of the accumulator. The accumulator must first be loaded with an initial value. The And function occurs on a bit-by-bit basis. The low order bit of the specified byte is Anded with the low order bit of the A register; if both the bit from the outside byte AND the bit from the A register are a 1, the bit in the A register is left as a 1. If either the bit position of the outside byte or the bit position in the A register, or both, contained 0's, that bit position in the A register is reset to 0. Identical actions occur on the other seven bit positions at the same time. The result, left in the accumulator, is a bit pattern which indicates, with 1's left on, in which positions of the bytes both the A register and the outside byte contained 1's. This is valuable for testing the conditions of specific bits within a byte, and reacting accordingly. All condition flags are involved, but the Carry flag is always cleared by an And.

AND immediate: The And Immediate (ANI data) instruction allows the programmer to match the byte in the accumulator with a fixed mask byte, contained in the second byte of the

instruction. The A register must first be loaded with the byte to be tested. The Anding function occurs exactly as shown above. All condition flags are involved, but the Carry flag is cleared.

OR Instructions: The Or Register (ORA r) and Or Memory (ORA M) instructions perform inclusive Or's between the specified byte, either in a register or in the address contained in the H&L registers, and the contents of the accumulator. All condition flags are affected, but the Carry and Auxiliary Carry flags are always cleared.

OR immediate: The Or Immediate (ORI data) instruction allows the programmer to match the contents of the accumulator against a fixed mask byte which is contained in the second byte of the instruction. The first byte must be loaded into the A register prior to execution of the instruction. All condition flags are affected, but the Carry and Auxiliary Carry flags are always cleared.

Exclusive or Instructions: The Exclusive Or Register (XRA r) and the Exclusive Or Memory (XRA M) instructions perform exclusive Or functions between a specified byte, either in a register or in a byte contained in the address in the H&L register, and the contents of the accumulator. All condition flags are affected, but the Carry and Auxiliary Carry flags are always cleared.

Exclusive or Immediate: The Exclusive Or Immediate (XRI data) instruction allows the programmer to perform an Exclusive Or between a mask byte stored as the second byte of the instruction and the contents of the accumulator. The first byte must be loaded into the A register prior to the execution of the instruction. The Exclusive Or function occurs on a bit-by-bit basis exactly as outlined above. All the condition flags are affected, but the Carry and Auxiliary Carry flags are cleared.

Compare Instructions: The Compare Register (CMP r) and Compare Memory (CMP M) instructions compare the contents of the specified byte, either in a register or in the address contained in the H&L registers, to the contents of the accumulator. This is accomplished by subtracting the byte from the contents of the accumulator. The contents of the accumulator remain unchanged, and the actual answer of subtraction is lost. The condition flags are all affected .Particularly, the Zero flag, if set on, will indicate that the two values compared are equal, since the result of subtracting one from the other is zero. Also, the Carry flag will be set if the value in the A reg is smaller than the byte to be compared. If neither the Z nor the C flags are left on, the value in the A register is larger than the byte.

Compare Immediate: The Compare Immediate (CPI data) instruction compares the contents of the accumulator to a fixed value provided by the second byte of the instruction. The first value must be loaded into the A register prior to the execution of the instruction.. The contents of the A register are left unchanged.

Rotate Instructions: The Rotate Left (RLC) and Rotate Right (RRC) instructions rotate the accumulator's contents one bit position left or right, respectively. In the RLC, all the bits move one position to the left; the high order bit which is shifted out of the A register is moved around to the low order bit position. It is also moved to the Carry flag. In the RRC, all the bits move one position to the right; the bit shifted out of the low order position of the A register is moved around to the high order position. It is also moved to the Carry flag. Thus, the Carry flag in either case indicates whether a bit was shifted out of the accumulator. Only the Carry flag is affected by these instructions.

Rotate through Carrys: The Rotate Left through Carry (RAL) and the Rotate Right through Carry (RAR) instructions rotate the accumulator's contents one bit position left or right, respectively. Unlike the rotates above, however, these instructions use the Carry flag as a ninth bit in the circle. In the RAL, the bits in the A register are shifted left one position; the high order bit moved to the Carry flag; the Carry flag is moved to the low order position of the A register. In the RAR, the bits in the A register are shifted right one position; the low order bit is moved to the Carry flag; the Carry flag is moved to the high order position of the A register. Only the Carry flag is affected.

Complement Accumulator: The Complement Accumulator (CMA) instruction provides a 1's complement of the 8 bits in the A register, i.e., the 1's are set to 0's, and the 0's are set to 1's. A two's complement may be effected by following the CMA with an INR A instruction. No condition flags are affected.

Carry Instructions: The Complement Carry (CMC) and Set Carry (STC) instructions allow direct control of the Carry flag by the programmer. The CMC will change the flag from 1 to 0, or 0 to 1, depending upon its initial condition. The STC forces the flag to a 1, regardless of its previous state. No other flags are affected.

Branch Group

This group of instructions permits the programmer to alter the flow of program execution from a normal straight line. There are two major types of these instructions in the 8085. The first type is the Jump, in which the flow is altered with no intention of returning to the place where the Jump occurred. The second type is the Call, which provides linking, via the system stack, to save the address of the next instruction following the Call, proceed to a subordinate routine, and return to the saved address when that routine is completed.

Further, both Jumps and Calls may be conditional or unconditional. An unconditional Jump or Call causes the function to be executed absolutely. The conditional Jump or Call causes the function to be executed if the conditions specified are met. In the first byte of these instructions, three bits labeled CCC will contain a code which specifies the conditions to be tested. These may be specified by the programmer in assembly language by putting together a mnemonic composed of a J, for Jump, or a C, J for Call, followed by one or two more characters which specify the conditions to be tested. The breakdown follows:

Mnemonic	Condition	CCC Bits
NZ	Not Zero (Z=0)	000
Z	Zero (Z=1)	001
NC	Not Carry (C=0)	010
C	Carry (C=1)	011
PO	Parity Odd (P=0)	100

PE	Parity Even (P=1)	101
P	Plus (S=0)	110
M	Minus (S=1)	111

Jump Instructions: The Jump (JMP addr) and Jump Conditional (Jxx addr) instructions allow program flow to be altered by loading the contents of the two bytes following the instruction to be loaded into the Program Counter. The next instruction to be fetched, therefore, will the first of the new routine. The JMP instruction is unconditional; the Jump occurs absolutely. The Jxx instruction will alter program flow if the conditions specified by the "xx" bits are true; otherwise, program flow remains in a straight line. No condition codes are affected.

CALL Instructions: The Call (CALL addr) and Call Conditional (Cxx addr) instructions allow linkage to permit a subroutine to be invoked, with the address of the next sequential instruction saved for later reference. The Call will move the high byte of the PC into the address pointed to by the Stack Pointer minus 1, and the low byte of the PC into the address below that. The SP is then decremented by two, to update it to the new stack position. The two bytes following the Call instruction will then be moved to the PC, with the second byte of the instruction containing the low order byte of the address, and the third byte of the instruction containing the high order byte of the address. Thus, the address of the instruction following the Call is saved on the system stack, and the address of the first instruction of the subroutine is fetched next. The Call Conditional executes exactly the same way, providing that the conditions specified by the CCC bits are true. None of the flags are affected.

Return Instructions: The Return (RET) and Return Conditional (Rxx) instructions provide a means, at the end of a subroutine, of resuming program execution at the instruction following the Call instruction which invoked the subroutine. These instructions are placed at the end of the subroutine, not in the body of the main program. When encountered, the Return will move the byte pointed to by the Stack Pointer into the lower byte of the PC, the next byte higher in RAM to the higher byte of PC, and add 2 to the contents of SP. Thus, the address of the instruction following the Call, previously saved on the stack, is now in PC, and will be fetched next. Also, the stack pointer is updated accordingly. The Return Conditional executes exactly the same way, providing that the conditions specified by the CCC bits are true. None of the flags are affected.

Restart: The Restart (RST n) instruction provides part of the vectored interrupt system by which any one of eight different levels of interrupt may stop the execution of the program currently in progress, save the address of the next instruction onto the stack, and then jump to any one of eight different locations in low core, depending upon the contents of the bits marked NNN in the instruction. Thus, as many as eight different external events, i.e. I/O devices, etc., may ask for service; the place where the program left off is saved; and one of eight different interrupt handling routines may be entered, which correspond to the level of the interrupt. This will be fully explained in the section on interrupts.

Jump Indirect: The Jump H&L Indirect (PCHL) instruction moves the contents of the H&L registers, assumed to be a valid address, into the Program Counter. The original contents of the PC are destroyed, so this is a one-way jump.

Machine Control Group

This group is a collection of miscellaneous instructions which control bodily functions of the MP, or provide utilities. Explanations follow:

PUSH and POP: The Push Register Pair (PUSH rp) and Pop Register Pair (POP rp) instructions allow programmers to manipulate the system stack. The Push will place the contents of the BC, DE, or HL register pairs onto the stack, and update the SP accordingly. The Pop instruction will return the last two items on the stack to the specified register pair, and update the SP. The condition flags are not affected; the SP register pair may not be specified, for obvious reasons.

PSW Instructions: The Push Processor Status Word (PUSH PSW) and the Pop Processor Status Word (POP PSW) instructions will allow the programmer to save the contents of the A register and of the condition flags on the stack, or to retrieve them from the stack. The Processor Status Word (PSW) of the 8085 is defined as a “Flag Byte” which contains the condition flag bits in a specific sequence:

In addition, the contents of the A register is also saved as part of the PSW. When the PUSH PSW is encountered, the contents of the A register is pushed onto the stack first, followed by the Flag byte. The SP is then updated. When the POP is executed, the Flag byte is retrieved first, and the bits are loaded into their proper flip-flops. The A register is then loaded with the next byte retrieved. This allows programmers to save conditions at the beginning of subroutines so that the execution of the instructions within the routines will not alter the conditions under which the original program was operating.

Exchange Stack Top: The Exchange Stack Top with H&L (XTHL) instruction causes the contents of the H&L registers to be exchanged with the two bytes which are currently on the top of the system stack. These will be the last two bytes pushed. It is a two-way instruction; the stack receives the original contents of H&L, while H&L receives the two bytes from the stack. The contents of SP remain unchanged. No flags are affected.

Move H&L to SP: The Move H&L Register to Stack Pointer (SPHL) instruction will directly move the contents of the H&L registers into the Stack Pointer; the original contents of SP are destroyed. This may be used to permit multiple stacks to exist at one time in the system. No flags are affected.

I/O Instructions

The Input (IN port) and Output (OUT port) instructions allow the MP to communicate with the outside world. In both cases, the address byte of the device to be used is contained in the byte following the instruction. This byte is presented at once to both the upper and lower bytes of the A0-A15 address lines. In the case of IN, the byte accepted on the D0-D7 data lines by the MP is placed in the A register. For the OUT, the byte to be sent on the data lines is placed in the A register prior to execution of the instruction. No flags are affected.

Interrupt Instructions: The Enable Interrupts (EI) and Disable Interrupts (DI) instructions allow the MP to permit or deny interrupts under program control. For the EI, the interrupts will be enabled following the completion of the next instruction following the EI. This allows at least one more instruction, perhaps a RET or JMP, to be executed before the MP allows itself to again be interrupted. For the DI, the interrupts are disabled immediately. No flags are affected.

Halt and NOP: The Halt (HLT) and No-Operation (NOP) instructions serve general utility purposes. The Halt will stop the processor from further execution; it can be restarted again only by an interrupt. A reset signal applied to the MP will abort the Halt. The MP may enter a Hold state, as the result of another device wanting the bus, from a Halt, but will return to the Halt state when the Hold is canceled.

Instruction and Machine cycle

Consider a simple instruction: MOV A, B which states that data from register B is to copied to register A. The instructions are all stored in the memory. The computer spends certain period of time on Fetching, Decoding and Executing this instruction.

Instruction Cycle: It is defined as the time required to complete execution of an instruction.

Machine cycle: It is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request.

T-state: It is defined as one subdivision of the operation performed in one clock period.

Addressing Modes

The 8085 provides four different modes for addressing data, either in its registers or in memory. These are described below:

DIRECT MODE - This mode creates instructions three bytes long. The first byte contains the operation to be performed. The second and third bytes contain the address in memory where the data byte may be found. Thus, the instruction directly specifies the absolute location of the data. Note that the second byte of the instruction contains the low order byte of the address, while the third byte contains the high order byte of the address. This illustrates the inverse addressing of the device.

OUT 32H

where 32H is an 8-bit device address. It moves the content of accumulator to specified address (Port) <– A. Since the byte (32H) is not the data but it points directly to where data is located this is called direct addressing.

REGISTER MODE - This mode results in single-byte instructions. The byte contains bits which specify a register or register pair in which the data is located.

REGISTER INDIRECT MODE - This mode results in single-byte instructions. The byte contains bits which specify a register pair, which in turn contains the address of the data in memory. Thus, the instruction indirectly specifies the address of the data by referring to a register pair for the absolute address. Note that the high order byte of the address is stored in the leftmost register of the pair, while the low order byte of the address is

stored in the rightmost register of the pair. The address 3000H, therefore, would be stored in the HL register pair as 30 in H, and 00 in L.

IMMEDIATE MODE - This mode results in a two or three byte instruction. The first byte contains the instruction itself. The second and third bytes contain the immediate data, either as a single 8-bit value, or as a 16-bit value. If the 16-bit value is used, the bytes are reversed as discussed previously, with the second byte containing the low order byte, and the third byte containing the high order byte.

MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

Register transfer Language (RTL):

The internal network organization of a digital computer is defined by specifying the following criteria :

- The set of register it contains and their functions.
- The sequence of micro operation performed on the binary information stored in the registers.
- The control that initiates the sequence of micro operation.

Depending upon the instruction they are executed accordingly. These operations are expressed using a language. Such type of language, which is basically used to express the transfer of data among the registers, is called Register Transfer Language (RTL).It is the symbolic notation used to describe the micro-operation transfer among register. If a data is transferred from register A to register B the in RTL

Register B ← Register A

During the execution of an instruction we have fetch cycle and execute cycle.

The operations performed during fetch cycle are:

The PC contains the address of the next instruction to be executed. As first operation of fetch cycle, the contents of program counter will be transferred to the memory address register (MAR). The memory address register then uses the address bus to transmit its contents that specifies the address of the memory location from where the instruction code is to be fetched. Let T1 be period of this operation.

T1: MAR ←PC

Now as soon as the control unit issues the memory read signal, the contents of the addressed memory location specified by MAR will be transferred to the memory buffer register (MBR). Let T2 be the period of this operation.

T2: MBR← [MAR] or Memory, M

Now the contents of MBR will be transferred to the instruction register and the program counter (PC) gets incremented to fetch another instruction and the fetching cycle is thus complete. Here two operations take place within a single time unit T3. Let T3 be the time required by the CPU for this operation.

T3: IR ←MBR
PC← PC+1

The control unit of the computer maintains the timing sequence of the all of the above operations that constitute the fetch cycle. The operations are sequenced on the basis of the single time period called the period of the clock. The complete RTL for fetch cycle is:

T1: MAR ←PC
T2: MBR← [MAR] or Memory, M
T3: IR ←MBR
PC← PC+1

All three time units are of equal duration. A time unit is defined by a regularly spaced clock pulses. The operations performed within this single unit of time are called micro-operations. Since each micro-operation specifies the transfer of data into or out of a register, such type of language is called Register Transfer Language. In other words, the convenient to adopt suitable Symbols to describe the sequence of transfers between registers and various arithmetic and logic micro-operations associated with the symbolic notation used to describe the micro-operation transfer among registers is called Register Transfer Language (RTL). RTL- provides an organized and concise manner for listing for micro-operations sequences in registers and the control functions that initiate them.

After the fetch of the instruction is complete, the execution cycle starts. Different instructions are executed in different fashions. Let us consider execution of a simple instruction MOV A,B.The control signal from control unit after decoding the instruction activates the load register signal for register A as well as sends signal to release the data from register B to data bus simultaneously. So register A is filled with data of register B.

More examples of RTL

1. **RTL for MOV A, B**

T1: MAR← PC
T2: MBR← [MAR] or Memory, M
T3: IR← MBR
PC← PC+1
T4: Decode; [A]← [B]

2. **RTL for MVI A, 01H**

T1: MAR← PC
T2: MBR← [MAR] or Memory, M
T3: IR ←MBR
PC← PC+1
T4: Decode;
T5: MAR ←PC
T6: MBR ←[MAR]
T7: IR← MBR
PC ←PC+1

3. **RTL for LXI B,2000H**

T1: MAR← PC
T2: MBR← [MAR] or Memory, M
T3: IR← MBR ; copying opcode of LXI B, 16 bit data
PC ←PC+1
T4: Decode;
T5: MAR← PC
T6: MBR← [MAR] ; copying 00(lower byte data)
T7: IR← MBR
PC← PC+1
T6: MBR← [MAR] ; copying oo(higher byte data)
T7: IR ←MBR
PC ←PC+1

4. ***RTL for LDA 2000H***

T1: MAR ←PC
T2: MBR← [MAR] or Memory, M
T3: IR ←MBR ; copying opcode of LDA instruction
PC ←PC+1
T4: Decode;
T5: MAR ←PC
T6: MBR← [MAR]
T7: temp[Z] ←MBR ; copying the lower byte (00) into Z temporary register
PC← PC+1
T8: MAR ←PC
T9: MBR← [MAR]
T10: temp[W] ←MBR ; copying the lower byte (20) into W temporary register
PC← PC+1
T11: MAR← [WZ] ; copying 16 bit data used for address
T12: MBR← [MAR] ;copying data from [2000]
T13: IR ←MBR ; [A]← [2000]

Instruction cycle, machine cycle and T states

Machine cycle: Machine cycle is the time required to transfer data to or from memory or I/O devices.It is defined as the time required to complete one operation of accessing memory , i/p, o/p or acknowledging and external request. This cycle may consists of 3 to 6 T states.

T-states: Each machine cycle consists of many clock periods/cycles, called T-states. Each and every operation inside the microprocessor is under the control of the clock cycle. The clock signal determines the time taken by the microprocessor to execute any instruction. The clock cycle shown in Fig. below has two edges (leading and trailing or lagging).T- State is defined as the time interval between 2-trailing or leading edges of the clock. It is sub division of the operation performed in one clock period. These sub division are internal states synchronized with system clock and each T states precisely equal to one clock period.

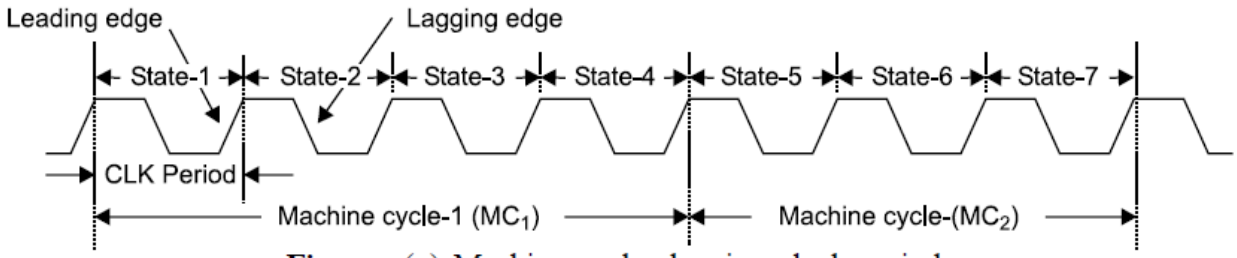


Fig. (*a*) Machine cycle showing clock periods

Instruction cycle: The necessary steps that the cpu carries out to fetch an instruction and necessary data from he memory and to execute it constitute and instruction cycle it is defined as the time required to complete the execution of an instruction. An instruction cycle consists of fetch cycle and executes cycle. In fetch cycle CPU fetches upcode from the memory . The necessary steps which are carried out to fetch an upcode from memory constitute a fetch cycle. The necessary steps which are carried out to get data if any from the memory and to perform the specific operation specified in an instruction constitute an execute cycle . The total time required to execute an instruction given by IC = Fc+ Ec The 8085 consists of 1-6 machine cycles or operations.

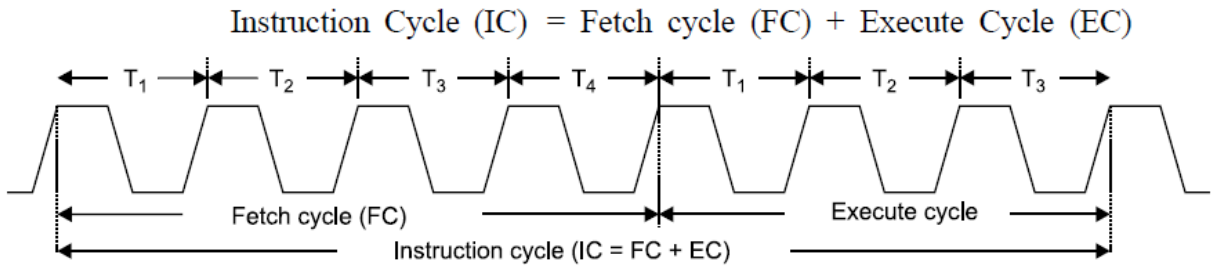
Fetch cycle: The first byte of an instruction is its op-code. The program counter keeps the memory address of the next instruction to be executed in the beginning of fetch cycle the content of the program counter, which is the address of the memory location where op-code is available, is send to the memory. The memory places the op-code on the data bus so as to transfer it to CPU. The entire process takes 3 clock cycles.

Execute cycle/Operation:- The op-code fetched from the memory goes to IR from the IR it goes to the decoder which decodes instruction. After the instruction is decoded execution begins. If the operand is in general purpose register, execution is performed immediately. i,e in one clock cycle. If an instruction is contains data or operand address, then CPU has to perform some read operations to get the desired data. In some instruction write operation is performed. In write cycle data are sent from the CPU to the memory of an o/p device. In some cases execute cycle may involve one or more read or write cycle or both.

Machine cycle of 8085 microprocessor: Op-code fetch, Memory read, memory write, I/O read, I/O write, Interrupt

The function of the microprocessor is divided into fetch and executes cycle of any instruction of a program. The program is nothing but number of instructions stored in the memory in sequence. In the normal process of operation, the microprocessor fetches (receives or reads) and executes one instruction at a time in the sequence until it executes the halt (HLT) instruction. Thus, an instruction cycle is defined as the time required fetching and executing an instruction. For executing any program, basically 2-steps are followed sequentially with the help of clocks **Fetch and Execute.**

The time taken by the μP in performing the fetch and execute operations are called fetch and execute cycle. Thus, sum of the fetch and execute cycle is called the instruction cycle as indicated in Fig



The 1st machine cycle (M1) of every instruction cycle is the op-code fetch cycle. In the op-code fetch cycle, the processor comes to know the nature of the instruction to be executed. The processor during (M1 cycle) puts the program counter contents on the address bus and reads the opcode of the instruction through read process. The T1, T2, and T3 clock cycles are used for the basic memory read operation and the T4 clock and beyond are used for its interpretation of the opcode.

Based on these interpretations, the μP comes to know the type of additional information/data needed for the execution of the instruction and accordingly proceeds further for 1 or 2-machine cycle of memory read and writes.The Op. code fetch cycle is of fixed duration (normally 4-states), whereas the instruction cycle is of variable duration depending on the length of the instruction. As an example, STA instruction, requires opcode fetch cycle, lower-order address fetch cycle and higher order fetch cycle and then the execute cycle.

One machine cycle is required each time the μP access I/O port or memory. A fetch opcode cycle is always 1-machine cycle, whereas, execute cycle may be of one or more machine cycle depending upon the length of the instruction. The instruction is decoded and translated into specific activities during the execution phase. Duration of machine cycle varies according to microprocessor architecture.

Opcode Fetch operation

A microprocessor either reads or writes to the memory or I/O devices. The time taken to read or write for any instruction must be known in terms of the μP clock. The 1st step in communicating between the microprocessor and memory is reading from the memory. This reading process is called opcode fetch. The 1st machine cycle of any instruction is always the fetch cycle that provides identification of the instruction to be executed. The process of opcode fetch operation requires minimum 4- clock cycles T1, T2, T3, and T4 and is the 1st machine cycle (M1) of every instruction.In order to differentiate between the data byte pertaining to an opcode or an address, the machine cycle takes help of the status signal IO/M, S1, and S0. The IO/M = 0 indicates memory operation and S1 = S0 = 1 indicates Opcode fetch operation.The opcode fetch machine cycle M1 consists of 4-states (T1, T2, T3, and T4). The 1st 3- states are used for fetching (transferring) the byte from the memory and the 4th-state is used to decode it.

Figs. below depict these first-steps for implementation of the instruction **ADI 05H**. Let us assume that the accumulator contains the result of previous operation i.e., **03H** and instruction is held at memory locations **2030H** and **2031H**.

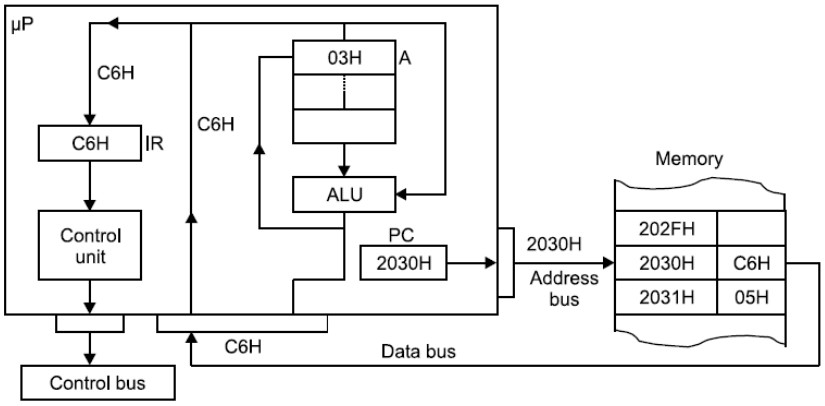


Fig. Instruction fetch : reads 1st byte (Opcode) in instruction register (IR)

The fetch part of the instruction is the same for every instruction. The control unit puts the contents of the program counter (PC) **2030H** on the address bus. The 1st byte (op-code **C6H** in this example) is passed to the instruction register.

Execute Operation:

In the execute cycle of the instruction, the control unit examines the op-code and as per interpretation further memory read or write operations are performed depending upon whether additional information/ data are required or not.

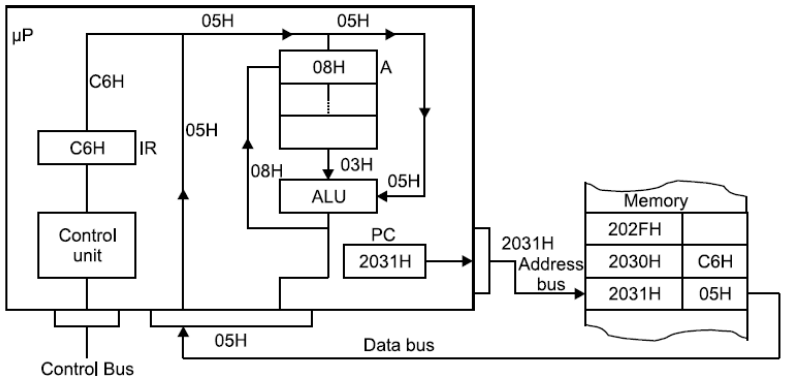


Fig. Instruction execute : reads 2nd byte from memory and adds to accumulator

In this case, the data **05H** from the memory is transferred through the data bus to the ALU. At the same time the control unit sends the contents of the accumulator (**03H**) to the ALU and performs the addition operation. The result of the addition operation **08H** is passed to the accumulator overriding the previous contents **03H**. On the completion of one instruction, the program counter is automatically incremented to point to the next memory location to execute the subsequent instruction.

Timing diagram of MOV, MVI, IN, OUT, LDA, STA

Timing diagram: The necessary steps which are carried in a machine cycle can represented graphically. Such graphical representation is called timing diagram. Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO / M , S1, and S0.

The 8085 microprocessor is designed to execute 74 different instruction types. Each instruction has two parts: operation code, known as op-code, and operand. To execute an instruction, the 8085 needs to

perform various operations such as Memory Read/Write and I/O Read/ Write. The total operations of a microprocessor can be classified into the following operations.

- Op-code Fetch
- Memory Read and Write
- I/O Read and Write
- Request Acknowledge

Timing Diagram for Opcode fetch

The following steps occur during Op-code Fetch Cycle

1. A low IO/M means microprocessor wants to communicate with memory. The Program counter (PC) places the 16 bit memory address on the address bus. At T1 high order address is placed at A 8-A 15 and lower order address is placed at AD 0-AD 7 and the ALE signal goes high, both s0 and s1 goes high; which identifies the op-code fetch cycle.

2. The control unit sends the control signal RD to enable the memory chip and remains active till two clock periods.

3. Now the op-code from memory location is placed on the multiplexed data bus. In this case 3E is placed on AD 7- AD 0 . The transition of RD indicates the end of this step.

4. The op-code byte is now placed on instruction decoder and the execute cycle is carried out.

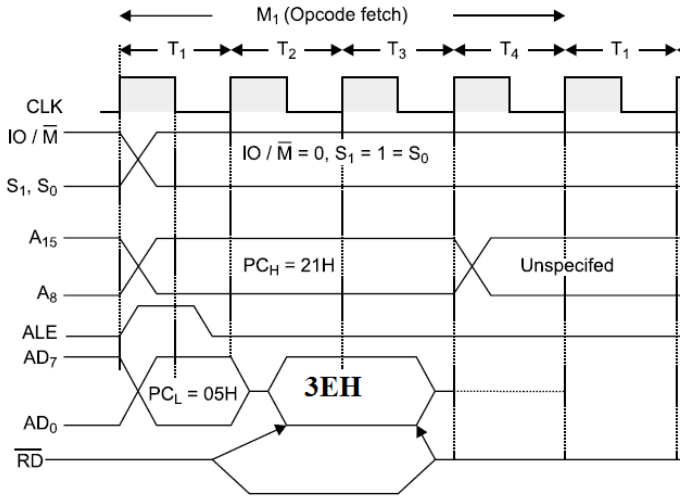


Fig. Opcode fetch

Example : Timing Diagram for MOV B,C.

Assume memory location as below.

Mnemonics	Machine code	Memory Locations
MVI B, 05H	06H	1000H
	05H	1001H

T1: The 1st clock of 1st machine cycle (M1) makes ALE high indicating address latch enabled which loads low-order address 00H on AD7 ⇔ AD0 and high-order address 10H simultaneously on A15 ⇔ A8. The address 00H is latched in T1.

T2: During T2 clock, the microprocessor issues RD control signal to enable the memory and memory places 41H from 1000H location on the data bus.

T3 : During T3, the 41H is placed in the instruction register and RD = 1 (high) disables signal. It means the memory is disabled in T3 clock cycle. The opcode cycle is completed by end of T3 clock cycle.

T4 : The opcode is decoded in T4 clock and the action as per 41H is taken accordingly. In other word, the content of C-register is copied in B-register.

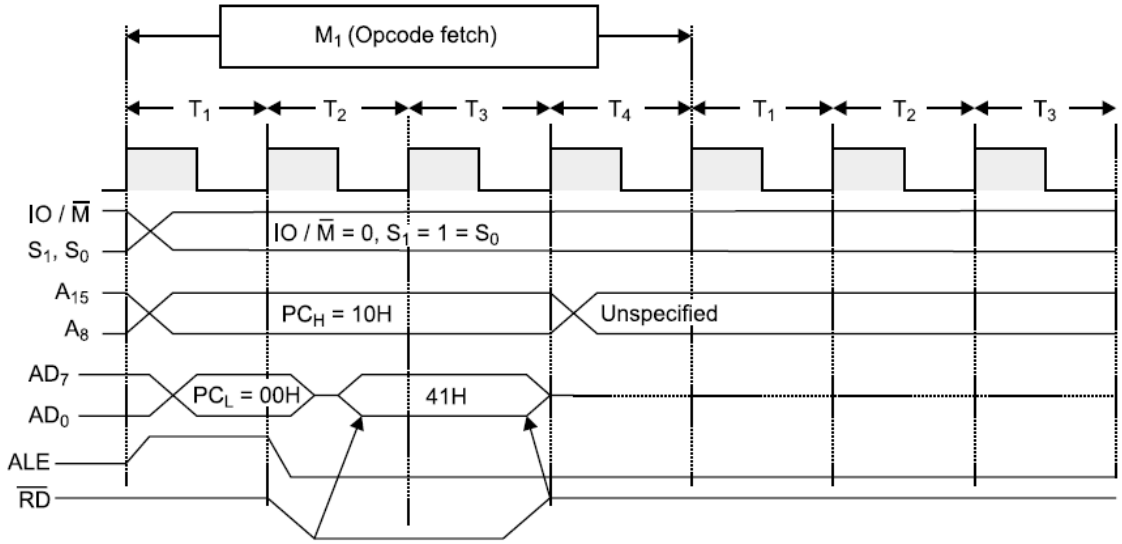


Fig. Opcode fetch (MOV B,C)

Timing Diagram for Memory Read /Write

Memory Read cycle: The Op-code fetch cycle is a memory read cycle. Thus, other memory read cycles are similar to the op-code fetch cycle. Let us take the same example as above.

EXAMPLE Timing diagram of MVI B, 05H

The instruction cycle of **MVI B, 05H** is shown below: The total cycle consist of **7 T** states and 2 machine cycles: **op-code fetch** and **memory read**. At the end of op-code fetch the PC is incremented thus the address is now **1001h** and instruction decoder has **06h**. Now the operand is to be read from the memory to register B. The second machine cycle is the memory read and consists of **3 T** states. The signal content of this cycle are similar to the first three T states of op-code fetch except the status signal, In this case **S0=0** and **S1=1**. The byte read in **T3** of memory read cycle will be copied into the accumulator in the same cycle.

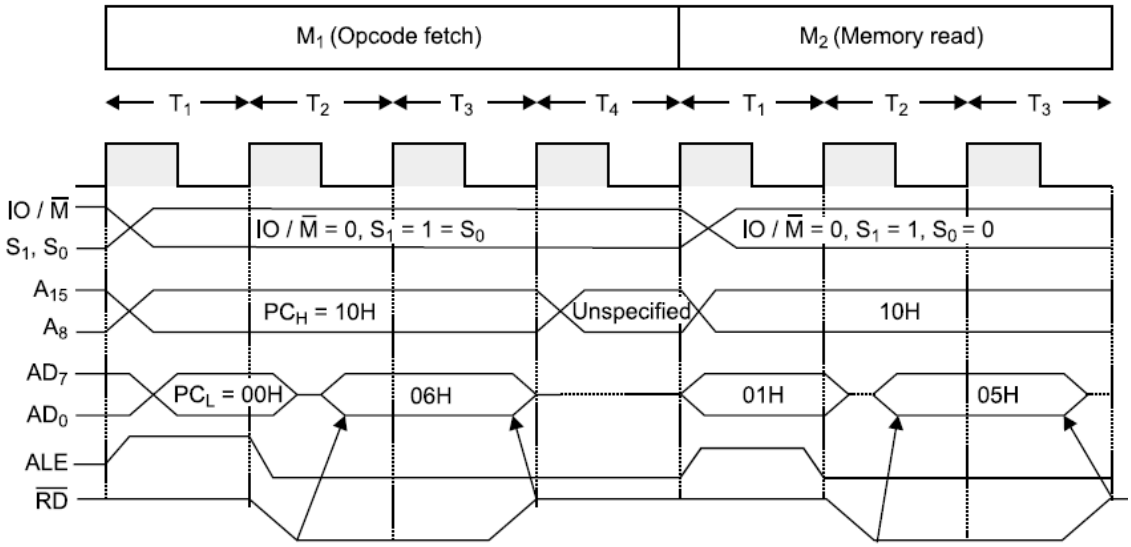


Fig. Timing diagram for MVI B,05H

The **MVI B, 05H** instruction requires 2-machine cycles (M1 and M2). M1 requires 4-states and M2 requires 3-states, total of 7-states as shown in Fig. above. Status signals IO/M, S1 and S0 specifies the 1st machine cycle as the op-code fetch. In T1-state, the high order address {10H} is placed on the bus A15 ⇔ A8 and low-order address {00H} on the bus AD7 ⇔ AD0 and ALE = 1. In T2 -state, the RD line goes low and the data 06 H from memory location 1000H are placed on the data bus. The fetch cycle becomes complete in T3-state. The instruction is decoded in the T4-state. During T4-state, the contents of the bus are unknown. With the change in the status signal, IO/M = 0, S1 = 1 and S0 = 0, the 2nd machine cycle is identified as the memory read. The address is 1001H and the data byte [05H] is fetched via the data bus. Both M1 and M2 perform memory read operation, but the M1 is called op-code fetch i.e., the 1st machine cycle of each instruction is identified as the op-code fetch cycle.

16 /32 Bit Architecture (Intel x86)

8086 is a 16 bit microprocessor. We will study it here .

Internal Architecture of 8086

8086 has two blocks BIU and EU. The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue. EU executes instructions from the instruction system byte queue. Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance. BIU contains Instruction queue, Segment registers, Instruction pointer, and Address adder. EU contains Control circuitry, Instruction decoder, ALU, Pointer, Index register and Flag register.

Bus Interface Unit:

It provides a full 16 bit bidirectional data bus and 20 bit address bus. The bus interface unit is responsible for performing all external bus operations.

Instruction fetches Instruction queuing, Operand fetch and storage, Address relocation and Bus control. The BIU uses a mechanism known as an instruction stream queue to implement pipeline architecture. This queue permits prefetching of up to six bytes of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction. These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle. After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.

The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory, these intervals of no bus activity, which may occur between bus cycles, are known as Idle state. If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle. The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address. For example**:** The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register. The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

Execution Unit

The Execution unit is responsible for decoding and executing all instructions. The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O and perform the operation specified by the instruction on the operands. During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue. When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

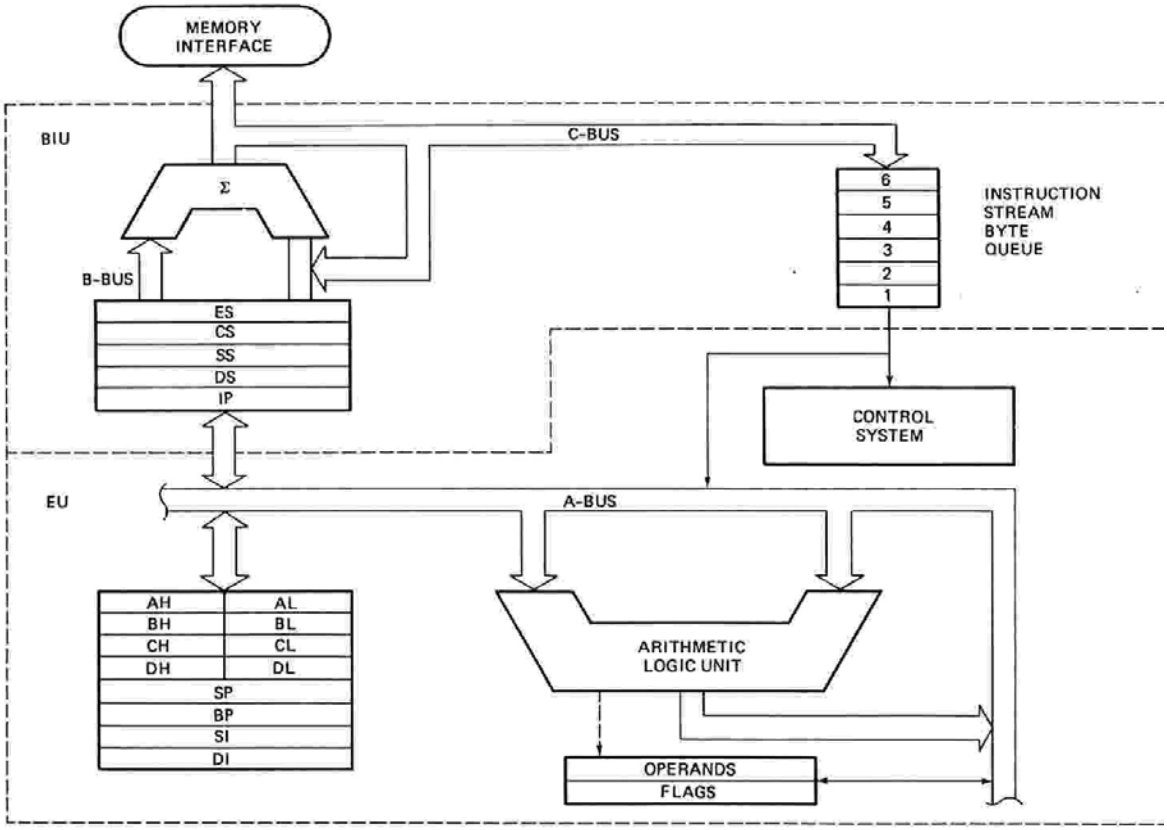


FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

Figure 1 : Block Diagram of 8086

Internal Registers of 8086

The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the status register, with 9 of bits implemented for status and control flags. Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It contains the starting address of a program's code segment. This segment address plus an offset value in the IP register indicates the address of an instruction to be fetched for execution.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. It permits the implementation of a stack in memory .By default; the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. It stores the starting address of a program's stack segment the SS register. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, and DX) and index register (SI, DI) is located in the data segment. It contains the starting address of a program's data segment .Instruction uses this address to locate data. This address plus an offset value in an instruction causes a reference to a specific byte location in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra Segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data.

By default, the processor assumes that the DI register points to the ES segment in string manipulation

Instructions. ES register can be changed directly using POP and LES instructions.. It is used by some

string operations to handle memory addressing.

Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,.

Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

The following registers are both general and index registers:

POINTER REGISTERS

The 16 bit Pointer Registers are IP,SP and BP respectively .SP and BP are located in EU whereas IP is located in BIU

Stack Pointer (SP) is a 16-bit register pointing to program stack. The 16 bit SP Register provides an offset value, which when associated with the SS register (SS: SP)

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. The 16 bit BP facilitates referencing parameters, which are data and addresses that a program passes via the stack. The processor combines the addresses in SS with the offset in BP.BP can also be combined with DI and SI as a base register for special addressing. BP register is usually used for based, based indexed or register indirect addressing.

INDEX REGISTERS

The 16 bit Index Registers are SI and DI

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions. SI is associated with the DS Register.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In this context, DI is associated with the ES register.

Other registers:

Instruction Pointer (IP) is a 16-bit register. The 16 bit IP Register contains the offset address of the next instruction that is to execute .IP is associated with CS register as (CS:IP) .For each instruction that executes, the processor changes the offset value in IP so that IP in effect directs each step of execution.

The Queue

While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instructions bytes for the following instructions. The BIU Stores pre-fetched bytes in First in First out register set called a queue. When the EU is ready for its next instruction, it simply reads the instruction bytes for the instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction bytes or bytes. Fetching the next instruction while the current instruction executes is called pipelining.

Flags are a 16-bit register containing 9 one bit flags.

Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.

Direction Flag (DF) - if it is set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.

Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.

Single-step Flag (TF) - if set then single-step interrupt will occur after the next instruction.

Sign Flag (SF) - set if the most significant bit of the result is set.

Zero Flag (ZF) - set if the result is zero.

Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.

Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.

Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

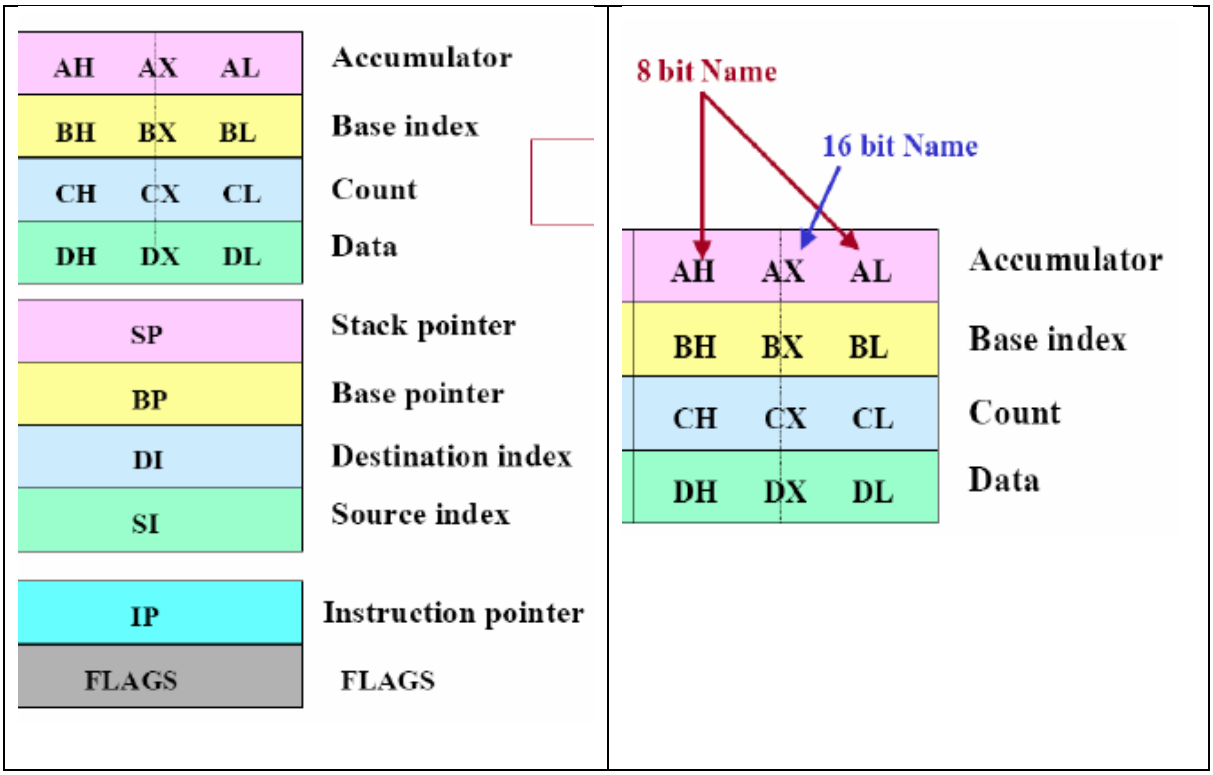


Figure 2 : Internal Registers of 8086

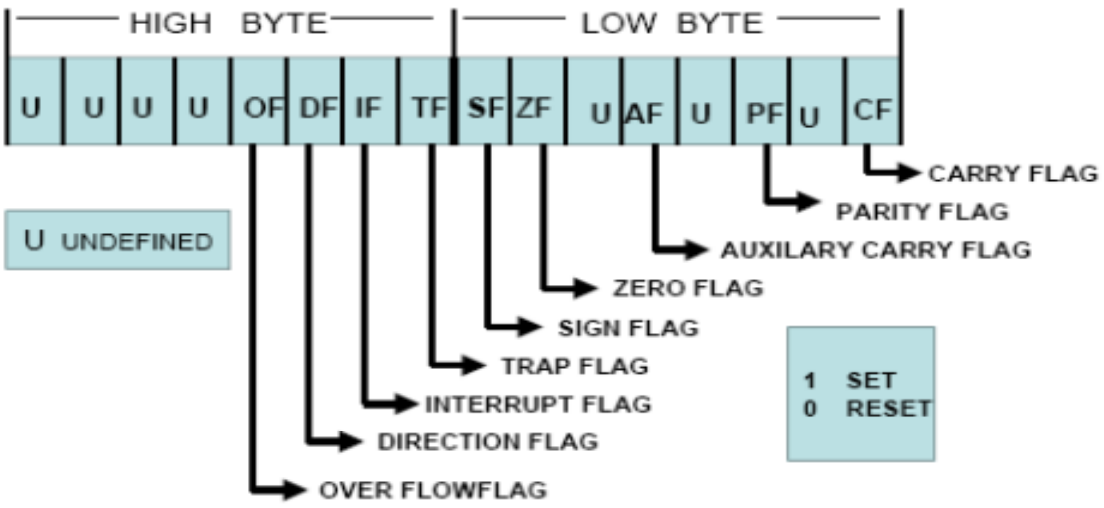


Figure 3 : Flag register of 8086

CONDITIONAL FLAGS

CF = CARRY FLAG [Set by Carry out of MSB]

PF = PARITY FLAG [Set if Result has even parity]

AF= AUXILIARY CARRY FLAG FOR BCD

ZF = ZERO FLAG [Set if Result is 0]

SF = SIGN FLAG [MSB of Result]

OF = OVERFLOW FLAG

CONTROL FLAG

TF = SINGLE STEP TRAP FLAG

IF = INTERRUPT ENABLE FLAG

DF = STRING DIRECTION FLAG

Memory Organization

INTRODUCTION

The processor provides a 20-bit address to memory, which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF (H). The

memory is logically divided into code, data, and extra and stacks segments of up to 64k bytes each, with each segment falling on 16-byte boundaries (see Figure). All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. All information in one segment type shares the same logical attributes (e.g. code or data). By structuring memory into re-locatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster and more structured.

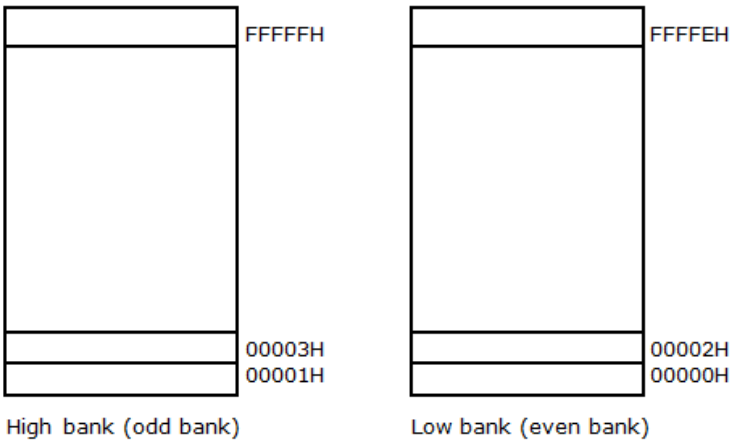


Figure: The Physical Memory System of 8086

The Intel 8086 is a 16 bit Microprocessor that is intended to be used as the CPU in a Microcomputer. The 8086 has a 20 bit address bus so it can address any one of 2^{20} or 1,048,576 memory locations. Each of the 1,048,576 memory address of the 8086 represents a byte-wide location.

16 bit word will be stored in two consecutive memory locations. If the first byte of a word is at an even address, the 8086 can read the entire word in one operation. If the first byte of a word is at an odd address, the 8086 will read the first byte with one bus cycle and the second byte with another bus cycle.

ACCESSING DATA IN MEMORY

An important point here is that an 8086 always stores the low byte of word in lower address and stores high byte of word in higher address. It follows little Endian format .

Low Byte – Low Address: High Byte – High Address (Little Endian)

- As an example, suppose we have the hexadecimal number 12345678.
- The big endian and little endian arrangements of the bytes are shown below.

Address →	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

CALCULATING PHYSICAL ADDRESS (Segment Arithmetic)

To perform segment arithmetic successfully, it helps to understand how the processor combines a 16-bit segment and a 16-bit offset to form a 20-bit linear address. In effect, the segment selects a 64K region of memory, and the offset selects the byte within that region. Here it is shown how it works:

1. The processor shifts the segment address to the left by four binary places, producing a 20-bit address ending in four zeros. This operation has the effect of multiplying the segment address by 16.

2. The processor adds this 20-bit segment address to the 16-bit offset address. The offset address is not shifted.

3. The processor uses the resulting 20-bit address, called the "physical address," to access an actual location in the 1-megabyte address space.

Figure below illustrates this process.

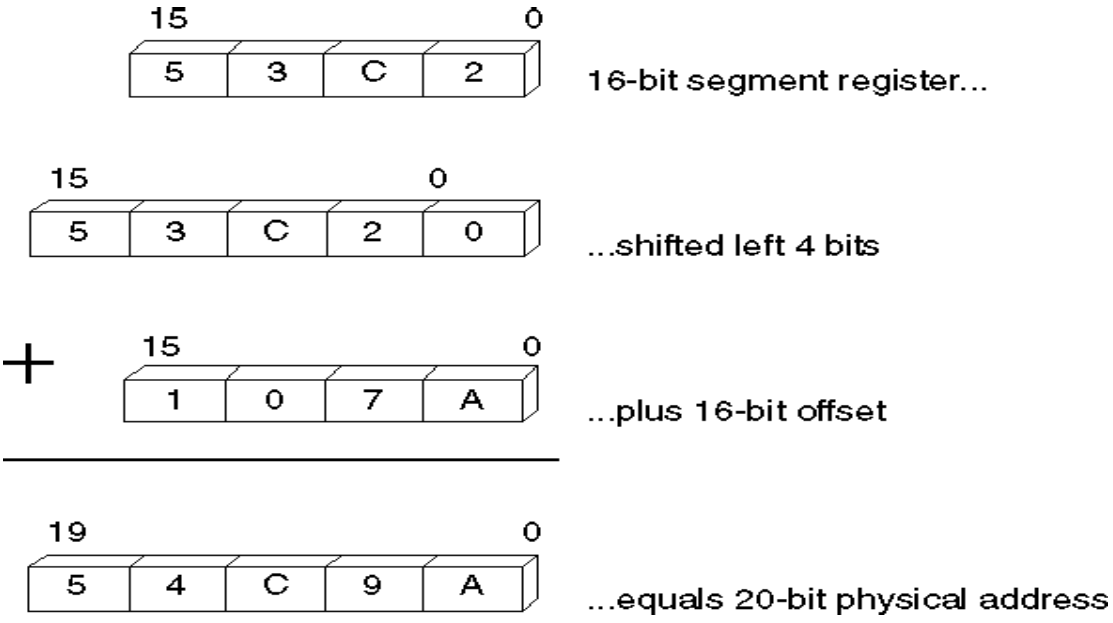


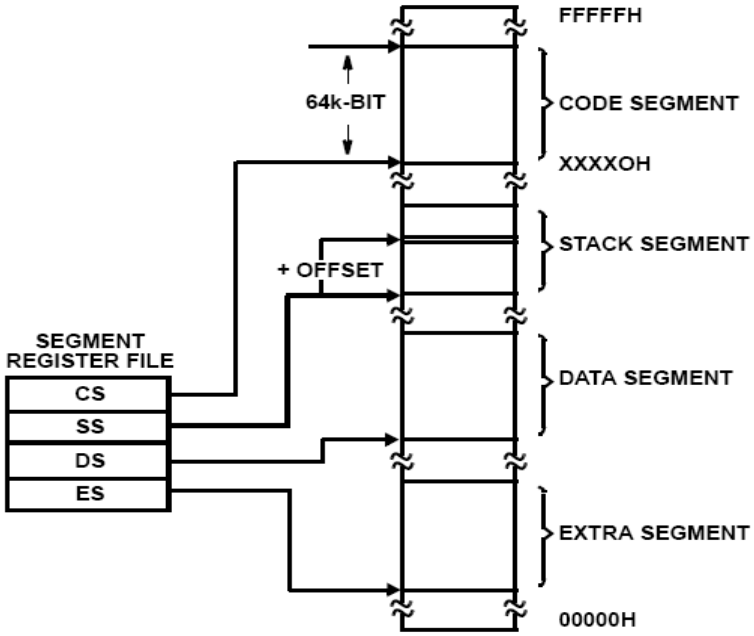
Figure: Calculating Physical Addresses

A 20-bit physical address may actually be specified by 4,096 equivalent *segment*:*offset* addresses. For example, the addresses 0000:F800, 0F00:0800, and 0F80:0000 all refer to the same physical address 0F800.

SEGMENTED MEMORY

The 8086 family of processors employs a segmented architecture - that is, each address is represented as a segment and an offset. Segmented addresses affect many aspects of assembly-language programming, especially addresses and pointers. Segmented architecture was originally designed to enable a 16-bit processor to access an address space larger than 64K. (The section "Segmented Addressing," later in this chapter, explains how the processor uses both the segment and offset to create addresses larger than 64K.) MS-DOS is an example of an operating system that uses segmented architecture on a 16-bit processor. With the advent of protected-mode processors such as the 80286, segmented architecture gained a second purpose. Segments can separate different blocks of code and data to protect them from undesirable interactions. Windows takes advantage of the protection features of the 16-bit segments on the 80286. Segmented architecture went through another significant change with the release of 32-bit processors, starting with the 80386. These processors are compatible with the older 16-bit processors, but allow flat model 32-bit offset values up to 4 gigabytes. Offset values of this magnitude remove the memory limitations of segmented architecture

The 8086 BIU sends out 20 bit address so it can address any of 2^{20} or 1,048,576 bytes in memory. However at any given time the 8086 works with only four 65536 bytes (64 Kbyte) segment within this 1,048,576 byte (1M Byte) Range .Four segments are: Code Segment, Stack Segment, Data Segment and Extra Segment .Four segment registers in BIU are used to hold the upper 16 bits of the starting address of 4 memory segments that the 8086 is working with at a particular time. The 4 segment registers are code segment register (CS), stack segment register (SS), data segment register (DS) and the extra segment register (ES). For small programs which do not need all 64 Kbytes in each segment can overlap. For example, the code segment holds the upper 16 bits of the starting address for the segment from which the BIU is currently fetching instruction code bytes. The BIU always inserts zero for the lowest 4 bits of the 20 bit starting address. If the code segment register contains 348A H then the code segment will start at address 348A0 H .A 64 Kbytes segment can be located anywhere within the 1 MByte address space, but the segment will always start at an address with zeros in the lowest 4 bits.



ADVANTAGES OF SEGMENTATION [SEGMENT: OFFSET SCHEME]

Intel designed the 8086 family devices to access memory using the segment: offset approach rather than accessing memory directly with 20 bit.

The advantages are listed below.

- The segment: offset scheme requires only a 16 bit number to represent the base address for a segment and only a 16 bit offset to access any location in a segment. This means that 8086 has to manipulate and store only 16 bit quantities instead of 20 bit quantities.
- This makes easier interface with 8 and 16 bit wide memory boards and with 16 bit registers in the 8086.
- It allows programs to be relocated in memory system. A re-locatable program is one that can be placed in any area of memory and executed without change.
- It allows programs written to function in the real mode to operate in protected mode.
- Segmentation also makes easy to keep user's program and data separate from one another and segmentation makes it easy to switch from one user's program to another user's program.

DISADVANTAGE OF SEGMENT: OFFSET APPROACH

The segment: offset scheme needs complex hardware and software i.e. overhead of complex memory management.

Addressing Modes

The different ways in which a processor can access data are referred to as its addressing modes. The addressing mode is indicated in the instruction itself. The various addressing modes are

1.Register Addressing Mode

2.Immediate Addressing Mode

3.Direct Addressing Mode

4.Register Indirect Addressing Mode

5.Base plus Index Addressing Mode

6.Register Relative Addressing Mode

7.Base Relative Plus Index Addressing Mode

1. **REGISTER ADDRESSING MODE**
   It is the most common form of data addressing. It transfers a copy of a byte/word from source register to destination register. It is carried out with 8 bit registers AH,AL,BH,BL,CH,CL,DH & DL or with 16 bit registers AX,BX,CX,DX,SP,BP,SI and DI.It is important to use registers of same size.

   EXAMPLES

   MOV AH,BH : Copies BH into AH

   MOV ES,DS : Copies DS into ES

   MOV AX,CX : Copies CX into AX

2. **IMMEDIATE ADDRESSING MODE**
   The term immediate implies that the data immediately follow the hexadecimal opcode in the memory. The immediate data are constant data. It transfers the source immediate byte/word of data in destination register or memory location.

   EXAMPLES

   MOV AL,77 : Copies 77 into AL

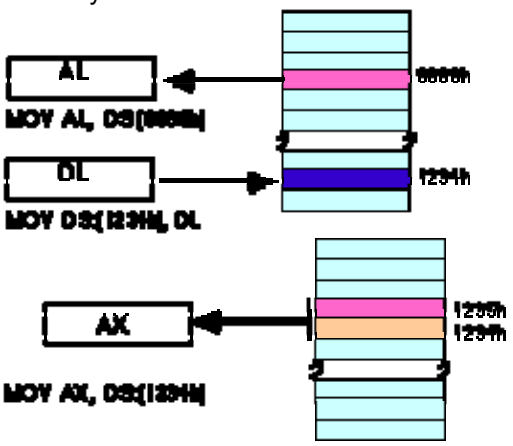
   MOV AX,6234H : Copies 6234H into AX

   MOV CL,10000011B : Copies 100000011 binary value into CL 27

3. **DIRECT ADDRESSING MODE**

   The most common addressing mode is the (displacement-only) direct addressing mode. The direct addressing mode consists of a 16 bit constant that specifies the address of the target location. The instruction mov al,ds:[8088h] loads the al register with a copy of the byte at memory

location 8088h. Likewise, the instruction mov ds:[1234h],dl stores the value in the dl register to memory location 1234h:



The direct addressing mode is perfect for accessing simple variables.By default, all displacement-only values provide offsets into the data segment. If we want to provide an offset into a different segment, we must use a segment override prefix before the address. For example, to access location 1234h in the extra segment (es) we would use an instruction of the form mov ax,es:[1234h]. Likewise, to access this location in the code segment we would use the instruction mov ax, cs:[1234h]. The ds: prefix in the previous examples is not a segment override. The CPU uses the data segment register by default. These specific examples require ds: because of MASM's syntactical limitations. If we use MOV AL, [1234H] then it uses DS segment by default.

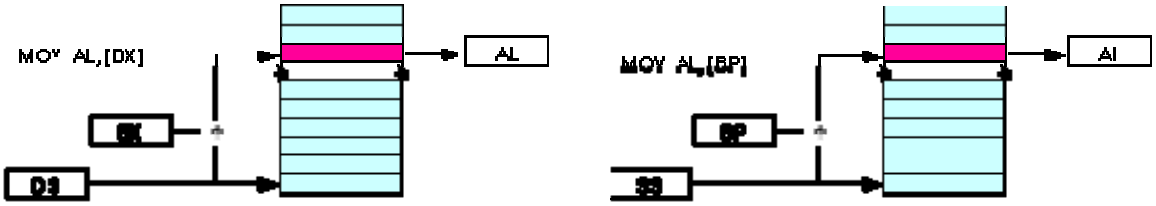
EXAMPLES

MOV AL,[1234H] ; Copies the byte content of data segment memory location 11234H into AL.

MOV AL, NUMBER ; Copies the byte content of data segment memory location NUMBER into AL.

4. REGISTER INDIRECT ADDRESSING MODE

In 80x86, memory can be accessed indirectly through a register using the register indirect addressing modes. Register Indirect Addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI. The Index and Base registers are used to specify the address of data. It transfers byte/word between a register and a memory location addressed by an index or base registers. The symbol [ ] denote indirect addressing. The data segment is used by default with register indirect addressing or any other addressing mode that uses BX, DI or SI to address memory. If BP registers addresses memory, the stack segment is used by default. Intel refers to [bx] and [bp] as base addressing modes and bx and bp as base registers (in fact, bp stands for base pointer). Intel refers to the [si] and [di] addressing modes as indexed addressing modes (si stands for source index, di stands for destination index). However, these addressing modes are functionally equivalent.



EXAMPLES

MOV CX, [BX]: Copies the word contents of the data segment memory location addressed by BX into CX.

MOV [DI], BH: Copies BH into the data segment memory location addressed by DI.

MOV [DI], [BX]: Memory to Memory moves are not allowed except with string instructions.

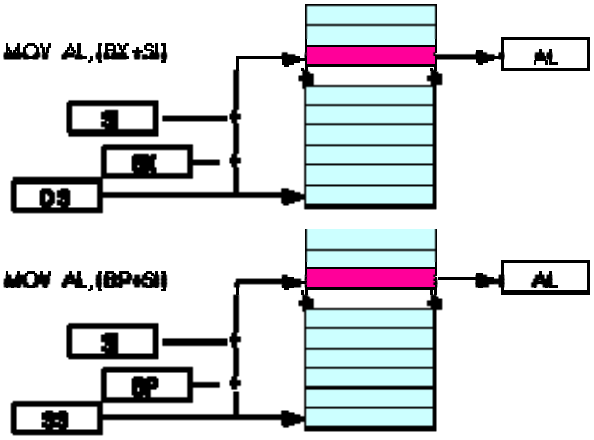
5. **BASE PLUS INDEX ADDRESSING MODE**

The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). Suppose that bx contains 1000h and si contains 880h. Then the instruction

Mov al,[bx][si] or mov al,[bx][si]

would load al from location DS:1880h.

Likewise, if bp contains 1598h and di contains 1004, **mov ax,[bp+di]** will load the 16 bits in ax from locations SS:259C and SS:259D. The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.



EXAMPLES

MOV CX,[BX+DI] : Copies the word contents of the data segment memory location addressed by BX plus DI into CX.

MOV CH,[BP+SI] : Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH.

6. **REGISTER RELATIVE ADDRESSING MODE** (Indexed Addressing Modes)

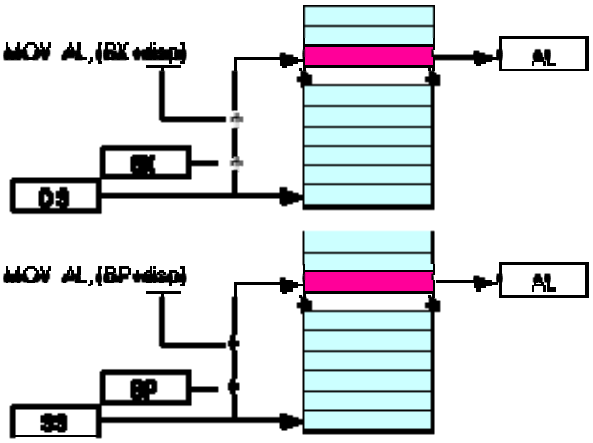
The data in a segment of memory are addressed by adding the displacement to the content of base or an index register (BP, BX, DI or SI). It transfers a byte/word between a register and the

memory location addressed by an index or base register plus a displacement. The offsets generated by these addressing modes are the sum of the constant and the specified register.

If **bx** contains **1000h**, then the instruction **mov cl,20h[bx]** will load **cl** from memory location **ds:1020h**. Likewise, if **bp** contains **2020h**, **mov dh,1000h[bp]** will load dh from location **ss:3020**.

The addressing modes involving bx, si, and di all use the data segment, the **disp[bp]** addressing mode uses the stack segment by default. As with the register indirect addressing modes, we can use the segment override prefixes to specify a different segment:

mov al, ss:disp[bx]



EXAMPLES

MOV ARRAY [SI], BL: Copies BL into the data segment memory location addressed by ARRAY plus SI.

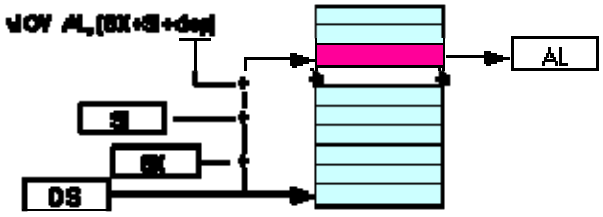
MOV LIST [SI+2], CL: Copies CL into the data segment memory location addressed by sum of LIST, SI and 2.

7. BASE RELATIVE PLUS INDEX ADDRESSING MODE

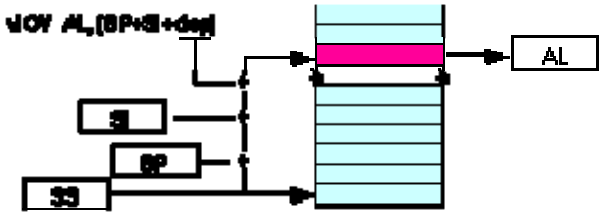
The base relative plus index addressing mode is similar to the base plus index addressing mode but it adds a displacement to form a memory address. These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant.

Example:

mov al, disp[bx][si]



mov al, [bp][di][disp]



Suppose **bp** contains **1000h, bx** contains **2000h, si** contains **120h**, and **di** contains **5**. Then

mov al,10h[bx+si] loads **al** from address **DS:2130**;

mov ch,125h[bp+di] loads **ch** from location **SS:112A**; and

mov bx,cs:2[bx][di] loads **bx** from location **CS:2007**.