INTRODUCTION TO MICROPROCESSORS

(Dr. Prince P. R., Govt. Arts College, Thiruvananthapuram)

We will start with some fundamental ideas, which could be helpful in dealing with microprocessors.

What is a number system?

In simple words, it is a method to count and represent numbers. The *decimal* number system, which we use, has got 10 symbols (from 0 to 9). Using these 10 symbols, we are able to represent any numbers and we say the base (or radix) of the number system is 10.

Binary number system uses only two symbols and its base is 2. This system is very much useful in dealing with digital circuits, which also have only two states- ON and OFF. A binary digit is called a bit, a group of four bits is called a *nibble* and a group of 8 bits is called a *byte*. 1 kilobyte (kB) means $2^{10} = 1024$ bytes.

Hexadecimal number system uses 16 symbols (base =16). The symbols are the ten digits from 0 to 9 and the six alphabets from A to F. This system is useful in microprocessor programming. A hexadecimal (or hexa) number is usually represented by a 'H' at the end. For example. The numbers 249H, 7AF58H are hexa numbers.

The following equivalent numbers in different number systems may please be noted.

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A

Decimal	Binary	Hexadecimal
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
21	10101	15

Hexadecimal to binary conversion

Hexadecimal numbers can be converted into the corresponding binary numbers by replacing each hexadecimal digit by the corresponding 4 bit binaries (use the following table).

Hexadecimal	4 bit Binary		Hexadecimal	4 bit Binary
0	0000		8	1000
1	0001		9	1001
2	0010		A	1010
3	0011		B	1011
4	0100		C	1100
5	0101		D	1101
6	0110		E	1110
7	0111		F	1111

Eg. $56AD\ H = (0101\ 0110\ 1010\ 1101)_2$
$2000\ H = (0010\ 0000\ 0000\ 0000)_2$

Similarly a binary number can be converted to hexadecimal number by replacing 4 bit binaries by the corresponding hexadecimal digit.

Eg. $(11100011)_2 = (1110\ 0011)_2 = D3\ H$
$(10101)_2 = (0001\ 0101)_2 = 15\ H$

**Reader is advised to have a thorough understanding on binary and hexadecimal arithmetic too.*

What is a computer?

A computer, essentially, is a computing and controlling device. Three components of a computer are

1. Central Processing Unit (CPU)
2. Input/Output devices (I/O)
3. Memory

CPU, sometimes called the brain of the computer, is responsible for the overall computations, control and decision making. A CPU of a modern computer is an assembly of complicated electronic circuits. If all these circuitry is designed and fabricated on a single IC, it is called a *microprocessor*. And a computer whose CPU is a microprocessor is called a *microcomputer*.

The CPU collects information using the input devices (key board, mouse, etc.) and gives information out using output devices (monitor, LED/LCD display, printer, etc.)

Memory is the place where the instructions (given by us) to the CPU as well as the data are stored. Memories can be broadly classified into

1. Primary memory (main memory)
2. Secondary memory

Primary memory is basically a semiconductor memory (RAM and ROM). Secondary memory is for bulk storage of information (magnetic memories like hard disk, optical memories like CD, DVD etc.)

Note: 1. The communication of the CPU is directly with the main memory. Any information stored in the secondary memory is first to be loaded to the main memory (RAM) in order to be accessed by the CPU.

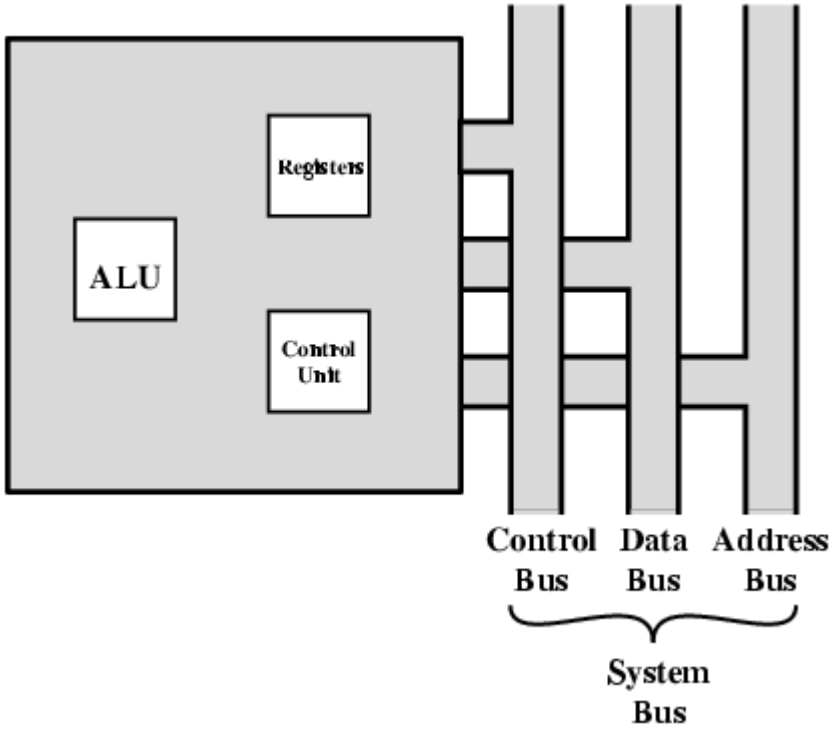
2. The microprocessor kits, we use in the laboratories are nothing but microcomputers, called single board microcomputer. In these, the CPU (the microprocessor 8085/8086), memory chips, I/O devices and other ICs along with the connecting buses are all arranged on a single board.

Microprocessor

As we have seen, a microprocessor is a CPU, in the form of an IC. It is called a *micro*processor, probably due to two reasons: One is that initially, the size of transistors used in such IC s were of *micro*metre dimensions. Other is that the basic time scale of processing by such a processor was in the order of *micro*seconds. All our PCs are microcomputers using microprocessors. Microprocessors (abbreviated as μPs) processes anything in the form of binary numbers. The number of bits a μP can process at a time is called the wordlength of the processor.

Eg. Intel 8085, Zilog 80, etc. are 8-bit processors (wordlength = 8bits)
Intel 8086 and 8088, Zilog 8000, are 16 bit processors
Intel 80386, 80486, pentium, etc, are 32 bit processors

A microprocessor generally can be divided in to three sections, viz. the Arithmetic and Logic Unit (ALU), the Control unit and the Register array. Internal buses internally connect these sections. The microprocessor communicates with other I/O and memory devices in a computer or a system using three buses viz. the address, the data and the control buses.



ALU performs all arithmetic and logical operations in the processor. Registers are used to store the information collected from the external memory (memory external to μP- that is the main memory of the computer). They are also used for the storage of data and results of a programme. There are some special purpose registers too. Registers can be understood as the internal memory elements of the μP. Timing and sending signals to the I/O and memory are the responsibility of the control unit.

Microprocessor programming

A microprocessor is an electronic machine, which can understand only two states ON and OFF, HIGH and LOW voltages, etc. Hence, our communication with a microprocessor must be in this language. So we are supposed to use binary numbers in communicating with the μP. A program written in binary number codes is called a *machine language* programme. But since such a programming is error prone and very difficult, we gave abbreviated names, called *mnemonics,* for such binary instruction codes. The programming using such mnemonics is called *assembly language programming.* Working with assembly language was also not very easy, and people developed user friendly programming languages with easily understandable English like instructions. They are known as *high level languages* (eg. FORTRAN,C, C++, etc.). It must be noted that whatever languages, one writes the program,

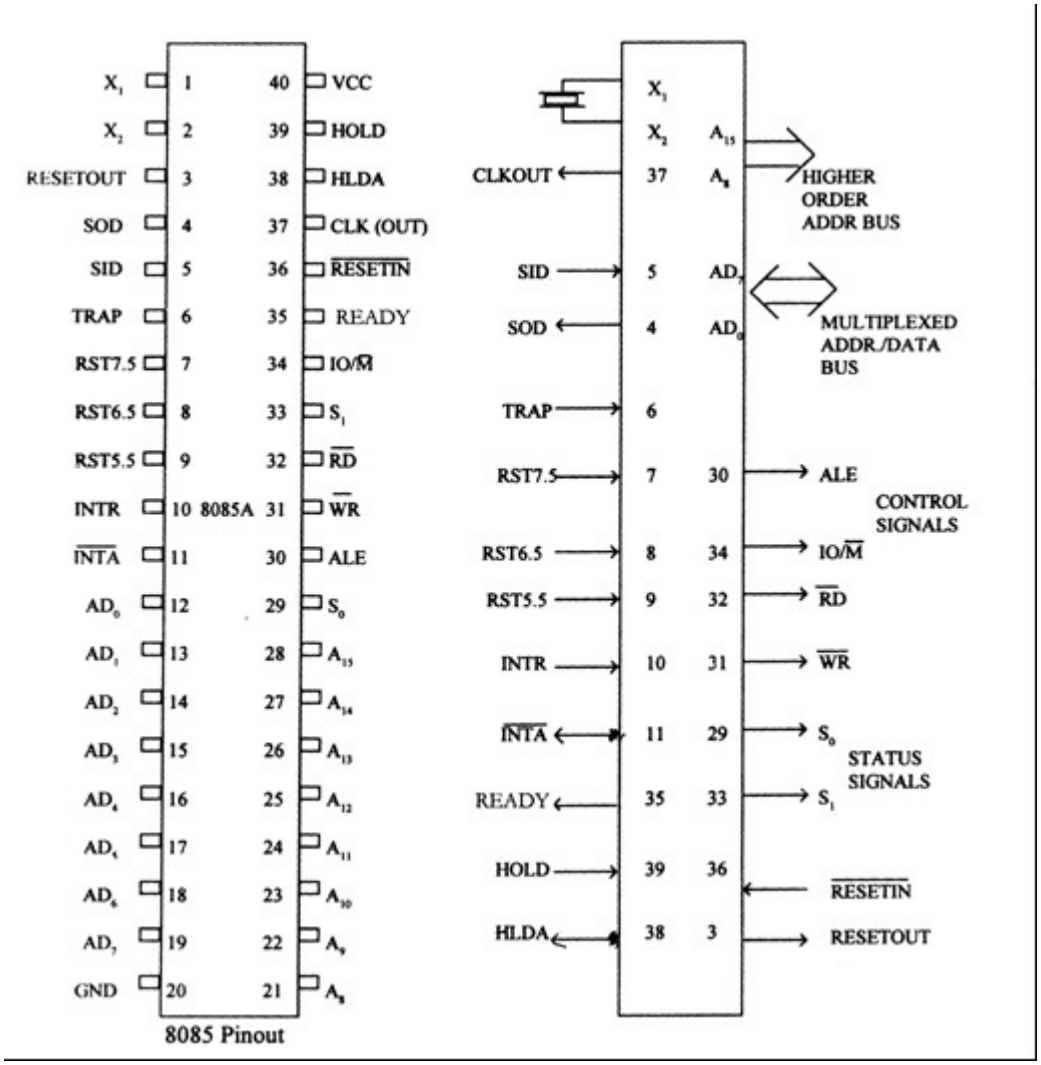
at last every thing must be converted to the machine language form, which the µP understands. An *assembler* converts assembly language program to machine language program. *Compilers and interpreters* convert high level language to machine language.

Note: When we use a microprocessor kit, without an assembler, the following is the sequence of reaching the machine language.

1. *The user designs the logic and writes the program in assembly language.*
2. *The user converts each instruction (in the form of mnemonics) to the corresponding hexadecimal codes with the help of tables called Instruction set.*
3. *A code converting program in the µP kit converts the hexa codes into the binary numbers (machine language).*

More about the microprocessor can be understood through a discussion on a typical microprocessor Intel 8085.

INTEL 8085

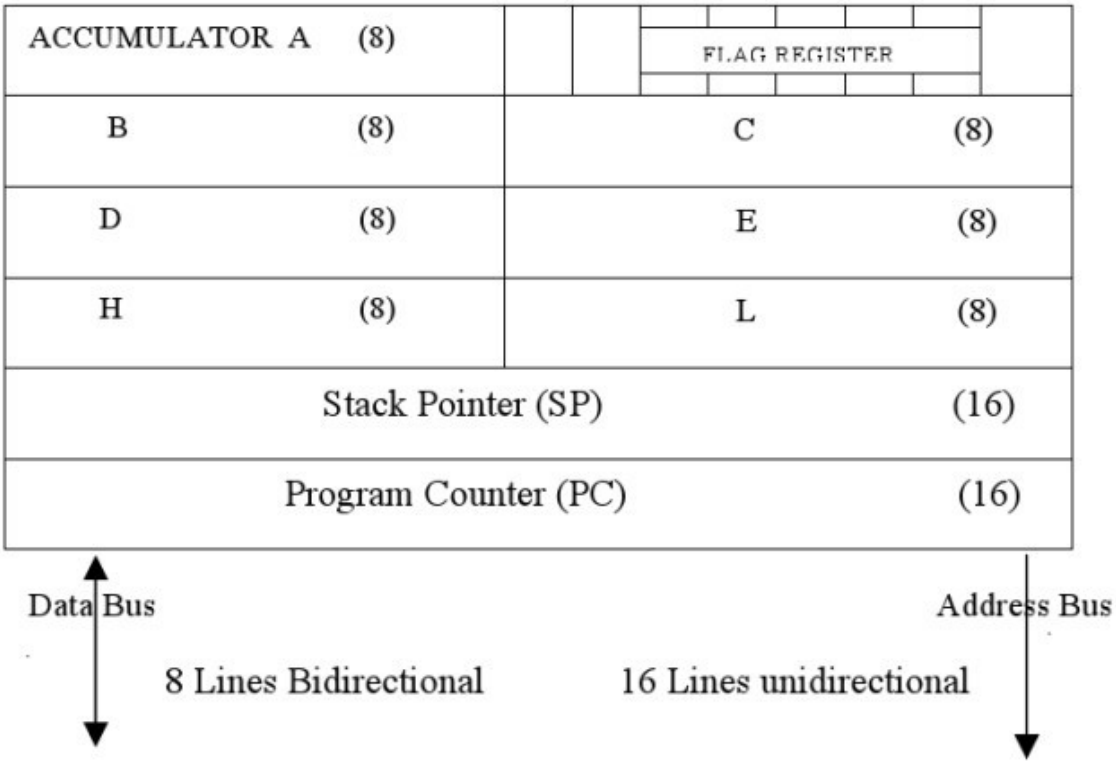


It is an 8 – bit microprocessor in the form of a 40 pin IC. 8085 has an 8 bit data bus to carry information between the μP and the peripherals (I/O and memory). It has got a 16 bit address bus, which means that the processor can address 2^{16} = 64 kilo locations. If all those locations are 8 bit (1 byte) memory elements in a memory chip, we can say that the processor is capable of addressing 64 kilobytes (kB) of information.

Internal architecture

Like any processor, 8085 also has an ALU, register array and control unit

Register array:



8085 has got seven 8-bit registers. They are *A (accumulator)*, B, C, D, E, H and L. Accumulator is a special register. There are many instructions, which operate on A by default. Many instructions make results in accumulator. B, C, D, E, H, L are called general-purpose registers. BC, DE and HL can be used as 16 bit registers too.

There are two 16 bit registers too. They are Programme counter (PC) and Stack pointer (SP). While programme counter is for keeping track of the programme execution sequence, stack pointer is for tracking some special memory locations called stack.

Flag register is an 8 bit register, whose 5 bits (5 flipflops) alone are used to reflect the different conditions of the result of execution of an instruction. There are 5 flags viz.

1. S- sign flag: reflects the sign of the result. Sets (to 1), when the result is negative (that is when the most significant bit of the result is 1) and resets (to 0), when the result is positive (that is when the most significant bit of the result is 0)
2. Z- Zero flag: reflects whether the result generated is a 00H (or binary 0000 0000). Sets to 1 when the result is 00H and resets to 0, when the result is not 00H.

3. AC- Auxiliary Carry flag: sets to 1, if a carry is generated in the D_3 position and carried over to D_4 position. (An 8 bit binary result is represented by $D_7D_6D_5D_4D_3D_2D_1D_0$).
4. P-Parity flag: sets to 1, when the result has an even parity and resets to 0, when the result has an odd parity
5. CY- Carry flag- sets to 1 if a carry =1 is generated in a an operation. That is when a D_8 =1 is generated. Resets to 0, if there is no carry or when D_8 = 0

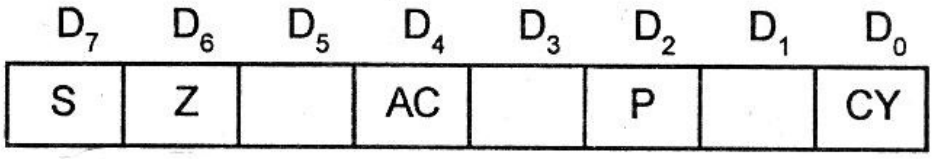


Fig 1.7 : Bit positions of various flags in the flag register of 8085

Note: Data transfer instructions do not affect the flags.

Note: *While A, B, C, etc. represent registers* [A], [B], [C], *etc. represents the data, which is present in the registers*

Instruction Set of 8085

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called *Instruction Set.* 8085 has **246** instructions.

Classification of Instruction Set

There are 5 categories of 8085 instruction sets.

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions

Data Transfer Instructions

These instructions move data between registers, or between memory and registers. By *moving we mean copying*. That is the contents of source are not modified.

Eg. MOV A, C copies the contents of register C to register A
MOV D, M copies the content of the memory to the register D.

(*What is M? Ans:* M is the memory location in the RAM. The address of the location is specified as the contents of H and L registers. This means if [H] = 21H AND [L] = 00H, then M means the location with address 2100H. And MOV D, M copies the contents of the memory location 2100H to register D)

MVI E, 39H stores data 39H in the register E. (Move immediate)
MVI M, F5H stores data F5H in the memory location M

LDA 2300H copies the contents of memory location 2300H to A (Load Accumulator)
STA 2325H copies the contents of A to memory location 2325H (Store Accumulator)

LXI H 2000H copies the 16 bit data 2000H to the register pair H and L
(** If HL pair contains 2000H then M means the memory location 2000H)

*** There are many more data transfer instructions. For a detailed list, please see the full Instruction set (with meaning) which is available in all related textbooks.*

Arithmetic Instructions

These instructions perform the operations like:

- Addition
- Subtraction
- Increment
- Decrement

Addition

Eg. ADD B adds the contents of register B to the contents of the accumulator and the result (sum) is stored in the accumulator. {This operation can be represented as [A] = [A] + [B]}

ADD M adds the contents of memory location to the contents of the accumulator and the result (sum) is stored in the accumulator {[A]= [A]+[M]}

ADI 89H adds the number 89H to the to the contents of the accumulator. (Add immediate) {[A]= [A]+89H}

ADC C adds the contents of register C and the content of the CY flag to the contents of the accumulator and the result is stored in the accumulator. {[A]= [A]+[C]+[CY]}

DAD D is a 16-bit addition instruction, which means [H and L] = [H and L] + [D and E]

Subtraction

Eg. SUB E means [A] = [A] - [E]
SUI 67H means [A] = [A] – 67H (Subtract immediate)

Increment/Decrement.

Eg. INR E means [E] = [E] + 1
DCR L means [L] = [L] – 1
INX H means [H and L] = [H and L] + 1 (16 bit increment)
DCX B means [B and C] = [B and C] - 1 (16 bit decrement)

Logical Instructions

These instructions perform logical operations like

- AND
- OR
- XOR
- Rotate
- Compare
- Complement

AND, OR, XOR

Any 8-bit data, or the contents of register, or memory location can logically have

- AND operation
- OR operation
- XOR operation

with the contents of accumulator. The result is stored in accumulator.
Eg. ANA D, ANI 87H, ORA M, ORI 5FH, XRA A, XRI 8EH, etc.

Rotate

Each bit in the accumulator can be shifted either left or right to the next position.
Eg. RLC, RRC, RAL, RAR

Compare

Any 8-bit data, or the contents of register, or memory location can be compared for:

- Equality
- Greater Than
- Less Than

with the contents of accumulator. The result is reflected in status flags.
Eg. CMP D, CMP M, CPI 63H, etc.

Complement

The contents of accumulator can be complemented. Each 0 is replaced by 1 and each 1 is replaced by 0.
Eg. CMA

Branching Instructions

The branching instruction alters the normal sequential flow of program execution. These instructions alter the sequence either unconditionally or conditionally.

Eg**.** JMP 2100 – the program execution sequence is *unconditionally* transferred to the address location 2100.

The program sequence can be transferred conditionally to the memory location specified by the 16-bit address gibased on the specified flag **Example:** JZ 2034 H.

Eg. JC 2105H - Jump if Carry CY = 1 to the memory location 2105H
JNC 2105H - Jump if No Carry CY = 0 to the memory location 2105H
JP Jump if Positive S = 0
JM Jump if Minus S = 1
JZ Jump if Zero Z = 1

JNZ Jump if No Zero Z = 0
JPE Jump if Parity Even P = 1
JPO Jump if Parity Odd P = 0

CALL and RET instructions are also branching instructions

Control Instructions

The control instructions control the operation of microprocessor.

Eg. NOP - No operation. The instruction is fetched and decoded but no operation is executed.

HLT - The CPU finishes executing the current instruction and *halts* any further execution.

Instruction format

An instruction is a command to a microprocessor to perform a given task on specified data. It has two parts: 1. the *opcode* (operation code) , which is the task to be performed and 2. the *operand*, which is the data to be operated on.

Instruction = opcode + operand

For eg. In the instruction MVI A, 9FH, MVI is the opcode and A and 9FH are the operands. (the register A to which the data 9FH is moved, is also treated as operand). The machine language for MVI A (move immediate to reg. A) is 0011 1110 whose hexadecimal equivalent is 3E. Therefore the complete hexadecimal code (machine code) corresponding to MVI A, 98H is 3E, 98.

The hexadecimal code (machine code) of each of the 246 instructions of 8085 is available as tables (instruction set).

Note: the abbreviations MVI. LXI, MOV, ADD, CMA, etc. are called mnemonics

Addressing modes.

The methods by which an operand is specified in the instruction are called addressing modes.

There are 5 different addressing modes in 8085. They are

1. *Immediate addressing:* The operand is immediately present in the instruction. (eg. MVI D, 08H)
2. *Register addressing*: The data (operand) is inside a register. (eg. MOV A,L)
3. *Direct addressing*: The exact data location (other than register) of the data is given. (eg. LDA 2500)
4. *Indirect addressing:* The exact data location is not directly given, Instead the location is specified somewhere else.(Eg. ADD M. The location of data is in a memory location M. That location address in actually specified in the HL register pair)
5. *Implicit addressing*: Operand need not be specified. (eg. CMA –operand is in the accumulator, RLC- operand is in the accumulator)

Program execution using microprocessor kit

When a microprocessor kit (without assembler) is used for program execution, the following steps are to be performed.

1. Identify the RAM location of your kit. (Say it is from 2000H to 3FFFH)
2. Write the program using mnemonics. The address of the first instruction and the whole program must be in the RAM area
3. Code the program to hexadecimal codes using the instruction set.
4. Enter the program using hexadecimal key board to the specified memory (RAM) locations. (For this, proper control keys like SUB, Exmmem, NEXT,etc. are to be used. These keys may vary with different make and models)
5. Enter the data in memory locations, if the program requires that.
6. Execute the program from the first memory location. (use appropriate control keys like GO, Exec, etc.)
7. If the program is logically correct, it will execute. If not, there will be error message; correct the errors. And execute again.
8. Check the result (using appropriate control keys)

Note: the program we write is entered into the memory (RAM) and is stored there. When we give the command for execute, the following is the course of action.

1. the microprocessor *fetches* (brings) the first instruction from the specified first memory location
2. the fetched instruction is *decoded* (meaning understood by comparing with the whole list of instructions and the corresponding actions)
3. the action as instructed by the instruction is *executed*.
4. the above steps are repeated for the next instruction till a HLT instruction is executed.

That is, the complete course of action of an instruction is accomplished in 3 steps viz..

FETCH DECODE and EXECUTE

Instruction timings

As seen, an instruction is fully executed in three different steps viz. fetch, decode and execute.

Fetching an instruction or data is a time consuming process because the micro processor has to interact with the external RAM. For fetching an instruction or data, the microprocessor has to place the appropriate address on the address bus, which has to be decoded by the memory chip, the instruction or data is to be placed on the data bus by the memory chip , etc.

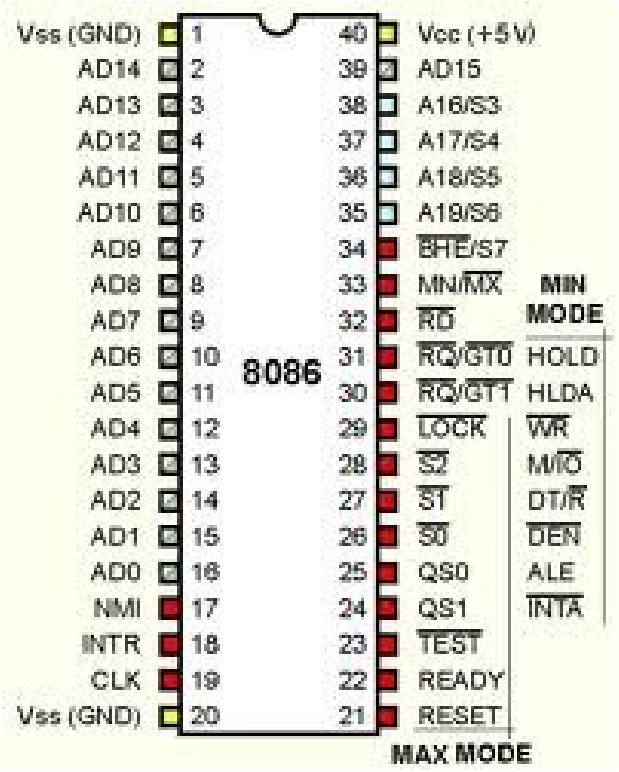
Decoding is a comparatively faster process as it is done inside the microprocessor. Execution timing depends on what the instruction actually demands to do.

This means the total execution time of an instruction varies with instructions. The instruction timing is specified in units called T-state (throttling states). A T-state is equal to one clock period of the microprocessor. (If the processor uses 3 MHz clock, the T –state is0.33 μs. For performing the smallest sub-operation, a microprocessor requires at least one T-state.)

For total execution, MOV A, B takes 4 T-states, MOV A, M takes 7 T-sates, JMP 2789H takes 10 T-states etc.

The number of T-states for each instruction is available in the full instruction set. This information is of great use in programs like square wave generation, traffic signal controller, etc.

INTEL 8086



It is a 16 bit - microprocessor on a 40 pin IC. 8086 has a 20-bit address bus and can access upto 2^{20} memory locations (1 MB). It has multiplexed address and data bus AD0- AD15 and A16 – A19. Its word length is 16 bits (two bytes). The processor can be used for 16 bits as well as for 8 bit arithmetic

Internal architecture

8086 has two blocks BIU (Bus interface unit) and EU (execution unit).

BIU

The BIU performs all bus operations such as instruction fetching, reading and writing data for memory and calculating the addresses of the memory locations.

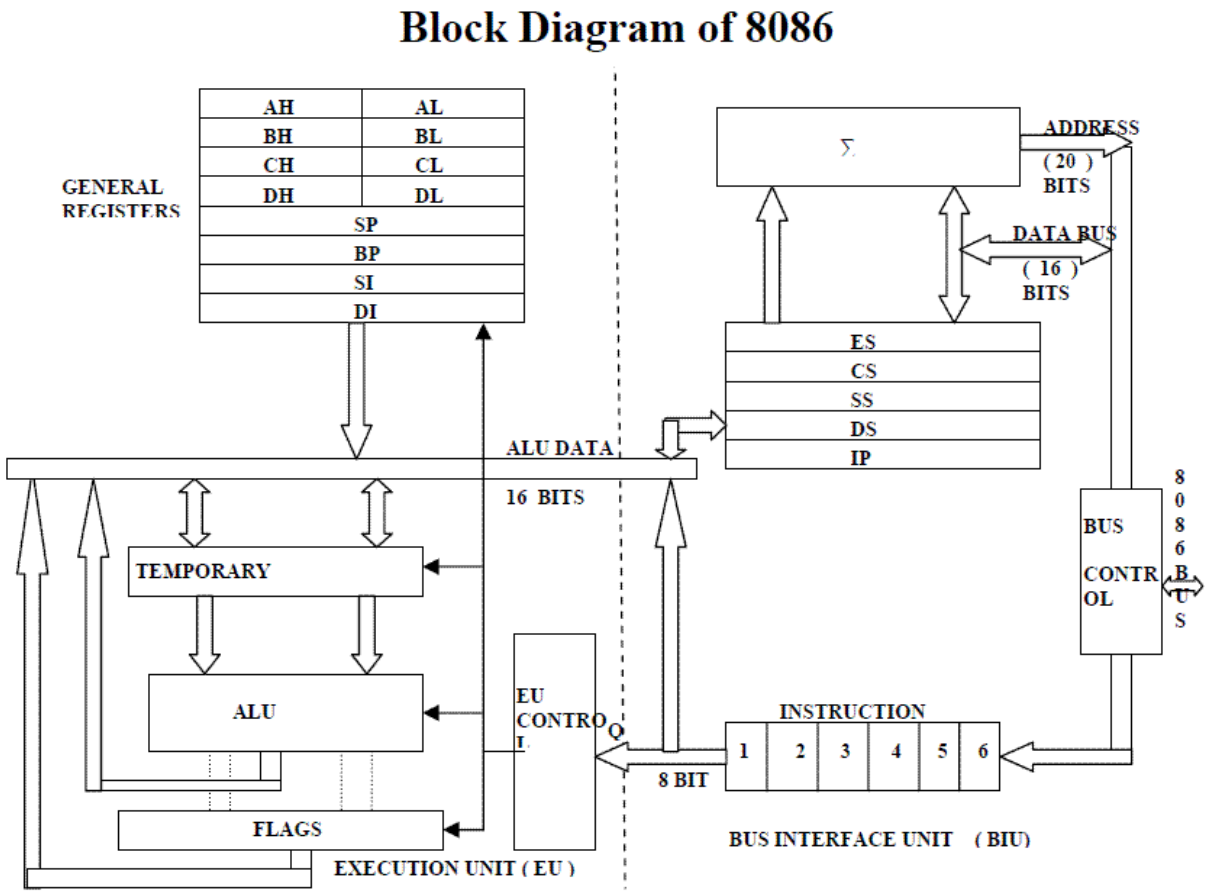
It mainly contains

1. *an Instruction queue* of 6 bytes length using which it can prefetch up to 6 instruction bytes from memory and queue them up. Whenever the EU asks for the instruction, to the BIU, it is readily available in the queue and hence the operation of the processor becomes faster. (this method is called *pipelining, means 'water' or the data is available whenever you*

are in need. Every time the water need not be brought from the tank or memory)

2. *Four 16 bit Segment registers* viz. Code Segment register (CS), Stack Segment register (SS), Data Segment register (DS) and Extra Segment register (ES). Segment registers are used for address generation
3. *a 16 bit register called Instruction pointer,* using which the address of the program locations are generated

** the use of these registers is explained in detail in the section Address generation*



EU

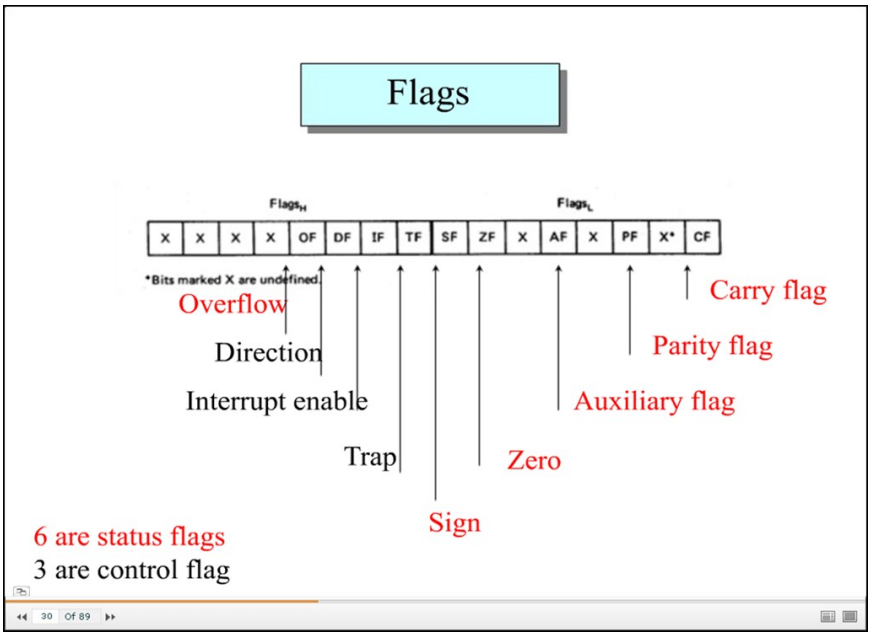
EU is responsible for the decoding and execution of instructions. EU executes instructions from the instruction queue.

It mainly contains

1. *Instruction decoder* to decode instructions
2. *ALU* for arithmetic and logic operations
3. *Four 16 bit general purpose registers* AX, BX, CX, DX to store data. AX is called 16 bit accumulator. Each of these 16 bit registers can be used as two 8 bit registers. Thus AX = AH+AL, BX= BH+BL. CX= CH+CL and DX = DH+DL, where AH, BH, etc. are higher order registers and AL, BL,

etc. are lower order registers. When used for 8 bit arithmetic, AL is the accumulator (AX-is called accumulator, BX, base register, CX, counter register and DX, data register.). These registers are user programmable

4. *Four 16-bit pointer and index registers* called Stack pointer (SP), Base pointer (BP), Source index (SI) an Destination Index (DI). These are used for address generation of the data locations.
5. *Flag register* , which is a 16 bit register, but only 9 flipflops re used as flags.



The most commonly used flags are the same as that used in 8085 (refer to 8085 flags).

Memory segmentation in 8086

Unlike 8085, where program, data and all can be stored in any location of the memory space (RAM), 8086 employs the technique of a segmented memory. The memory is divided into 4 segments viz.

1. Code segment to store the programs (instructions)
2. Stack segment to store those information, which are to be stored by a program to reuse later.
3. Data segment to store data
4. Extra segment to store data (slightly different use from data segment)

It is to be noted that if the CS register carries 2000H, the address of the memory locations of the code segment starts from 20000H, if the DS register carries 2567H, the address of the memory locations of the data segment starts from 25670H, etc.

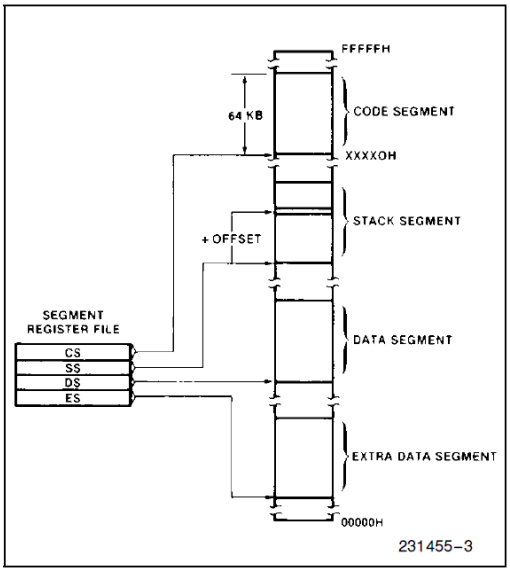


Figure 3a. Memory Organization

Segment register ≠ Segment
Segment register (CS, DS, SS, or ES) is a 16-bit register of 8086
Segment is a portion of the memory (code segment, data segment, stack segment, extra segment)

Address generation in 8086.

All registers of 8086 are 16 bits wide, but the address bus is 20 bits wide. This generates an issue: If the content of a register is to be placed as address, the address will be only 16 bits wide. So how to generate 20-bit address form 16 bit registers?

For address generation, we take two 16-bit numbers and *shift and add* them to generate a 20-bit number, as demonstrated below:

Let the 16 bit numbers be X and Y. Say, X = 2000H and Y = 0001H. Let us write these two numbers in the 16-bit binary form

X	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Let us write the same inside a 20-column table

X					0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Y					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

If we add X and Y now, we would get a 16 bit sum gain. Instead of directly adding, we now shift X by 4 positions, to the left and add with Y

X	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0				
Y					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
X+Y	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

So the sum is now a 20 bit binary number or a 5 digit hexadecimal number 20001H. The shift and add of 2000H and 0001H can be written in the hexadecimal format as follows (shifting of 4 bits in binary is equivalent to shifting of a single digit in hexadecimal)

X	2	0	0	0	
Y		0	0	0	1
X+Y	2	0	0	0	1

Let us take X = 2536H and Y =19F4H as another example. Then,

X	2	5	3	6	
Y		1	9	F	4
X+Y	2	6	D	5	4

In 8086, for address generation of any location in a particular segment, the first number is invariably the content of the corresponding segment register. For eg. Content of code segment register [CS] is the first number for generating address of any location in the code segment, [DS] is the first number for data segment address generation and so on.

The second number for the code segment address generation is the content of instruction pointer IP. Or the *code segment address is generated by shift and adding [CS] and [IP]*. (note that IP is similar to program counter, PC, in 8085).

Similar shift and add methods are used to generate, stack segment, data segment and extra segment addresses. The first number for these address generations is invariably the content of the corresponding segment registers. In order to avoid confusion, we do not attempt to discuss which/where is the second number here. The same would be understood to a little bit more, when we discuss addressing modes.

The 20-bit address is called the physical address and the *second number* is called the *effective address*

Physical address = Shifted segment register content + effective address

*Note: In 8086 programming using microprocessor kits**, we will be using the 16 bit effective address only.** Segment register contents are not relevant to the programmer*

Addressing modes.

There are no hard and fast rules to categorize instructions in terms of addressing modes. One common classification is there are 5 groups of addressing modes, with a total number of 12 addressing modes. These groups are

1. Addressing modes for accessing immediate and register data (2 modes in these)
2. Addressing modes for accessing data in memory (6 modes)
3. Addressing modes for accessing data in I/O ports (2 modes)
4. Relative addressing modes (1 mode)
5. Implied addressing modes (1 mode)

1. **Addressing modes for accessing immediate and register data**

   There are two modes in this group viz.

   a. the *register mode*: the operand (data) is in any of the registers. Eg. The instruction MOV AX, DX copies the [DX] to [CX].
   b. *Immediate mode*: an 8 bit or 16 bit data appearing as part of an instruction. Eg. MOV AL, 32H copies 32H to[AL]. *Note that the data (32H) present along with the instruction is stored in the code segment itself and not in the data segment.*

2. **Addressing modes for accessing data in memory**

   These are the methods by which the data present in memory locations are accessed through instructions. Six types of addressing is possible.

   a. *Direct addressing mode:* here the *effective address* of the data is present in the instruction. Eg. Instruction MOV CX, [0004H] copies data from the data segment memory location pointed by the effective address 0004H. Note that the programmer will be interested only in the effective address and not the physical address. In this example, If [DS] = 2000H, then the physical address of the data segment location is 20004H (shifted and added). The instruction copies data from 20004 H and 20005H to [CX]. (Since CX is a 16 bit register and memory location 20004H carries only 8 bit data, the instruction transfers the content of two consecutive locations to CX)
   b. *Register indirect addressing mode*: Here the 16 bit effective address of the data is present in the registers BX, SI, or DI. Eg. MOV [DI], DX copies [DX] to the data segment memory location pointed out by the physical address, which is present in [DI]
   c. *Based addressing mode*: Here the effective address is sum of [BX] or [BP] and a 8 or16 bit number (displacement) . Eg. MOV AX, 50H [BX] copies data from the memory location whose effective address is [BX]+50H
   d. *Indexed addressed mode*: This is similar to abased addressing mode except that [SI] or [DI] is used here instead of [BX] or [BP]. Eg. MOV DX, 50H [SI]
   e. *Based indexed addressing mode*: here the effective address is sum of [BX] or [BP] and [SI] or [DI]. Eg. MOV AX, [BX] [SI]
   f. *Relative based indexed addressing mode*: this is a combination of the based and indexed mode. Here the effective address is the sum of [BX] or [BP], [SI] or [DI] and an 8 or 16 bit displacement

3. **Addressing modes for accessing data in I/O ports**

   These are the methods for accessing data present in I/O ports. There are 2 types.

   a. Direct port addressing mode: herethe 8 bit port address is present along with the instruction. Eg. OUT 56H, AL copies data from [AL] to the output port whose address is 56H
   b. Indirect port addressing mode: Here the port address is specified in [DX]. Eg. IN AL, DX copies data from the input port whose address is specified in [DX] to [AL]

4. **Relative addressing modes**

   Instruction of this type specifies the operand as a signed (positive number as it is and negative number in 2' s complement form) 8 or 16 bit displacement relative to IP. Eg. JMP 04H changes the program execution sequence forward to a new IP ,

which is current [IP] + 04H. JMP F4H changes the program execution sequence backward to a new IP, which is current [IP] – 0CH (F4H = 1111 0100 is a negative number and as it starts with 1. The number is 2's complement of F4H , which is 0CH)

5. Implied addressing mode:

In this, no explicit operand is present. Eg. CLC means clear the carry flag, and the operand (flag register) is not mentioned in the instruction.