Bangladesh Army University of Engineering & Technology (BAUET)
Department of Computer Science and Engineering

Microprocessors and Micro-controller

Microcomputer System

Prof. Dr. Md. Rabiul Islam
Dept. of Computer Science & Engineering (CSE)
Rajshahi University of Engineering & Technology (RUET)
Bangladesh
rabiul_cse@yahoo.com

Outline

- History of Microprocessor [1]
- Basic Operation of a Computer
- Outlook of Various Microprocessor
- Intel 8086 Architecture [2]
  - Data Registers [2][3]
  - Segment Registers [2][3]
  - Pointer and Index Registers [2]
  - Instruction Pointer [2]
  - FLAG Register [2]
- Combination of Segment and Offset Address
- Data and Address Bus of 8086
- Memory into Disjoint Segments for 8086 [3]
- DOS and BIOS Routine [3]
- Memory Organization of the PC [3]
- I/O Port Address [3]
- Start-up Operation of the PC [3]
- Basics of Assembly Language [3]
  - Machine Code [3]
  - Assembly Language [3]
  - Compiler/Assembler [3]

References

[1] ***Chapter 1*** Barry B. Brey, “The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processor, Architecture, Programming, And Interfacing”, 4th Edition, Prentice Hall, 1997.

[2] ***Chapter 3*** Mohamed Rafiquzzaman, “Microprocessor and Microcomputer Based System Design”, 2nd Edition, CRC Press, 1995.

[3] ***Chapter 3*** Yutha Yu and Charles Marut, “Assembly Language Programming and Organization of the IBM PC", McGraw-Hill International Edition, 1992.

History of Microprocessors

- The Mechanical Age
- The Electrical Age
- Programming Advancements
- The Microprocessor Age
  - Intel 4004
  - Intel 4040
  - Intel 8008
  - Intel 8080
  - Motorola MC6800

TABLE Early 8-bit microprocessors

Manufacturer	*Part Number*
Fairchild	F-8
Intel	8080
MOS Technology	6502
Motorola	MC6800
National Semiconductor	IMP-8
Rockwell International	PPS-8
Zilog	Z-8

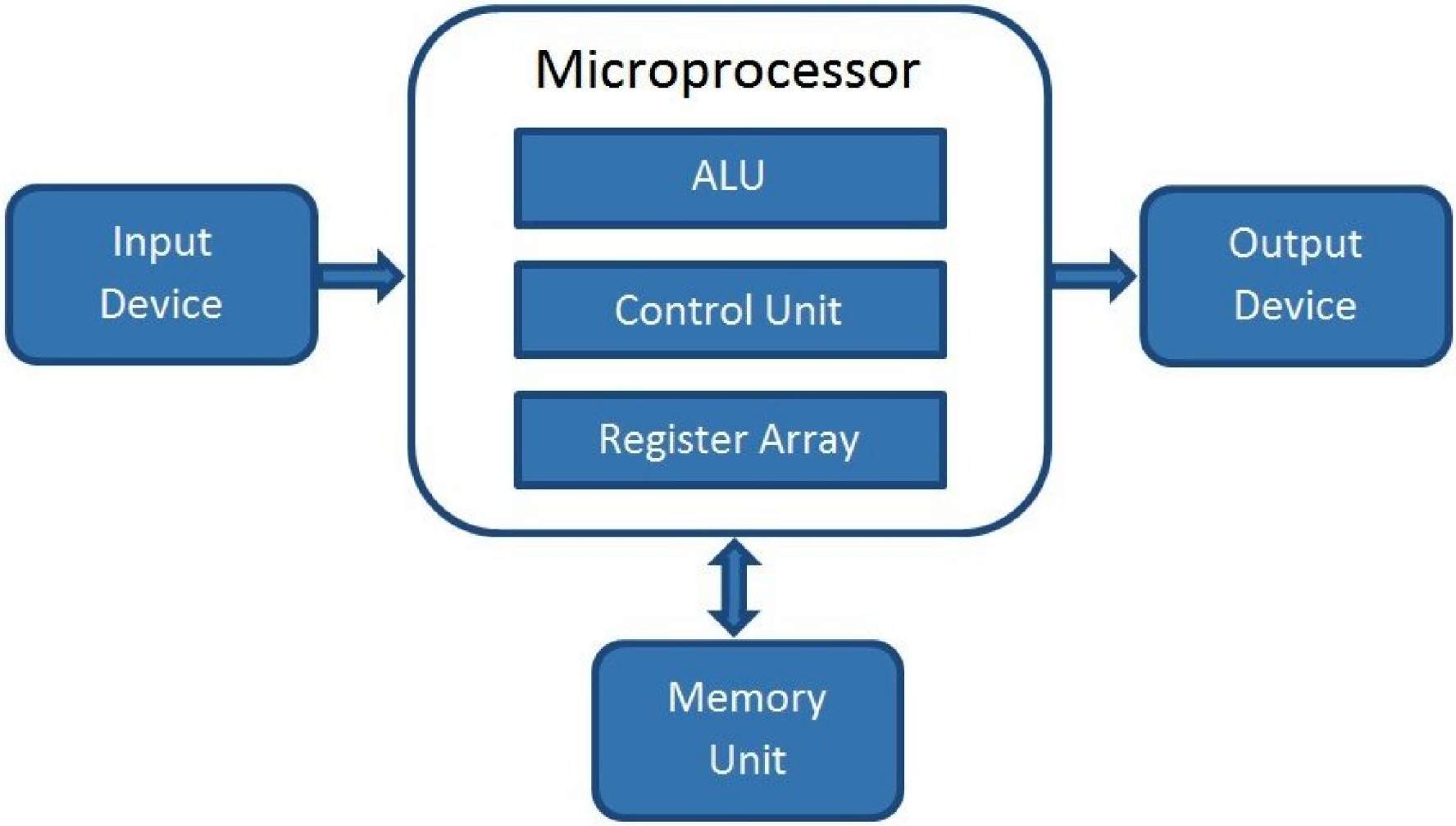
- The Modern Microprocessors
  - Features of 8086 and 8088 Microprocessor
  - RISC (Reduced Instruction Set Computer)
  - CISC (Complex Instruction Set Computer)

History of Microprocessors

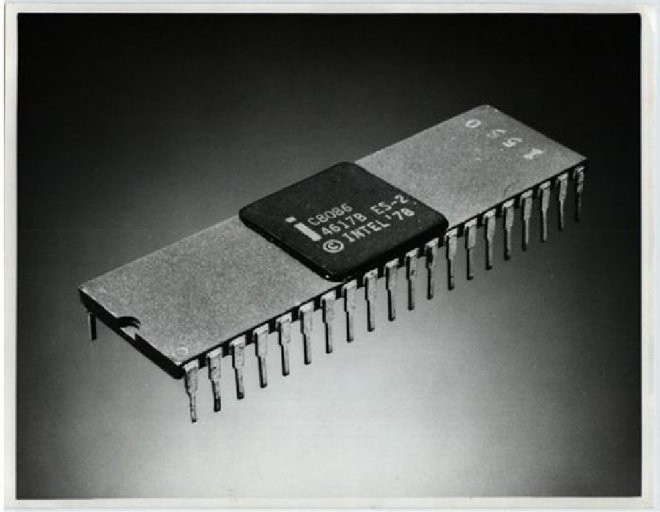
TABLE Many modern Intel and Motorola microprocessors

Manufacturer	*Part*	*Data Bus Width*	*Memory Size*
Intel	8048	8	2K internal
	8051	8	8K internal
	8085A	8	64K
	8086	16	1M
	8088	8	1M
	8096	16	8K internal
	80186	16	1M
	80188	8	1M
	80251	8	16K internal
	80286	16	16M
	80386EX	16	64M
	80386DX	32	4G
	80386SL	16	32M
	80386SLC	16	32M + 1K cache
	80386SX	16	16M
	80486DX/DX2	32	4G + 8K cache
	80486SX	32	4G + 8K cache
	80486DX4	32	4G + 16K cache
	Pentium	64	4G + 16K cache
	Pentium Overdrive (P24T)	32	4G + 16K cache
	Pentium Pro processor	64	64G + 16K L1 cache + 256K L2 cache
Motorola	6800	8	64K
	6805	8	2K
	6809	8	64K
	68000	16	16M
	68008Q	8	1M
	68008D	8	4M
	68010	16	16M
	68020	32	4G
	68030	32	4G + 256 cache
	68040	32	4G + 8K cache
	68050	32	Proposed, but never released
	68060	64	4G + 16K cache
	PowerPC	64	4G + 32K cache

Basic Operation of a Computer



Outlook of Various Microprocessors



Intel 8086 Microprocessor

Intel Core i9 Microprocessor

Intel 8086 Architecture

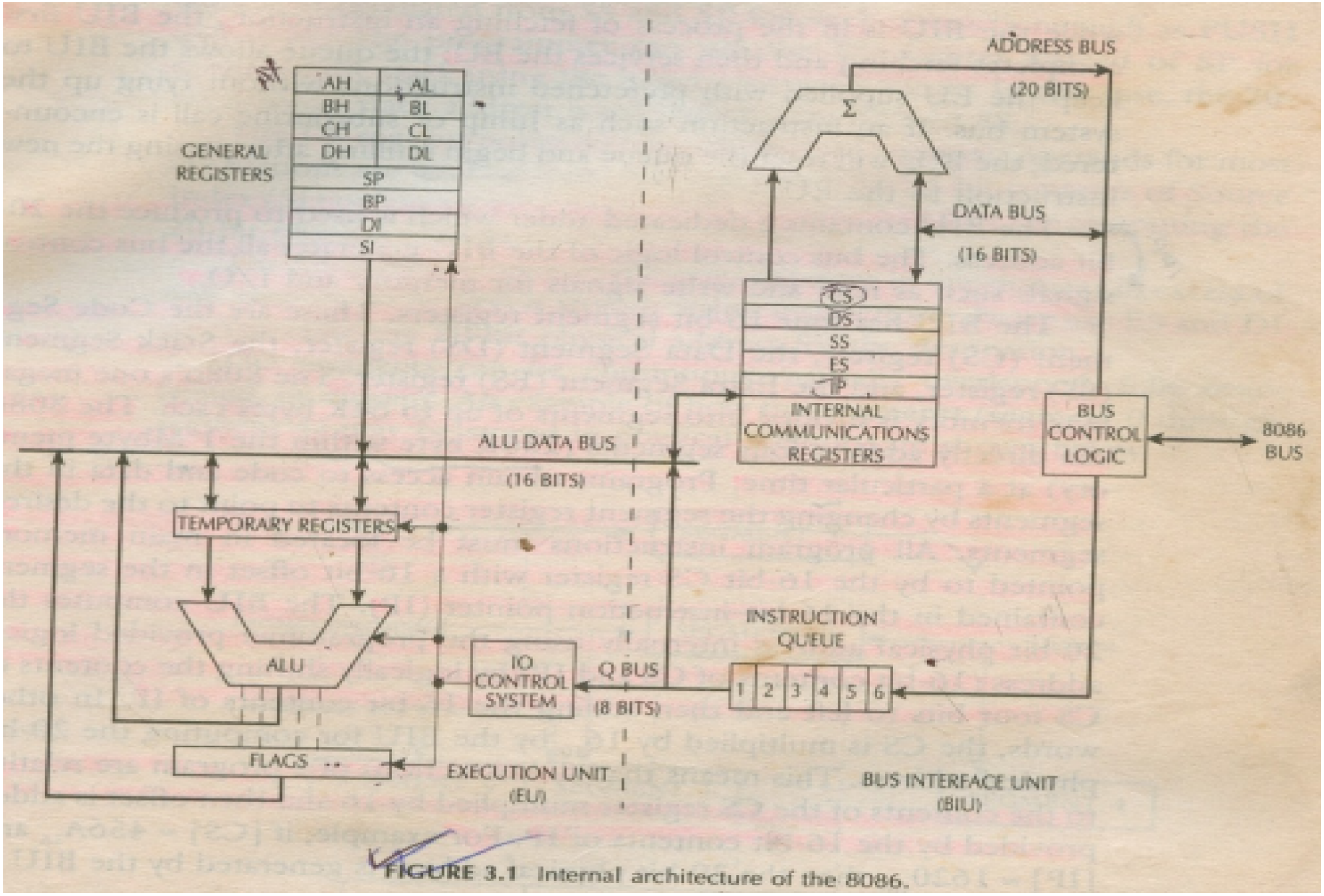


FIGURE 3.1 Internal architecture of the 8086.

Internal architecture of 8088 Microprocessor

- Bus Interface Unit (BIU)
- Execution Unit (EU)
- Data Registers
- Segment Registers
- Pointer and Index Registers
- Instruction Pointer (IP)
- FLAG Register

Intel 8086 Architecture: Bus Interface Unit (BIU)

❑Functions of Bus Interface Unit (BIU):

✓The BIU and EU function independently.

✓The BIU interface the 8086 to the outer world.

✓BIU fetch instruction, reads data from memory and ports and write data to memory and ports.

✓The BIU provides external bus operations.

✓The BIU contains segment registers, instruction pointer, instruction queue and address generator/bus control circuitry to provide functions such as fetching and queuing of instructions and bus control.

✓The BIU instruction queue is a FIFO group of registers in which up to 6 bytes of instruction code are prefetched from memory ahead of time. This is done in order to speed up the program execution. This mechanism id known as pipelining.

✓The BIU contains a dedicated adder which is used to produced 20 bit physical address.

✓The Bus Control Logic of the BIU generates all the bus control signals to read and write signals for memory and I/O.

Intel 8086 Architecture: Bus Interface Unit (BIU)

❑Functions of Execution Unit (EU):

✓The EU executes instructions that have already been fetched by the BIU.

✓The EU decodes and executes instructions.

✓A decoder in the EU control system translates instructions.

✓The EU has a 16 bit ALU to perform arithmetic and logic instructions.

✓The EU contains temporary register for holding operation for the ALU and FLAG register which individual bits reflect the result of a computation.

✓The BIU and EU are connected by an internal bus and they work together. While the EU is executing an instruction, the BIU fetches up to six bytes of the next instruction and places them in the instruction queue.

Data Registers: (AX, BX, CX and DX)

These four registers are available to the programmer for general data manipulation.

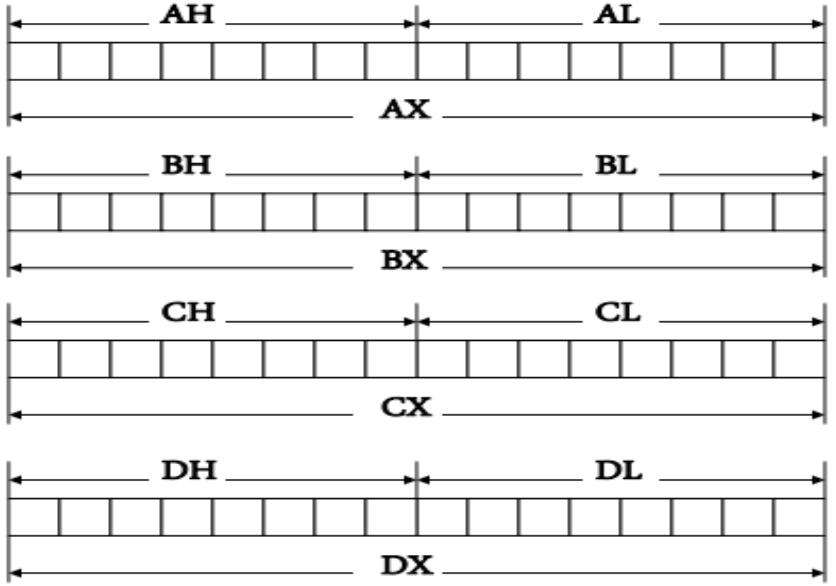
✓The high and low bytes of the data registers can be accessed separately. The high byte of AX is called AH and the low byte is AL. Similarly, the high and low bytes of BX, CX, and DX are BH and BL, CH and CL, DH and DL respectively. This arrangement gives us more registers to use when dealing with byte-size data.

AX (Accumulator Register): AX is the preferred register to use in arithmetic, logic, and data transfer instructions. Input and output operations also require the use of AL and AX.

BX (Base Register): BX also serves as an address register; an example is a table look-up instruction called XLAT (translate).

CX (Count Register): Program loop constructions arc facilitated by the use of CX, which serves as a loop counter.

DX (Data Register): DX Is used in multiplication and division. It is also used in 1/0 operations



Segment Registers: (CS, DS, SS and ES)

Code Segment:

✓The 8086's one megabyte of memory is divided into segments up to 64K bytes each. The 8086 can directly access four segments (256K bytes within 1 MB memory) at a particular time.

✓All program instructions must be located in main memory pointed to by 16 bit CS register with a 16-bit offset in the segment contained in the 16-bitINstructor Pointer (IP).

✓The BIU computes the 20-bit physical address internally by logically shifting by the contents of CS four bits to left and then adding the 16-bit contents of IP.

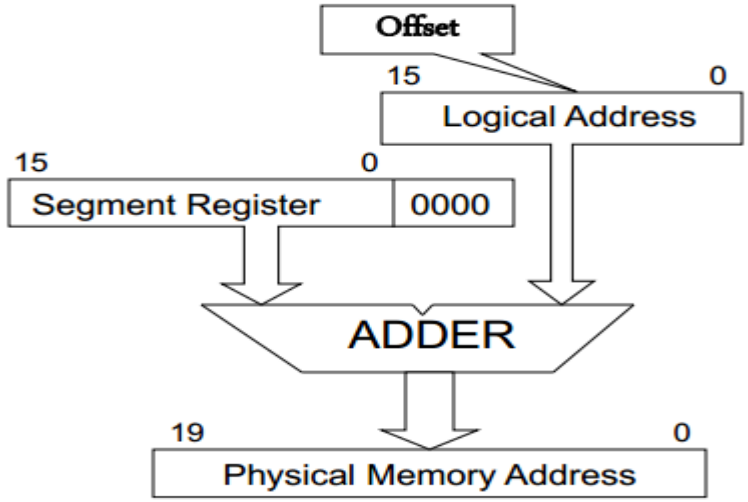
For example, if [CS] = $456A_{16}$ and [IP] = 1620_{16}, then the 20-bit physical address is generated by the BIU as follows:

Four times logically shifted [CS] to left ⇒ $456A0_{16}$

\+ [IP] as offset = 1620_{16}

20-bit physical address = $46CC0_{16}$

✓CS contains the base or start of the current code segment and the IP contains the distance or offset from this address to the nest instruction byte to be fetched.



Segment Registers: (CS, DS, SS and ES)

❑Exercise:

Example For the memory location whose physical address is specified by 1256Ah, give the address in segment:offset form for segments 1256h and 1240h.

Solution: Let X be the offset in segment 1256h and Y the offset in segment 1240h. We have

$$1256Ah = 12560h + X \text{ and } 1256Ah = 12400h + Y$$

and so

$$X = 1256Ah - 12560h = Ah \text{ and } Y = 1256Ah - 12400h = 16Ah$$

thus

$$1256Ah = 1256:000A = 1240:016A$$

Segment Registers: (CS, DS, SS and ES)

- **Exercise:**

Example A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?

Solution: We know that

$$\text{physical address} = \text{segment} \times 10\text{h} + \text{offset}$$

Thus

$$\text{segment} \times 10\text{h} = \text{physical address} - \text{offset}$$

in this example

$$\begin{array}{rl} \text{physical address} = & 80\text{FD2h} \\ -\ \text{offset} = & \text{BFD2h} \\ \hline \text{segment} \times 10\text{h} = & 75000\text{h} \end{array}$$

So the segment must be 7500h.

Segment Registers: (CS, DS, SS and ES)

Data Segment:

- The DS register points to the current data segment.
- The 16-bit contents of Source Index (SI) or Destination Index (DI) are used as offset for computing the 20-bit physical address.

Extra Segment:

- The ES register points to the extra segment in which data (in excess of 64K bytes pointer to by the DS) is stored.
- String instruction is always use ES and DI to determine the 20-bit physical address for the destination.

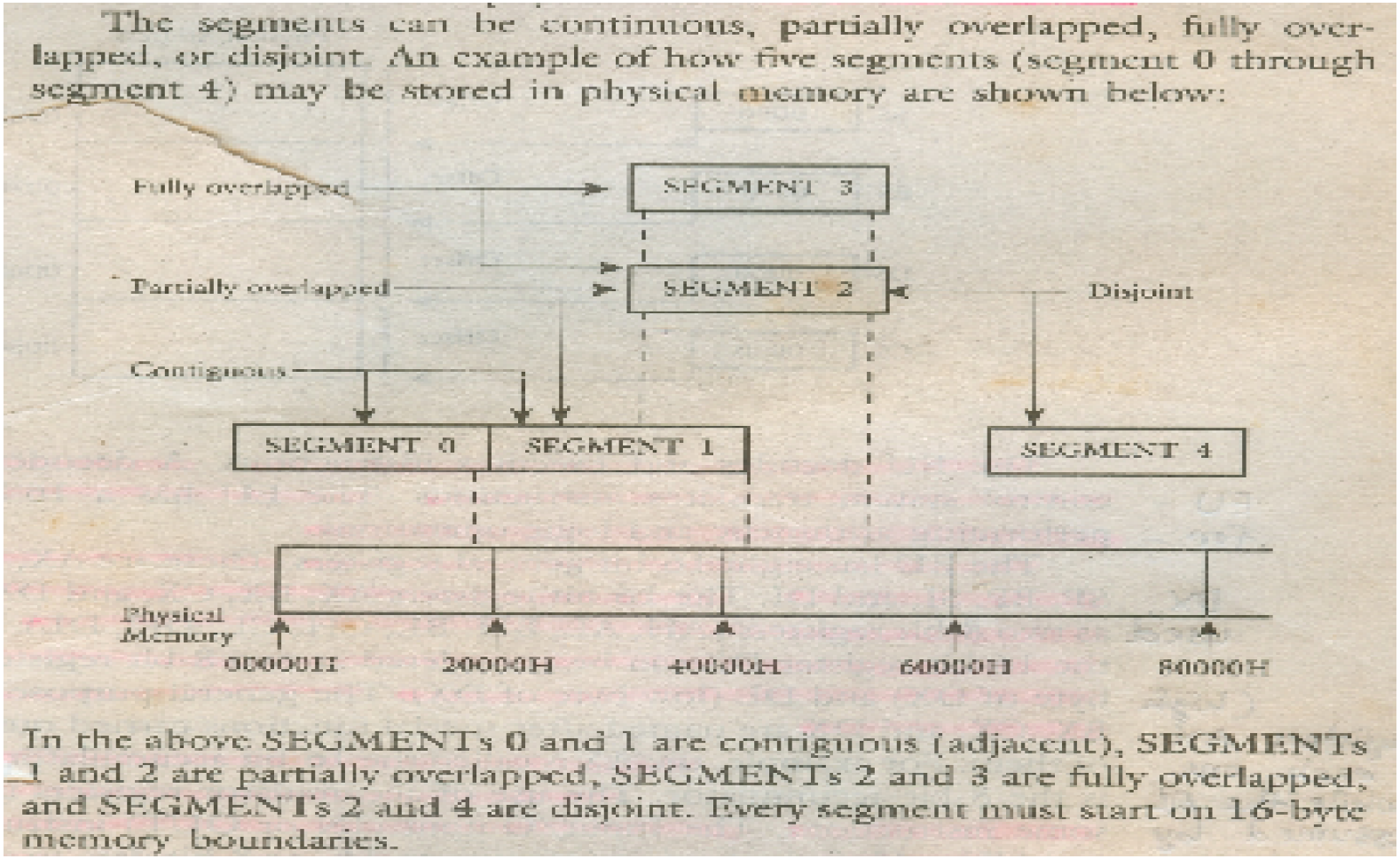
Stack Segment:

- The SS register points to the current stack.
- The 20-bit physical address is calculated from SS and SP for stack instructions such as PUSH and POP.

Segment Registers: (CS, DS, SS and ES)

❑Segments in Memory Location:

✓A program segment need not occupy the entire 64 kilobytes in a memory segment. The overlapping nature of the memory segments permits program segments that arc less than 64 KB to be placed close together.



Segment Registers: (CS, DS, SS and ES)

- Typical layout of a program segments:

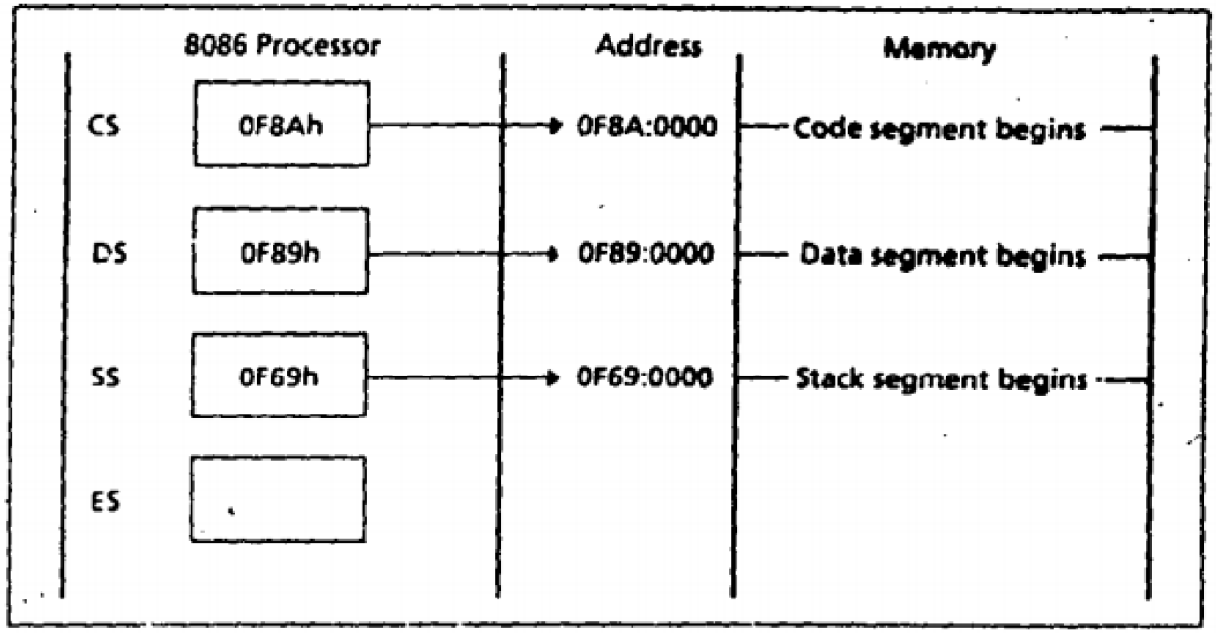


Figure Segment Registers

Pointer and Index Register: (SP, BP, SI and DI)

The registers SP, BP, SI, and DI normally point to (contain the offset addresses of) memory locations.

SP (Stack Pointer):

- The SP (stack pointer) register Is used in conjunction with SS for accessing the stack segment.

BP (Base Pointer):

- The BP (base pointer) register is used primarily to access data on the stack. However, unlike SP, we can also use BP to access data in the other segments.

SI (Source Index) :

- The SI (source index) register is used to point to the memory locations.
- In the data segment addressed by DS. By incrementing the contents of SI, we can easily access consecutive memory locations.

DI (Destination Index):

- The DI (destination Index) register performs the same functions as SI.
- There is a class of instructions, called string operations, that use DI to access memory locations addressed by ES.

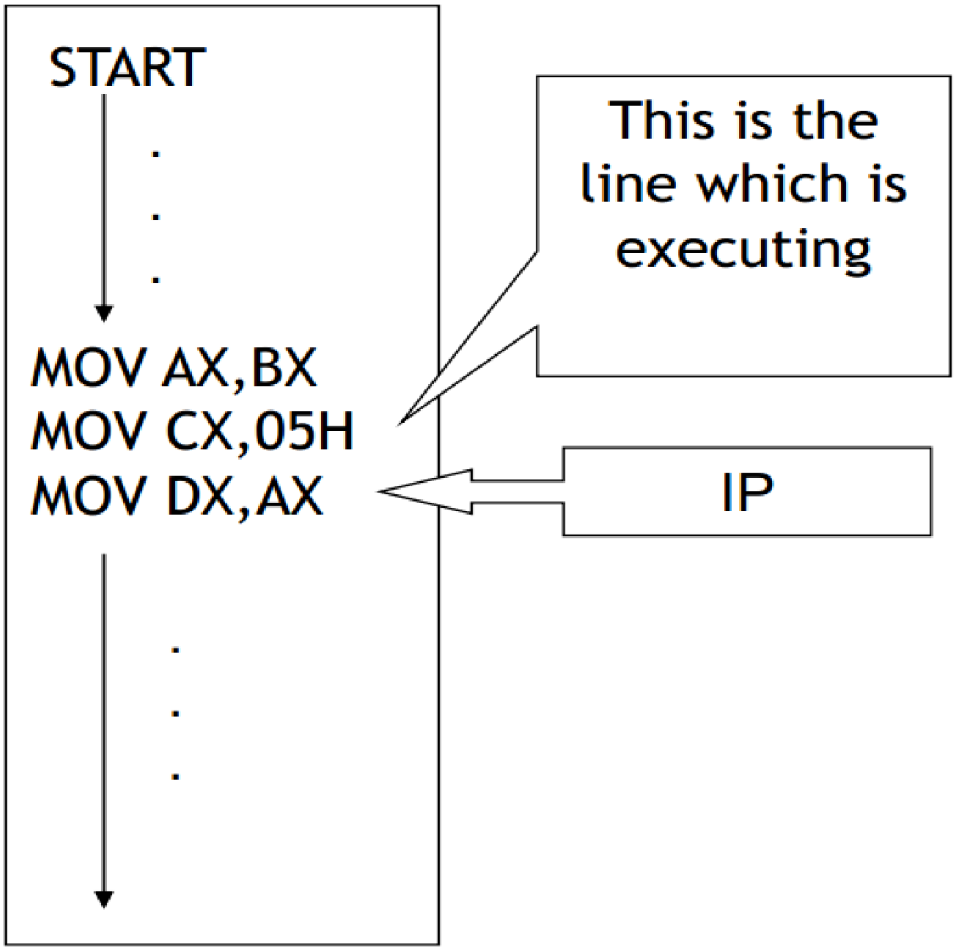
Instructor Pointer (IP)

✓The computer keeps track of the next line to be executed by keeping its address in a special register called the Instruction Pointer (IP) or Program Counter.

✓To access instruction, the 8086 uses the registers CS and IP. The CS register contains the segment number of the next instruction, and the IP contains the offset.

✓IP is updated each time an instruction is executed so that it will point to the next Instruction. Thus a program is executed sequentially line by line.

➢Unlike the other registers, the IP cannot be directly manipulated by an instruction; that is, an interaction may not contain IP as its operand.



FLAG Register

✓The purpose of the FLAGS register is to indicate the status of the microprocessor. It does this by the setting of individual bits called flags.

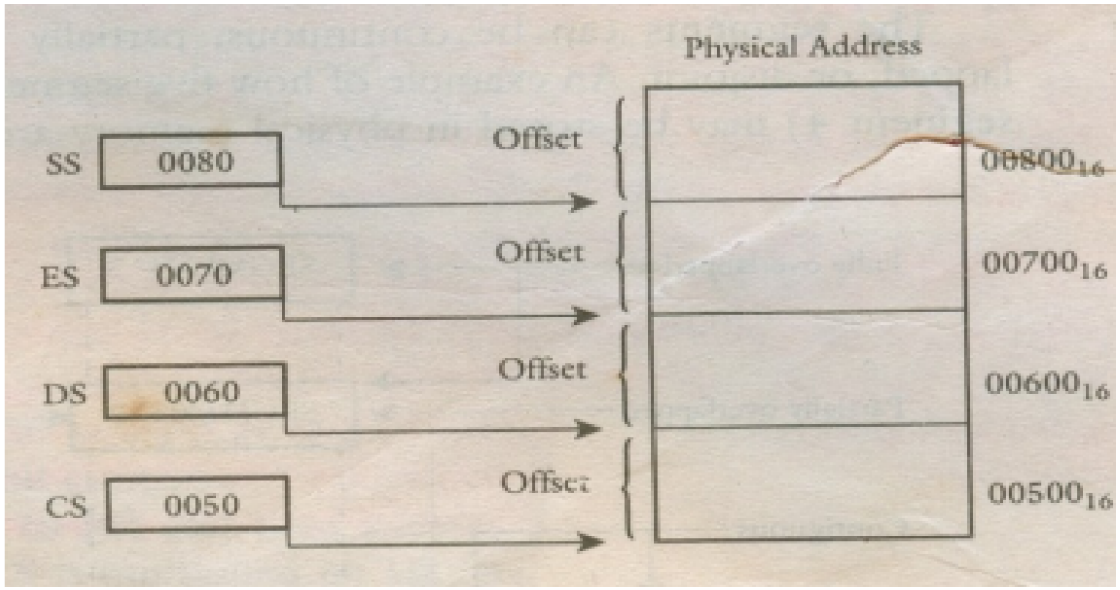
✓There arc two kinds of flags: status flags and control flags.

✓The status flags reflects the result of an instruction executed by the processor. For example, when a subtraction operation results in a 0, the ZF (zero flag) is set to 1 (true).

✓The control flags enable or disable certain operations of the processor; for example, if the IF (interrupt flag) is cleared (set to 0), inputs from the keyboard are ignored by the processor.

Combination of Segment and Offset Address:

The registers SP, BP, SI, and DI normally point to (contain the offset addresses of) memory locations.



Segment	: Offset
Code Segment (CS)	: Instruction Pointer (IP)
Data Segment (DS)	: Source Index (SI)
Stack Segment (SS)	: Stack Pointer (SP) or Base Pointer (BP)
Extra Segment (ES)	: Destination Index (DI)

Data and Address Bus of 8086

8086 has 8 bit data bus and 20 bit address bus. As a result, it can address up to 1MB of memory.

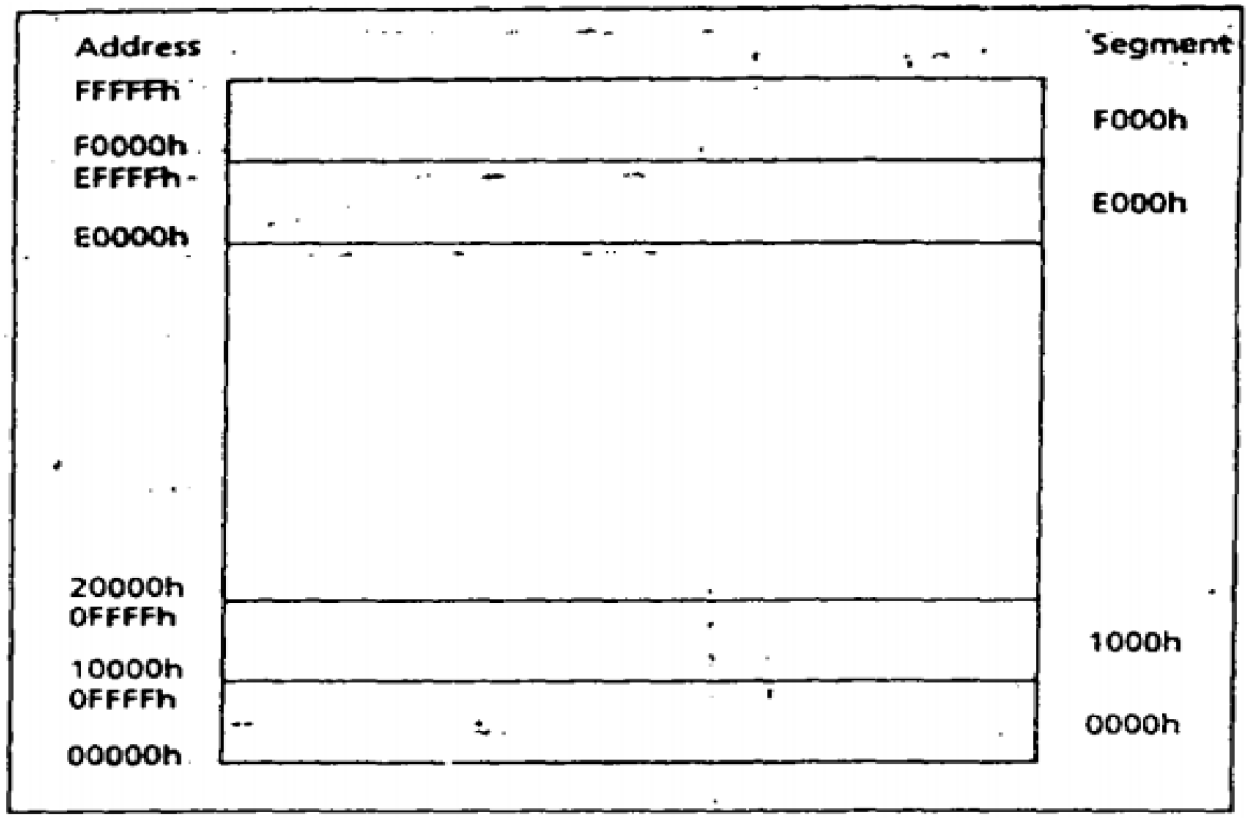


Figure: Size of memory representation of 8086

Memory Organization of the PC

✓The 8086/8088 processor is capable of addressing 1 megabyte of memory. However, not all the memory can be used by an application program. Some memory locations have special meaning for the processor.

✓Figure shows the memory fragmentation for 8086 microprocessor based system.

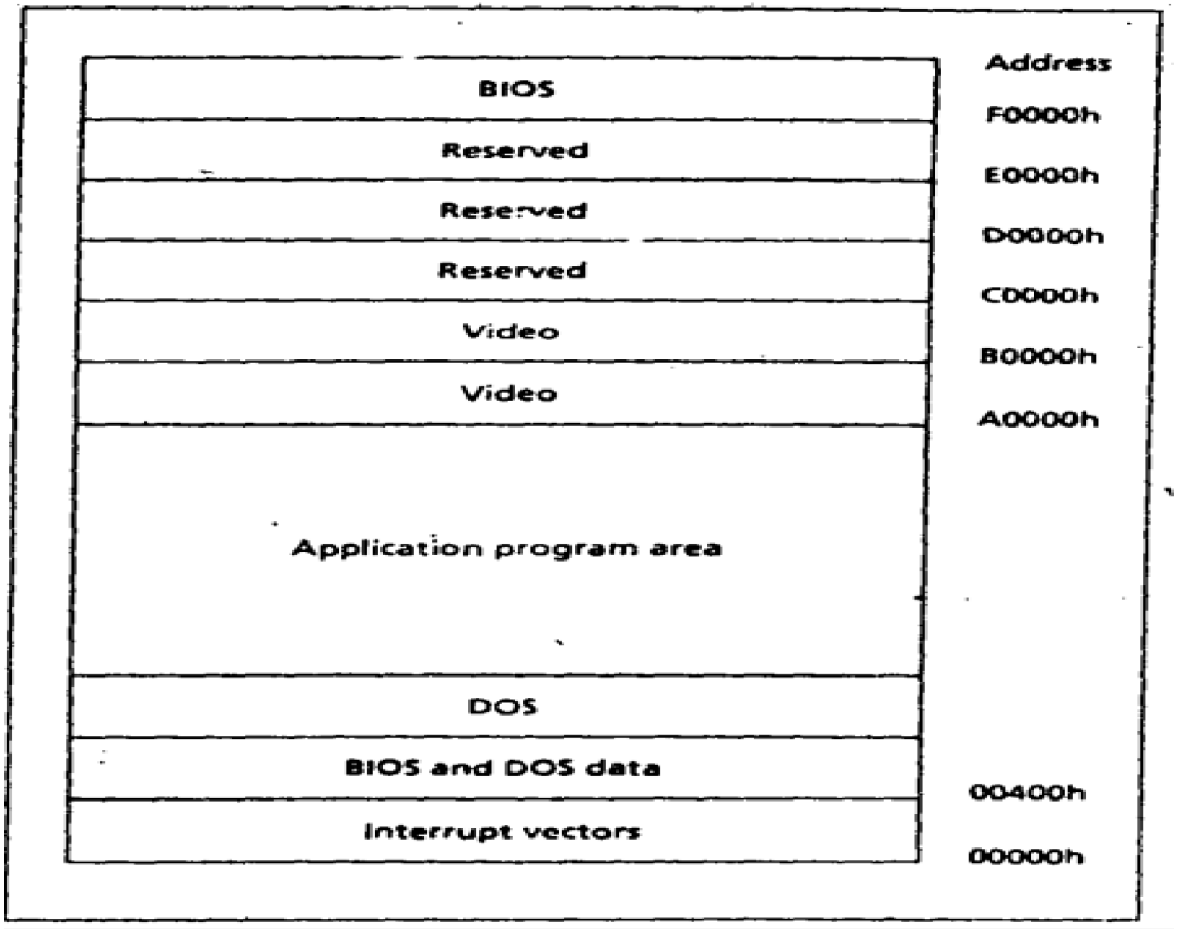


Figure: Memory map of the PC.

DOS and BIOS Routine

✓ Disk Operating System (DOS) was designed for the 8086/8088-based computers. Because of this, it can manage only 1 megabyte of memory and it does not support multitasking. However, It can be used on 80286, 80386, and 80486-based machines when they run in real address mode.

✓Infernal commands are performed by DOS routines that have been loaded into memory, external commands may refer to DOS routines that have not been loaded or to application programs. In normal operations, many DOS routines are not loaded into memory so as to have memory space.

✓Because DOS routines reside on disk, a program must use operating when the computer is powered up to read the disk. The system routines stored in ROM that are not destroyed when the power is off. In the PC, they are called BIOS (Basic Input/Output System) routines.

✓The BIOS routines perform 1/0 operations for the PC. Unlike the DOS routines, which operate over the entire PC family, the BIOS routines are machine specific. Each PC model has its own hardware configuration and its own BIOS routines, which invoke the machine's 1/0 port registers for input and output. Tile DOS 110 operations arc ultimately carried out by the BIOS routines.

✓Other important functions performed by BIOS are circuit checking and loading of the DOS routines.

✓To let DOS and other programs use the BIOS routines, the addresses of the BIOS routines, called interrupt vectors, arc placed at memory, starting at 0000h.

I/O Port Address

The 8086/8088 supports 64 KB of 1/0 ports. Some common port addresses are given in the following table:

Table Some Common I/O Ports for the PC

Port Address	*Description*
20h–21h	interrupt controller
60h–63h	keyboard controller
200h–20Fh	game controller
2F8h–2FFh	serial port (COM 2)
320h–32Fh	hard disk
378h–37Fh	parallel printer port 1
3C0h–3CFh	EGA
3D0h–3DFh	CGA
3F8h–3FFh	serial port (COM1)

Start-up Operations of the PC

✓When the PC is powered up, the 8086/8088 processor is put in a reset state, the CS register is set to FFFFh, and IP is set to 0000h. So the first instruction it executes is located at FFFF0h. This memory location is in ROM, and it contains an instruction that transfers control to the starting point of the BIOS routines.

✓The BIOS routines first check for system and memory errors, and then initialize the interrupt vectors and BIOS data area.

✓Finally, BIOS loads the operating system from the system disk. This is done in two steps; first, the BIOS loads a small program, called the boot program. Then the boot program loads the actual operating system routines. The boot program is so named because it is part of the operating system; having it load the operating system is like the computer pulling itself up by the bootstraps.

✓ Using the boot program isolates the BIOS from any changes made to the operating system and lets it be smaller in size. After the operating system is loaded into memory, COMMAND.COM is then given control.

Basics of Assembly Language

❑Machine Code:

✓There are occasions when the programmer must program at the machine's own level.

✓Machine Code programs are tedious to write and highly error prone.

✓In situations where a high-level language is inappropriate, we avoid working in machine code most of the time by making the computer do more of the work. Thus we write in assembly language and then the computer converts this assembly language program into machine code.

```
0000111100001111
0010010101010100
1010101010100101
```

Basics of Assembly Language

❑Assembly Language:

✓In assembly language, a mneumonic (i.e. memory aid) is used as a short notation for the instruction to be used.

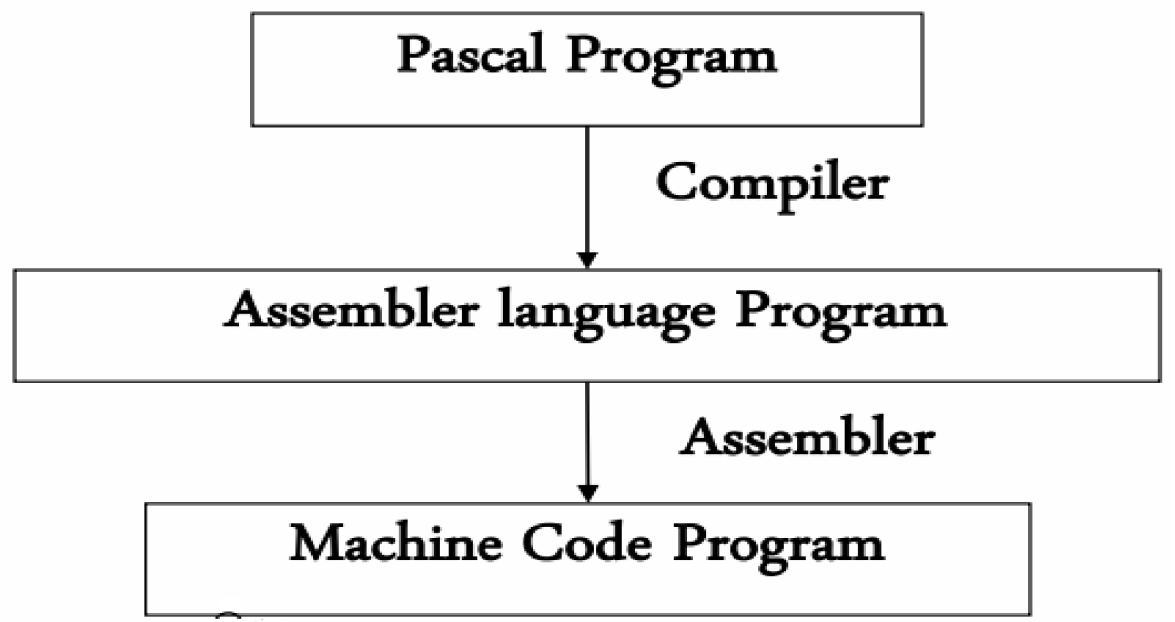
✓Assembly language is an intermediate step between high level languages and machine code. Most features present in HLL are not present in Assembly Language as type checking etc.

Assembly Language	Machine Code
SUB AX,BX	001010111000011
MOV CX,AX	100010111001000
MOV DX,0	10111010000000000000000

Basics of Assembly Language

❑Compiler/Assembler:

✓High-level Languages such as Pascal programs are sometimes converted firstly to assembly language by a computer program called compiler and then into machine code by another program called assembler.



Thanks

For any additional query please contact with me through
rabiul_cse@yahoo.com