CHAPTER 1

THE 8086 MICROPROCESSOR

1.1 INTRODUCTION TO MICROPROCESSORS

1.1.1 TERMS USED IN MICROPROCESSOR LITERATURE

Bit : A digit of the binary number or code is called a bit.

Nibble : The 4-bit (4-digit) binary number or code is called a nibble.

Byte : The 8-bit (8-digit) binary number or code is called a byte.

Word : The 16-bit (16-digit) binary number or code is called a word.

Double Word : The 32-bit (32-digit) binary number or code is called a double word.

Multiple Word : The 64, 128, ... bit / digit binary numbers or codes are called multiple words.

Data : The quantity (binary number/code) operated by an instruction of a program is used data. The size of data is specified as bit, byte, word,etc.

Address : Address is an identification number (in binary) for memory locations. The 8086 processor uses a 20-bit address for memory.

Memory Word Size (or Addressability) : The memory word size or addressability is the size of binary information that can be stored in a memory location. The memory word size for an 8086 processor-based system is 8-bit.

[Address and program codes in a microprocessor system are given in binary (i.e., as a combination of "0" and "1"). With n-bit binary we can generate 2^n *different binary codes or addresses.]*

Microprocessor : The microprocessor is a program-controlled semiconductor device (IC), which fetches data (from memory), decodes and executes instructions. It is used as a CPU (**C**entral **P**rocessing **U**nit) in computers.

The basic functional blocks of a microprocessor are ALU (**A**rithmetic **L**ogic **U**nit), an array of registers and a control unit. The microprocessor is identified with the size of data the ALU of the processor can work with at a time. The 8086 processor has a 16-bit ALU; hence, it is called a 16-bit processor. The 80486 processor has a 32-bit ALU; hence, it is called a 32-bit processor.

Bus : A bus is a group of conducting lines that carries data, address and control signals. Buses can be classified into Data bus, Address bus and Control bus.

The group of conducting lines that carries data is called a data bus.

The group of conducting lines that carries address is called an address bus.

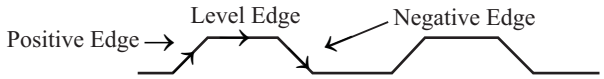
The group of conducting lines that carries control signals is called a control bus.

CPU Bus : The group of conducting lines that are directly connected to the microprocessor is called a CPU bus. In a CPU bus, the signals are multiplexed, i.e., more than one signal is passed through the same line but at different timings.

System Bus : The group of conducting lines that carries data, address and control signals in a microcomputer system is called System bus. Multiplexing is not allowed in a system bus.

[In microprocessor-based systems, each bit of information (data/address/control signal) is sent through a separate conducting line. Due to practical limitations, the manufacturers of microprocessors may provide multiplexed pins, i.e., one pin is used for more than one purpose. This leads to a multiplexed CPU bus. For example, in an 8086 processor, the address and data are sent through the same pins but at different timings. But when the system is formed, the multiplexed bus lines should be demultiplexed by using latches, ports, transceivers, etc. The demultiplexed bus lines are called system bus. In a system bus, separate conducting lines will be provided for each bit of data, address and control signals.]

Clock : A clock is a square wave used to synchronize various devices in the microprocessor and in the system. Every microprocessor system requires a clock for its functioning. The time taken for the microprocessor and the system to execute an instruction or program are measured only in terms of the time period of its clock.



A clock has three edges: rising edge (positive edge), level edge and falling edge (negative edge). The device is made sensitive to any one of the edges for better functioning (it means that the device will recognize the clock only when the edge is asserted or arrived).

Tristate Logic : Almost all the devices used in a microprocessor-based system use tristate logic. In devices with tristate logic, three logic levels will be available: **High** state, **Low** state and **High impedance** state.

The **high** and **low** level states are normal logic levels for data, address or control signals. The **high impedance** state is an electrical open-circuit condition. The **high impedance** state is provided to keep the device electrically isolated from the system. The tristate devices will normally remain in the **high impedance** state and their pins are physically connected in the system bus but electrically isolated. In the **high impedance** state, they cannot receive or send any signal or information. These devices are provided with chip enable/chip select pins. When the signal at this pin is asserted to the right level, they come out from the **high impedance** state to normal levels.

1.1.2 EVOLUTION OF MICROPROCESSORS

History tells us that it was the ancient Babylonians who first began using the abacus (a primitive calculator made of beads) in about 500 BC. This simple calculating machine eventually sparked the human mind into the development of calculating machines that use gears and wheels (Blaise Pascal in 1642). The giant computing machines of the 1940s and 1950s were constructed with relays and vacuum tubes. Next, the transistor and solid-state electronics were used to build the mighty computers of the 1960s. Finally, the advent of the **I**ntegrated **C**ircuit (IC) led to the development of the microprocessor and microprocessor-based computer systems.

In 1971, INTEL Corporation released the world's first microprocessor the INTEL 4004, a 4-bit microprocessor. It addresses 4096 memory locations of 4-bit word size. The instruction set consists of 45 different instructions. It is a monolithic IC employing large-scale integration in PMOS technology. The INTEL 4004 was soon followed by a variety of microprocessors, with most of the major semiconductor manufacturers producing one or more types.

First-Generation Microprocessors

The microprocessors introduced between 1971 and 1973 were the first-generation processors. They were designed using PMOS technology. This technology provided low cost, slow speed and low output currents and was not compatible with TTL (**T**ransistor **T**ransistor **L**ogic) levels.

The first-generation processors required a lot of additional support ICs to form a system, sometimes as high as 30 ICs. The 4-bit processors are provided with only 16 pins, but 8-bit and 16-bit processors are provided with 40 pins. Due to limitations of pins, the signals are multiplexed. A list of first-generation microprocessors are as follows:

1.	**INTEL 4004**	4-bit processors
2.	**INTEL 4040**	
3.	**FAIR CHILD PPS - 25**	
4.	**NATIONAL IMP - 4**	
5.	**ROCKWELL PPP - 4**	
6.	**MICRO SYSTEMS INTL. MC - 1**	
7.	**INTEL 8008**	8-bit processors
8.	**NATIONAL IMP - 8**	
9.	**ROCKWELL PPS - 8**	
10.	**AMI 7200**	
11.	**MOSTEK 5065**	
12.	**NATIONAL IMP/16**	16-bit processors
13.	**NATIONAL PACE**	

Second-Generation Microprocessors

The second-generation microprocessors appeared in 1973 and were manufactured using the NMOS technology. The NMOS technology offers faster speed and higher density than PMOS and it is TTL compatible. Some of the second-generation processors are as follows:

1. **INTEL 8080**
2. **INTEL 8085**
3. **FAIRCHILD F - 8**
4. **MOTOROLA M6800**
5. **MOTOROLA M6809**
6. **NATIONAL CMP -8**
7. **RCA COSMAC**
8. **MOS TECH. 6500**
9. **SIGNETICS 2650**

} 8-bit processors

10. **ZILOG Z80**
11. **INTERSIL 6100**
12. **TOSHIBA TLCS - 12**

} 12-bit processors

13. **TI TMS 9900**
14. **DEC - W.D. MCP - 1600**
15. **GENERAL INSTRUMENT CP 1600**
16. **DATA GENERAL μN601**

} 16-bit processors

Characteristics of Second-Generation Microprocessors

1. **Larger chip size (170 × 200 mil). [1mil = 10^{-3} inch]**
2. **40 pins.**
3. **More numbers of on-chip decoded timing signals.**
4. **The ability to address large memory spaces.**
5. **The ability to address more IO ports.**
6. **Faster operation.**
7. **More powerful instruction set.**
8. **A greater number of levels of subroutine nesting.**
9. **Better interrupt-handling capabilities.**

Third-Generation Microprocessors

After 1978, the third-generation microprocessors were introduced. These are 16-bit processors and designed using HMOS (**H**igh density **MOS**) technology. Some of the third-generation microprocessor are as follows:

1. **INTEL 8086**
2. **INTEL 8088**
3. **INTEL 80186**
4. **INTEL 80286**
5. **MOTOROLA 68000**
6. **MOTOROLA 68010**
7. **ZILOG Z8000**
8. **NATIONAL NS 16016**
9. **TEXAS INSTRUMENTS TMS 99000**

The HMOS technology offers better **S**peed **P**ower **P**roduct (SPP) and higher packing density than NMOS.

Speed Power Product (SPP) = Speed × Power
Unit of SPP = Nanoseconds × Milliwatts
= Picojoules

1. **Speed Power Product of HMOS is four times better than NMOS.**
   SPP of NMOS = 4 picojoules (pJ)
   SPP of HMOS = 1 picojoules (pJ)
2. **Circuit densities provided by HMOS are approximately twice those of NMOS.**
   Packing density of NMOS = 1852.5 gates/mm^2
   Packing density of HMOS = 4128 gates/mm^2 (1 mm = 10^{-6} metre)

Characteristics of Third-Generation Microprocessors

- **Provided with 40/48/64 pins.**
- **High speed and very strong processing capability.**
- **Easier to program.**
- **Allow for dynamically relocatable programs.**
- **Size of internal registers are 8/16/32 bits.**
- **The processor has multiply/divide arithmetic hardware.**
- **Physical memory space is from 1 to 16 megabytes.**
- **The processor has segmented addresses and virtual memory features.**
- **More powerful interrupt-handling capabilities.**
- **Flexible IO port addressing.**
- **Different modes of operations (e.g., user and supervisor modes of M68000).**

Fourth-Generation Microprocessors

The fourth-generation microprocessors were introduced in the year 1980. These generation processors are 32-bit processors and are fabricated using the low-power version of the HMOS technology called HCMOS. These 32-bit microprocessors have increased sophistications that compete strongly with mainframes. Some of the fourth-generation microprocessors are as follows:

1. **INTEL 80386**
2. **INTEL 80486**
3. **NATIONAL NS16032**
4. **MOTOROLA M68020**
5. **BELLMAC - 32**
6. **MOTOROLA M68030**
7. **MOTOROLA MC88100**

Characteristics of Fourth-Generation Microprocessors

1. **Physical memory space of 2^{24} bytes = 16 MB (megabytes).**
2. **Virtual memory space of 2^{40} bytes = 1TB (terabytes).**
3. **Floating-point hardware is incorporated.**
4. **Supports increased number of addressing modes.**

Fifth-Generation Microprocessors

In microprocessor technology, INTEL has taken a leading edge and is developing more and more new processors. The latest processor by INTEL is the **pentium** which is considered a fifth- generation processor. The pentium is a 32-bit processor with 64-bit data bus and is available in a wide range of clock speeds from 60 MHz to 3.2 GHz. With improvement in semiconductor technology, the processing speed of microprocessors has increased tremendously. The 8085 released in the year 1976 executes 0.5 **M**illion **I**nstructions **P**er **S**econd (0.5 MIPS). The 80486 executes 54 Million Instructions Per Second. The pentium is optimized to execute two instructions in one clock period. Therefore, a pentium processor working at 1 GHz clock can execute 2000 **M**illion **I**nstructions **P**er **S**econd (2000 MIPS). The various processors released by INTEL are listed in Appendix I.

1.1.3 BASIC FUNCTIONAL BLOCKS OF A MICROPROCESSOR

[AU May'15, 16 marks]

A microprocessor is a programmable IC which is capable of performing arithmetic and logical operations. The basic functional block diagram of a microprocessor is shown in Fig. 1.1.

The basic functional blocks of a microprocessor are ALU, Flag register, Register array, **P**rogram **C**ounter (PC)/**I**nstruction **P**ointer (IP), Instruction decoding unit, and the Timing and Control unit.

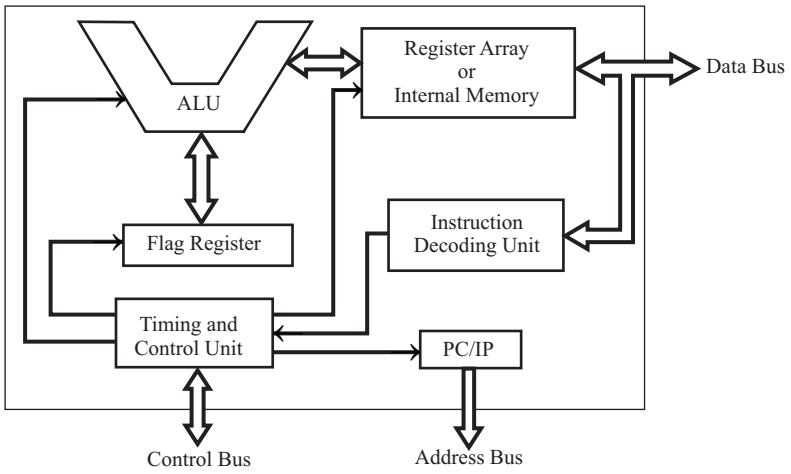


Fig. 1.1 : Block diagram showing basic functional blocks of a microprocessor.

The ALU is the computational unit of the microprocessor which performs arithmetic and logical operations on binary data. The various conditions of the result are stored as status bits called flags in the flag register. For example, consider sign flag. One of the bit positions of the flag register is called sign flag and it is used to store the status of the sign of the result of the ALU operation (output data of ALU). If the result is negative then "1" is stored in the sign flag and if the result is positive then "0" is stored in the sign flag.

The register array is the internal storage device and so it is also called internal memory. The input data for ALU, the output data of ALU (result of computations) and any other binary information needed for processing are stored in the register array.

For any microprocessor, there will be a set of instructions given by its manufacturer. For doing any useful work with the microprocessor, we have to first write a program using these instructions, and store them in a memory device external to the microprocessor.

The instruction pointer generates the address of the instructions to be fetched from the memory and sends it through the address bus to the memory. The memory will send the instruction codes and data through the data bus. The instruction codes are decoded by the decoding unit and it sends information to the timing and control unit. The data is stored in the register array for processing by the ALU.

The control unit will generate the necessary control signals for internal and external operations of the microprocessor.

1.2 INTRODUCTION TO 8086

INTEL 8086 is the first 16-bit processor released by INTEL in the year 1978. The 8086 was designed using the HMOS technology and it is now manufactured using HMOS III technology

and contains approximately 29,000 transistors. The 8086 is packed in a 40-pin DIP and requires a single 5 V supply.

The 8086 does not have an internal clock circuit. The 8086 requires an external asymmetric clock source with 33% duty cycle. The 8284 clock generator is used to generate the required clock for 8086. The maximum internal clock of 8086 is 5 MHz. The other versions of 8086 with different clock rates are 8086-1, 8086-2 and 8086-4 with maximum internal clock frequency of 10 MHz, 8 MHz and 4 MHz, respectively.

The 8086 uses a 20-bit address to access memory and hence it can directly address up to one megabyte (2^{20} = 1 Mega) of memory space. One megabyte (1 MB) of addressable memory space of 8086 are organized as two memory banks of 512 kilobytes each (512 kB + 512 kB = 1 MB). The memory banks are called even (or lower) bank and odd (or upper) bank. The address line A_0 is used to select the even bank and the control signal $\overline{BHE}$ is used to select the odd bank.

For accessing IO-mapped devices, the 8086 uses a separate 16-bit address and so the 8086 can generate 64 k(2^{16}) IO addresses. The signal M/$\overline{IO}$ is used to differentiate the memory and IO addresses. For memory address, the signal M/$\overline{IO}$ is asserted **high** and for IO address the signal M/$\overline{IO}$ is asserted **low** by the processor.

The 8086 can operate in two modes: minimum mode and maximum mode. The mode is decided by a signal at MN/$\overline{MX}$ pin. When the MN/$\overline{MX}$ is tied **high**, it works in minimum mode and the system is called a uniprocessor system. When MN/$\overline{MX}$ is tied **low**, it works in maximum mode and the system is called a multiprocessor system. Usually, the pin MN/$\overline{MX}$ is permanently tied to **low** or **high** so that the 8086 system can work in any one of the two modes. The 8086 can work with the 8087 coprocessor in maximum mode. In this mode, an external bus controller 8288 is required to generate bus control signals.

The 8086 has two families of processors. They are 8086 and 8088. The 8088 uses an 8-bit data bus externally but the 8086 uses a 16-bit data bus externally. The 8086 accesses memory in words but the 8088 accesses memory in bytes. IBM designed its first **P**ersonal **C**omputer (PC) using an INTEL 8088 microprocessor as the CPU.

1.3 8086 - MICROPROCESSOR ARCHITECTURE *[AU May'15, 10 marks]*

The 8086 has a pipelined architecture. In pipelined architecture, the processor will have a number of functional units and the execution time of each functional unit overlaps. Each functional unit works independently most of the time. The simplified block diagram of the internal architecture of an 8086 is shown in Fig. 1.2. The architecture of the 8086 can be internally divided into two separate functional units: **B**us **I**nterface **U**nit (BIU) and **E**xecution **U**nit (EU).

The BIU fetches instructions, reads data from memory and IO ports, and writes data to memory and IO ports. The BIU contains segment registers, instruction pointer, instruction queue, address generation unit and bus control unit. The EU executes instructions that have already been fetched by the BIU. The BIU and EU function independently.

The instruction queue is a FIFO (**F**irst-**I**n-**F**irst-**O**ut) group of registers. The size of the queue is 6 bytes. The BIU fetches instruction code from the memory and stores it in the queue. The EU fetches instruction codes from the queue.

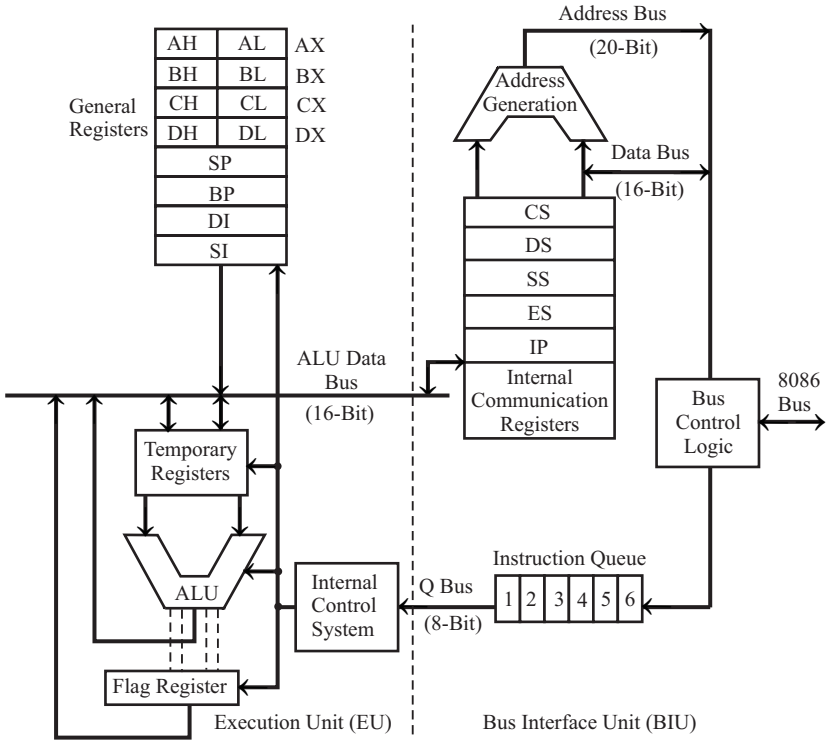


Fig. 1.2 : Internal architecture of 8086.

The BIU has four numbers of 16-bit segment registers. They are **C**ode **S**egment (CS) register, **D**ata **S**egment (DS) register, **S**tack **S**egment (SS) register and **E**xtra **S**egment (ES) register. The 8086 memory space can be divided into segments of 64 kilobytes (64 kB). The 4 segment registers are used to hold four segment base addresses. Hence the 8086 can directly address 4 segments of 64 kB at any time instant ($4 \times 64 = 256$ kB within 1 MB memory space). This feature of 8086 allows the system designer to allocate separate areas for storing program codes and data.

The contents of segment registers are programmable. Hence, the processor can access the code and data in any part of the memory by changing the contents of the segment registers. The memory segment can be continuous, partially overlapped, fully overlapped or disjoint.

> ***Note :*** *Since segment registers are programmable, it is possible to design multitasking and multiuser system using 8086. The program code and data for each task/user can be stored in separate segments. The program execution can be switched from one task/user to another by changing the content of the segment registers.*

The dedicated address generation unit generates the 20-bit physical address from the segment base and an offset or effective address. The segment base address is logically shifted left four times and added to the offset *[logically shifting left four times is equal to multiplying by 16_{10}]*.

The address for fetching instruction codes is generated by logically shifting the content

of the CS to the left four times and then adding it to the content of the IP (**I**nstruction **P**ointer). The IP holds the offset address of the program codes. The content of IP gets incremented by two after every bus cycle *[in one bus cycle, the processor fetches two bytes of the instruction code]*.

The data address is computed by using the content of DS or ES as base address and an offset or effective address specified by the instruction. The stack address is computed by using the content of the SS as base address and the content of the SP (**S**tack **P**ointer) as the offset address or effective address.

The bus control logic of the BIU generates all the bus control signals such as read and write signals for memory and IO. The EU consists of ALU, flag register and general-purpose registers. The EU decodes and executes the instructions. A decoder in the EU control system translates the instructions.

The EU has a 16-bit ALU to perform arithmetic and logical operations. The EU has eight numbers of 16-bit general-purpose registers. They are AX, BX, CX, DX, SP, BP, SI and DI.

TABLE 1.1: SPECIAL FUNCTIONS OF 8086 REGISTERS

Register	Name of the register	Special function
AX	16-bit accumulator	Stores 16-bit result of certain arithmetic and logical operations.
AL	8-bit accumulator	Stores 8-bit result of certain arithmetic and logical operations.
BX	Base register	Used to hold the base value in base addressing mode to access memory data
CX	Count register	Used to hold the count value in SHIFT,ROTATE and LOOP instructions
DX	Data register	Used to hold data for multiplication and division operations.
SP	Stack pointer	Used to hold the offset address of top of the stack memory.
BP	Base pointer	Used to hold the base value in base addressing using stack segment register to access data from stack memory.
SI	Source index	Used to hold the index value of source operand (data) for string instructions.
DI	Destination index	Used to hold the index value of destination operand (data) for string instructions.

Some of the 16-bit registers can also be used as two numbers of 8-bit registers as follows:

AX – can be used as AH and AL CX – can be used as CH and CL

BX – can be used as BH and BL DX – can be used as DH and DL

The general-purpose registers can be used for data storage, when they are not involved in special functions assigned to them. These registers are named after special functions carried out by each one of them as given in Table 1.1.

8086 Flag Register

The size of an 8086 flag register is 16-bit and in this, nine bits are defined as flags. The six flags are used to indicate the status of the result of the arithmetic or logical operations. Three flags are used to control the processor operation and so they are also called control bits. The various flags of an 8086 processor and their bit position in the flag register are shown in Fig. 1.3.

The **C**arry **F**lag (CF) is set if there is a carry from addition or borrow from subtraction. The **A**uxiliary carry **F**lag (AF) is set if there is a carry from low nibble to high nibble of the low order 8-bit of a 16-bit number.

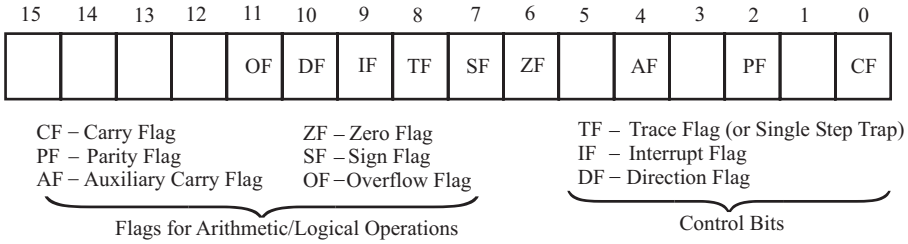


Fig. 1.3 : Bit positions of various flags in the flag register of 8086.

The **O**verflow **F**lag (OF) is set to **one** if there is an arithmetic overflow, that is, in signed arithmetic if the size of the result exceeds the maximum range. The **S**ign **F**lag (SF) is set to **one** if the most significant bit of the result is **one** and SF is cleared to **zero** for non-negative result. The **P**arity **F**lag (PF) is set to **one** if the result has even parity and PF is cleared to **zero** for odd parity of the result. The **Z**ero **F**lag (ZF) is set to **one** if the result is zero and ZF is cleared to **zero** for non zero result.

The three control bits in the flag register can be set or reset by the programmer. The **D**irection **F**lag (DF) is set to **one** for autodecrement and DF is reset to **zero** for autoincrement of SI and DI registers during string data accessing. Setting the **I**nterrupt **F**lag (IF) to **one** causes the 8086 to recognize external maskable interrupts and clearing IF to **zero** disables the interrupts.

Setting the **T**race **F**lag (TF) to **one** places the 8086 in the single-step mode. In this mode, the 8086 generates an internal interrupt after execution of each instruction. The single stepping is used for debugging a program.

1.3.1 INSTRUCTION AND DATA FLOW IN 8086

The 8086 microprocessor allows the user to define different memory areas for storing program and data. The program memory can be accessed by using the CS-register and the data memory can be accessed by using the DS, ES and SS registers.

The program instructions are stored in the program memory which is an external device. To execute a program in 8086, the base address and offset address of the first instruction of the program should be loaded in the CS-register and IP, respectively. The 8086 computes the 20-bit physical

address of the program instruction by multiplying the content of the CS-register by 16_{10} and adding it to the content of IP. The 20-bit physical address is given out on the address bus. Then $\overline{RD}$ is asserted **low**. Also, other control signals necessary for program memory read operation are asserted. The IP is incremented by two to point to the next instruction or the next word of the same instruction.

The address and control signals enable the memory to output one word (two bytes) of program memory on the data bus. After a predefined time, the $\overline{RD}$ is asserted **high** and at this instant, the content of the data bus is latched into two empty locations of the instruction queue. Then the BIU starts fetching the next word of the program code as explained above. The BIU keeps on fetching the program codes, word by word from consecutive memory locations whenever two locations of the queue are empty. When a branch instruction is encountered, the queue is emptied and then filled with program codes from the new address loaded in CS and IP by the branch instruction.

The EU reads the program instructions from a queue, and decodes and executes them one by one. If the execution of an instruction requires data from memory (or to store data in memory) then the BIU is interrupted to read (or write) data in memory. When BIU is interrupted, it completes the fetching of the current instruction word and then starts reading/writing the data by generating a 20-bit data memory address. The 20-bit data memory address is obtained by multiplying the content of the segment base register specified by the instruction by 16_{10} and adding it to an effective or offset address specified by the instruction.

1.3.2 EVEN AND ODD MEMORY BANKS

The 8086 microprocessor uses a 20-bit address to access memory. With a 20-bit address, the processor can generate 2^{20} = 1 Mega address. The basic memory word size of the memories used in the 8086 system is 8-bit or 1-byte (i.e., in one memory location an 8-bit binary information can be stored). Hence, the physical memory space of the 8086 is 1 MB (1 megabyte).

For the programmer, the 8086 memory address space is a sequence of one megabyte in which one location stores an 8-bit binary code/data and two consecutive locations store 16-bit binary code/data. But physically (i.e., in the hardware), the 1 MB memory space is divided into two banks of 512 kB (512 kB + 512 kB = 1 MB). The two memory banks are called Even (or Lower) bank and Odd (or Upper) bank. The organization of even and odd memory banks in the 8086-based system is shown in Fig. 1.4.

The 8086-based system will have two sets of memory IC's, one set for the even bank and another set for the odd bank. The data lines D_0-D_7 are connected to the even bank and the data lines D_8-D_{15} are connected to the odd bank. The even memory bank is selected by the address line A_0 and the odd memory bank is selected by the control signal $\overline{BHE}$. The memory banks are selected when these signals are **low** (active low). Any memory location in the memory bank is selected by the address line A_1 to A_{19}.

The organization of memory into two banks and providing bank select signals allows the programmer to read/write the byte (8-bit) operand in any memory address through a 16-bit data bus. It also allows the programmer to read/write the word (16-bit) operand starting from an even address or odd address.

The memory access for byte and word operand from the even and odd bank by the 8086 processor will be as follows:

Case i **: Byte access from the even bank**

For read/write operation of a byte in the even memory address, A_0 is asserted **low** and $\overline{BHE}$ is asserted **high** (i.e., $A_0 = 0$ and $\overline{BHE} = 1$). Now the even bank alone is enabled and the data transfer takes place through D_0-D_7 data lines.

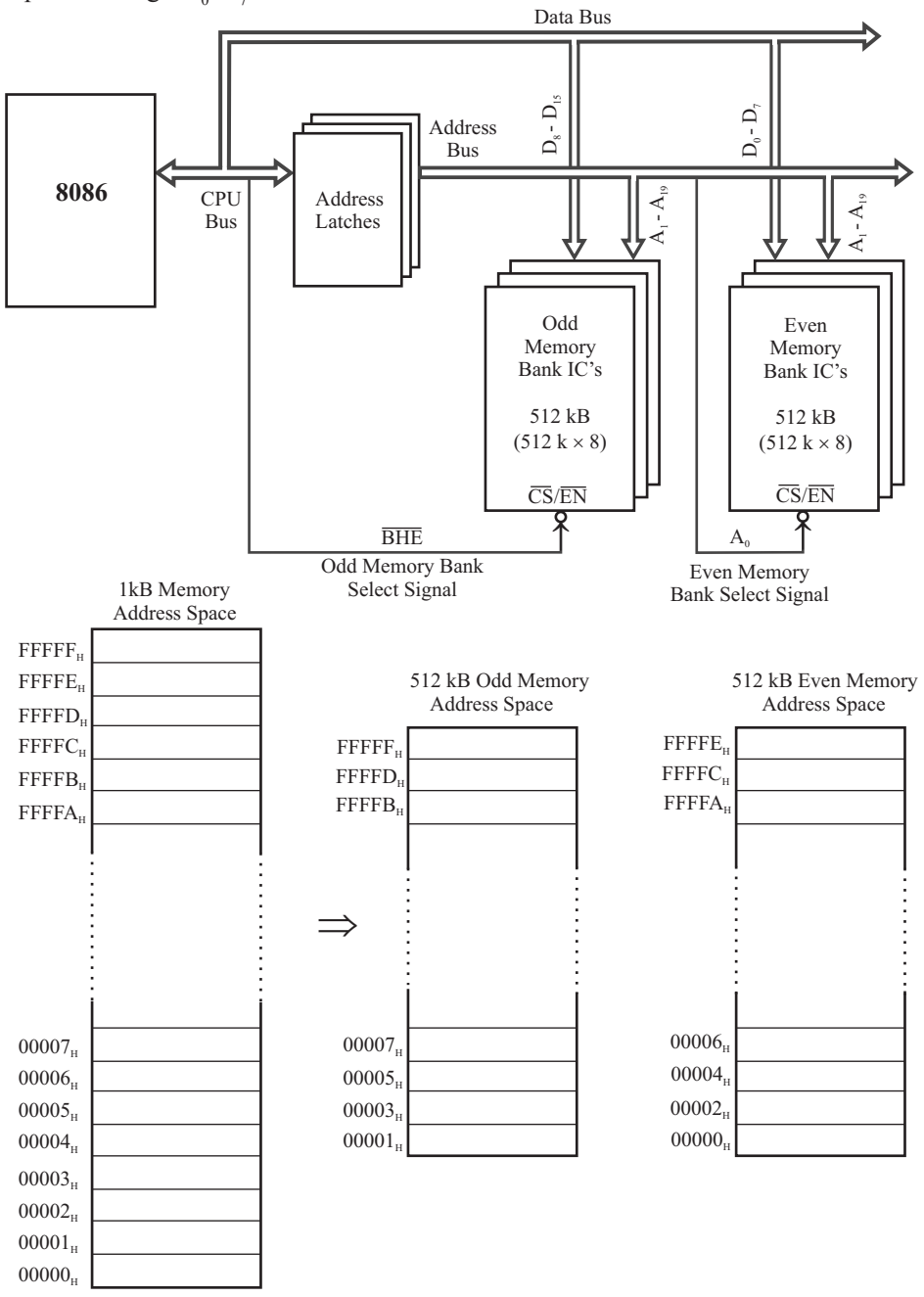


Fig. 1.4 : Organization of even and odd memory banks in the 8086-based system.

Case ii **: Byte access from the odd bank**

For read/write operation of a byte in the odd memory address, A_0 is asserted **high** and $\overline{BHE}$ is asserted **low** (i.e., $A_0 = 1$ and $\overline{BHE} = 0$). Now, the odd bank alone is enabled and the data transfer takes place through D_8-D_{15} data lines.

Case iii **: Word access from the even boundary**

For read/write operation of a word (16-bit) in the even boundary [i.e., low byte in even address and high byte in next address (odd address)], both A_0 and $\overline{BHE}$ are asserted **low** (i.e., $A_0 = 0$ and $\overline{BHE} = 0$). Now both the memory banks are enabled simultaneously and the processor read/write the 16-bit operand in one bus cycle through D_0-D_{15} data lines.

Case iv **: Word access from the odd boundary**

For read/write operation of a word (16-bit) in the odd boundary [i.e., low byte in odd address and high byte in the next address (even address)], the processor executes two bus cycles to read/write the word (16-bit) operand. In the first bus cycle, A_0 is asserted **high** and $\overline{BHE}$ is asserted **low** (i.e., $A_0 = 1$ and $\overline{BHE} = 0$). Now the odd bank alone is enabled and the low byte of the 16-bit operand is read/write through D_8-D_{15} data lines. In the second bus cycle, A_0 is asserted **low** and $\overline{BHE}$ is asserted **high** (i.e., $A_0 = 0$ and $\overline{BHE} = 1$). Now the even bank alone is enabled and the high byte of the 16-bit operand is read/write through D_0-D_7 data lines.

The status of A_0 and $\overline{BHE}$ for byte and word memory access are listed in Table 1.2.

TABLE 1.2: STATUS OF A_0 AND $\overline{BHE}$ DURING MEMORY ACCESS

Memory bank	Operand type	Status of		Data lines used for memory access	No. of bus cycle
		A_0	$\overline{BHE}$		
Even	Byte	0	1	D_0-D_7	One
Odd	Byte	1	0	D_8-D_{15}	One
Even	Word	0	0	D_0-D_{15}	One
Odd	Word	1	0	D_8-D_{15}	First cycle
		0	1	D_0-D_7	Second cycle

Note : *1. The processor may access the low byte operand via the upper data lines and high byte operand via the lower data lines, but while placing the operand in the registers, it places in the appropriate locations.*

2. When the word operand is accessed from an odd boundary the instruction execution will take extra time due to two bus cycles of memory access.

1.4 ADDRESSING MODES *[AU May'15, 2 marks]*

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called addressing. The 8086 has 12 addressing modes and they can be classified into the following five groups.

1. Register addressing	**Group I : Addressing modes for register and immediate data**
2. Immediate addressing	
3. Direct addressing	**Group II : Addressing modes for memory data**
3. Register indirect addressing	
5. Based addressing	
6. Indexed addressing	
7. Based index addressing	
8. String addressing	
9. Direct IO port addressing	**Group II : Addressing modes for memory data**
10. Indirect IO port addressing	
11. Relative addressing	**Group IV : Relative addressing mode**
12. Implied addressing	**Group V : Implied addressing mode**

Notes:

1. *The "register" or "register + constant" enclosed by square brackets in the operand field of instructions refer to the method of effective address calculation of memory. The 16-bit constant enclosed by square brackets in the operand field of instructions refers to the effective address of memory data. The 8-bit/16-bit constants which are not enclosed by square brackets in the operand field refer to immediate data.*
2. *The term MA used in the symbolic description of instructions refer to physical memory address of data segment memory, MA_S refers to physical memory address of stack segment memory; and MA_E refers to physical memory address of extra segment memory.*
3. *The register/memory enclosed by brackets in symbolic description refers to the content of register/memory.*
4. *For hexadecimal constant (data/address), the letter H is included at the end of 8-bit/16-bit constants(data/address), and the numeral 0 is included in the front of the hexadecimal constant starting with A through F.*

Register Addressing

In register addressing, the instruction will specify the name of the register which holds the data to be operated by the instruction.

Examples:

(a) MOV CL,DH **(CL) ← (DH)**

The content of the 8-bit register DH is moved to another 8-bit register CL.

(b) MOV BX,DX **(BX) ← (DX)**

The content of the 16-bit register DX is moved to another 16-bit register BX.

Immediate Addressing

In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction.

Examples:

(a) MOV DL,08H **(DL) ← 08_H**

The 8-bit data (08_H) given in the instruction is moved to DL-register.

(b) MOV AX,0A9FH **(AX) ← $0A9F_H$**

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX-register.

Direct Addressing

*In direct addressing, an unsigned 16-bit displacement or signed 8-bit displacement will be specificied in the instruction.The displacement is the **E**ffective **A**ddress (EA) or offset. In case of 8-bit displacement, the effective address is obtained by sign extending the 8-bit displacement to 16-bit*

The 20-bit physical address of memory is calculated by multiplying the content of DS-register by 16_{10} (or 10_H)and adding to effective address. When segment override prefix is employed, the content of the segment register specified in the override prefix will be used for segment base address calculation instead of DS-register.

Examples:

(a) MOV DX,[08H]

EA = 0008_H (sign extended 8-bit displacement)
BA = (DS) $\times 16_{10}$; MA = BA + EA
(DX) ← (MA) or DL ← (MA)
DH ← (MA+1)

*The **E**ffective **A**ddress (EA) is obtained by sign extending the 8-bit displacement given in the instruction to 16-bit. The segment **B**ase **A**ddress (BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress (MA) is computed by adding the **E**ffective **A**ddress (EA) to segment **B**ase **A**ddress (BA).*

The content of memory whose address is calculated as explained above is moved to DL-register and the content of the next memory location is moved to DH-register.

(b) MOV AX, [089DH]

EA = $089D_H$; BA = (DS) $\times$ 16_{10} ; MA = BA + EA
(AX) ← (MA) or (AL) ← (MA)
(AH) ← (MA+1)

*Here, the 16-bit displacement given in the instruction is the effective address. The segment **B**ase **A**ddress(BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress (MA) is computed by adding the **E**ffective **A**ddress (EA) to segment **B**ase **A**ddress (BA).*

The content of memory whose address is calculated as explained above is moved to AL-register and the content of the next memory location is moved to AH-register.

Register Indirect Addressing

*In register indirect addressing, the name of the register which holds the **E**ffective **A**ddress (EA) will be specified in the instruction. The registers used to hold the effective address are BX, SI and DI. The content of DS is used for segment base address calculation. When segment override prefix is employed, the content of segment register specified in the override prefix will be used for base address calculation instead of DS-register.*

The base address is obtained by multiplying the content of the segment register by 16_{10}. The 20-bit physical address of the memory is computed by adding the effective address to the base address.

Examples:

(a) MOV CX, [BX]

EA = (BX) ; BA = (DS) $\times$ 16_{10} ; MA = BA + EA
(CX) ← (MA) or (CL) ← (MA)
(CH) ← (MA+1)

*The content of BX is the **E**ffective **A**ddress (EA). The segment Base Address (BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress (MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to CL-register and the content of the next memory location is moved to CH-register.

(b) MOV AX,[SI]

EA = (SI) ; BA = (DS) × 16_{10} ; MA = BA + EA

(AX) ← (MA) or (AL) ← (MA)

(AH) ← (MA + 1)

*The content of SI is the **E**ffective **A**ddress (EA). The segment **B**ase **A**ddress (BA) is computed by multiplying the content of DS by 16_{10}. The memory address is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to AL-register and the content of the next memory location is moved to AH-register.

Based Addressing

*In this addressing mode, the BX or BP-register is used to hold a base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. The displacement is added to the base value in BX or BP to obtain the **E**ffective **A**ddress(EA). In case of 8-bit displacement, it is sign extended to 16-bit before adding to base value.*

When BX is used to hold the base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of DS by 16_{10} adding to EA.

When BP is used to hold the base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of SS by 16_{10} and adding to EA.

Example:

MOV AX, [BX+08H]

0008_H 08_H ; EA = (BX) + 0008_H

BA = (DS) × 16_{10} ; MA = BA + EA

(AX) ← (MA) or (AL) ← (MA)

(AH) ← (MA+1)

*The effective address is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of BX-register. The Base Address (BA) is obtained by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress (MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to AL-register and the content of the next memory is moved to AH-register.

Indexed Addressing

*In this addressing mode, the SI or DI-register is used to hold an index value for memory data and a signed 8-bit displacement or unsigned 16-bit displacement will be specified in the instruction. The displacement is added to index value in SI or DI-register to obtain the **E**ffective **A**ddress (EA). In case of 8-bit displacement, it is sign extended to 16-bit before adding to the index value.*

*The 20-bit memory address is calculated by multiplying the content of **D**ata **S**egment (DS) by 16_{10} and adding to EA.*

Note: *In general, Effective Address = Reference + Modifier.*

In this context, the based and indexed addressing looks similar, but in based addressing, the base value is the reference and displacement is the modifier, whereas in indexed addressing, displacement is the reference and index value is the modifier.