Introduction to Microprocessor

- ✓ The microprocessor is a general purpose **programmable logic device.**
- ✓ It is the brain of the computer and it performs all the
  computational tasks,
  calculations
  data processing etc. inside the computer.
- ✓ In the computers, the most **popular type of the processor is the Intel Pentium.**
- ✓ The microprocessors can be classified based on the following features:

Instruction set:

It is the **set of the instructions that the microprocessor can execute.**

Bandwidth:

The **number of bits processed by the microprocessor in a single instruction**, or the number of bits which are **transfer via the bus in parallel.**
For example, the 8086 microprocessor called 16-bits, i.e. because it can process, send, and evaluate 16 bits in parallel.

Clock Speed:
It determines **the speed of the processing the instruction inside the processor** and it is measured in Hz.

Historical Overview:

- ✓ The first microprocessor is the **Intel 4004**, was a **4-bit microprocessor**. It has **12-bit address lines** to access 4096 memory location. The 4004 microprocessor has only **45 instructions.**

- ✓ The Intel released the **4040 microprocessor**, as updated version of 4004 with **enhancement in speed**, and without any improvement in word length and memory size.

- ✓ In **1971 announced the 8008 microprocessor**, **8-bit and faster** version of 4004 microprocessor. This version came up with **expanded memory size up to 16 Kbytes** and additional instructions to make **total of 48 instructions**.

- ✓ **In 1977 Intel introduced updated version of 8080,** which was 8085 microprocessor. The 8085 is an 8-bit microprocessor.

- ✓ **In 1978, Intel came out with the 8086** processor which is our study in this course.

- ✓ **Today**, the development of the microprocessors reached to the Pentium, Core-2-duo,Core i.

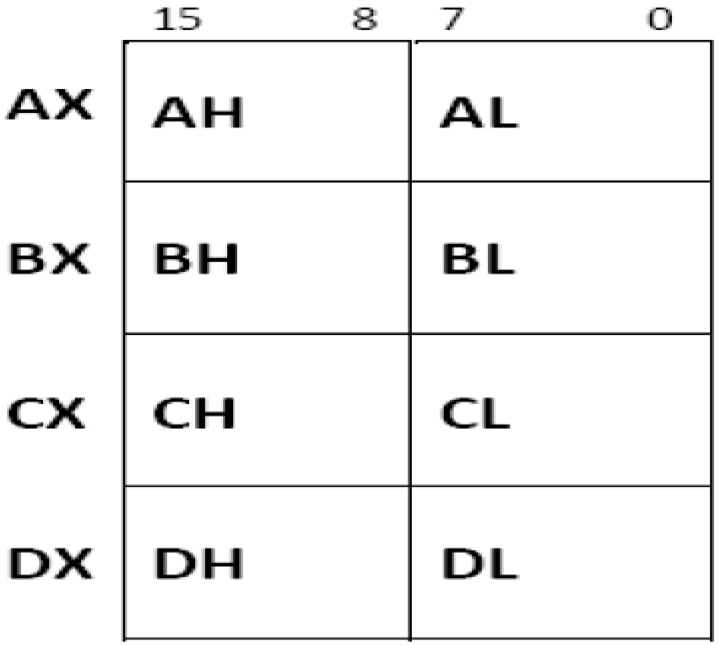
Basic Characteristics and features of the 8086 Microprocessor.

- ✓ The 8086 microprocessor is a **16-bit microprocessor**. The term “16-bit” means that its **arithmetic logic unit, internal registers and most of its instructions are designed to work with 16-bit binary words.**
- ✓ The 8086 has a **16-bit data bus**, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time.
- ✓ The 8086 has a **20-bit address bus**, so it can directly access 2^20 or 1,048,576 locations.
- ✓ The 8086 can generate **16-bit I/O address**; hence it can access 2^16 = 65536 I/O ports.
- ✓ The 8086 has **14 registers** each one with 16-bit.
- ✓ The 8086 has **multiplexed address and data bus** which reduced the number of pins needed.

- ✓ The 8086 is **designed to operate in two modes,** namely the **minimum mode** and the **maximum mode**. When **only one 8086 CPU is to be used** in a microcomputer system, the 8086 is used in the **minimum mode** of operation. In **multiprocessor** (more than one processor in the system) system 8086 operates in **maximum mode**.
- ✓ The 8086 supports **multiprogramming**. In multiprogramming, the code for two or more processes is in memory at the same time and is executed in a time-multiplexed fashion.
- ✓ An interesting feature of the 8086 is that it **fetches up to six instruction bytes from memory and queue stores** them in order to speed up instruction execution.

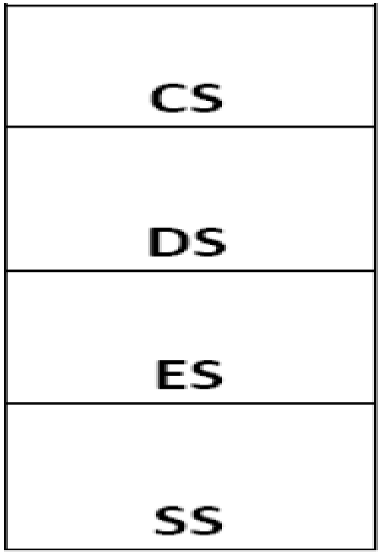
Register Organization of 8086

- ✓ The 8086 has a powerful set of registers. It includes
  general purpose registers,
  segment registers,
  pointers, index registers,
  flag register.
- ✓ The registers organization of 8086 is shown below. All registers are 16-bit registers.



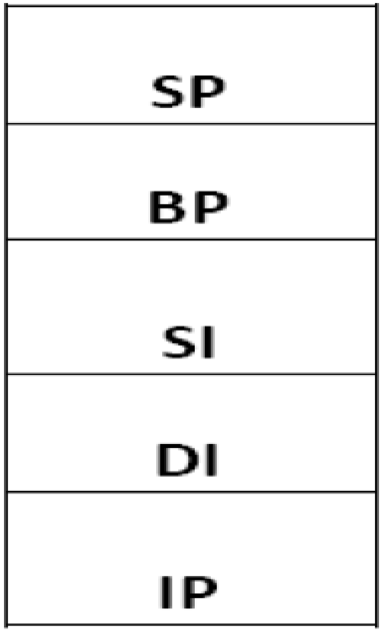
	15 8	7 0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

"General Purpose Register"



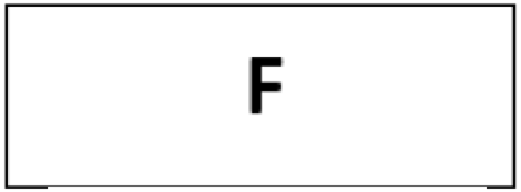
CS
DS
ES
SS

"Segment Register"



SP
BP
SI
DI
IP

"Pointer & Index Register"



“Flag Register”

General Purpose Register:

- ✓ The 8086 has four 16-bit general purpose registers labeled AX, BX, CX, and DX.
- ✓ Each 16-bit general purpose register can be split into two 8-bit registers.
- ✓ The letters L and H specify the lower and the higher bytes of a particular register. For example, the BH means the higher byte (8-bits) of the BX register and BL means the lower byte (8-bit) of the BX register.
- ✓ The letter X is used to specify the 16-bit register.
- ✓ The general purpose register are either used for

  holding data,

  variables

  intermediate result temporarily.

- ✓ The Accumulator register (AX) is used as 16-bit accumulator. It is used for all

  Input/output operations

  some arithmetic operations

  string manipulation

For example, multiply, and divide.

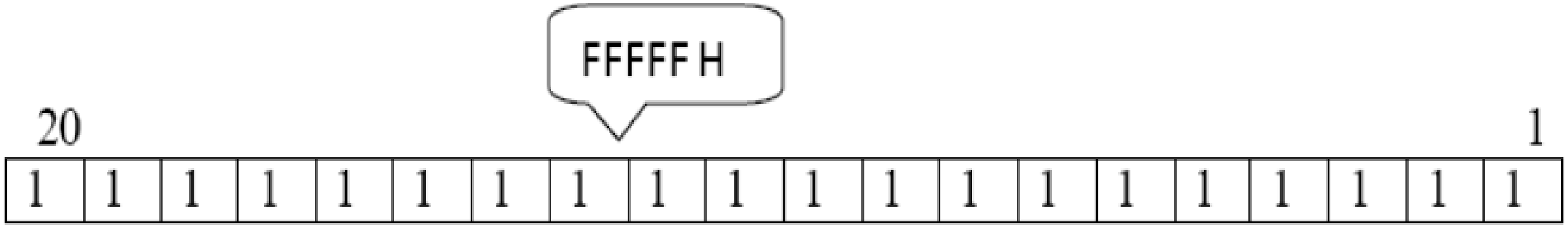
- ✓ The Base register(BX) is also **used as offset storage for generating physical addresses** in case of certain addressing modes.
- ✓ The Count register(CX) is also used as a default of counter in case of

  string and loop instructions.

  string manipulation and shift/rotate instructions.
- ✓ Data register(DX) can be **used as a port number in I/O operations.**

Segment Registers

- ✓ The physical address of the 8086 **is 20-bits wide** to access 1M byte memory locations.



20-bit physical address (in binary)

- ✓ We know that the **logical addresses (size of registers in 8086) are just 16-bits wide.**

- ✓ How the 20-bits Physical address obtained?

- ✓ And how 1Mbyte memory is addressed?

- ✓ The **1Mbyte memory is divided into segments**, with maximum size of segments with **16 Bits** (i.e., 2^16) as shown in the calculation below:

- ✓ We have 16-bits logical address (because the registers are in 16-bits), so we have maximum memory size locations which are addressed by 16-bits as below:

  2^16 = 65,536

- ✓ But in memory system (in binary system) it is written as 64K.

- ✓ Achieving 20-bits physical addresses will be discussed later.

- ✓ Thus, **any location within the segment can be accessed using 16-Bits**.

- ✓ The 8086 allows **only four active segments at a time,** as shown in the figure.

- ✓ For the selection of four active segments the 16-bit segment registers are provided by the **bus interface unit** (BIU) of the 8086.

The four segment registers are:

- ✓ Code Segment (CS) Register
- ✓ Data Segment (DS) Register
- ✓ Stack Segment (SS) Register
- ✓ Extra Segment (ES) Register

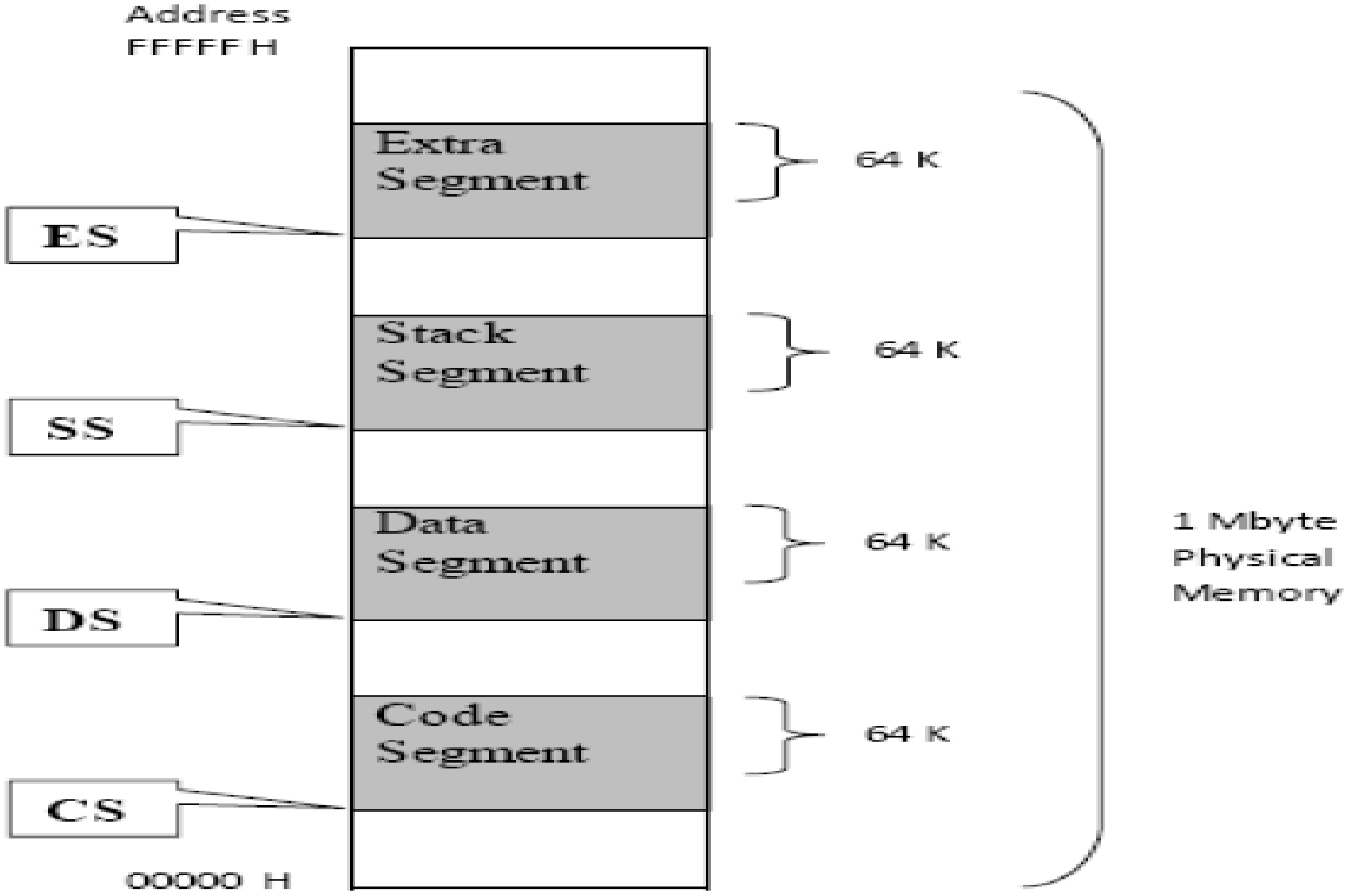
- ✓ These are used to **hold the 16-bits of the starting addresses** of the four memory segments.

- ✓ The starting address also **called base address or segment address**.

Functions of Segment Registers

- ✓ The CS register **holds the upper 16-bits of the starting address** of the segment from which **the BIU is currently fetching the instruction code byte**.

- ✓ The SS register is used for the **upper 16-bits of the starting address for the program stack (all stack related instructions will operate on stack).**
- ✓ ES register and DS register are used to hold the upper 16-bits of the starting address of the two memory segments **which are used for data**.



Note:

What is the relation between the size of the register and the probability of addressing locations?

Answer/

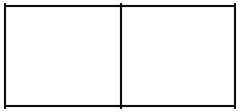
- When we have 1 bit, to find the probability of addressing location number we say that:

No. of address =2^(no. of bits) = 2^1 =2

- ✓ This mean that we have two addresses for 1 bit, the addresses may be 1 or 0.

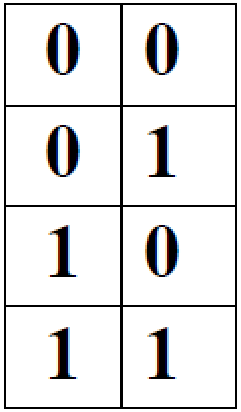
$\boxed{1}$ Or $\boxed{0}$

- ✓ **-Now if we have two bits???**



We have the following addresses:

2^2=4 addresses



0	0
0	1
1	0
1	1

- ✓ Now for the 8086 MP, there are 20-bits address bits, so we have the following locations in memory to be addressed:

2^20 =1,048,576 =1M locations

Pointers and Index Registers

- ✓ All **segment registers are 16-bit wide**. But it is necessary to **generate 20-bit address (physical address) on the address bus.**

- ✓ To get 20-bit physical address **one or more pointer or index registers are associated with each segment register.**

- ✓ The following pointer registers are associated with the adjacent segment registers:

  - ✓ **IP (Instruction Pointer) associated with Code (CS) Register**

  - ✓ **BP (Base Pointer) associated with Data (DS) Register**

  - ✓ **SP (Stack Pointer) associated with Stack (SS) Register**

The Index registers are

Source Index (SI)
Destination Index (DI) registers.

The index registers are used as a **general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing mode.**

Flag Register (FR)

- ✓ The content of the flag register of 8086 **indicate some conditions produced by the execution of an arithmetic or logic instruction.**

- ✓ It also **control some flag bits** the certain operations of the execution unit.

Architecture of 8086

The architecture of 8086 includes

- ✓ **Arithmetic Logic Unit (ALU)**
- ✓ **Flags**
- ✓ **General registers**
- ✓ **Instruction byte queue**
- ✓ **Segment registers**

- ✓ It is internally divided into two separate functional units.

- ✓ These are the **Bus Interface Unit (BIU) and the Execution Unit (EU).**

- ✓ These two functional units can work **simultaneously to increase system and hence throughput.**

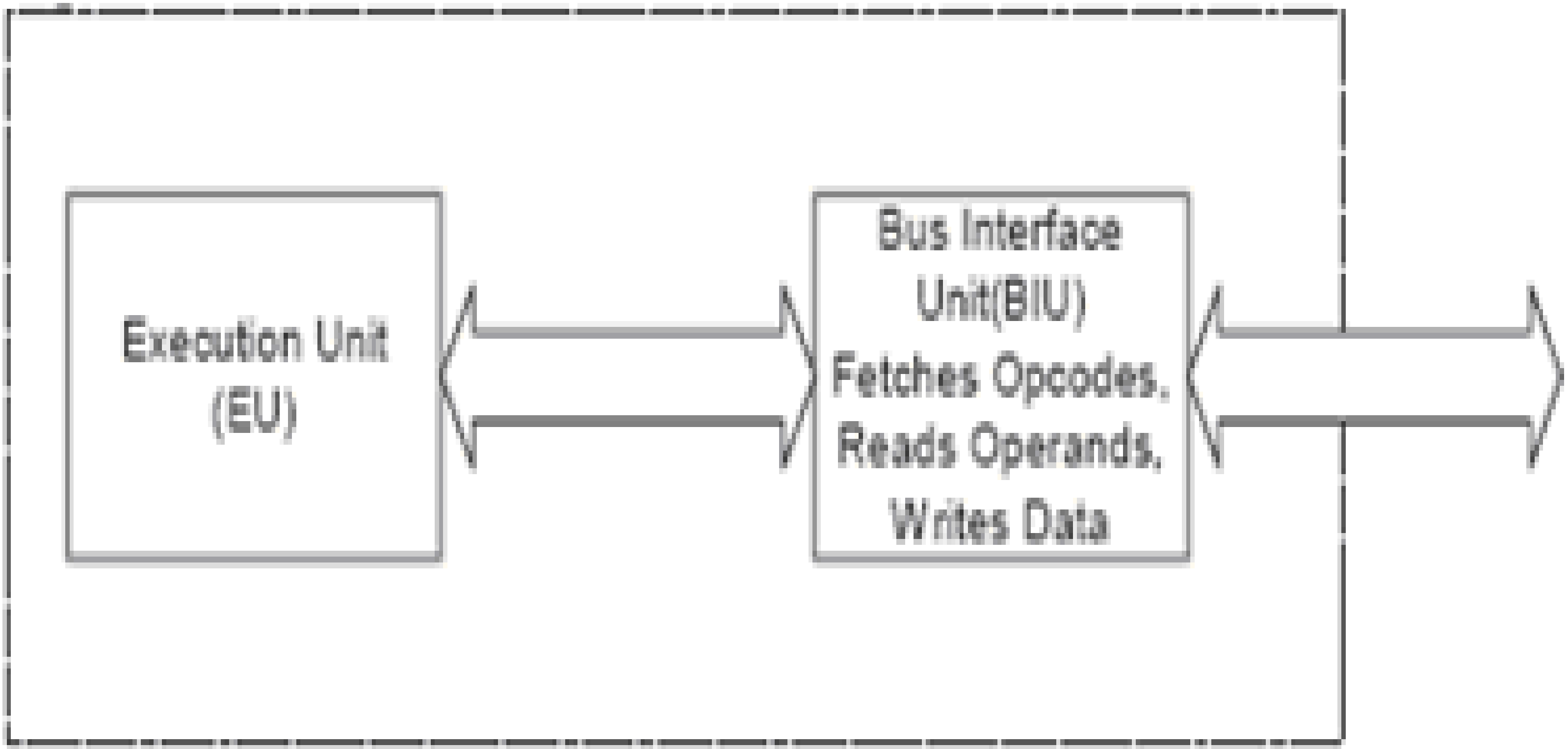
- ✓ The throughput is a measure of number of instructions executed per unit time.

Or

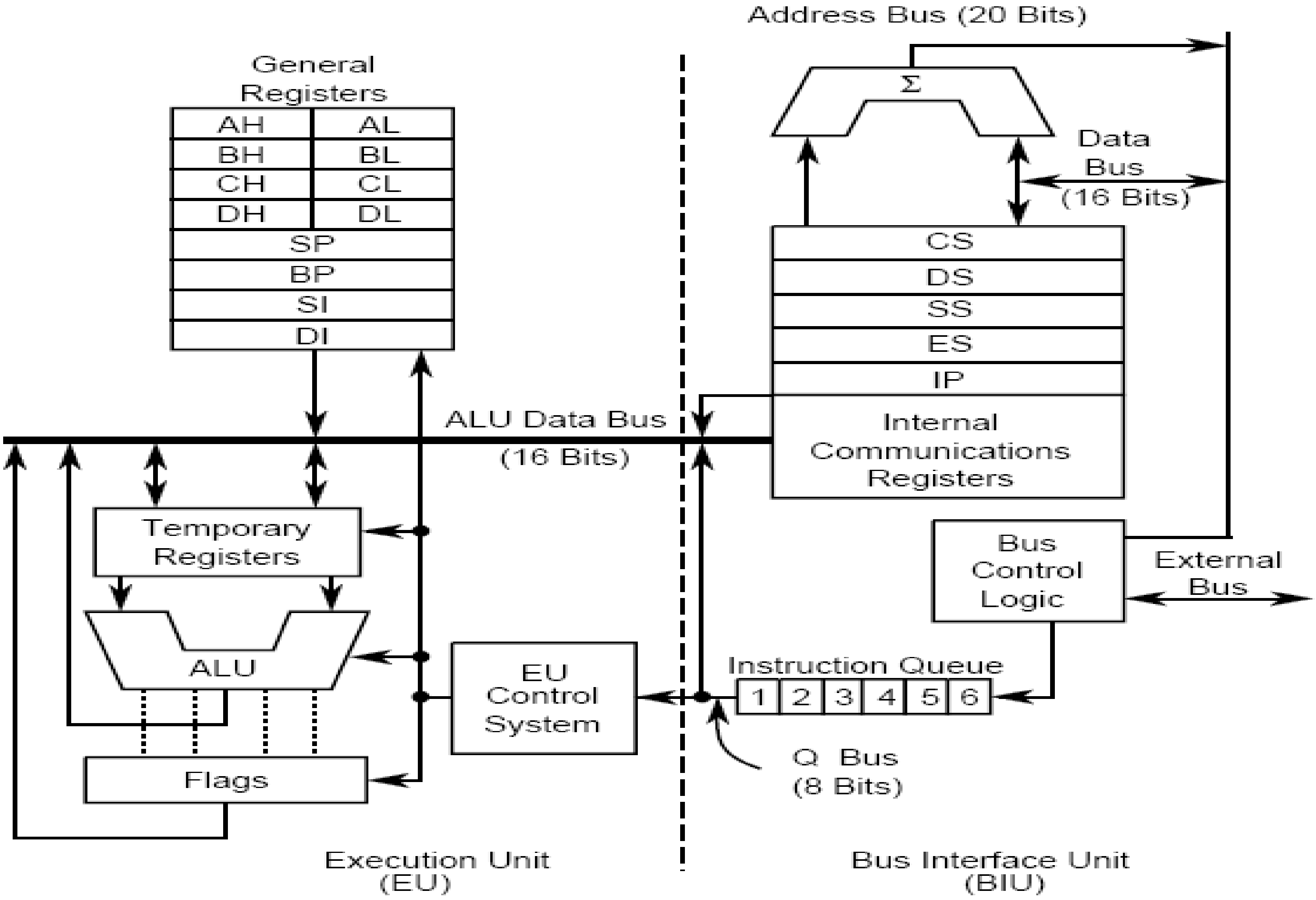
- ✓ The major reason for this separation is to **increase the processing speed of the processor.**

- ✓ The **BIU has to interact with memory and input and output devices in fetching the instructions** and data required by the EU

- ✓ EU is responsible for **executing the instructions of the programs and to carry out the required processing.**



Architecture Diagram



Bus Interface Unit (BIU)

- ✓ The BIU is the 8086‟s interface to the outside world.
- ✓ It provides a full 16-bit bi-directional data bus and 20-bit address bus.
- ✓ The BIU is responsible for performing all external bus operation, as listed below.

Functions of BIU

- ✓ It sends address of the memory or I/O.
- ✓ It fetches instruction from memory.
- ✓ It reads data from port/memory.
- ✓ It writes data into port/memory.
- ✓ It supports instruction queuing.
- ✓ It provides the address relocation facility.

To provide these functions the BIU contains

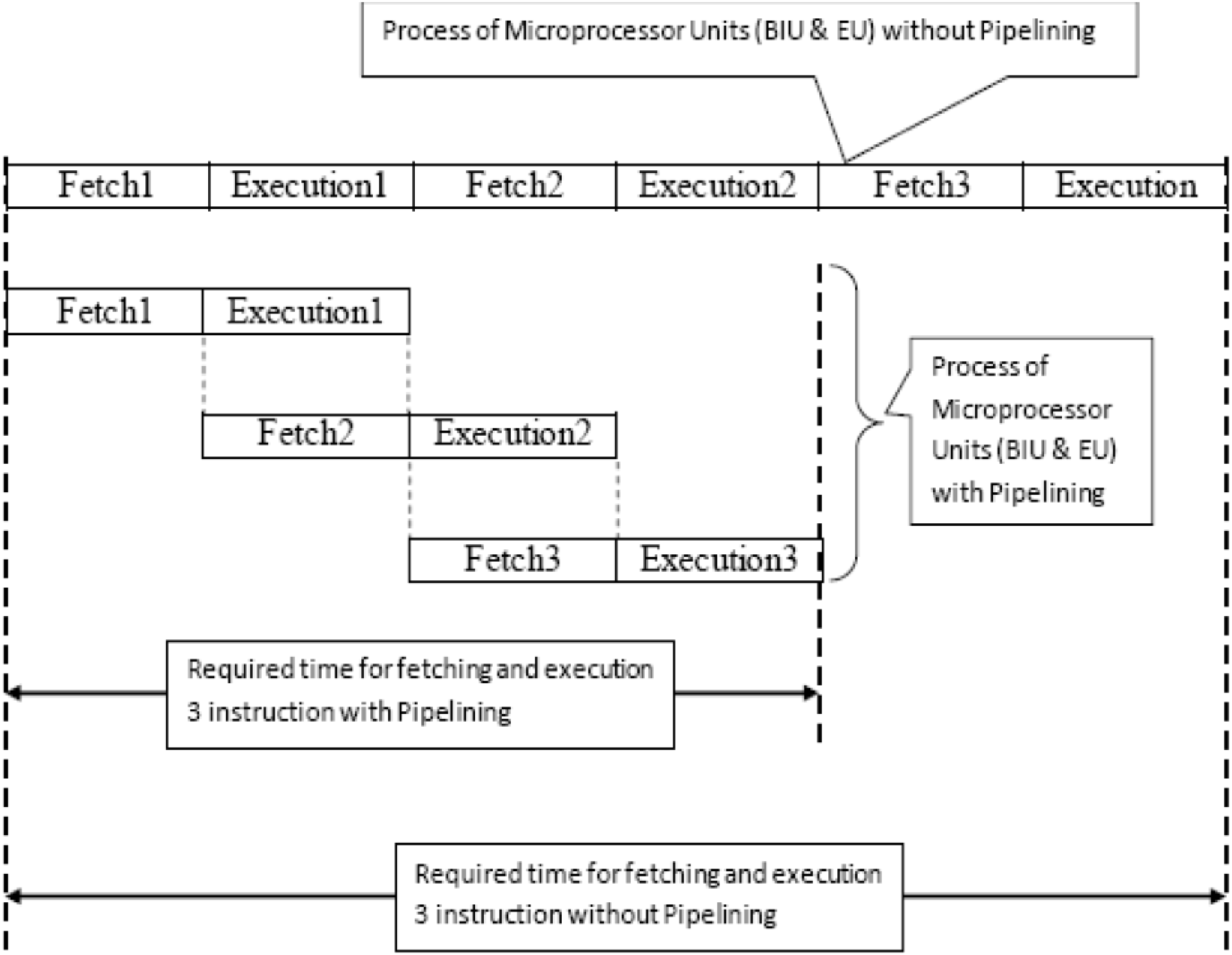
A. Instruction Queue
B. Segment Registers
C. Instruction Pointers
D. Address Summer
E. Bus Control Logic

Instruction Queue and Pipelining

- ✓ To speed up program execution, the BIU fetches **six instruction bytes** a head of time from the memory.

- ✓ Feature of fetching the next instruction while the current (first) instruction is executing is called **Pipelining** as shown in figure.

- ✓ From figure is seen that the required time for fetching and execution 3 instructions without pipelining is mare than with pipelining.


Process of Microprocessor Units (BIU & EU) without Pipelining
Fetch1
Execution1
Fetch2
Execution2
Fetch3
Execution
Fetch1
Execution1
Fetch2
Execution2
Fetch3
Execution3
Process of Microprocessor Units (BIU & EU) with Pipelining
Required time for fetching and execution 3 instruction with Pipelining
Required time for fetching and execution 3 instruction without Pipelining

- ✓ The BIU also contains a **dedicated adder which is used to generate the 20 bit physical address** that is output on the address bus.

- ✓ 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 (EU)

- ✓ The EU of 8086 tells the BIU from where to **fetch instructions or data, decodes instructions and execute instructions**. It contains:

- ✓ **Control Circuitry**
- ✓ **Instruction Decoder**
- ✓ **Arithmetic Logic Unit (ALU)**
- ✓ **Flag Register**
- ✓ **General Purpose Registers**
- ✓ **Pointers and Index Registers**

Control Circuitry

Control circuitry in the EU **directs the internal operations.**

Instruction Decoder

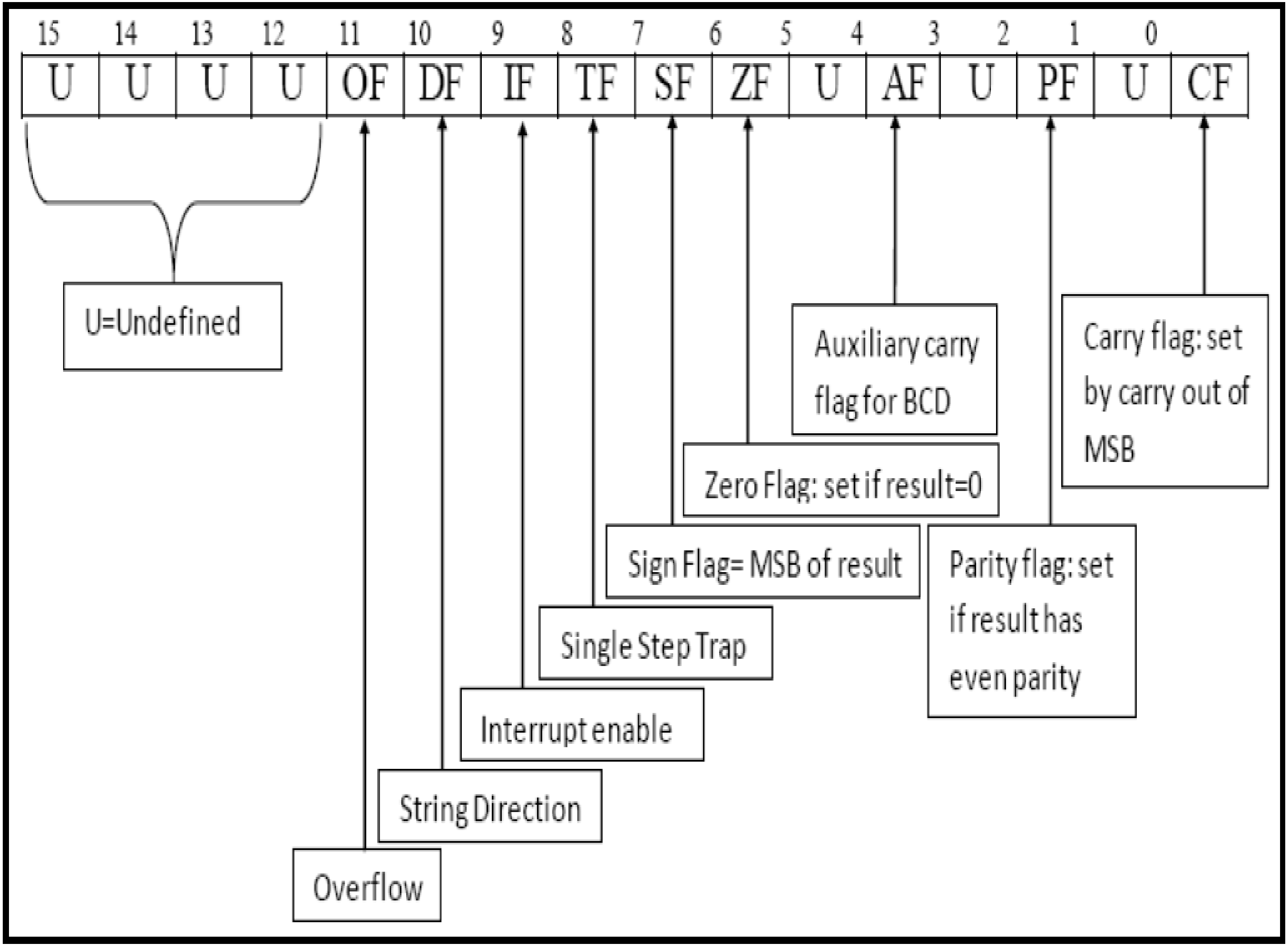
A decoder in the **EU translates the instructions fetched from memory into a series of actions which the EU performs.**

Arithmetic Logic Unit (ALU)

It can **add, subtract, AND, OR, XOR, increment, decrements, complement and shift binary number.**

Flag Register

A flag is a Flip-Flop (FF) which indicates some **condition produced by the execution of an instruction** or controls certain operations of the EU. The flag register contains (9) flags as shown in figure



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

U=Undefined
Overflow
String Direction
Interrupt enable
Single Step Trap
Sign Flag= MSB of result
Zero Flag: set if result=0
Auxiliary carry flag for BCD
Parity flag: set if result has even parity
Carry flag: set by carry out of MSB

Carry flag (CF)

In the case of addition this **flag is set if there is a carry out of the MSB**. The carry flag also serves as a borrow flag for subtraction. In case of subtraction it is set when borrow is needed.

Parity Flag (PF)

It is set to 1 if result of byte operation or lower byte of the word operation **contains an even number** of ones; otherwise it is zero.

Auxiliary Flag (AF)

This flag is set if there is an overflow out of bit i.e., carry from lower nibble to higher nibble (D3 bit to D4 bit). This flag is used for BCD operations and it is not available for the programmer.

Zero Flag (ZF)

The zero flag sets if the result of **operation in ALU is zero and flag resets if the result is non-zero**. The zero flag also set if certain register content becomes zero following an increment or decrement operation of that register.

Sign Flag (SF)

After the execution of arithmetic or logical operations, if the **MSB of the result is 1, the sign bit is set**. **Sign bit 1 indicates the result is negative; otherwise it is positive.**

Overflow Flag (OF)

This **flag is set if the result is out of range**. For addition this flag is set when there is a carry into the MSB and no carry out of the MSB or vice-versa. For subtraction, it is set when the MSB needs a borrow and there is no borrow from the MSB, or vice-versa.

Trap Flag (TF)

Used for **on-chip debugging**.

Interrupt enable Flag (IF)

when this flag is set to **1** CPU reacts to interrupts from external devices.

Direction Flag (DF)

this flag is used by some instructions to process data chains, when this flag is set to **0** - **the processing is done forward**, when this flag is set to **1** the **processing is done backward**.

Overflow Flag (OF)

set to **1** when there is a **signed overflow**

Memory Segmentation

- ✓ Two types of memory organizations are commonly used. There are

  linear addressing
  segmented addressing.

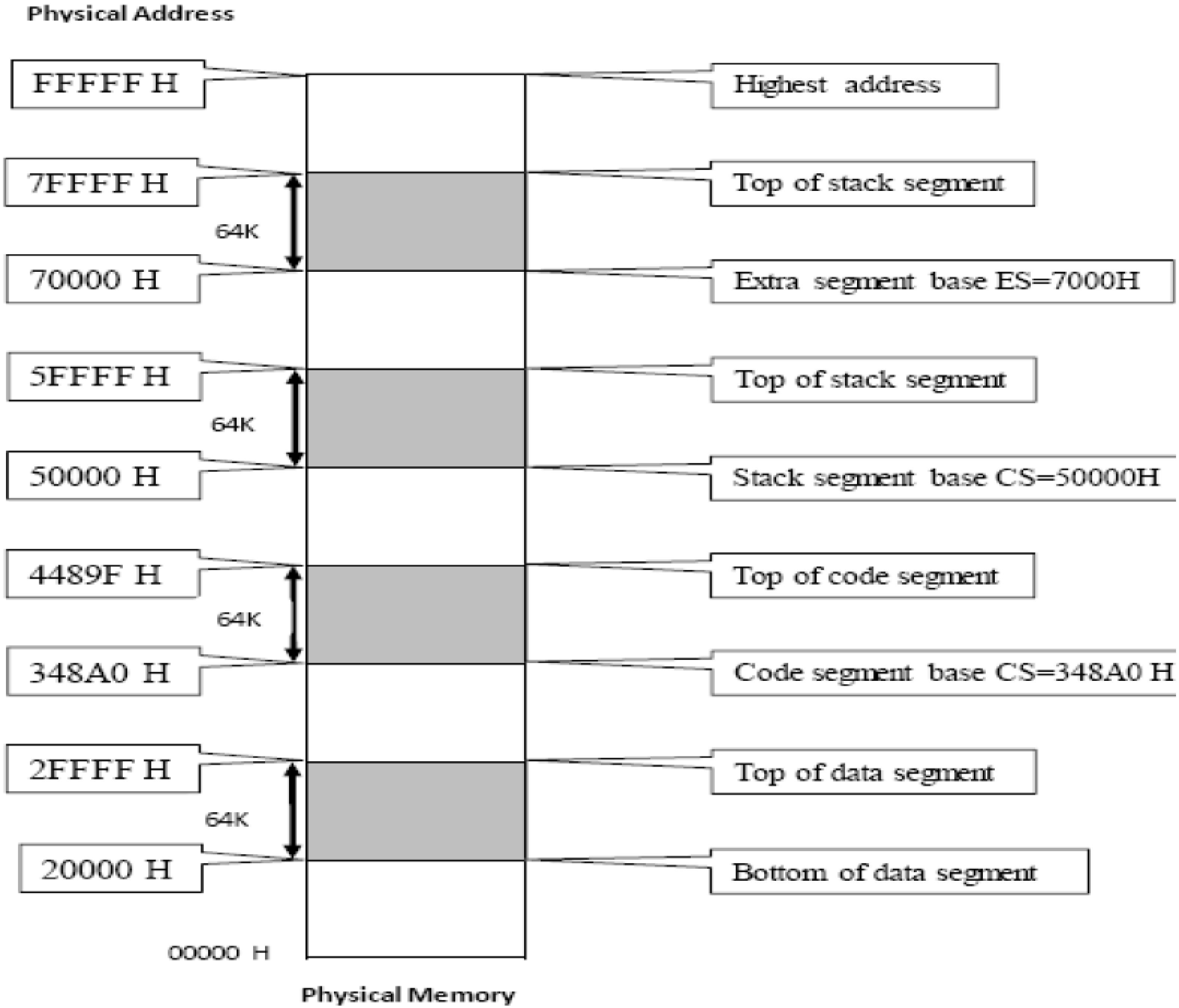
- ✓ In linear addressing: the entire memory space is available to the process in one linear array.

- ✓ In segmented addressing: on the other hand the available memory space is divided into „chunks‟ called segments.

- ✓ Such memory is known segmented memory. In 8086 system the available memory space is 1Mbytes.

- ✓ This memory is divided into number of logical segments. Each segment is 64Kbytes in size and addresses by one of the segment registers.

- ✓ The 16-bit contents of the segment register give the starting/base address of a particular segment, as shown in figure



Physical Address

FFFFF H — Highest address

7FFFF H — Top of stack segment

64K

70000 H — Extra segment base ES=7000H

5FFFF H — Top of stack segment

64K

50000 H — Stack segment base CS=50000H

4489F H — Top of code segment

64K

348A0 H — Code segment base CS=348A0 H

2FFFF H — Top of data segment

64K

20000 H — Bottom of data segment

00000 H

Physical Memory

Generation of 20-bit Address

- ✓ The logical addresses that occur in the 8086 program are always 16 bits in length.

- ✓ This is because all registers are 16 bits.

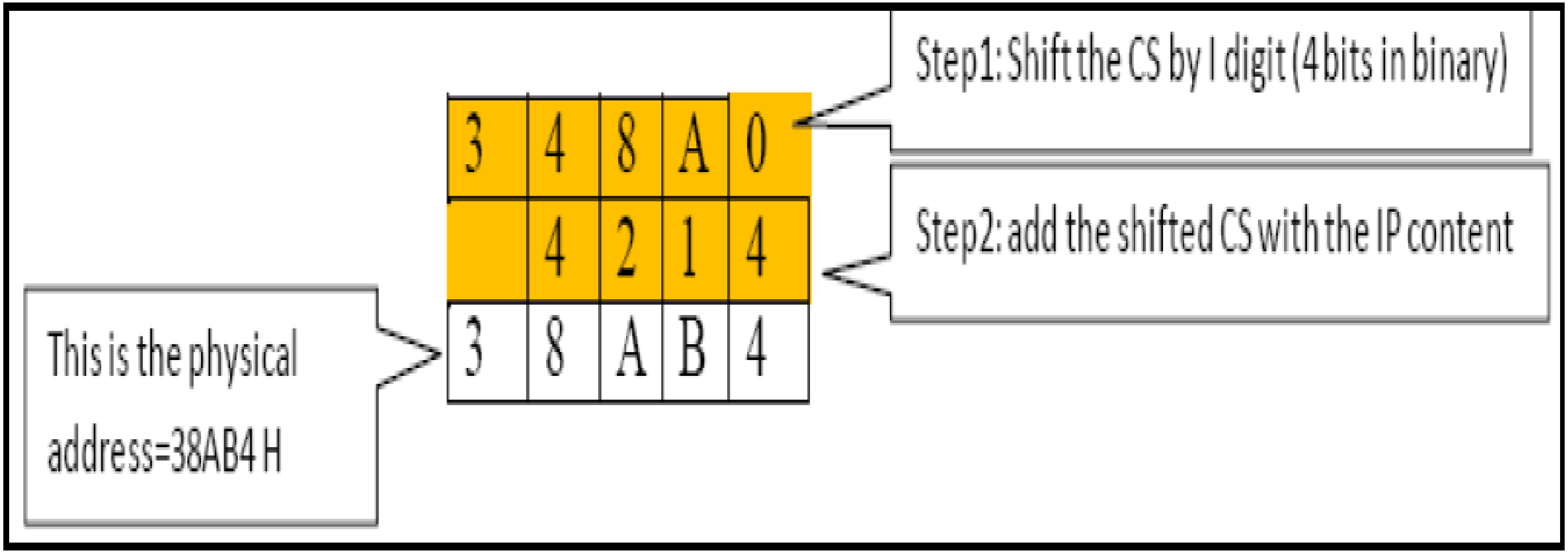
- ✓ However, the physical addresses that are used to access memory are 20 bits in length.

- ✓ The generation of the physical address involves combining a 16-bit *offset* values that is located in either an *index register or pointer register* and a 16-bit *base* value that is located in one of the *segment register*.

Decimal pattern (Hex Value)	Binary numbers
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10 - (A)	1010
11 - (B)	1011
12 - (C)	1100
13 - (D)	1101
14 - (E)	1110
15 - (F)	1111
16 - (10)	10000

Example: The following operation illustrates the generation of 20-bit physical address from CS and IP registers.

Let: CS = 3 4 8 A H , IP= 4 2 1 4 H



Step1: Shift the CS by l digit (4bits in binary)

Step2: add the shifted CS with the IP content

3	4	8	A	0
	4	2	1	4
3	8	A	B	4

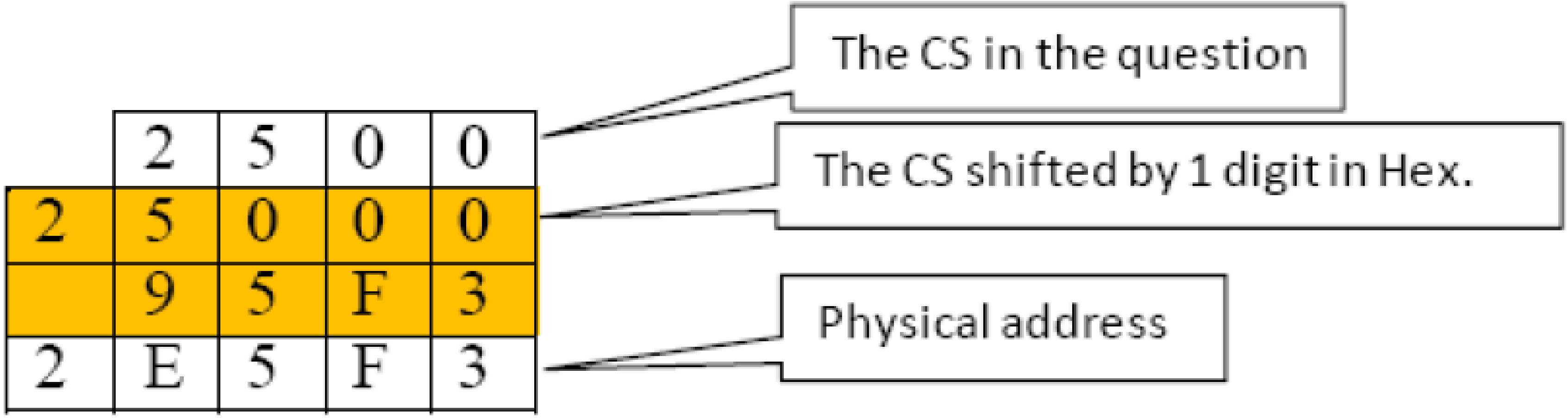
This is the physical address=38AB4 H

Therefore,
Physical Address = Offset + Segment Register (Shifted by 1 digit in Hexadecimal)

The table below shows the offset registers with their related segment registers.

Segment Register	Its Offset Register
CS	IP
DS	BP
SS	SP

Example:
If **CS=2500 h** & **IP=95F3 h,** what is the physical address?
Solution:



	2	5	0	0
2	5	0	0	0
	9	5	F	3
2	E	5	F	3

The CS in the question

The CS shifted by 1 digit in Hex.

Physical address

Example:

If **CS=24F6h** & **IP=634Ah**, show the;

1- The logical address

2- The offset address

3- The physical address

4- The lower range of the segment

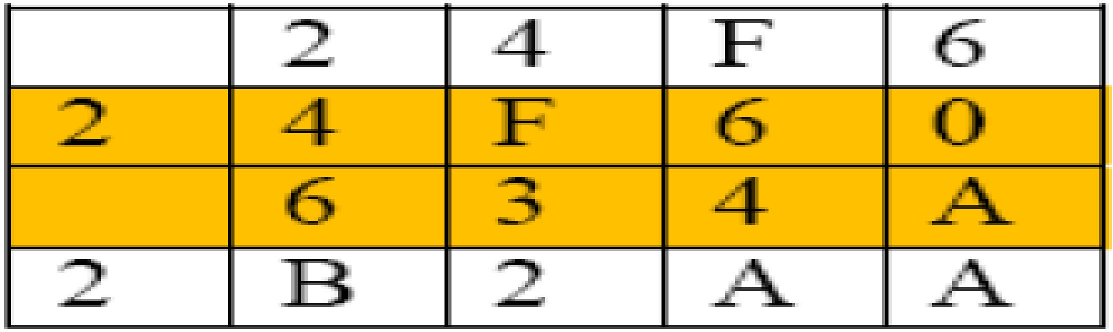
5- The upper range of the segment

Solution:

1- The logical address is the CS: IP content which is: 24F6:634A

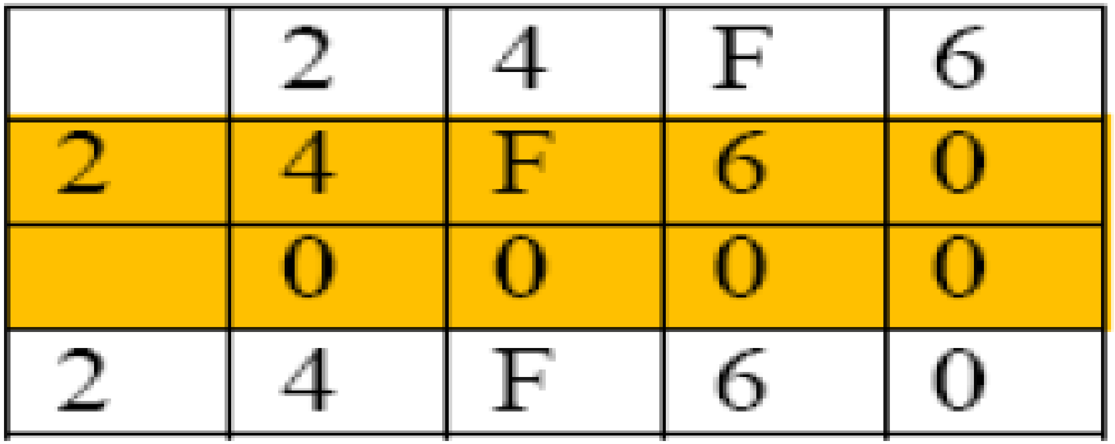
2- The offset address is the content of the IP register which is: 634A

3- The physical address:



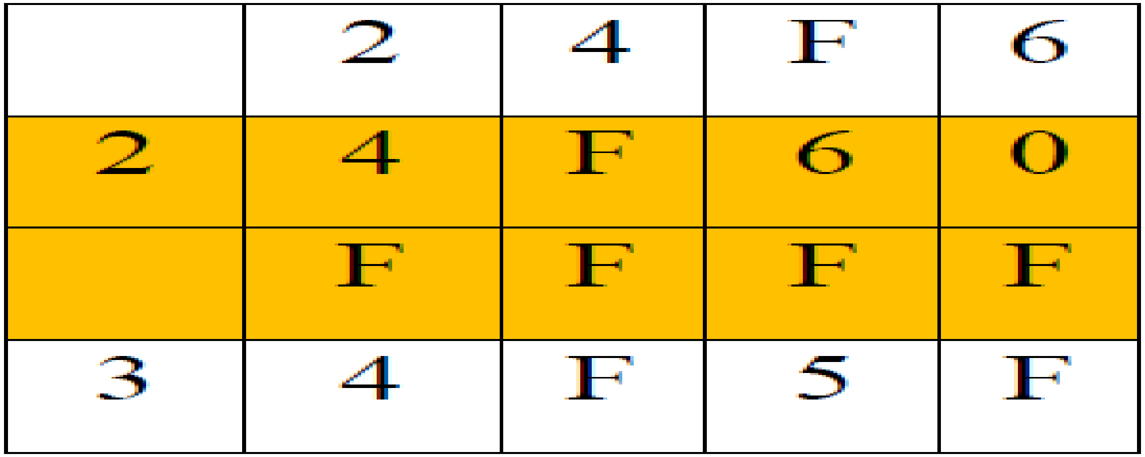
	2	4	F	6
2	4	F	6	0
	6	3	4	A
2	B	2	A	A

4- The lower range of the segment:

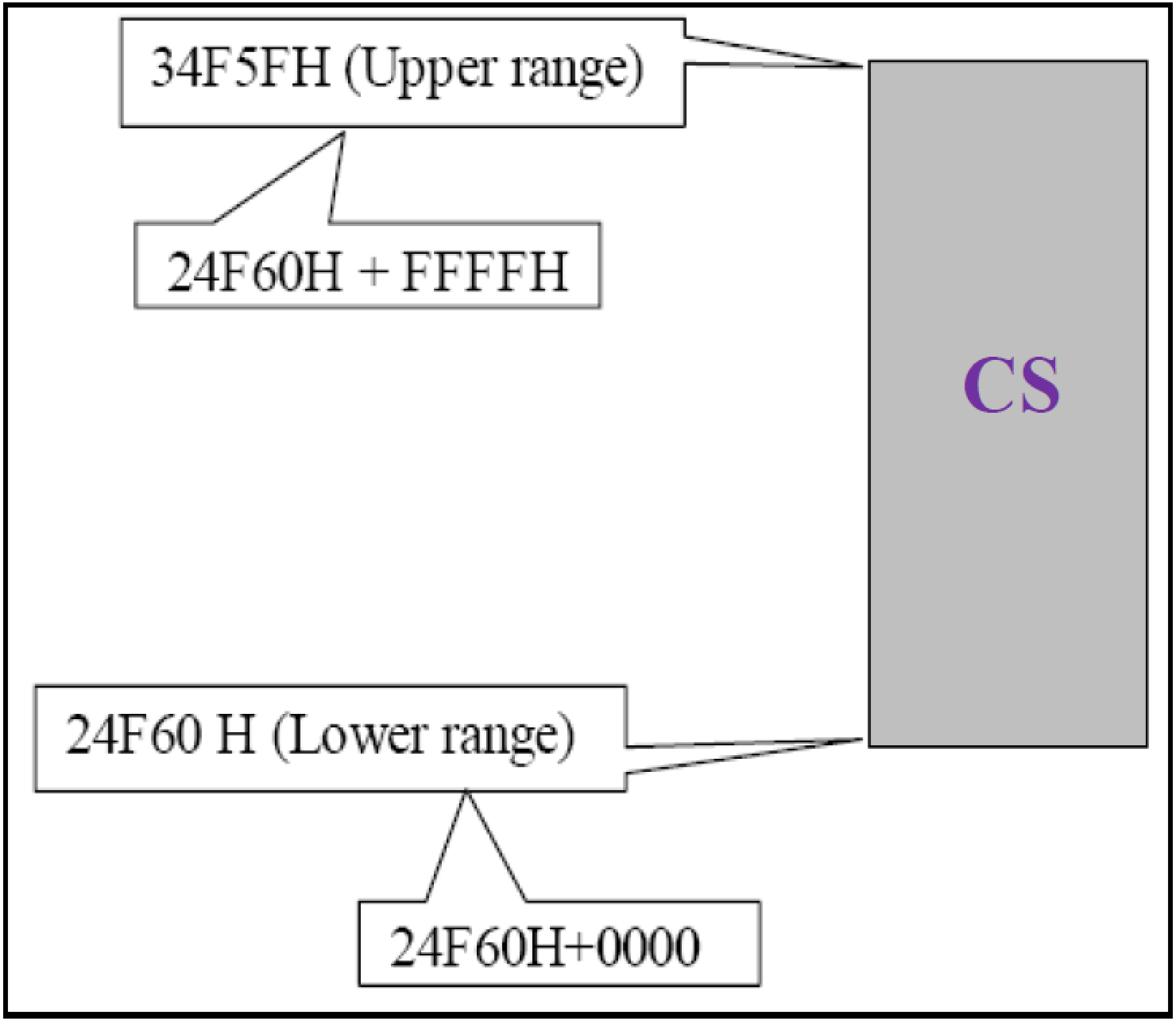


	2	4	F	6
2	4	F	6	0
	0	0	0	0
2	4	F	6	0

5- The upper range of the segment:



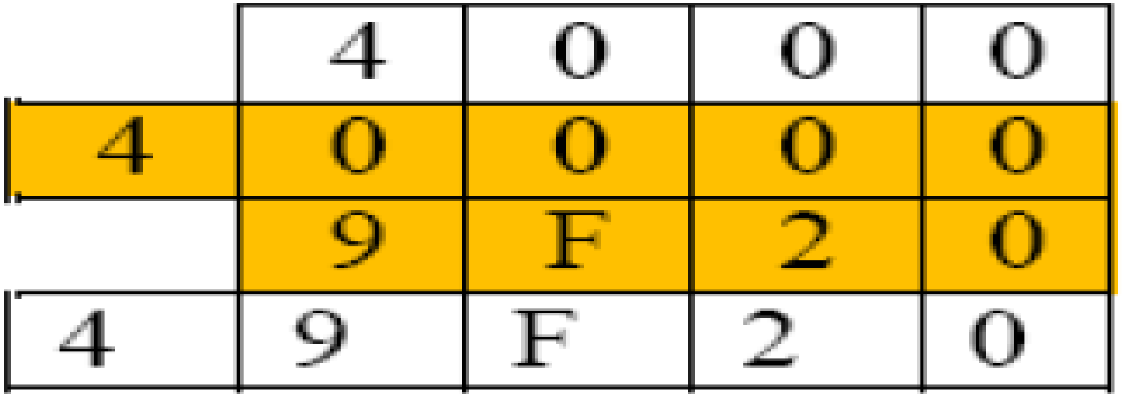
	2	4	F	6
2	4	F	6	0
	F	F	F	F
3	4	F	5	F


34F5FH (Upper range)
24F60H + FFFFH
CS
24F60 H (Lower range)
24F60H+0000

Example:

If SS=4000 H and SP= 9F20 H what is physical address.

Solution:



	4	0	0	0
4	0	0	0	0
	9	F	2	0
4	9	F	2	0

Physical address=49F20 H

Addressing Modes

The addressing modes of any processor can be broadly classified as

Data addressing modes
Program memory addressing modes
Stack memory addressing modes

Data addressing modes

The data addressing modes can be further classified as:

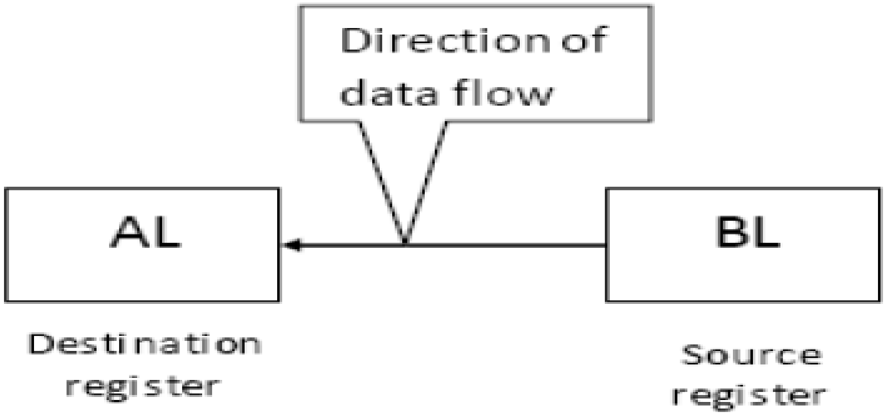
- ✓ Addressing modes for accessing **immediate and register data (Register and Immediate modes)**
- ✓ Addressing modes for accessing **data in memory (memory modes)**
- ✓ Addressing modes for **accessing I/O ports (I/O modes)**

Addressing modes for accessing immediate and register data

Register Addressing Mode

This mode specifies the source operand, destination operand or both to be contained in an 8086 register. Figure below shows the operation:

MOV AL, BL



Examples:

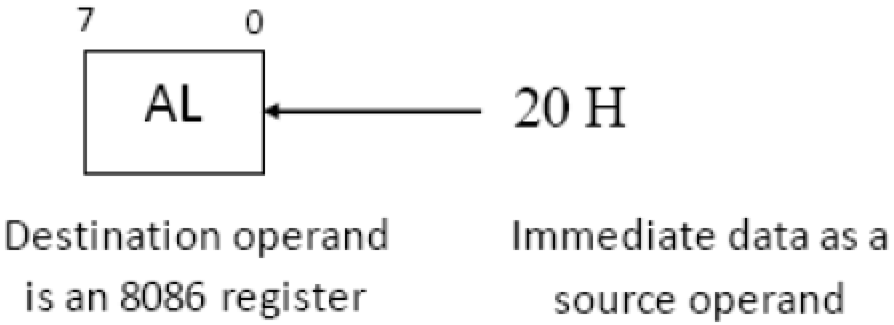
MOV BX, CX; Copies the 16-bit contents of CX into BX.

MOV CL, BL; Copies the 8-bit contents of BL into CL.

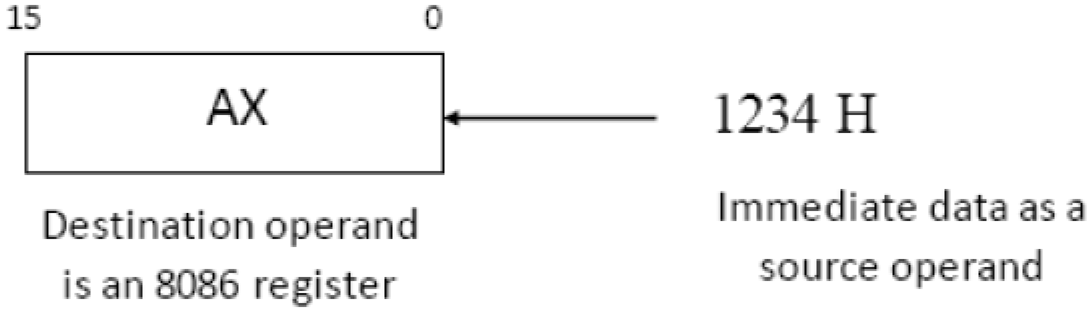
Immediate Addressing Mode

In an immediate mode, 8 or 16-bit data can be specified as a part of instruction.

MOV AL, 20 H


7
0
AL
20 H
Destination operand is an 8086 register
Immediate data as a source operand

MOV AX, 1234 H


15
0
AX
1234 H
Destination operand is an 8086 register
Immediate data as a source operand

Arrow indicates direction of data flow.

Examples:

MOV BL, 26 H ; Copies the 8-bit data 26 H into BL.

MOV CX, 4567 H ; Copies the 16-bit data 4567H into CX.

Addressing modes for accessing Data in Memory

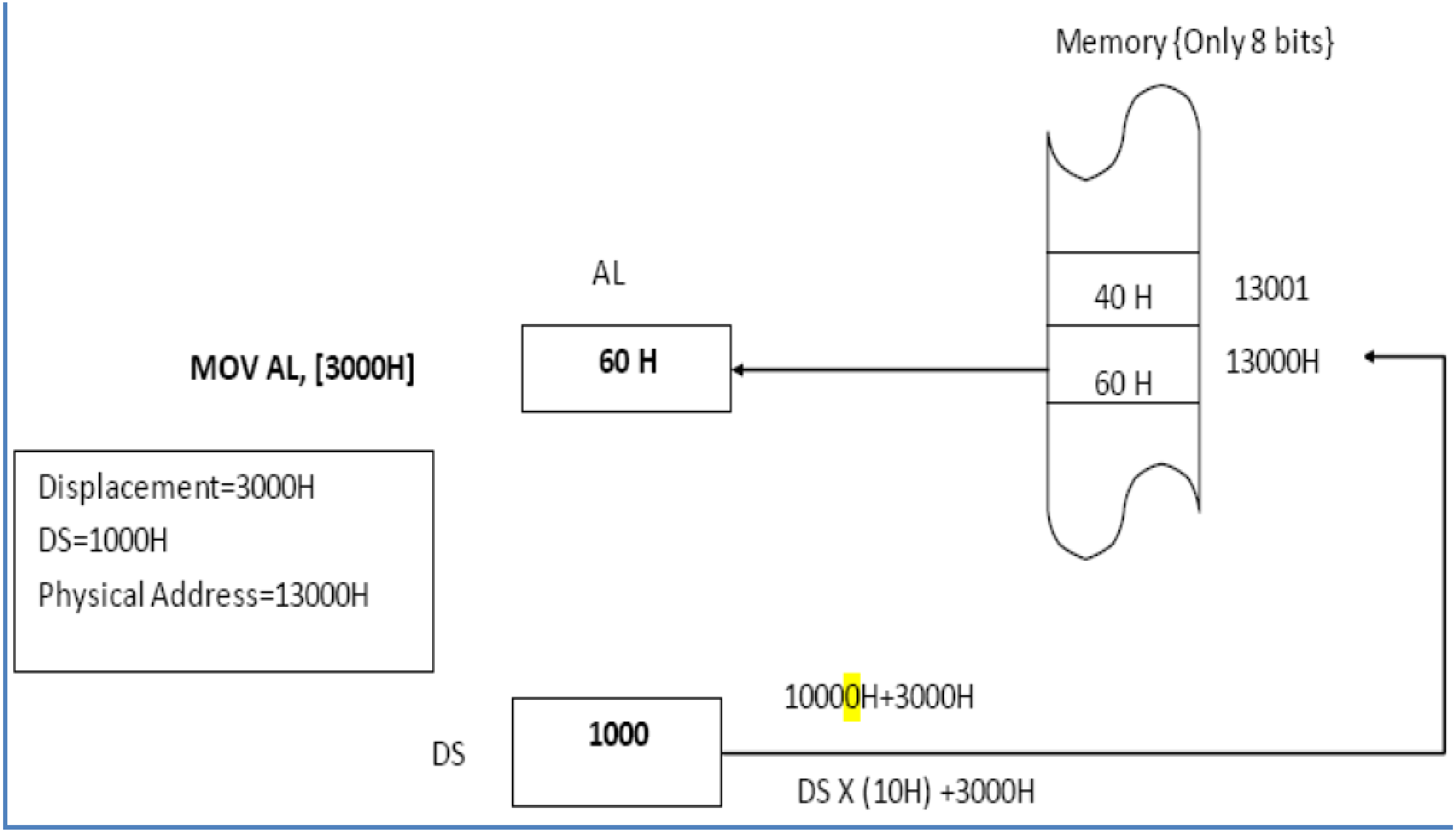
- ✓ As mentioned before, the Execution Unit (EU) has **direct access to all registers and data for register and immediate operands.**
- ✓ However, the EU **cannot directly access the memory operands.**
- ✓ It must use **the BIU segment registers to access memory operands**.
- ✓ For example, when EU needs to access memory location, it sends an offset value to the BIU. This offset is also called the **Effective Address (EA)**.
- ✓ Note that the effective address is displacement form the desired location from the segment base. As mentioned before, the BIU generates a 20-bit physical address after shifting the contents of the desired segment register four bits to the left and then adding the 16-bit EA to it.

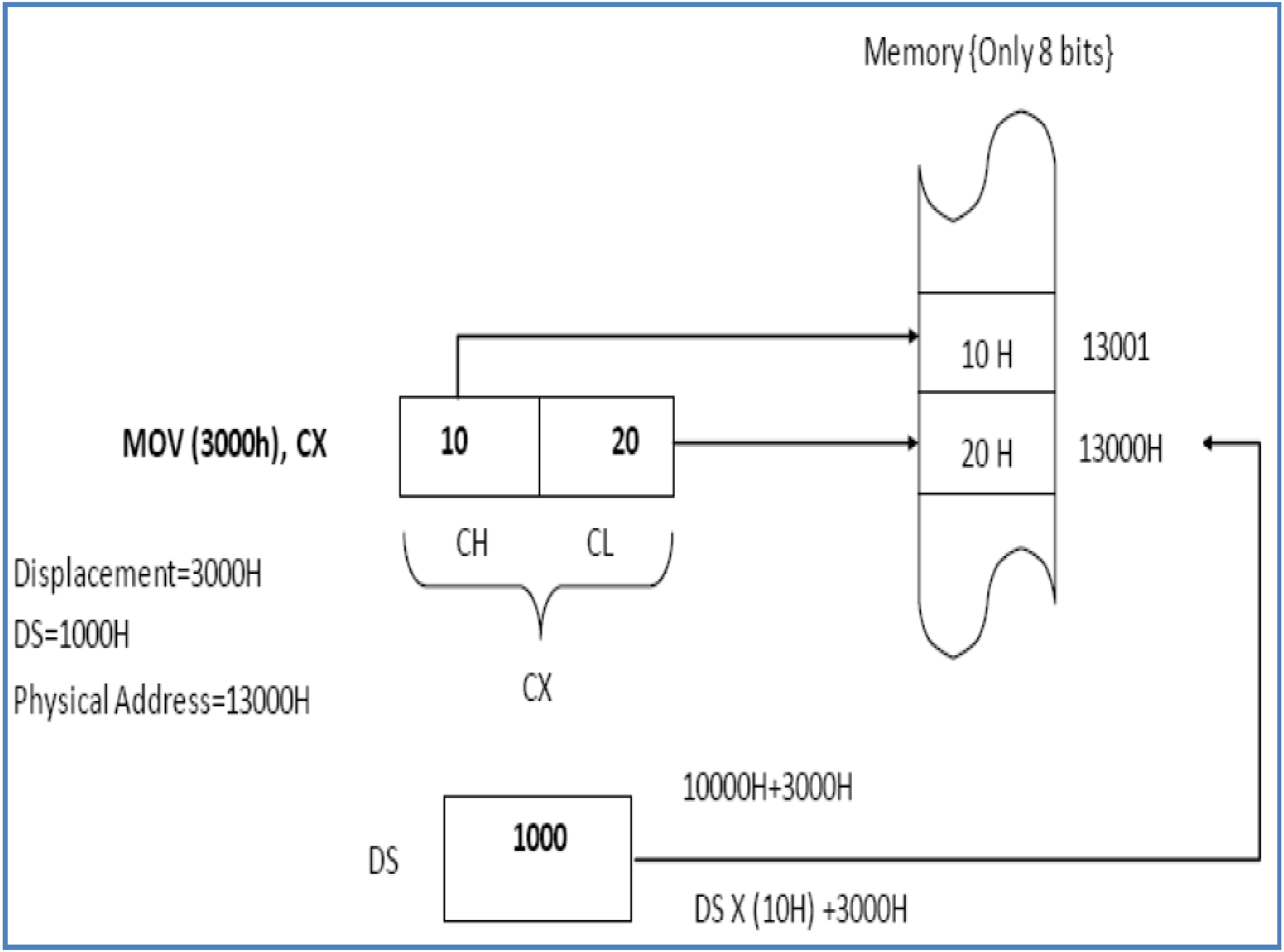
There are six ways to specify effective address (EA) in the instruction:

- ✓Direct addressing mode
- ✓Register indirect addressing mode
- ✓Based addressing mode
- ✓Indexed addressing mode
- ✓Based indexed addressing mode
- ✓String addressing mode

Direct Addressing Mode:

In this mode, the 16-bit effective address (EA) is taken directly from the displacement field of the instruction.





MOV (3000h), CX

Displacement=3000H

DS=1000H

Physical Address=13000H

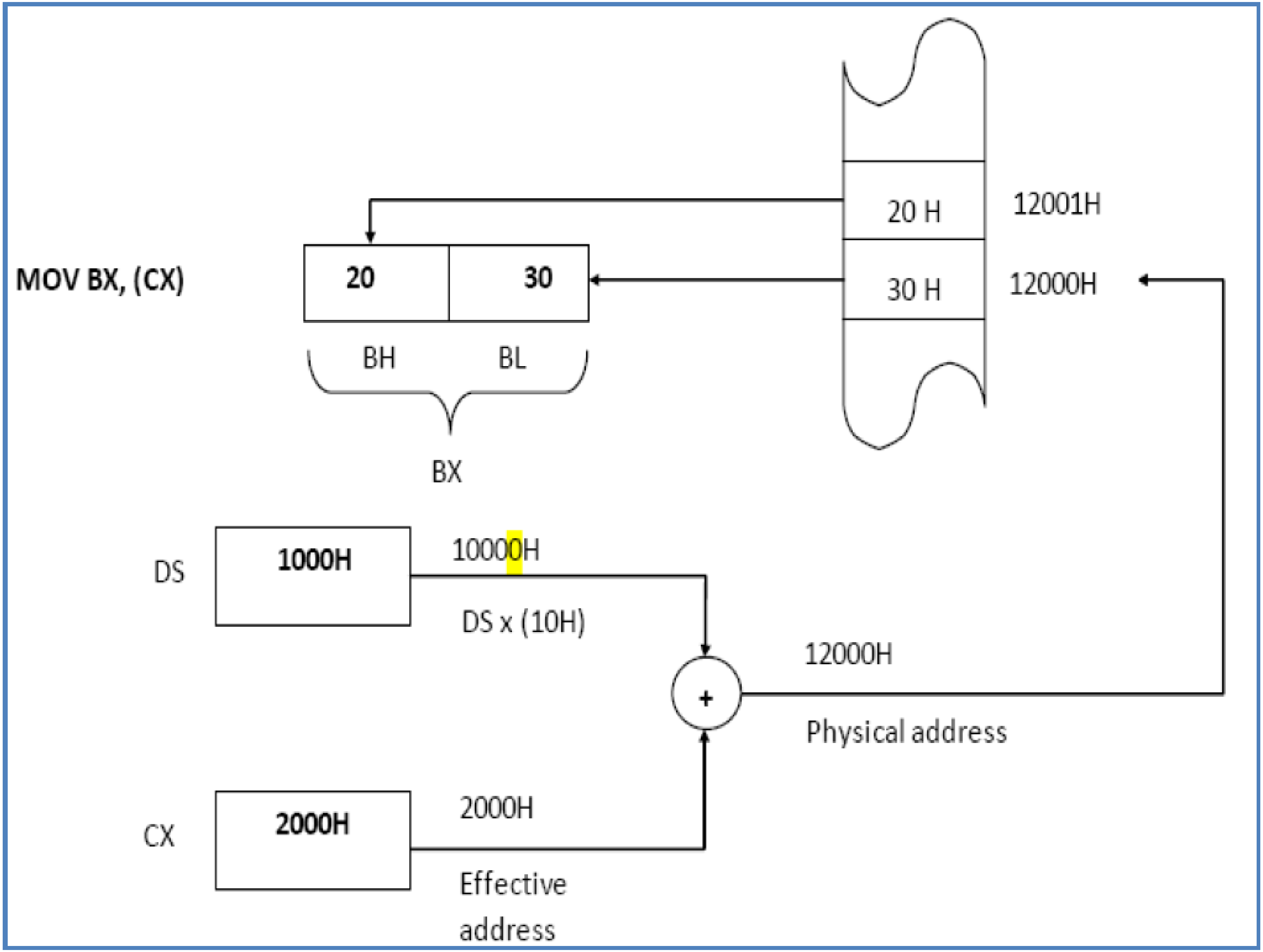
MOV CL, [9823] ; this instruction will copy the contents of the memory Location at a displacement of 9823H from the data Segment base, into the CL register. Here, 9823H is the Effective address (EA) which is written directly in the Instruction.

Register Indirect Addressing Mode

- ✓ In this mode, the EA is specified in either a **pointer register or an index register**.

- ✓ The **pointer register can be either base register BX or base pointer BP and index register can be either Source Index (SI) register or Destination Index (DI) register.**

- ✓ The 20-bit physical address is computed using DS and EA.



MOV BX, (CX)
20
30
BH
BL
BX
20 H
30 H
12001H
12000H
DS
1000H
10000H
DS x (10H)
12000H
+
Physical address
CX
2000H
2000H
Effective
address

Example:
MOV [DI], BX:

The instruction copies the 16-bit contents of BX into a Memory location offset by the value of EA specified in DI from the current contents in DS.
Now, if [DS]=7205H, [DI]=0030H, and [BX]=8765H, then after MOV [DI], BX, content of the BX (8765H) is copied to Memory location 72080H.

Physical Address=DI + DS*(10H)
= 0030H + 7205H * 10H
= 0030H + 72050H
= 72080H

Example:

MOV DL, [BP]

this instruction copies the 8-bit contents in DL from the Memory location offset by the value of EA specified In BP from the contents of SS. Because data addressed by BP are by default located in stack segment (SS).

Addressing Modes

- ✓ Implied Addressing

  The data value/data address is **implicitly associated with the instruction.**

- ✓ Register Addressing

  The data is specified by referring the register or the register pair in which the data is present.

- ✓ Immediate Addressing

  The **data itself is provided in the instruction**

- ✓ Direct Addressing

  The instruction operand **specifies the memory** address where data is located

✓ Register indirect addressing

The instruction specifies a register containing an address, where data is located

✓ Based

8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides

✓ Indexed

8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides

- ✓ Based Indexed

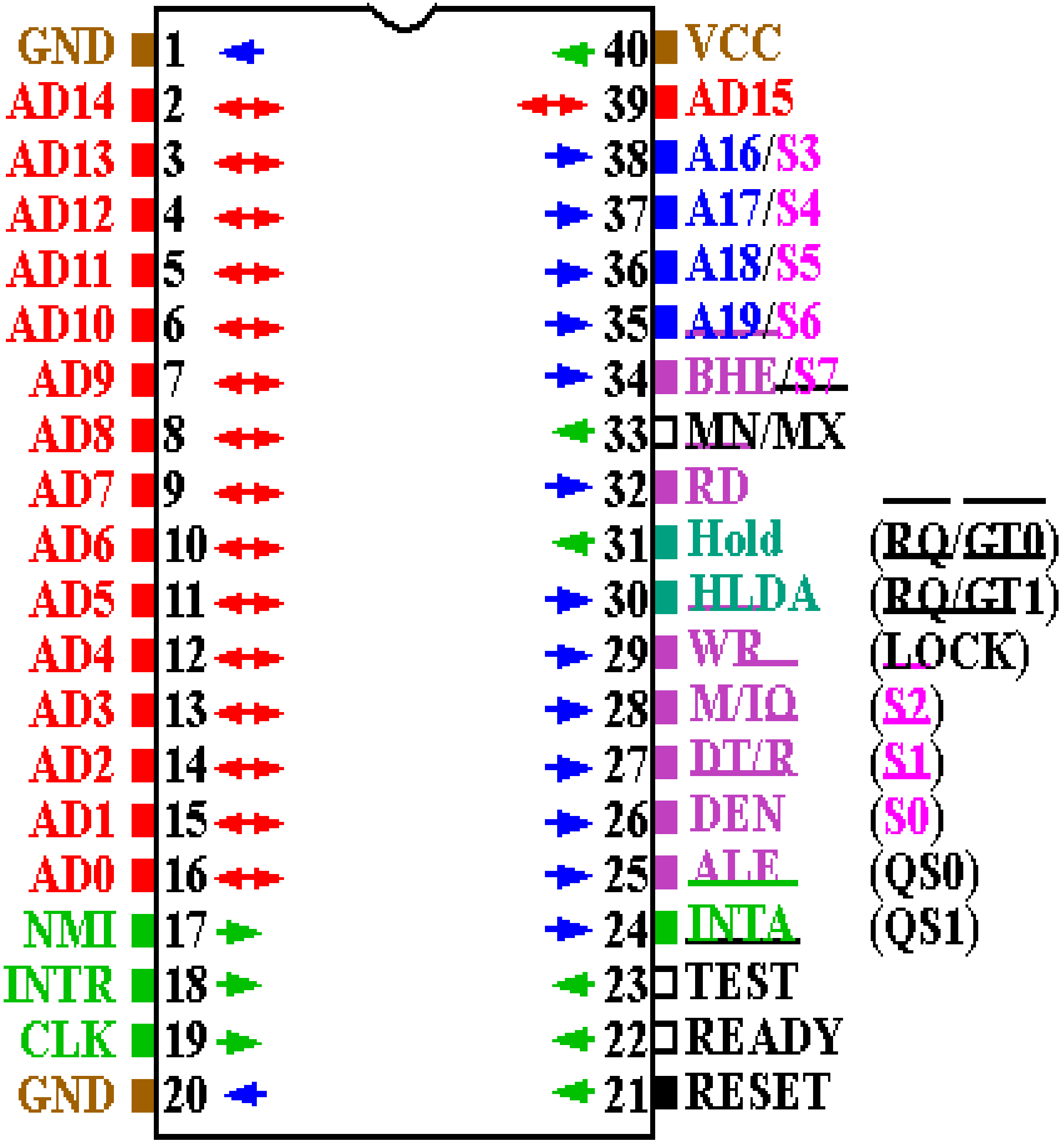
the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

- ✓ Based Indexed with displacement

8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

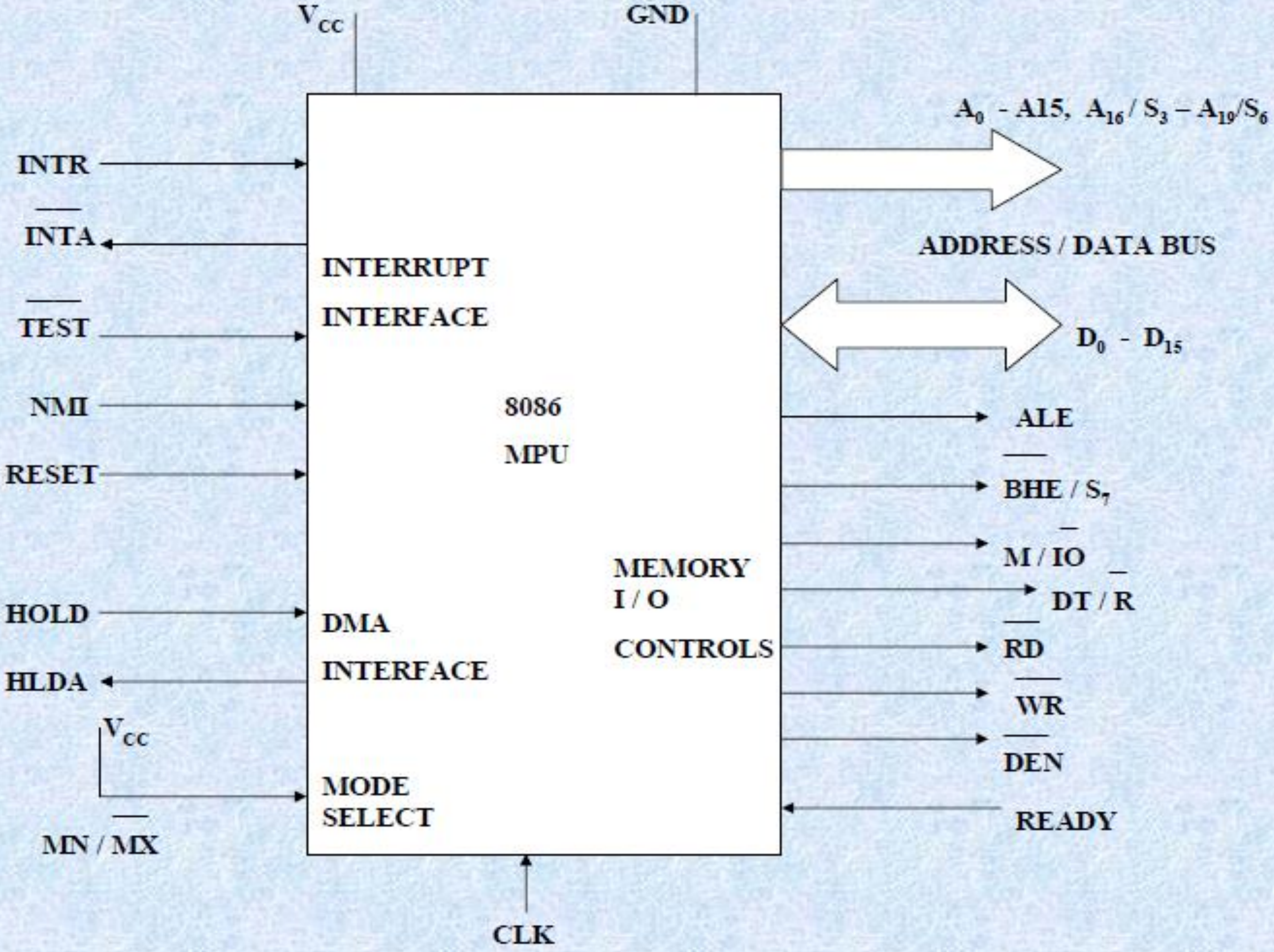
8086 CPU

MIN MODE (MAX MODE)



GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20

40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 Hold (RQ/GT0)
30 HLDA (RQ/GT1)
29 WR (LOCK)
28 M/IO (S2)
27 DT/R (S1)
26 DEN (S0)
25 ALE (QS0)
24 INTA (QS1)
23 TEST
22 READY
21 RESET



V_{CC}

GND

A_0 - A15, A_{16} / S_3 – A_{19}/S_6

ADDRESS / DATA BUS

D_0 - D_{15}

INTR

$\overline{\text{INTA}}$

INTERRUPT

INTERFACE

$\overline{\text{TEST}}$

NMI

8086

MPU

RESET

ALE

$\overline{\text{BHE}}$ / S_7

M / $\overline{\text{IO}}$

DT / $\overline{\text{R}}$

MEMORY

I / O

CONTROLS

HOLD

DMA

INTERFACE

HLDA

$\overline{\text{RD}}$

$\overline{\text{WR}}$

$\overline{\text{DEN}}$

V_{CC}

MODE

SELECT

READY

MN / $\overline{\text{MX}}$

CLK

COMMON SIGNALS

Name	Function	Type
$AD_{15} – AD_{0}$	Address/ Data Bus	Bidirectional 3 - state
$A_{19} / S_{6} – A_{16} / S_{3}$	Address / Status	Output 3 - State
$\overline{BHE}$ / S 7	Bus High Enable / Status	Output 3 - State
MN / $\overline{MX}$	Minimum / Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output 3 - State
TEST	Wait On Test Control	Input
READY	Wait State Controls	Input
RESET	System Reset	Input
NMI	Non - Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
Vcc	+ 5 V	Input
GND	Ground	

Minimum Mode Signals (MN/ $\overline{\text{MX}}$ = Vcc)

Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{\text{WR}}$	Write Control	Output 3- state
M/$\overline{\text{IO}}$	Memory or IO Control	Output 3-State
DT/$\overline{\text{R}}$	Data Transmit / Receiver	Output 3- State
$\overline{\text{DEN}}$	Date Enable	Output 3-State
ALE	Address Latch Enable	Output
$\overline{\text{INTA}}$	Interrupt Acknowledge	Output

Maximum mode signals (MN / $\overline{\text{MX}}$ = GND)		
Name	Function	Type
RQ / $\overline{\text{GT1, 0}}$	Request / Grant Bus Access Control	Bidirectional
$\overline{\text{LOCK}}$	Bus Priority Lock Control	Output, 3- State
$\overline{S_2} - \overline{S_0}$	Bus Cycle Status	Output, 3- State
QS1, QS0	Instruction Queue Status	Output

AD15±AD0

ADDRESS DATA BUS: These lines constitute the time multiplexed memory/IO address and data bus.

ALE

Address Latch Enable. A HIGH on this line causes the lower order 16bit address bus to be latched that stores the addresses and then, the lower order 16bit of the address bus can be used as data bus.

READY

READY is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer.

INTR

INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.

INTA

Interrupt Acknowledge from the MP

NMI

NON-MASKABLE INTERRUPT: an edge triggered input which causes an interrupt request to the MP. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software.

RESET

causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution.

MN/$\overline{\text{MX}}$

MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

M/$\overline{IO}$

Differentiate between the Memory and I/O operation. A LOW on this pin indicated I/O operation and a HIGH indicated a Memory Operation.

HOLD

The 8086 has a pin called HOLD. This pin is used by external devices to gain control of the busses.

HLDA

When the HOLD signal is activated by an external device, the 8086 stops executing instructions and stops using the busses. This would allow external devices to control the information on the.