INTEL 8086 MICROPROCESSOR ARCHITECTURE

Introduction

- Intel 8086 was launched in 1978.
- It was the first 16-bit microprocessor.
- This microprocessor had major improvement over the execution speed of 8085.
- It is available as 40-pin Dual-Inline-Package (DIP).
- It is available in three versions:
  - 8086 (5 MHz)
  - 8086-2 (8 MHz)
  - 8086-1 (10 MHz)

Features

- It is **16 bit processor**. So that it has 16 bit ALU, 16 bit registers and internal data bus and 16 bit external data bus which accounts for its faster processing.

- 8086 has a **20 bit address bus** that can access up to 2^{20} memory locations (1 MB).

- It can support up to **64K I/O ports**. 8086 has **16-bit address lines** to access I/O devices, hence it can access 2^16 = 64K I/O location.

- It provides **14, 16 -bit registers**.

- **Word size** is 16 bits.

- It has **multiplexed address and data bus** AD0-AD15 and A16 –A19.

- **8086 is designed to operate in two modes, Minimum and Maximum.**

a) Minimum Mode: A system with only one microprocessor.
b) Maximum Mode: A system with multiprocessor.

- **Pipelining:-** Pipelining improves the performance of the processor so that operation is faster. 8086 uses two stage of pipelining. First is Fetch Stage and the second is Execute Stage.

a) Fetch stage that pre-fetch up to 6 bytes of instructions stores them in the queue.
b) Execute stage that executes these instructions.

- It requires +5V **power supply**.

- A **40 pin** dual in line package.

- **Address ranges** from 00000H to FFFFFH.(2^20-1)

- **8086 uses memory banks:-**The 8086 uses a memory banking system, memory is split into two 8-bit banks. It means entire data is not stored sequentially in a single memory of 1 MB but memory is divided into two banks of 512KB.

- **Interrupts:-**8086 has 256 interrupts.

- It supports **multiplication and division**.

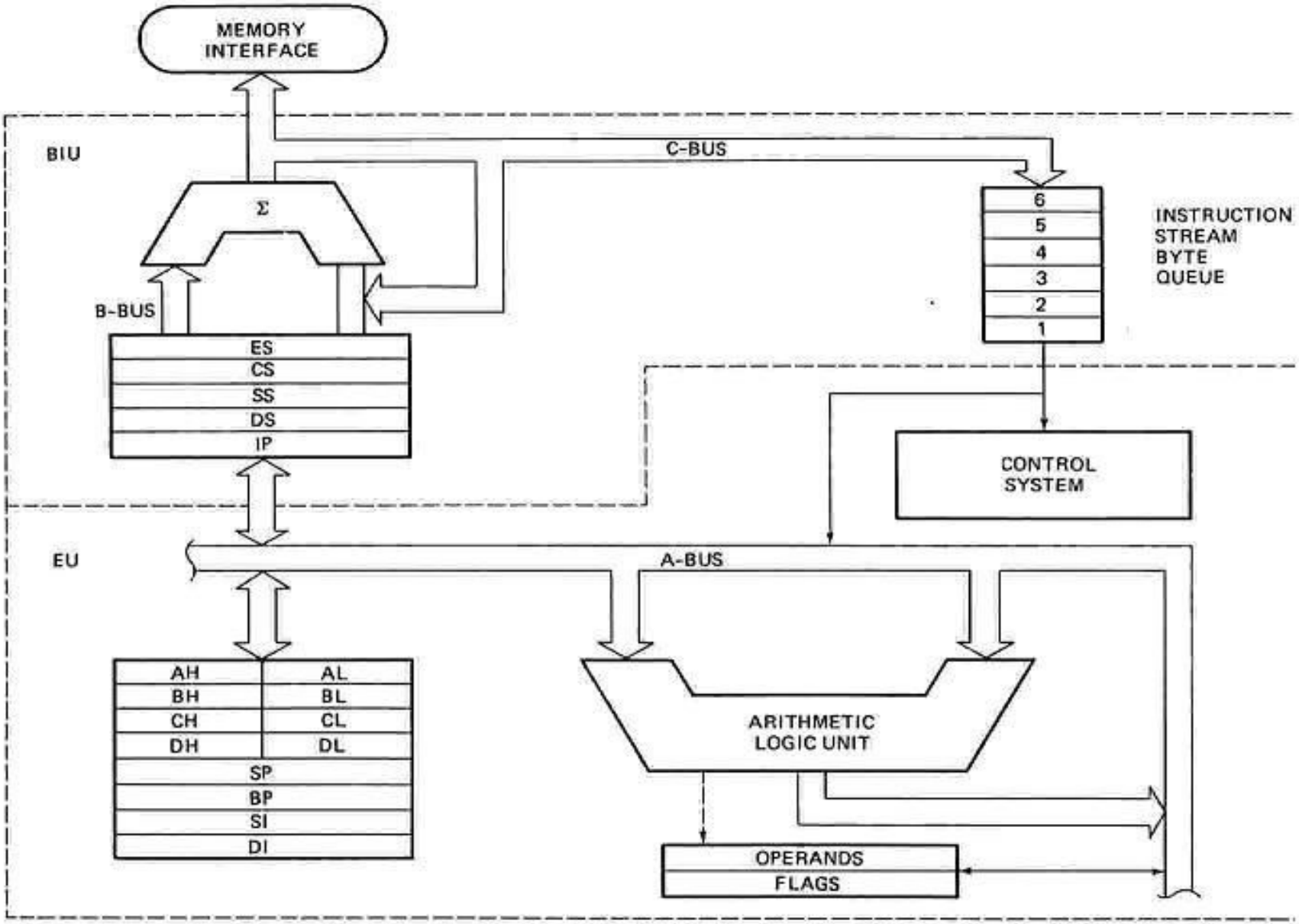
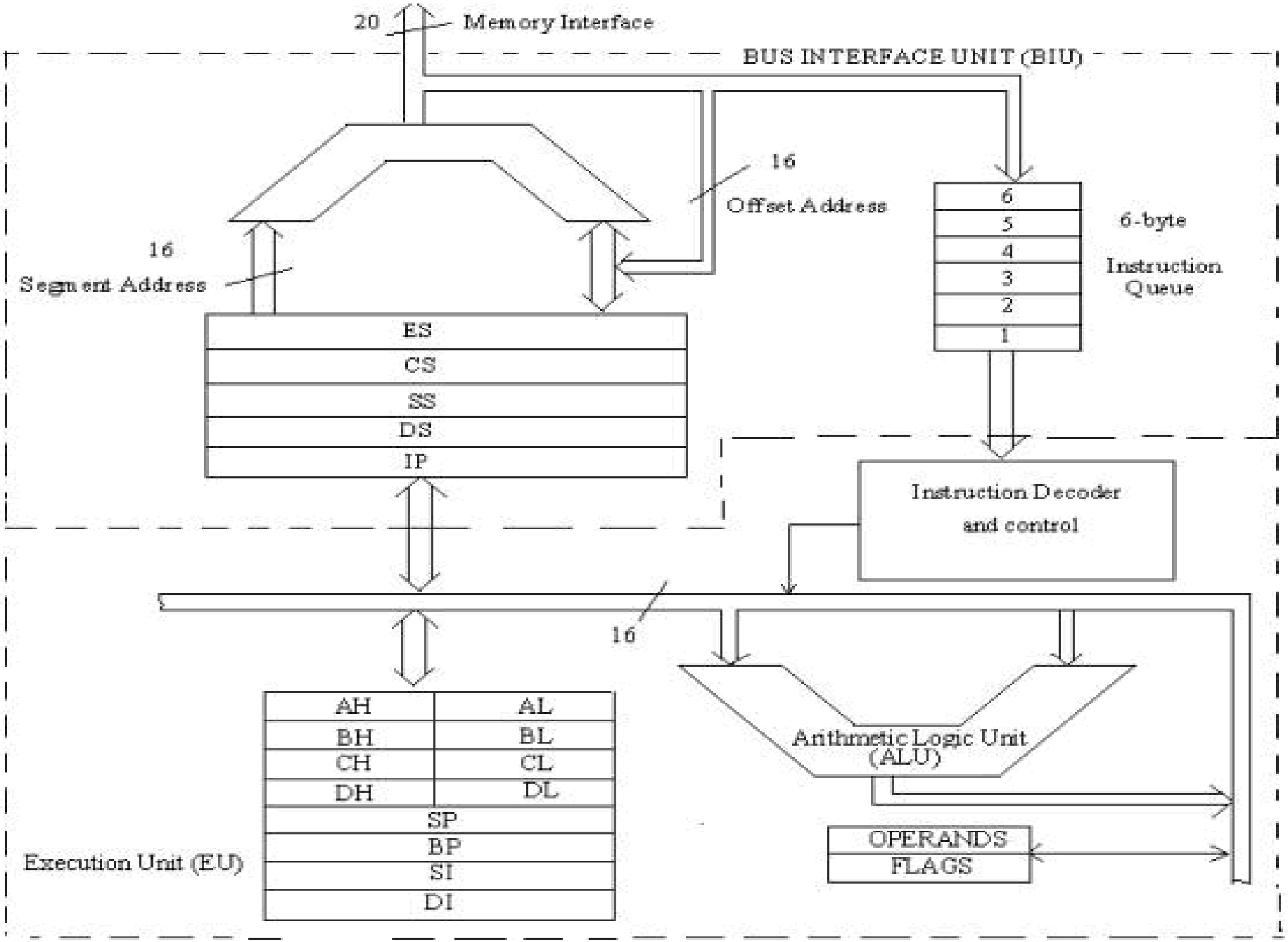


FIGURE 8086 internal block diagram. (*Intel Corp.*)


20
Memory Interface
BUS INTERFACE UNIT (BIU)
16
Offset Address
16
Segment Address
6
5
4
3
2
1
6-byte
Instruction
Queue
ES
CS
SS
DS
IP
Instruction Decoder
and control
16
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
Execution Unit (EU)
Arithmetic Logic Unit
(ALU)
OPERANDS
FLAGS

Internal architecture of 8086

8086 has two blocks BIU and EU.

Bus Interface Unit (BIU)

- The BIU handles all transactions of data and addresses on the buses for EU.

- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.

Functions of Bus Interface Unit

- It handles transfer of data and addresses between the processor and memory / IO.
- It reads data from memory and I/O devices.
- It writes data to memory and I/O devices.
- It computes and sends out addresses.
- It fetches instruction codes.

- Bus Interface Unit contains:
  - Segment Registers
  - Instruction Pointer
  - 6-Byte Instruction Queue

  - It stores fetched instruction codes in a FIFO register called QUEUE.

Instruction Queue

- To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory.
- All six bytes are then held in first-in-first-out 6-byte register called instruction queue.
- Then all bytes are given to EU one by one.
- This pre-fetching operation of BIU may be in parallel with execution operation of EU, which improves the execution speed of the instruction.

Instruction Pointer

- The Instruction Pointer (IP) in 8086 acts as a Program Counter.
- It points to the address of the next instruction to be executed.
- Its content is automatically incremented when the execution of a program proceeds further.
- The contents of the IP and Code Segment Register are used to compute the memory address of the instruction code to be fetched.
- This is done during the Fetch Cycle.

Execution Unit (EU)

- Execution Unit contains:
  - General Purposes Registers
  - Stack Pointer
  - Base Pointer (function)
  - Index Registers
  - ALU
  - Flag Register
  - Instruction Decoder
  - Timing & Control Unit

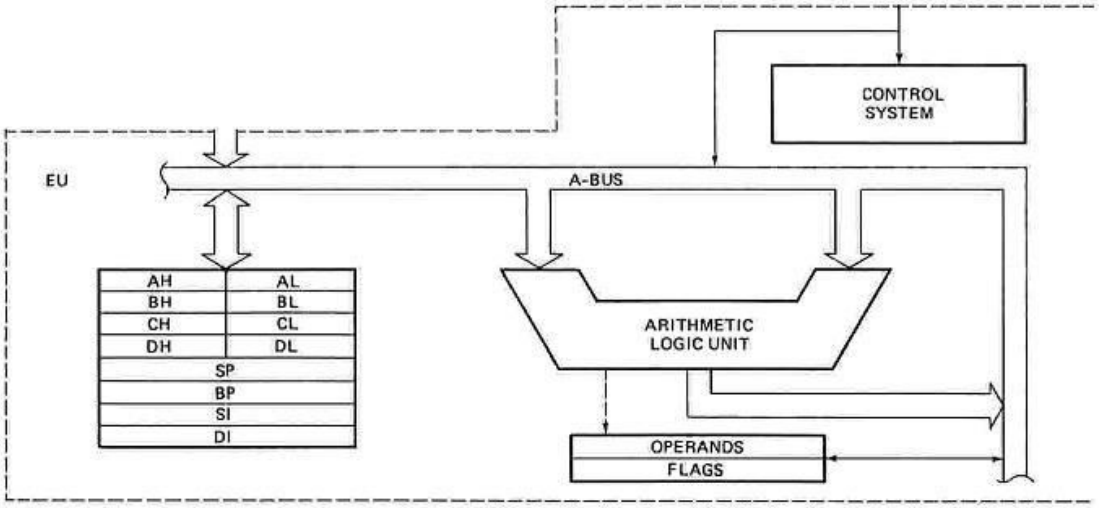
• EU executes instructions from the instruction system byte queue.

• EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

• Decodes instructions fetched by the BIU, Generate control signals and Executes instructions.

• The main parts are:

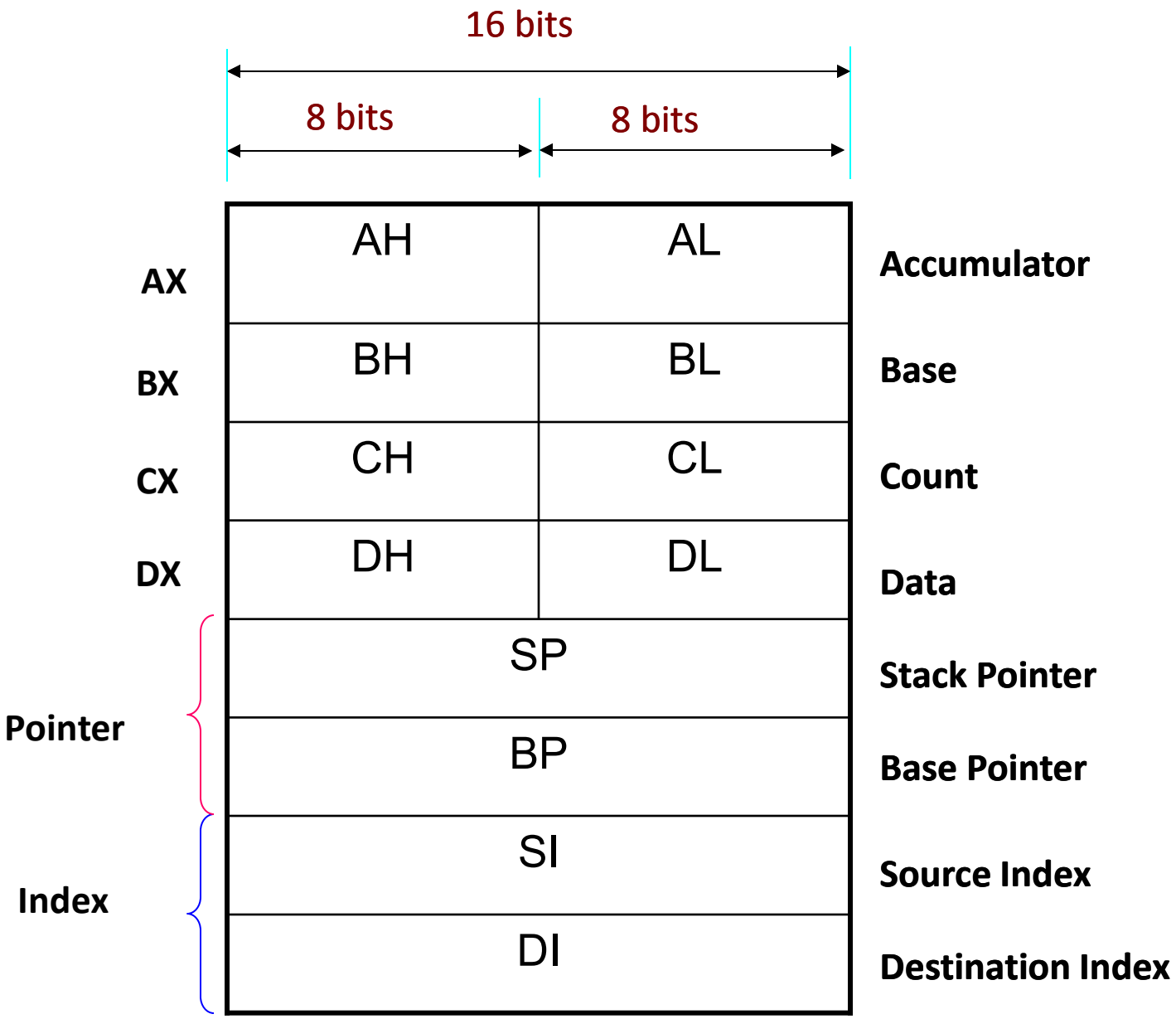
✓Control Circuitry
✓Instruction decoder
✓ALU



Functions of Execution Unit

- It receives op-code of an instruction from the QUEUE.
- It decodes it and then executes it.
- It tells BIU where to fetch the instructions or data from.
- It contains the control circuitry to perform various internal operations.
- It has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit data.

EXECUTION UNIT – General Purpose Registers



Registers of Intel 8086

- Intel 8086 contains following registers:

  - General Purpose Registers
  - Pointer and Index Registers
  - Segment Registers
  - Instruction Pointer
  - Status Flags

I. General Purpose Registers

- There are four 16-bit general purpose registers. Each of these 16-bit registers are further subdivided into two 8-bit registers:

- **AX Register**: AX register is also known as accumulator register that stores operands for arithmetic operation like divided, rotate.
- **BX Register**: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.
- **CX Register**: It is defined as a counter. It is primarily used in loop instruction to store loop counter.
- **DX Register**: DX register is used to contain I/O port address for I/O instruction. holds the high 16 bits of the product in multiply (also handles divide operations)

NOTE :- AX & DX registers:

In 8 bit multiplication, one of the operands must be in AL. The other operand can be a byte in memory location or in another 8 bit register. The resulting 16 bit product is stored in AX, with AH storing the MS byte. In 16 bit multiplication, one of the operands must be in AX. The other operand can be a word in memory location or in another 16 bit register. The resulting 32 bit product is stored in DX and AX, with DX storing the MS word and AX storing the LS word.

II. Pointer And Index Registers

- used to keep offset addresses.
- Used in various forms of memory addressing.
- In the case of SP and BP the default reference to form a physical address is the Stack Segment (SS-will be discussed under the BIU).
- The index registers (SI & DI) and the BX generally default to the Data segment register (DS).

- SP: Stack pointer
  - Used with SS to access the stack segment

- BP: Base Pointer
  - Primarily used to access data on the stack
  - Can be used to access data in other segments

- SI: Source Index register
  - is required for some string operations.
  - When string operations are performed, the SI register points to memory locations in the data segment which is addressed by the DS register. Thus, SI is associated with the DS in string operations.

- DI: Destination Index register
  - is also required for some string operations.
  - When string operations are performed, the DI register points to memory locations in the data segment which is addressed by the ES register. Thus, DI is associated with the ES in string operations.

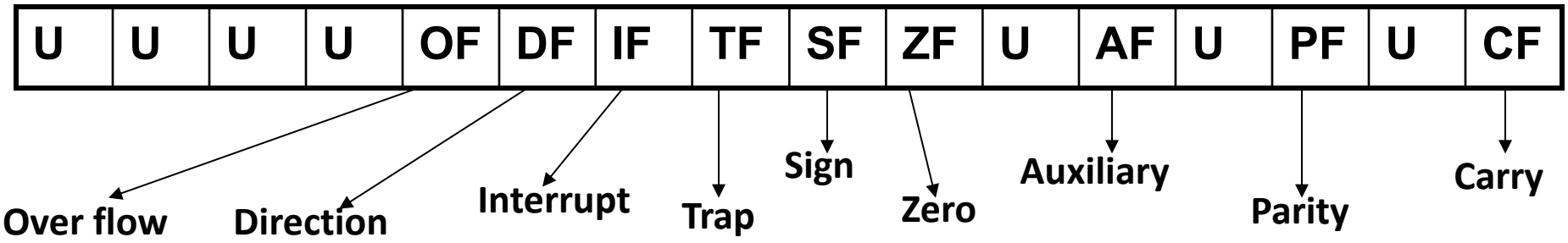
NOTE :- The SI and the DI registers may also be used to access data stored in arrays

EXECUTION UNIT – Flag Register

- A flag is a flip flop which indicates some conditions produced by the execution of an instruction or controls certain operations of the EU . Status Flags determines the current state of the accumulator.

- In 8086 The EU contains :

➢ a 16 bit flag register.



NOTE :- U - Unused

- 9 of the 16 are active flags and remaining 7 are undefined.

  → 6 flags indicates some conditions- status flags

  → 3 flags –control Flags

- Status Flags determines the current state of the accumulator.
- They are modified automatically by CPU after mathematical operations.
- This allows to determine the type of the result.
- It is also called Flag Register or Program Status Word (PSW).

Status Flags

- 8086 has 9 flags and they are divided into two categories:

  - Condition Flags
  - Control Flags

Status Flags

- Following are the nine flags:

Condition Flags	Control Flags
1. Carry Flag	1. Trap Flag
2. Auxiliary Carry Flag	2. Interrupt Flag
3. Zero Flag	3. Directional Flag
4. Sign Flag	
5. Parity Flag	
6. Overflow Flag	

Condition Flags

Condition flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:

i. **Carry Flag (CF):** This flag is set if there is a carry / borrow after an integer arithmetic.

ii. **Auxiliary Carry Flag (AF):** If an operation performed in ALU generates a carry / borrow from lower nibble (i.e. $D_0 – D_3$) to upper nibble (i.e. $D_4 – D_7$), then AF is set. It is used in BCD Addition.

iii. **Parity Flag (PF):** This flag is used to indicate the parity of result. If the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset.

Condition Flags

iv. **Zero Flag (ZF):** It is set; if the result of arithmetic or logical operation is zero else it is reset.

v. **Sign Flag (SF):** In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

vi. **Overflow Flag (OF):** It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

- **Trap Flag (TP):**
  - It is used for single step control.
  - It allows user to execute one instruction of a program at a time for debugging.
  - When trap flag is set, program can be run in single step mode.

Control Flags

- **Interrupt Flag (IF):**
  - ⦿ It is an interrupt enable / disable flag.
  - ⦿ If it is set, the INTR interrupt of 8086 is enabled and if it is reset then INTR is disabled.
  - ⦿ It can be set by executing instruction STI and can be cleared by executing CLI instruction.

Control Flags

- **Directional Flag (DF):**
  - It is used in string operation.
  - If it is set, string bytes are accessed from higher memory address to lower memory address.
  - When it is reset, the string bytes are accessed from lower memory address to higher memory address.
  - It is set with STD instruction and cleared with CLD instruction.

EXECUTION UNIT – Flag Register

Flag	Purpose
Carry (CF)	Holds the carry after addition or the borrow after subtraction. Also indicates some error conditions, as dictated by some programs and procedures .
Parity (PF)	PF=0;odd parity, PF=1;even parity.
Auxiliary (AF)	Holds the carry (half – carry) after addition or borrow after subtraction between bit positions 3 and 4 of the result (for example, in BCD addition or subtraction.)
Zero (ZF)	Shows the result of the arithmetic or logic operation. Z=1; result is zero. Z=0; The result is 0
Sign (SF)	Holds the sign of the result after an arithmetic/logic instruction execution. S=1; negative, S=0

Flag	Purpose
Trap (TF)	A control flag. Enables the trapping through an on-chip debugging feature.
Interrupt (IF)	A control flag. Controls the operation of the INTR (interrupt request) I=0; INTR pin disabled. I=1; INTR pin enabled.
Direction (DF)	A control flag. It selects either the increment or decrement mode for DI and /or SI registers during the string instructions.
Overflow (OF)	Overflow occurs when signed numbers are added or subtracted. An overflow indicates the result has exceeded the capacity of the Machine

Execution unit – Flag Register

- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL,1 is executed then the following happen

AL = 80h

CF = 0; there is no carry out of bit 7

PF = 0; 80h has an odd number of ones

AF = 1; there is a carry out of bit 3 into bit 4

ZF = 0; the result is not zero

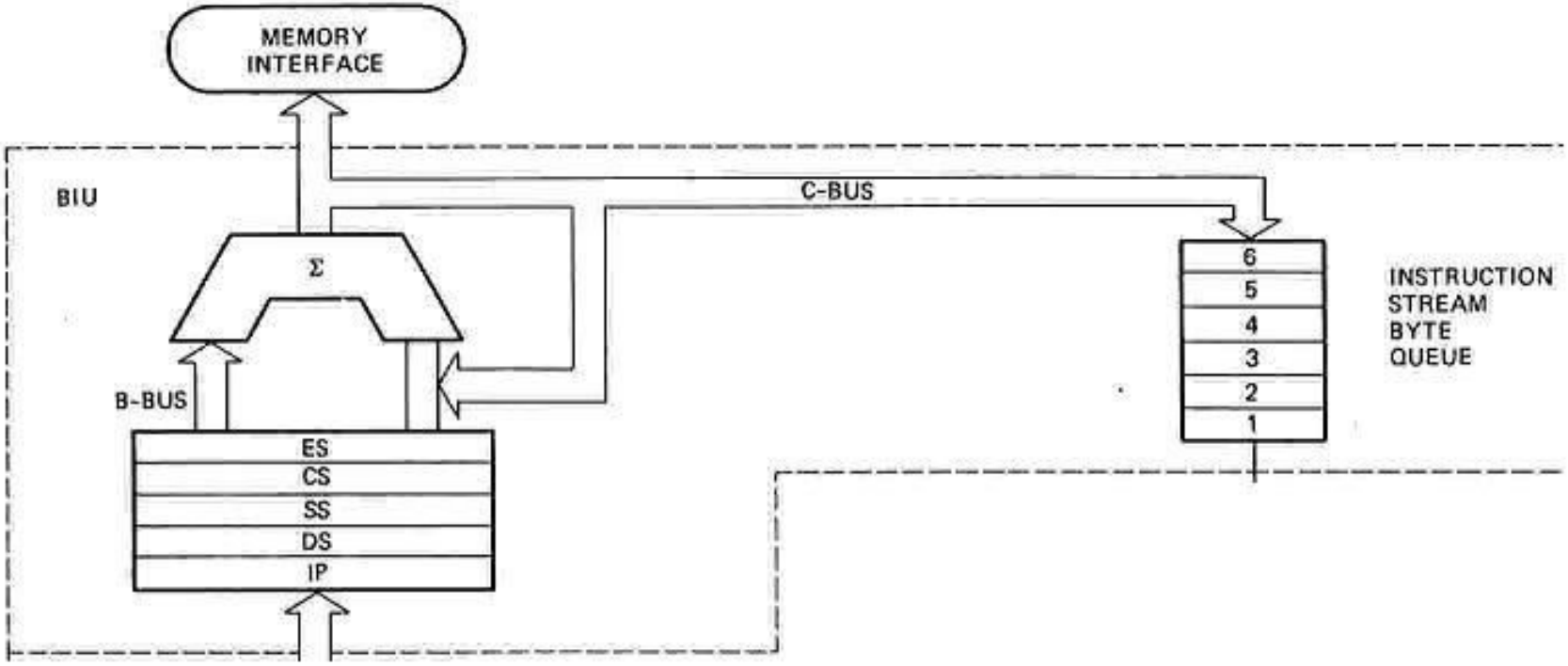
SF = 1; bit seven is one

OF = 1; the sign bit has changed

BUS INTERFACE UNIT (BIU)

It Contains :

- **6-byte Instruction Queue (Q)**
- **The Segment Registers (CS, DS, ES, SS).**
- **The Instruction Pointer (IP).**
- **The Address Summing block (Σ)**



THE QUEUE (Q)

- The BIU uses a mechanism known as an **instruction stream queue** to implement a ***pipeline architecture.***

- This queue permits pre-fetch of up to **6 bytes** of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by pre-fetching the next sequential instruction.

- These pre-fetching instructions are held in its **FIFO queue**. With its **16** bit data bus, the BIU fetches two instruction bytes in a single memory cycle.

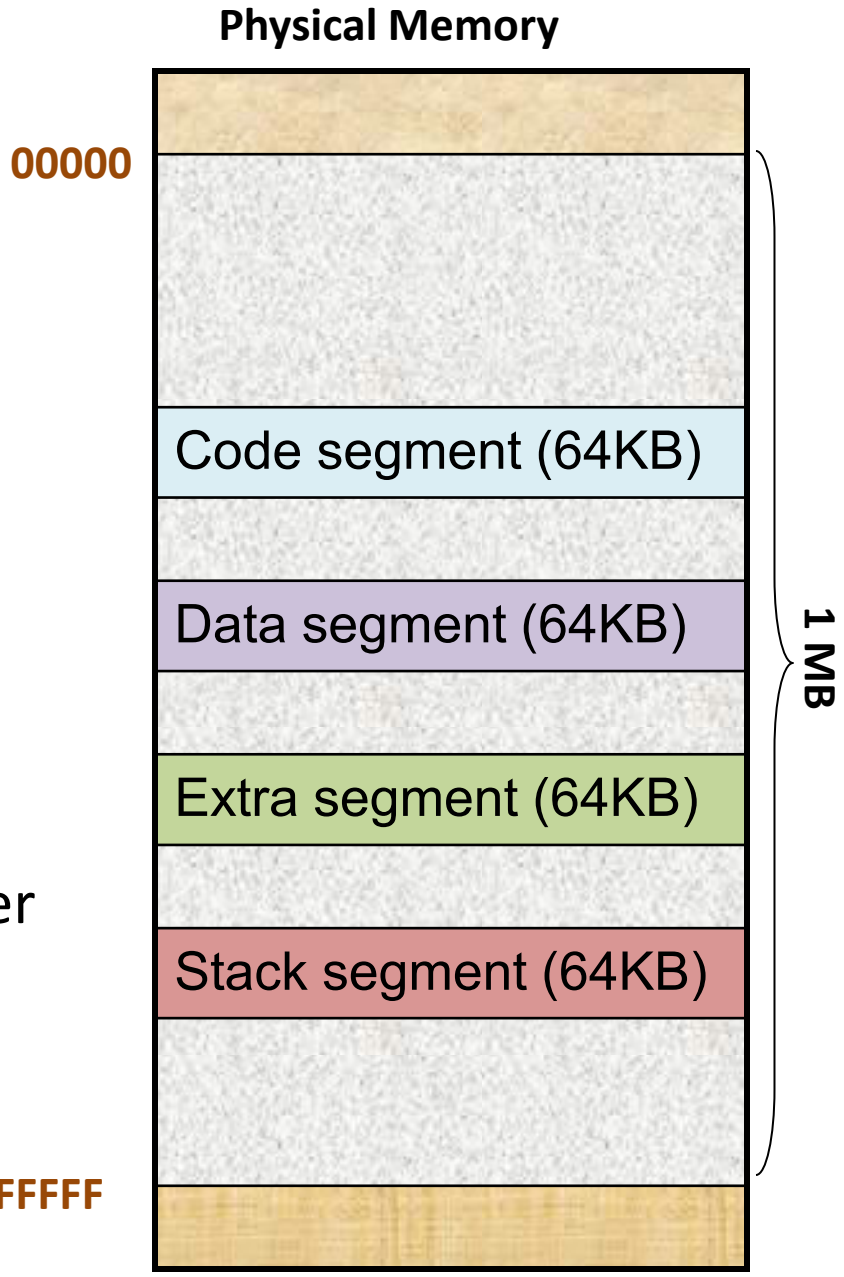
- After a byte is loaded at the input end of the queue, it automatically shifts up through the **FIFO** to the empty location nearest the output.

- The **EU accesses the queue from the output end**. It reads one instruction byte after the other from the output of the queue.

- The intervals of no bus activity, which may occur between bus cycles are known as ***Idle state*.**

Segmented Memory

- The memory in an 8086/88 based system is organized as segmented memory.
- The CPU 8086 is able to address 1Mbyte of memory.
- The Complete physically available memory may be divided into a number of logical segments.



- The size of each segment is 64 KB.
- A segment is an area that begins at any location which is divisible by 16.
- A segment may be located any where in the memory.
- Each of these segments can be used for a specific function.
  - Code segment is used for storing the instructions.
  - The stack segment is used as a stack and it is used to store the return addresses.
  - The data and extra segments are used for storing data byte.

NOTE:- In the assembly language programming, more than one data/ code/ stack segments can be defined. But only one segment of each type can be accessed at any time.

- The 4 segments are Code, Data, Extra and Stack segments.
- A Segment is a 64kbyte block of memory.
- The 16 bit contents of the segment registers in the BIU actually point to the starting location of a particular segment.
- Segments may be overlapped or non-overlapped.

Advantages of Segmented memory Scheme

i. Allows the memory capacity to be 1MB although the actual addresses to be handled are of 16 bit size.

ii. Allows the placing of code, data and stack portions of the same program in different parts (segments) of the memory, for data and code protection.

iii. Permits a program and/or its data to be put into different areas of memory each time program is executed, i.e. provision for relocation may be done.

iv. The segment registers are used to allow the instruction, data or stack portion of a program to be more than 64Kbytes long.

v. The above can be achieved by using more than one code, data or stack segments.

Segment registers

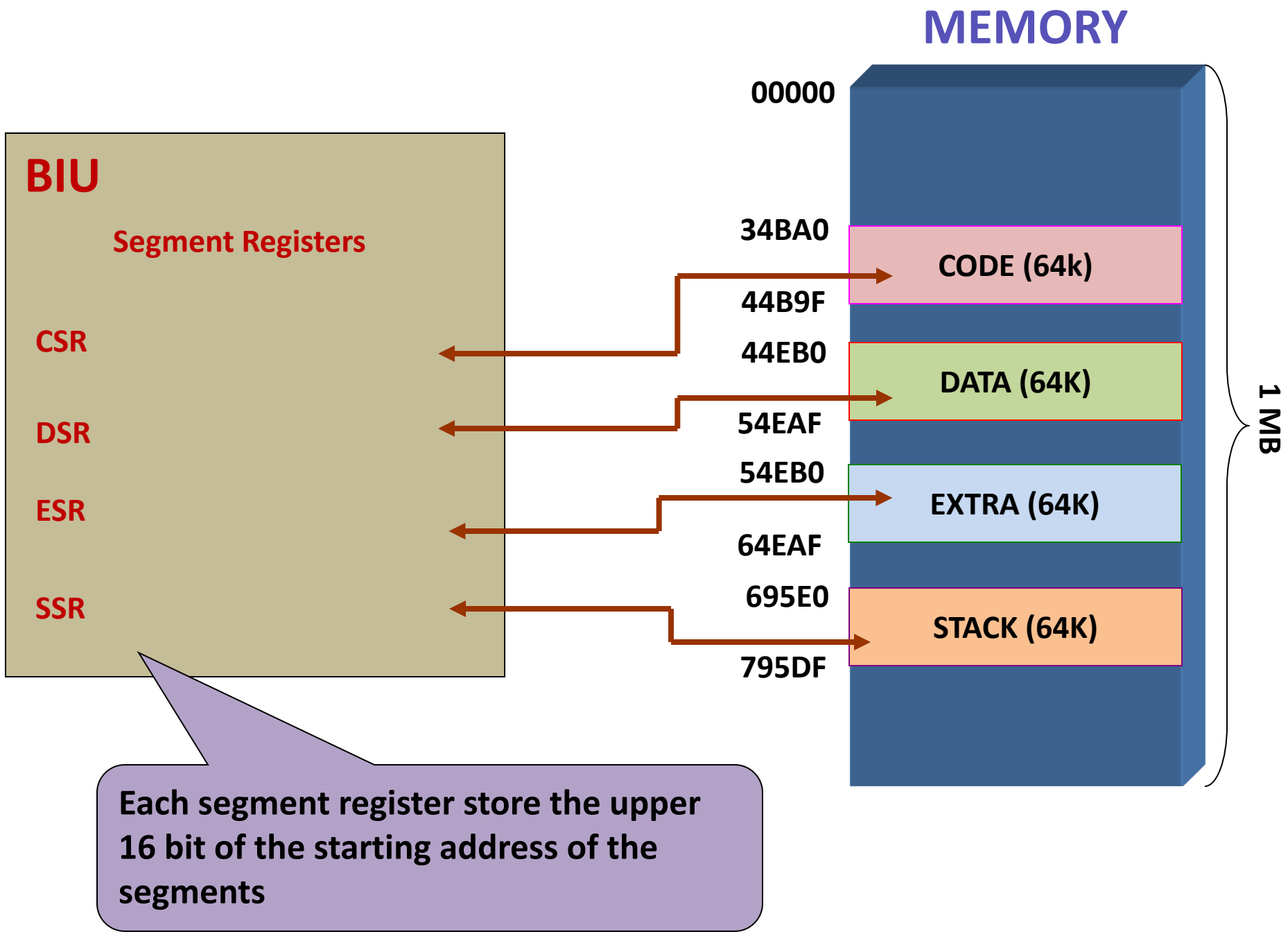
- In 8086/88 the processors have 4 segments registers.
- Code Segment register (CS), Data Segment register (DS), Extra Segment register (ES) and Stack Segment (SS) register.
- All are 16 bit registers.
- Each of the Segment registers store the upper 16 bit address of the starting address of the corresponding segments.

i. CS Register: This register contains the initial address of the code segment. This address plus the offset value contained in the instruction pointer (IP) indicates the address of an instruction to be fetched for execution.

ii. SS Register: The stack segment register contains the initial address of the stack segment. This address plus the value contained on the stack pointer (SP) is used for stack operations.

iii. DS Register: The data segment register contains the initial address of the current data segment. This address plus the offset value in instruction causes a reference to a specific location in the data segment.

iv. ES Register: Extra segment is used by some string operations. The extra segment register contains the initial address of the extra segment. String instructions always use the ES and DI registers to calculate the physical address for the destination.

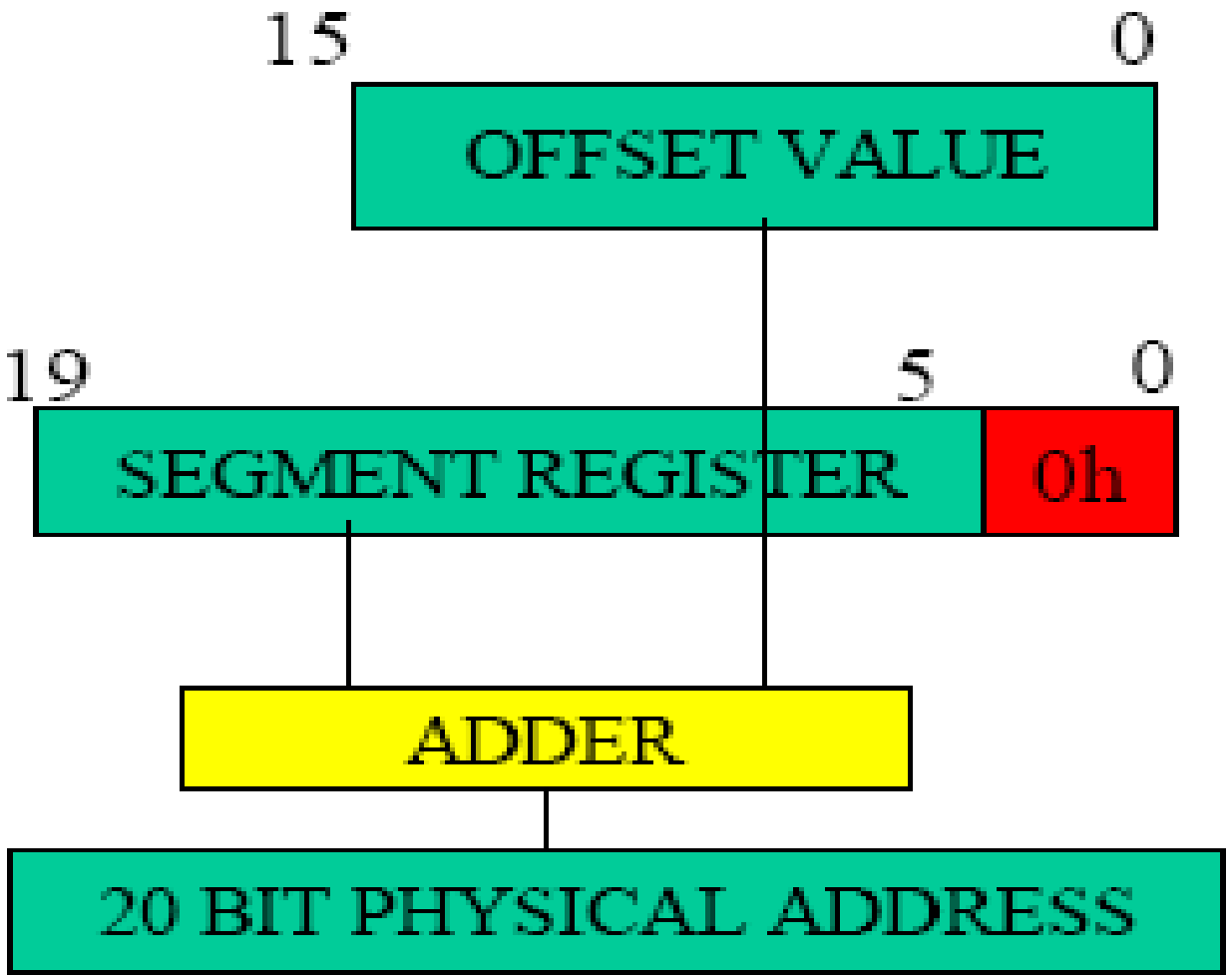

MEMORY
00000
BIU
Segment Registers
CSR
DSR
ESR
SSR
34BA0
CODE (64k)
44B9F
44EB0
DATA (64K)
54EAF
54EB0
EXTRA (64K)
64EAF
695E0
STACK (64K)
795DF
1 MB
Each segment register store the upper 16 bit of the starting address of the segments

Instruction pointer & summing block

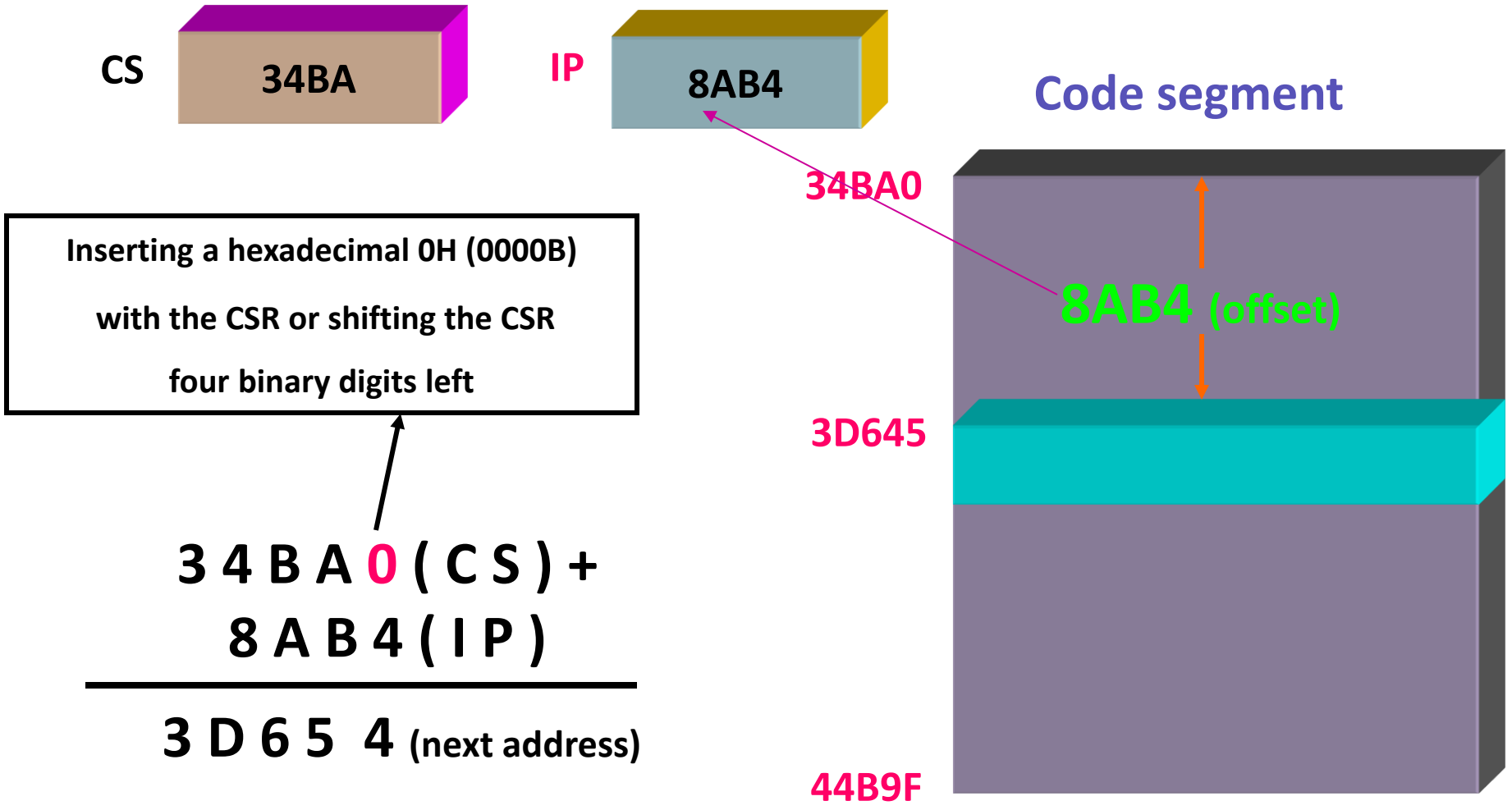
- The instruction pointer register contains a 16-bit offset address of instruction that is to be executed next.
- The IP always references the Code segment register (CS).
- The value contained in the instruction pointer is called as an offset because this value must be added to the base address of the code segment, which is available in the CS register to find the 20-bit physical address.

- The value of the instruction pointer is incremented after executing every instruction.

- To form a 20 bit address of the next instruction, the 16 bit address of the IP is added (by the address summing block) to the address contained in the CS , which has been shifted four bits to the left.

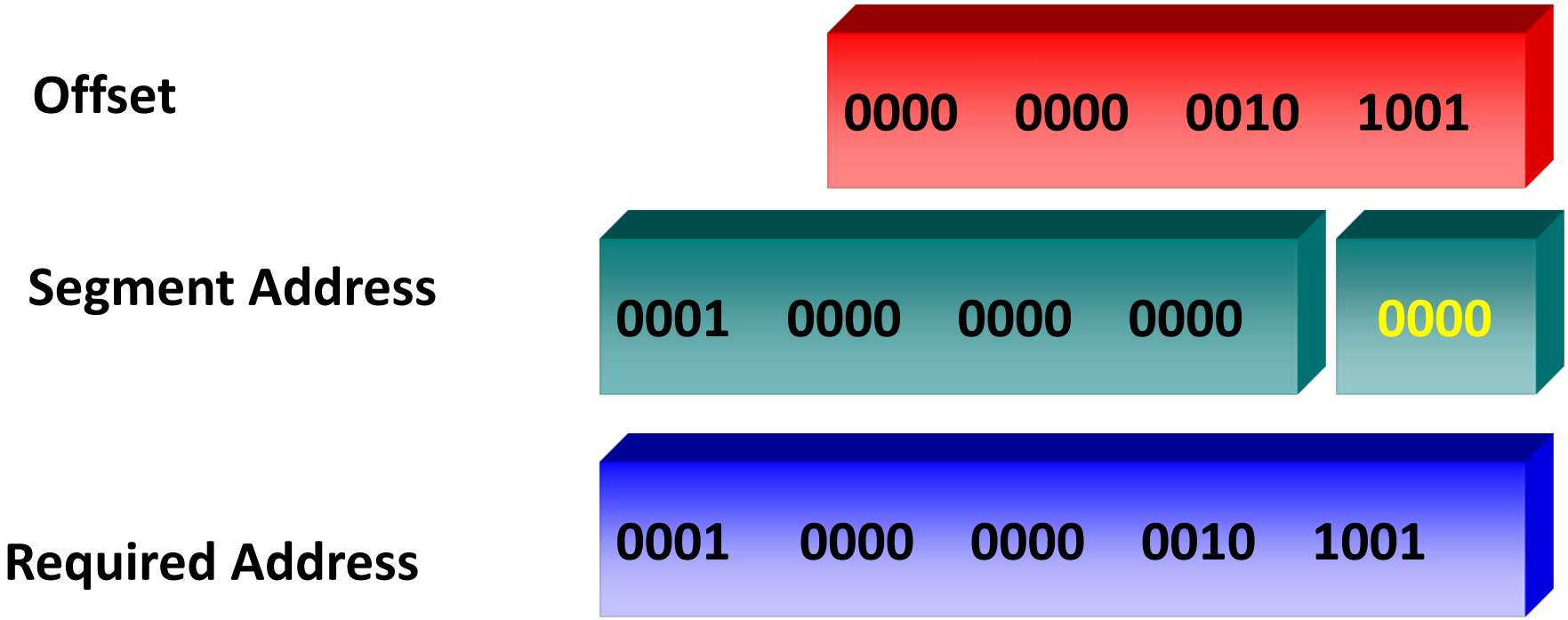

15
0
OFFSET VALUE
19
5
0
SEGMENT REGISTER
0h
ADDER
20 BIT PHYSICAL ADDRESS

- The following examples shows the CS:IP scheme of address formation:



Example For Address Calculation (segment: offset)

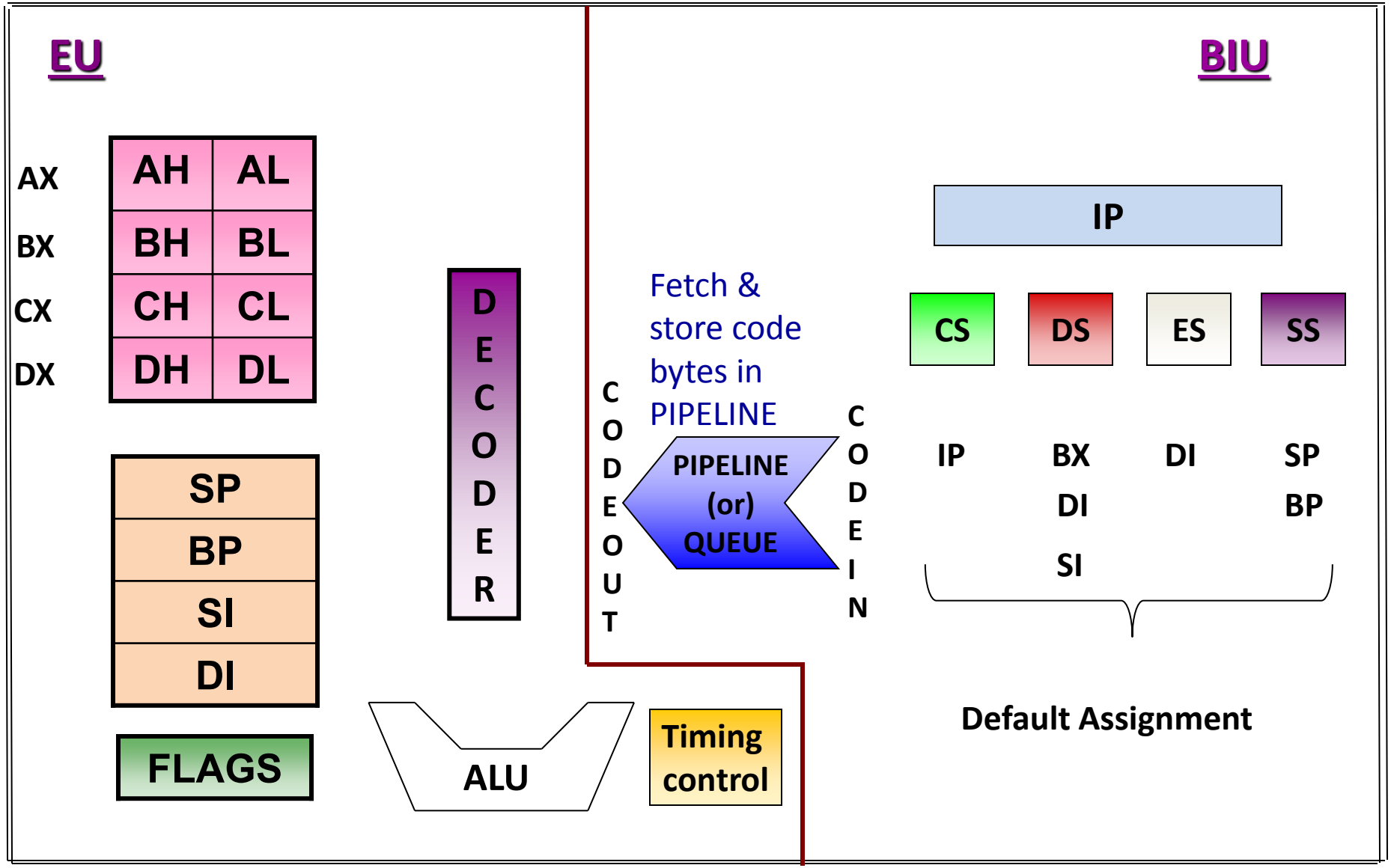
- If the data segment starts at location 1000H and a data reference contains the address 29H where is the actual data?



Segment and Address register combination

- **CS:IP**
- **SS:SP SS:BP**
- **DS:BX DS:SI**
- **DS:DI (for other than string operations)**
- **ES:DI (for string operations)**

Summary of Registers & Pipeline of 8086 µP



Physical Address (PA) generation :

Generally Physical Address (20 Bit) = Segment Base Address (SBA)+ Effective Address (EA)

Code Segment :
Physical Address (PA) = CS Base Address+ Instruction Pointer (IP)

Data Segment (DS)
PA = DS Base Address + EA can be in BX or SI or DI

Stack Segment (SS)
PA + SS Base Address + EA can be SP or BP

Extra Segment (ES)
PA = ES Base Address + EA in DI

Memory Segmentation

- The total memory size is divided into segments of various sizes.
- A segment is just an area in memory.
- The process of dividing memory this way is called **Segmentation**.

Memory Segmentation

- In memory, data is stored as bytes.
- Each byte has a specific address.
- Intel 8086 has 20 lines address bus.
- With 20 address lines, the memory that can be addressed is 2^{20} bytes.
- 2^{20} = 1,048,576 bytes (1 MB).
- 8086 can access memory with address ranging from 00000 H to FFFFF H.

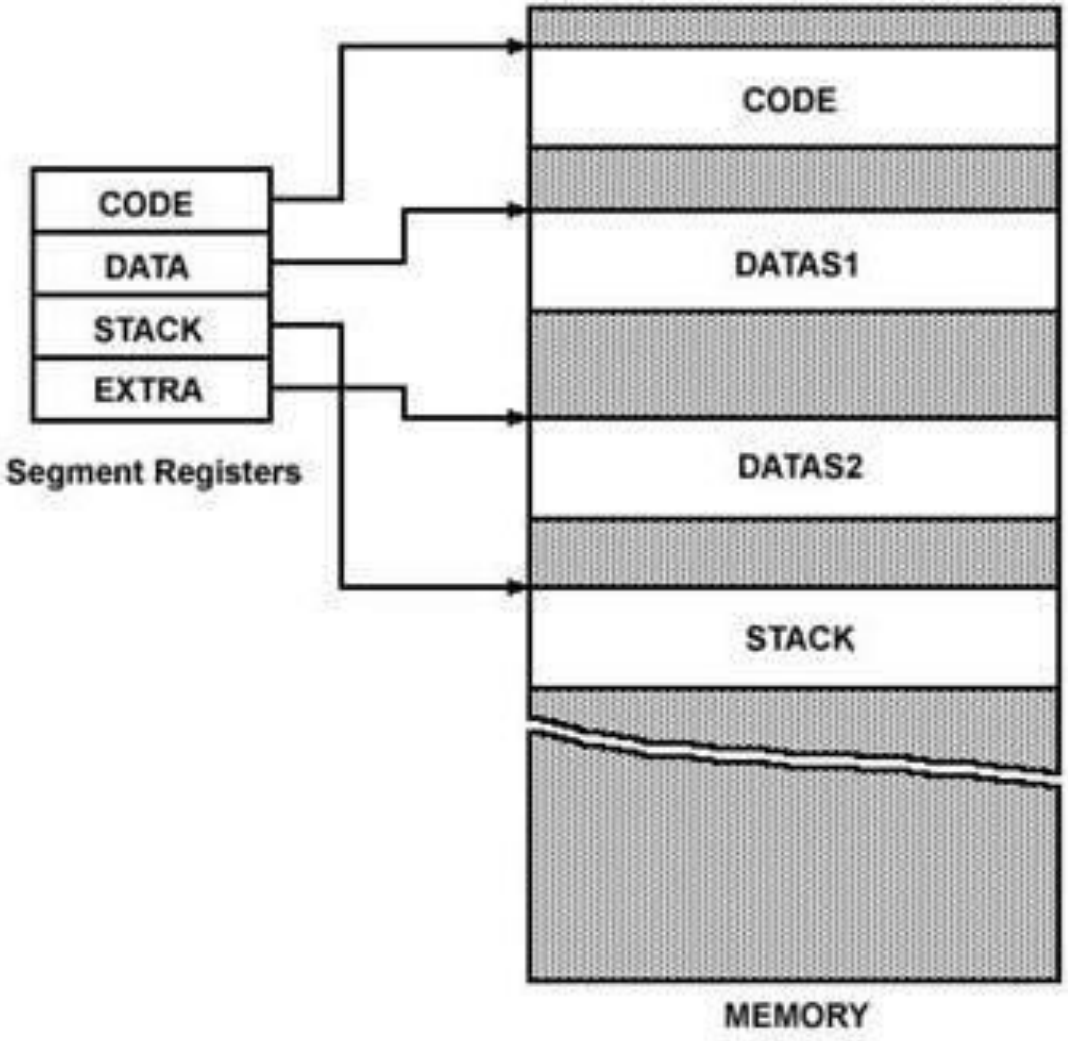
Memory Segmentation

- In 8086, memory has four different types of segments.
- These are:
  - Code Segment
  - Data Segment
  - Stack Segment
  - Extra Segment

Segment Registers

- Each of these segments are addressed by an address stored in corresponding segment register.
- These registers are 16-bit in size.
- Each register stores the base address (starting address) of the corresponding segment.
- Because the segment registers cannot store 20 bits, they only store the upper 16 bits.

Segment Registers



Segment Registers

- How is a 20-bit address obtained if there are only 16-bit registers?
- The answer lies in the next few slides.
- The 20-bit address of a byte is called its **Physical Address**.
- But, it is specified as a **Logical Address**.
- Logical address is in the form of:

Base Address : Offset

- Offset is the displacement of the memory location from the starting location of the segment.

Example

- The value of Data Segment Register (DS) is 2222 H.
- To convert this 16-bit address into 20-bit, the BIU appends 0H to the LSBs of the address.
- After appending, the starting address of the Data Segment becomes 22220H.
- If the data at any location has a logical address specified as:

2222 H : 0016 H

- Then, the number 0016 H is the offset.
- 2222 H is the value of DS.

Example (Contd.)

- To calculate the effective address of the memory, BIU uses the following formula:
  - Effective Address = Starting Address of Segment + Offset
- To find the starting address of the segment, BIU appends the contents of Segment Register with 0H.
- Then, it adds offset to it.

Example (Contd.)

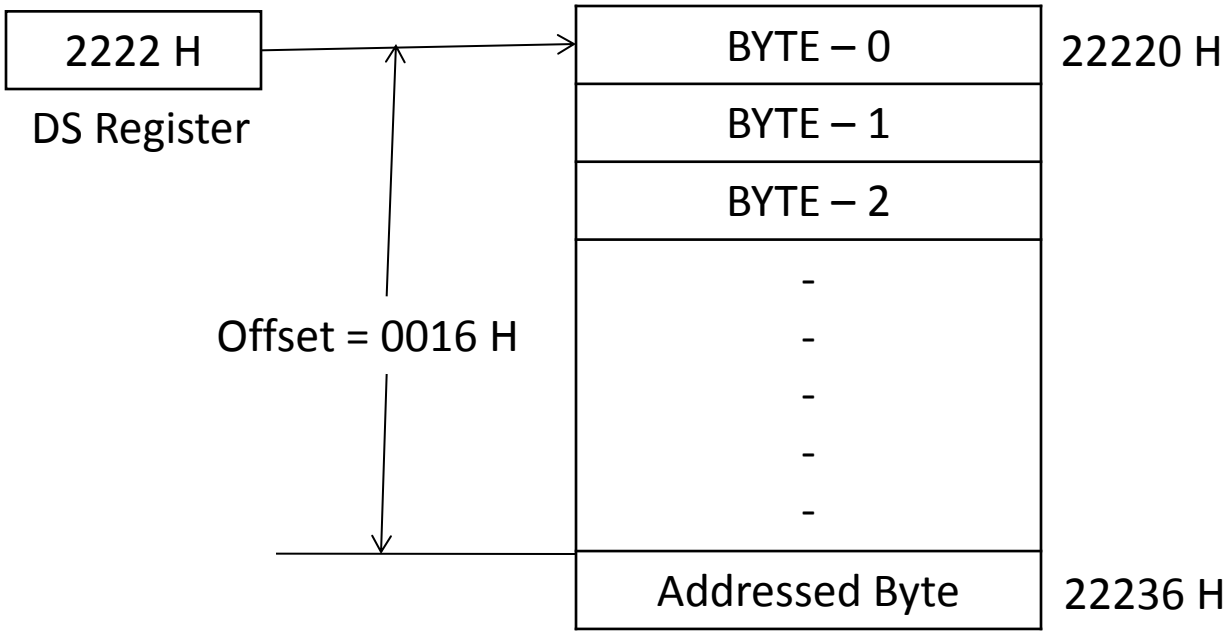
- Therefore,

EA = 22220 H

+ 0016 H

22236 H

Example (Contd.)



Max. Size of Segment

- All offsets are limited to 16-bits.
- It means that the maximum size possible for segment is 2^{16} = 65,535 bytes (64 KB).
- The offset of the first location within the segment is 0000 H.
- The offset of the last location in the segment is FFFF H.

Where to Look for the Offset ?

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Question

- The contents of the following registers are:

  CS = 1111 H

  DS = 3333 H

  SS = 2526 H

  IP = 1232 H

  SP = 1100 H

  DI = 0020 H

- Calculate the corresponding physical addresses for the address bytes in CS, DS and SS.

Solution

1. CS = 1111 H

- The base address of the code segment is 11110 H.
- Effective address of memory is given by 11110H + 1232H = 12342H.

2. DS = 3333 H

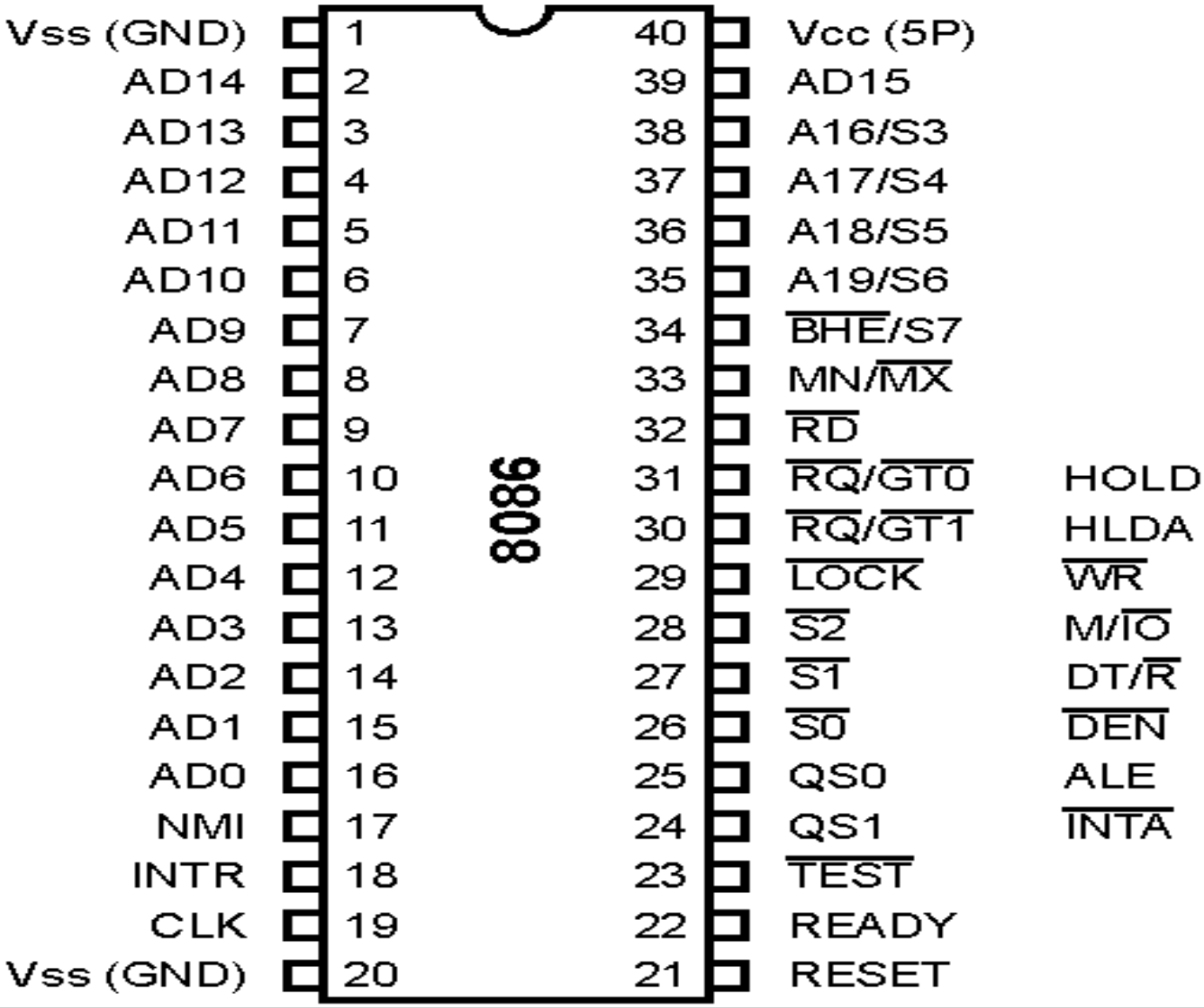
- The base address of the data segment is 33330 H.
- Effective address of memory is given by 33330H + 0020H = 33350H.

3. SS = 2526 H

- The base address of the stack segment is 25260 H.
- Effective address of memory is given by 25260H + 1100H = 26350H.

Pin Diagram 8086

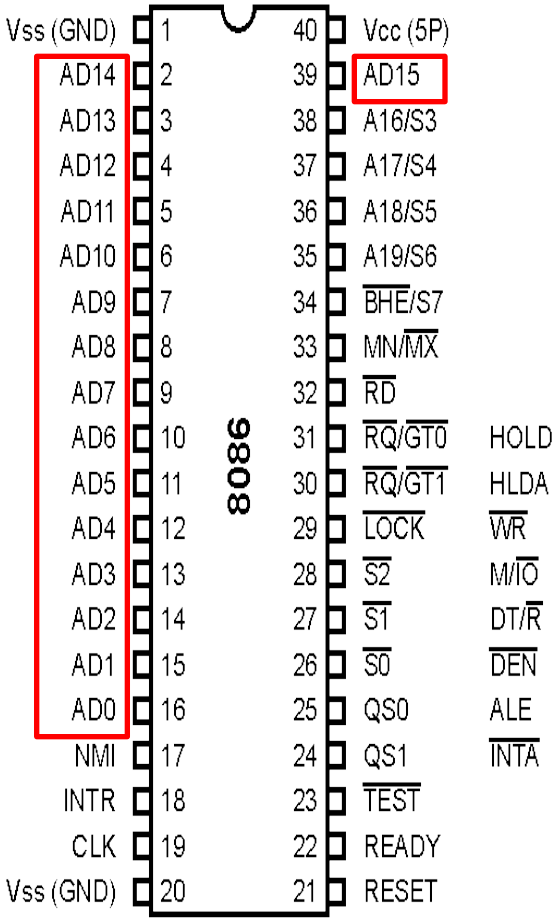
Pin Diagram of Intel 8086



$AD_0 – AD_{15}$

Pin 16-2, 39 (Bi-directional)

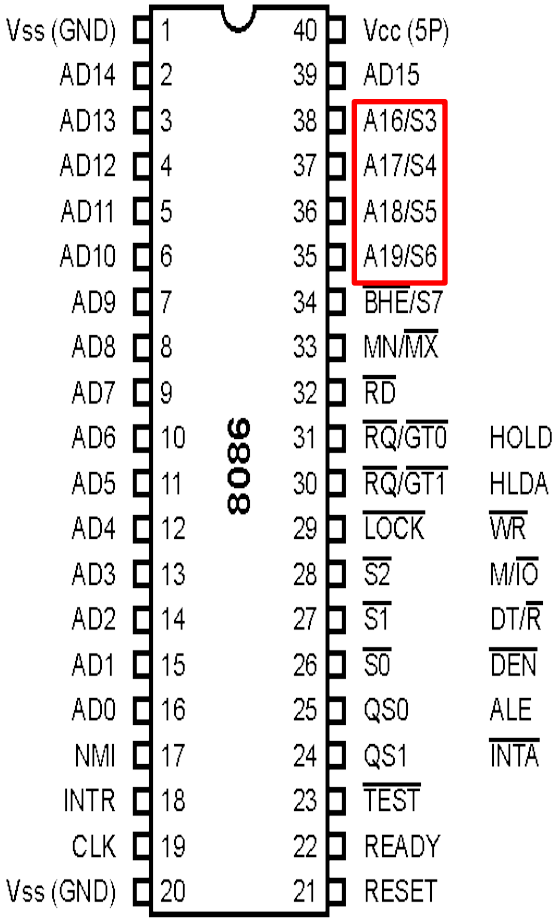
- These lines are multiplexed bi-directional address/data bus.
- During T_1, they carry lower order 16-bit address.
- In the remaining clock cycles, they carry 16-bit data.
- AD_0-AD_7 carry lower order byte of data.
- AD_8-AD_{15} carry higher order byte of data.



A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3

Pin 35-38 (Unidirectional)

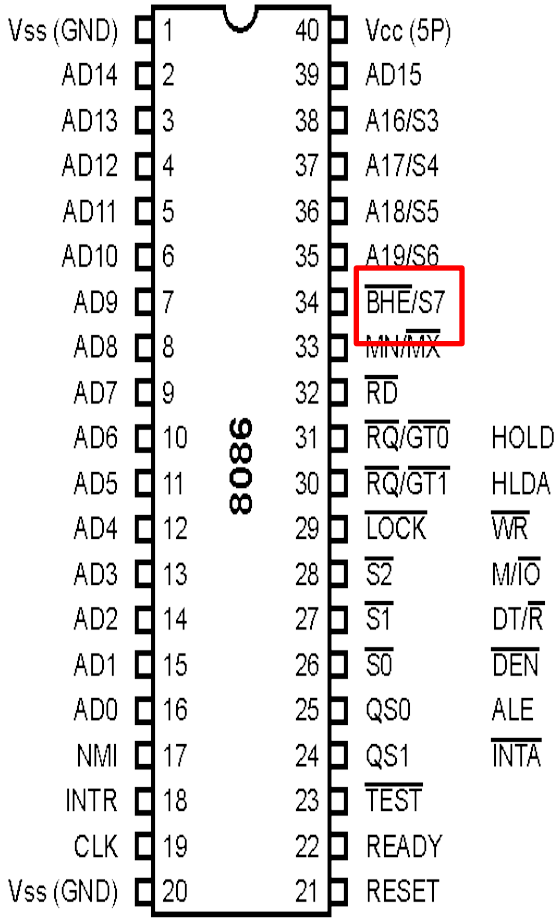
- These lines are multiplexed unidirectional address and status bus.
- During T_1, they carry higher order 4-bit address.
- In the remaining clock cycles, they carry status signals.



$\overline{BHE}$ / S_7

Pin 34 (Output)

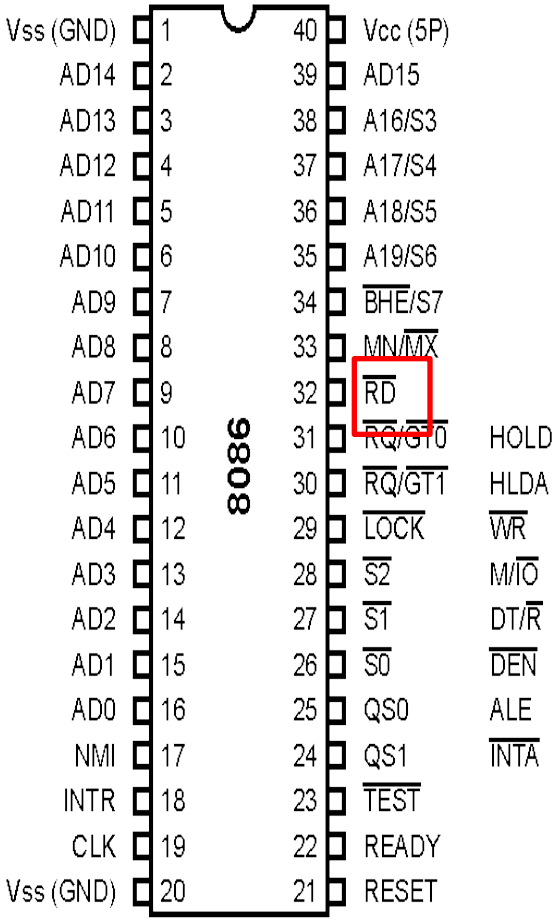
- BHE stands for Bus High Enable.
- BHE signal is used to indicate the transfer of data over higher order data bus ($D_8 – D_{15}$).
- 8-bit I/O devices use this signal.
- It is multiplexed with status pin S_7.



$\overline{RD}$ (Read)

Pin 32 (Output)

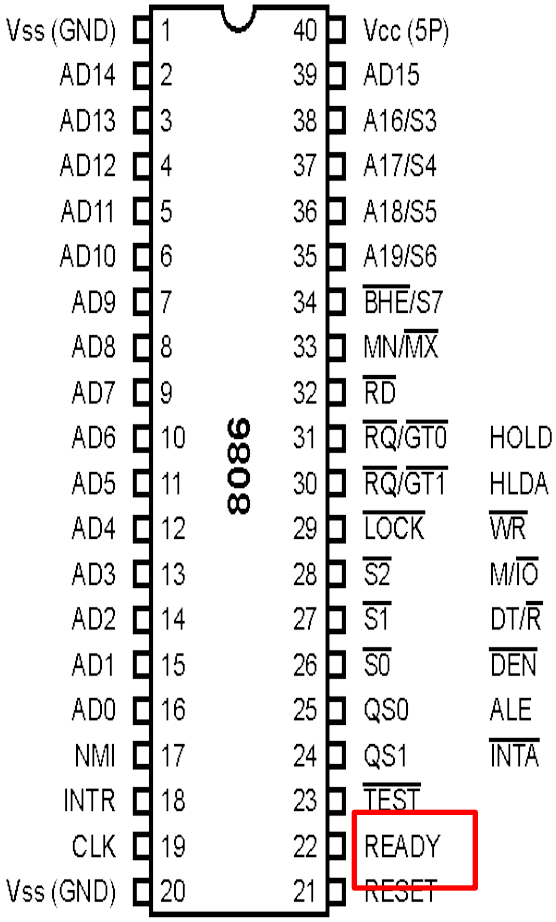
- It is a read signal used for read operation.
- It is an output signal.
- It is an active low signal.



READY

Pin 22 (Input)

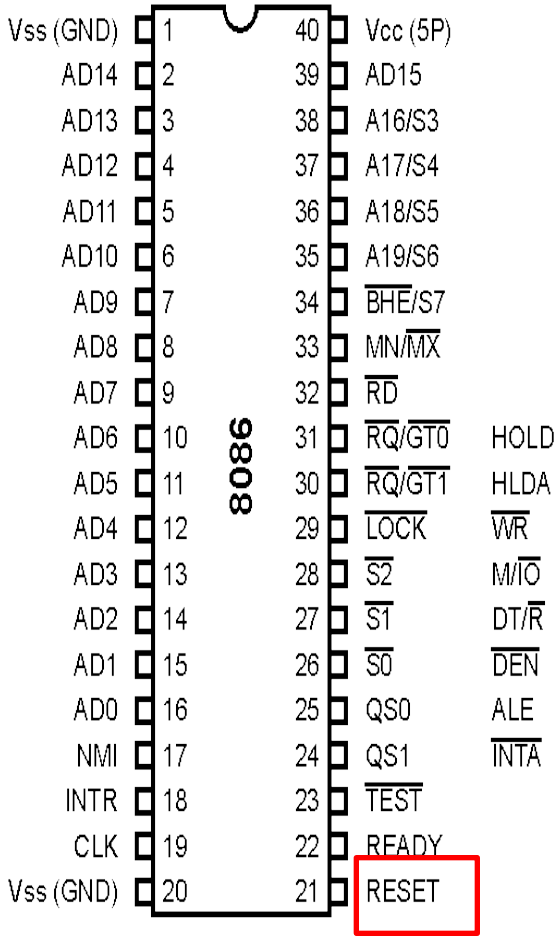
- This is an acknowledgement signal from slower I/O devices or memory.
- It is an active high signal.
- When high, it indicates that the device is ready to transfer data.
- When low, then microprocessor is in wait state.



RESET

Pin 21 (Input)

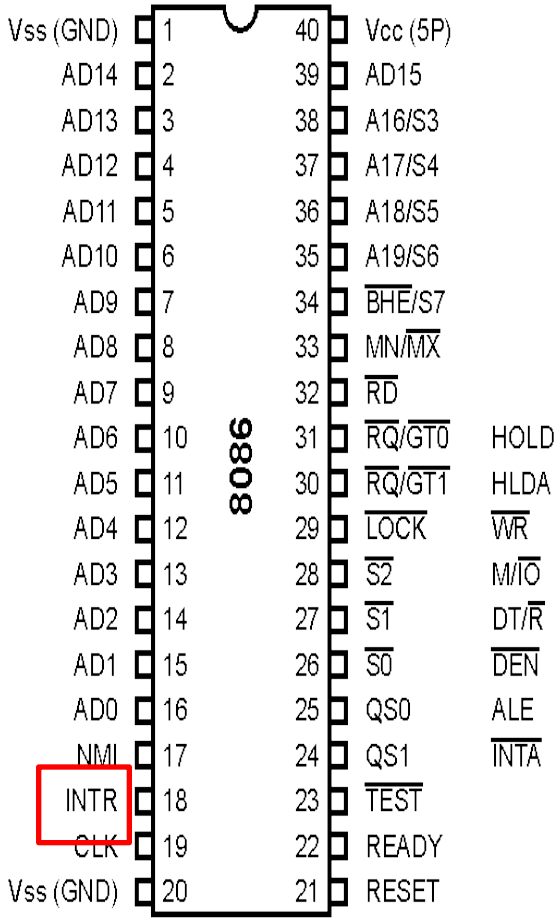
- It is a system reset.
- It is an active high signal.
- When high, microprocessor enters into reset state and terminates the current activity.
- It must be active for at least four clock cycles to reset the microprocessor.



INTR

Pin 18 (Input)

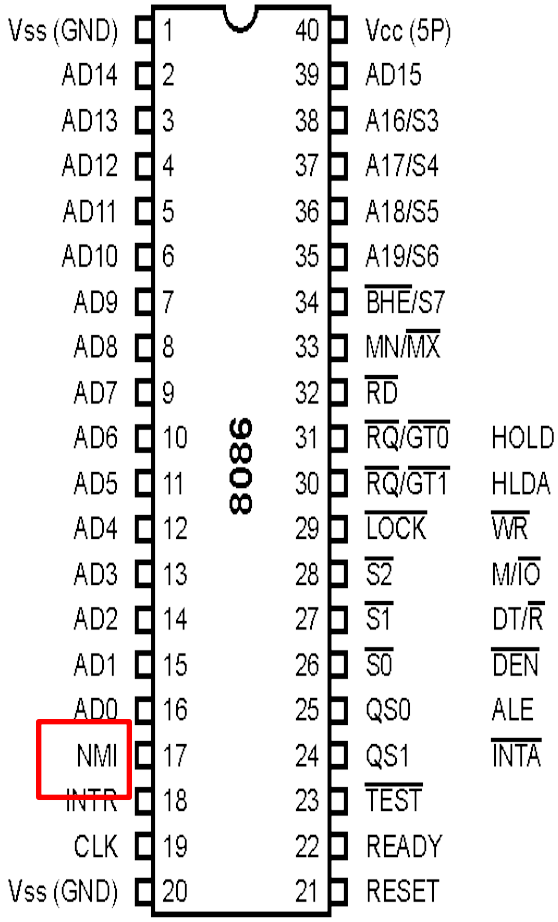
- It is an interrupt request signal.
- It is active high.
- It is level triggered.



NMI

Pin 17 (Input)

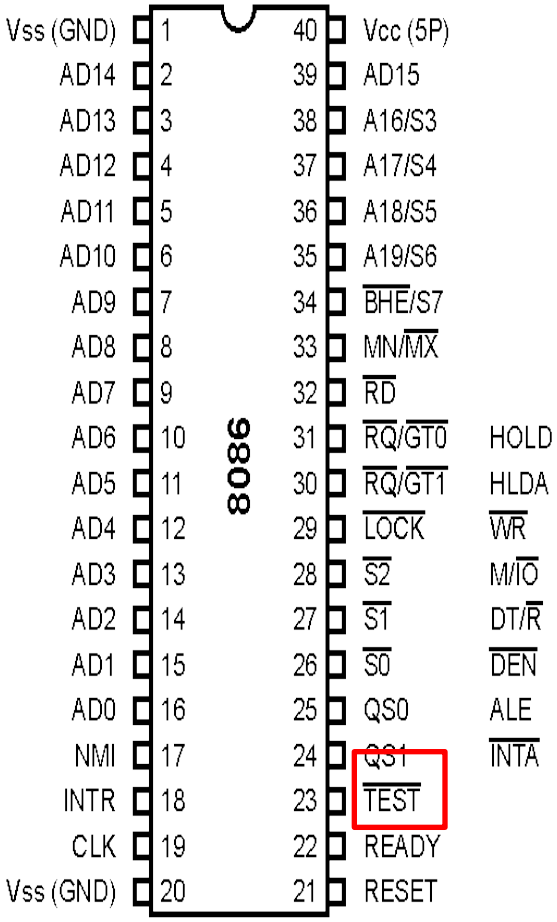
- It is a non-maskable interrupt signal.
- It is an active high.
- It is an edge triggered interrupt.



$\overline{\text{TEST}}$

Pin 23 (Input)

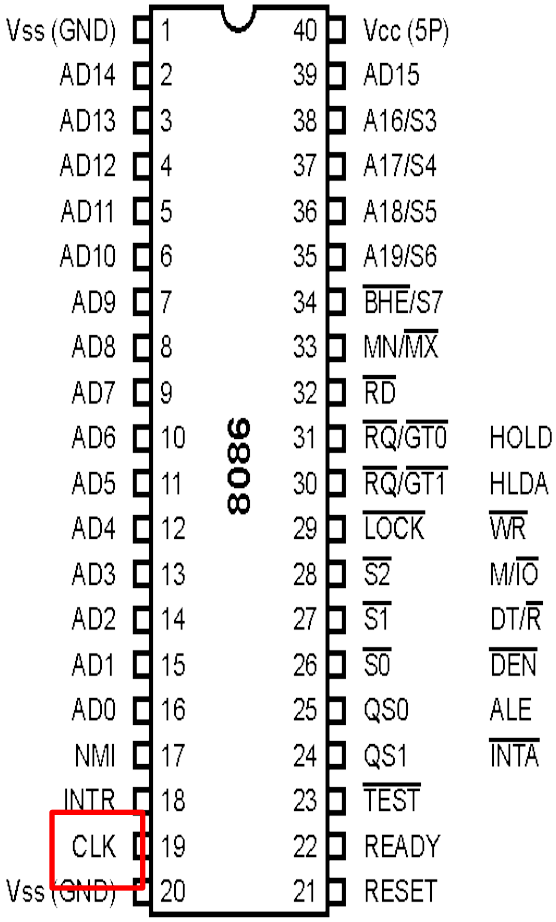
- It is used to test the status of math co-processor 8087.
- The $\overline{\text{BUSY}}$ pin of 8087 is connected to this pin of 8086.
- If low, execution continues else microprocessor is in wait state.



CLK

Pin 19 (Input)

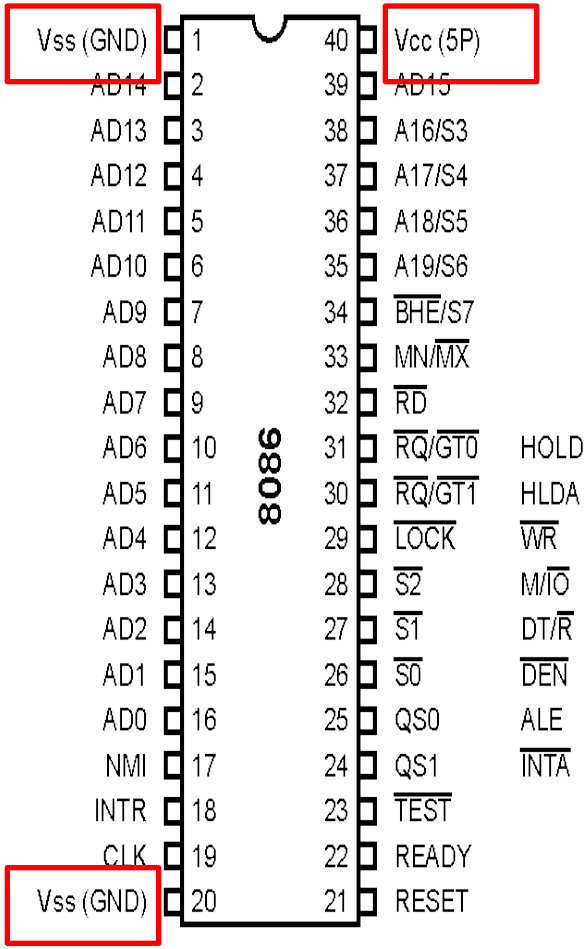
- This clock input provides the basic timing for processor operation.
- It is symmetric square wave with 33% duty cycle.
- The range of frequency of different versions is 5 MHz, 8 MHz and 10 MHz.



V_{CC} and V_{SS}

Pin 40 and Pin 20 (Input)

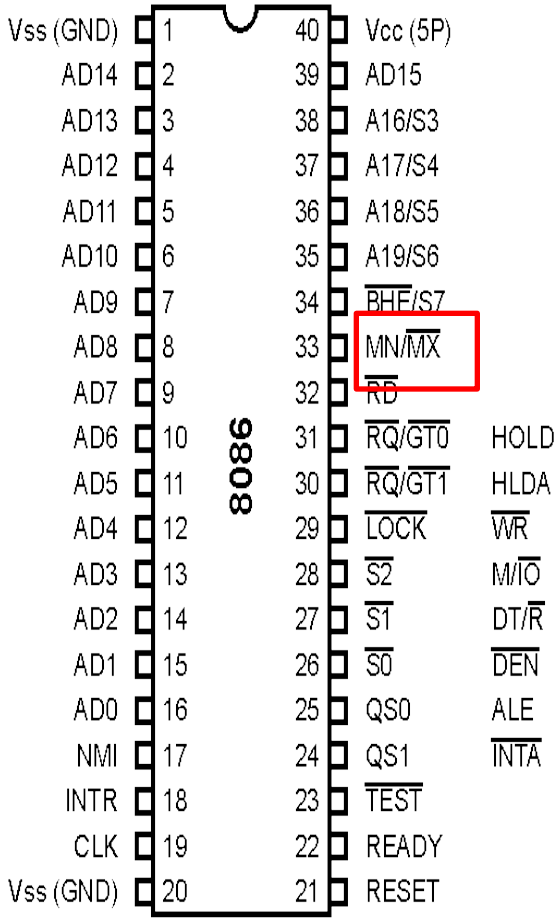
- V_{CC} is power supply signal.
- +5V DC is supplied through this pin.
- V_{SS} is ground signal.



MN / $\overline{MX}$

Pin 33 (Input)

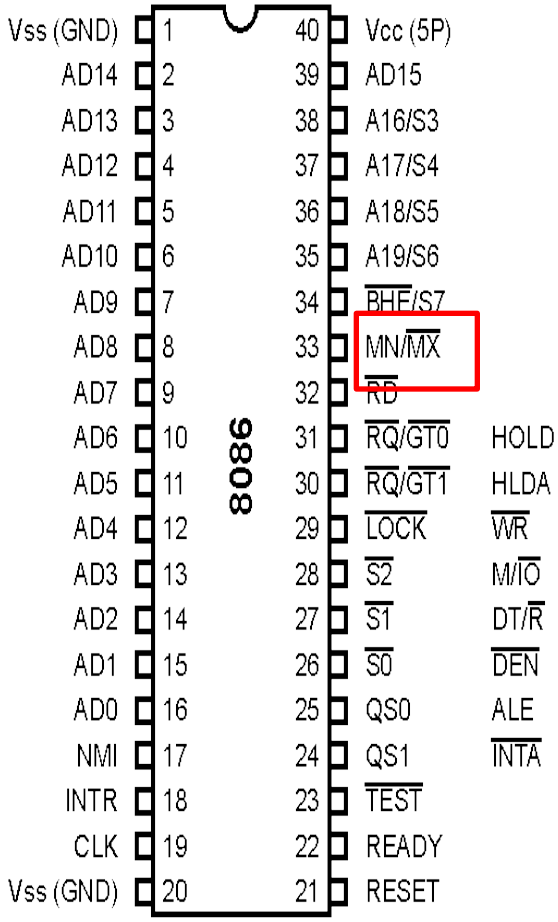
- 8086 works in two modes:
  - Minimum Mode
  - Maximum Mode
- If MN/$\overline{MX}$ is high, it works in minimum mode.
- If MN/$\overline{MX}$ is low, it works in maximum mode.



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

Pin 33 (Input)

- Pins 24 to 31 issue two different sets of signals.
- One set of signals is issued when CPU operates in minimum mode.
- Other set of signals is issued when CPU operates in maximum mode.

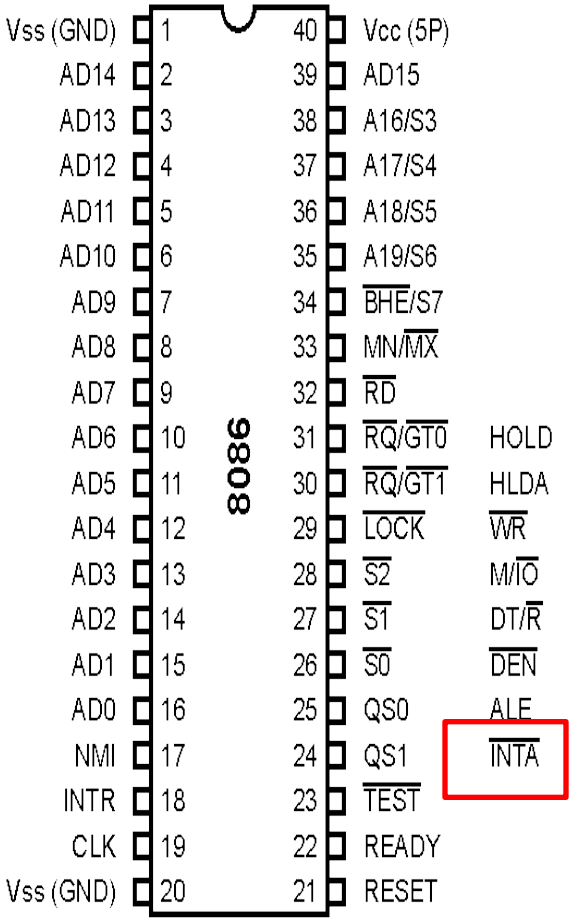


Pin Description for Minimum Mode

$\overline{\text{INTA}}$

Pin 24 (Output)

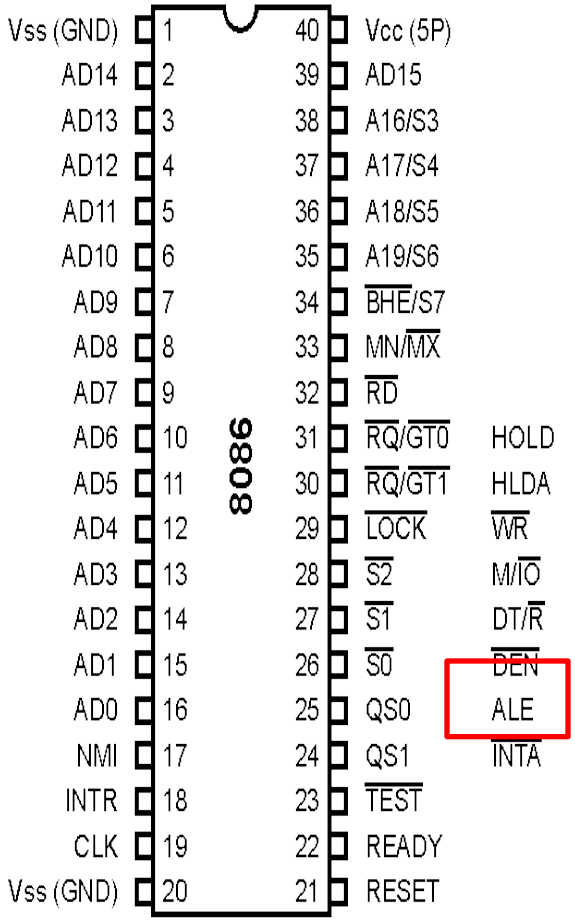
- This is an interrupt acknowledge signal.
- When microprocessor receives INTR signal, it acknowledges the interrupt by generating this signal.
- It is an active low signal.



ALE

Pin 25 (Output)

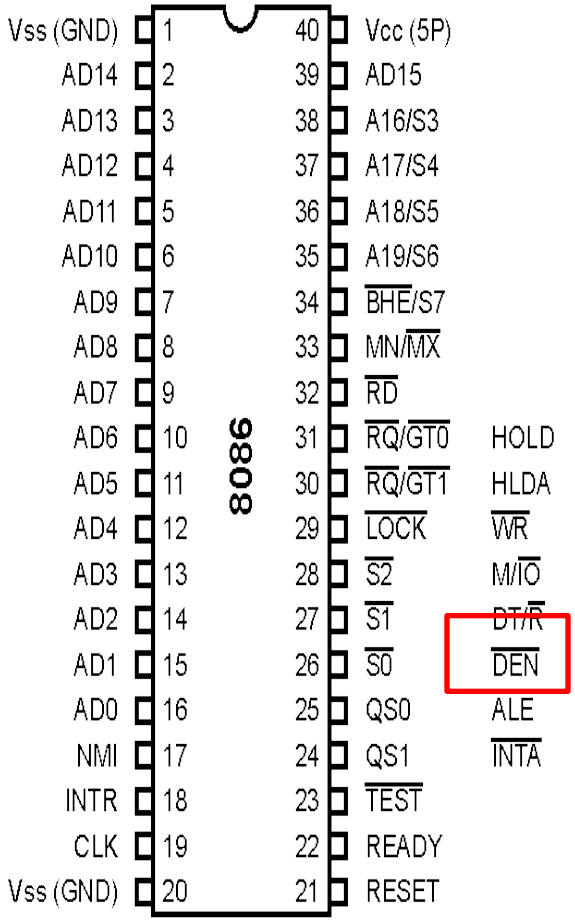
- This is an Address Latch Enable signal.
- It indicates that valid address is available on bus $AD_0 – AD_{15}$.
- It is an active high signal and remains high during T_1 state.
- It is connected to enable pin of latch 8282.



$\overline{\text{DEN}}$

Pin 26 (Output)

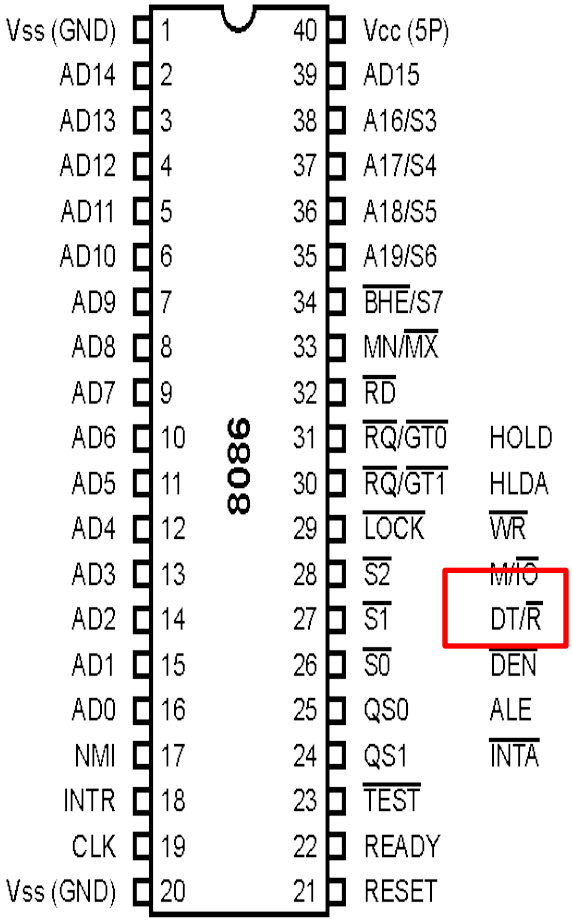
- This is a Data Enable signal.
- This signal is used to enable the transceiver 8286.
- Transceiver is used to separate the data from the address/data bus.
- It is an active low signal.



DT / $\overline{R}$

Pin 27 (Output)

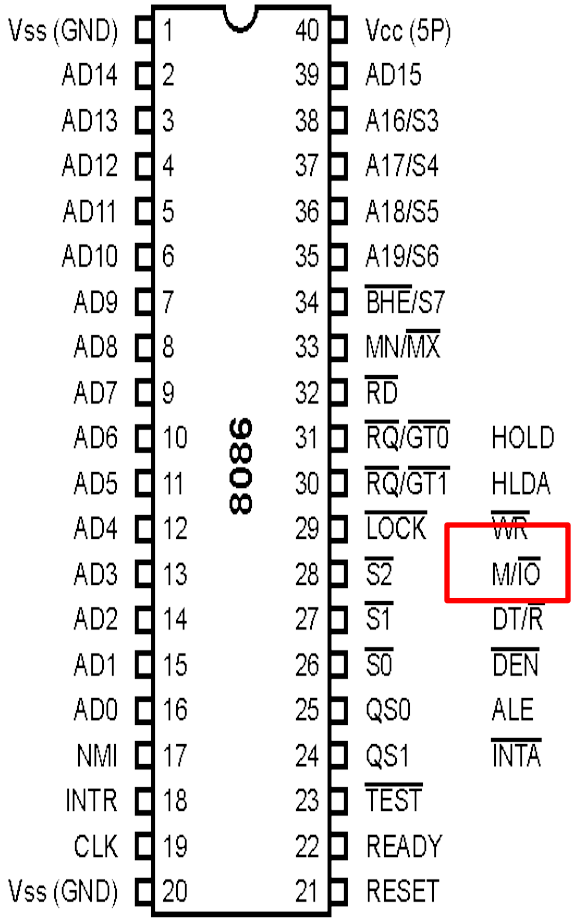
- This is a Data Transmit/Receive signal.
- It decides the direction of data flow through the transceiver.
- When it is high, data is transmitted out.
- When it is low, data is received in.



M / $\overline{IO}$

Pin 28 (Output)

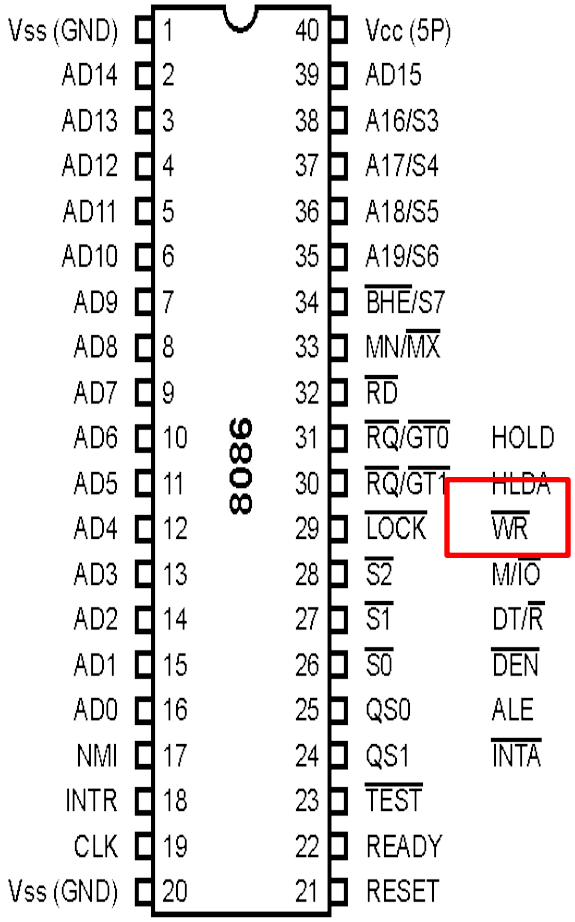
- This signal is issued by the microprocessor to distinguish memory access from I/O access.
- When it is high, memory is accessed.
- When it is low, I/O devices are accessed.



WR

Pin 29 (Output)

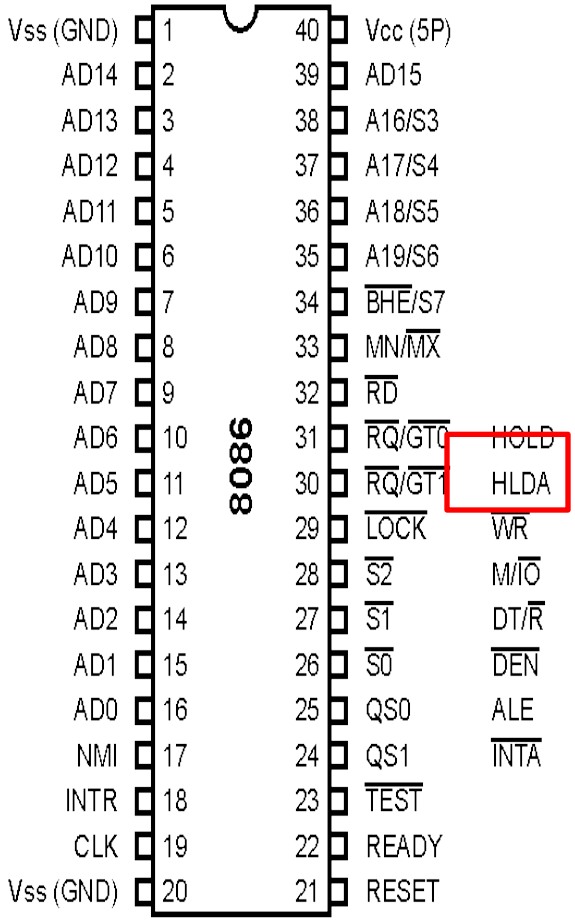
- It is a Write signal.
- It is used to write data in memory or output device depending on the status of M/$\overline{IO}$ signal.
- It is an active low signal.



HLDA

Pin 30 (Output)

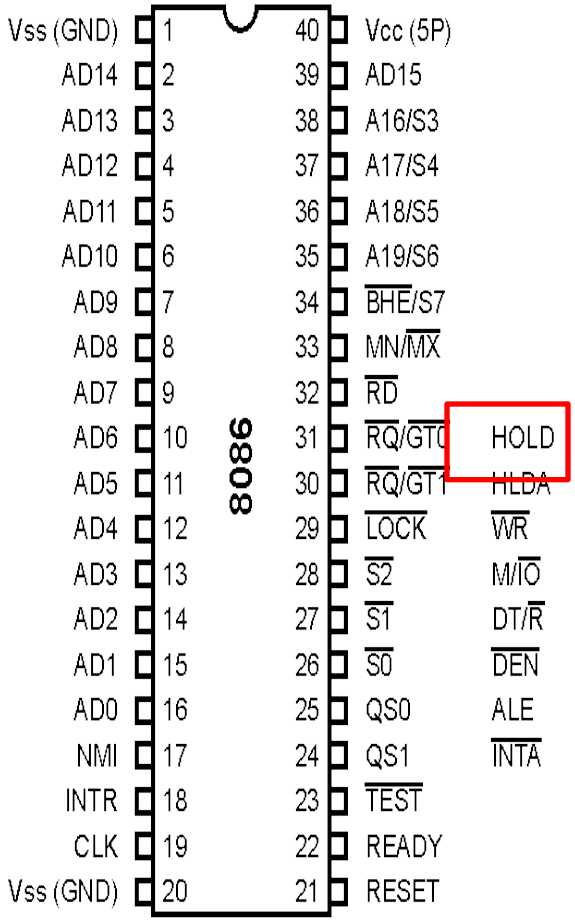
- It is a Hold Acknowledge signal.
- It is issued after receiving the HOLD signal.
- It is an active high signal.



HOLD

Pin 31 (Input)

- When DMA controller needs to use address/data bus, it sends a request to the CPU through this pin.
- It is an active high signal.
- When microprocessor receives HOLD signal, it issues HLDA signal to the DMA controller.



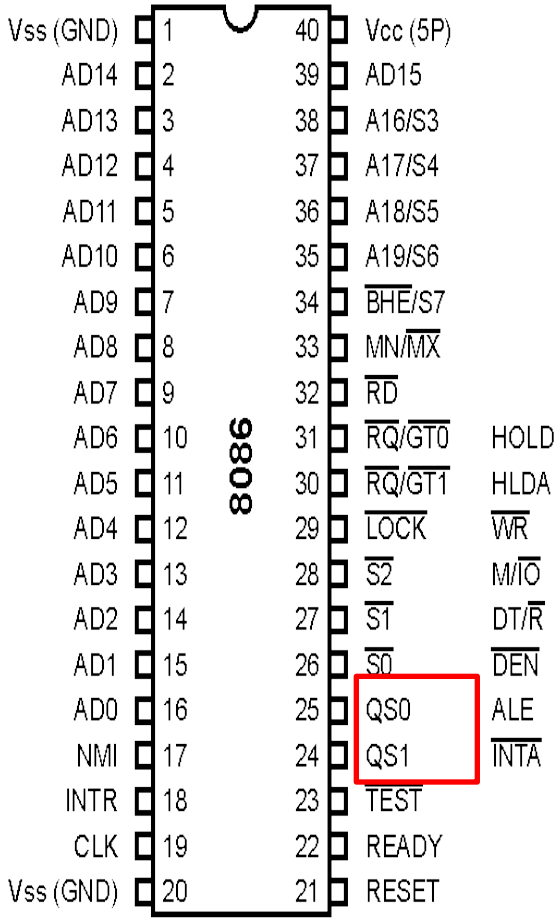
Pin Description for Maximum Mode

QS_1 and QS_0

Pin 24 and 25 (Output)

- These pins provide the status of instruction queue.

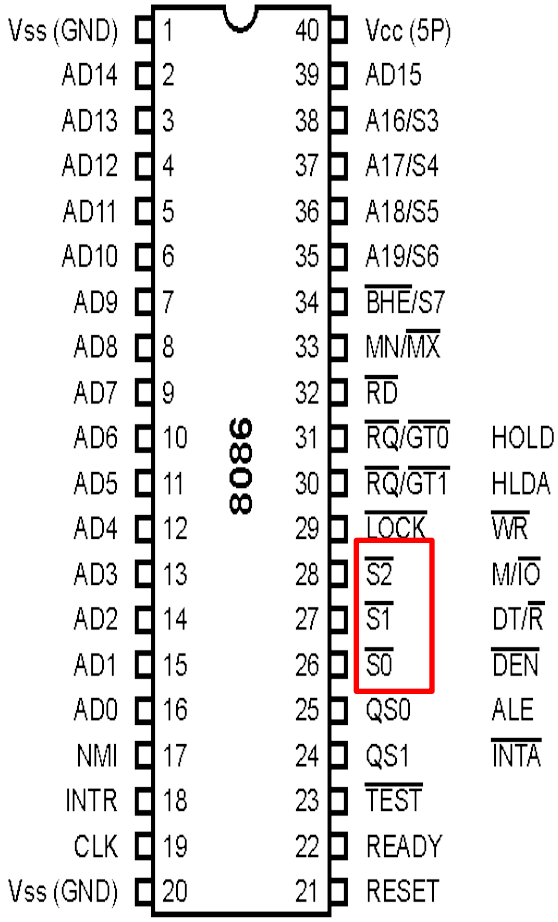
QS_1	QS_0	Status
0	0	No operation
0	1	1st byte of opcode from queue
1	0	Empty queue
1	1	Subsequent byte from queue



$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$

Pin 26, 27, 28 (Output)

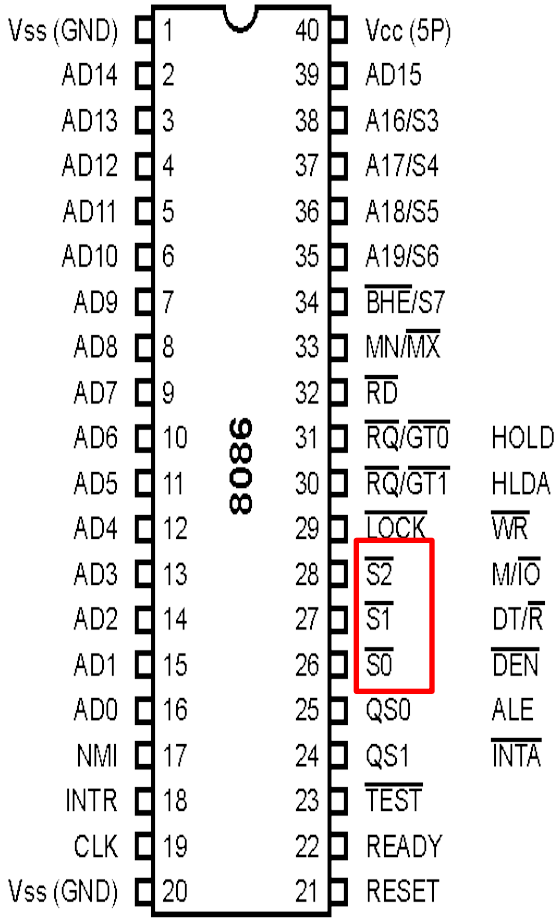
- These status signals indicate the operation being done by the microprocessor.
- This information is required by the Bus Controller 8288.
- Bus controller 8288 generates all memory and I/O control signals.



$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$

Pin 26, 27, 28 (Output)

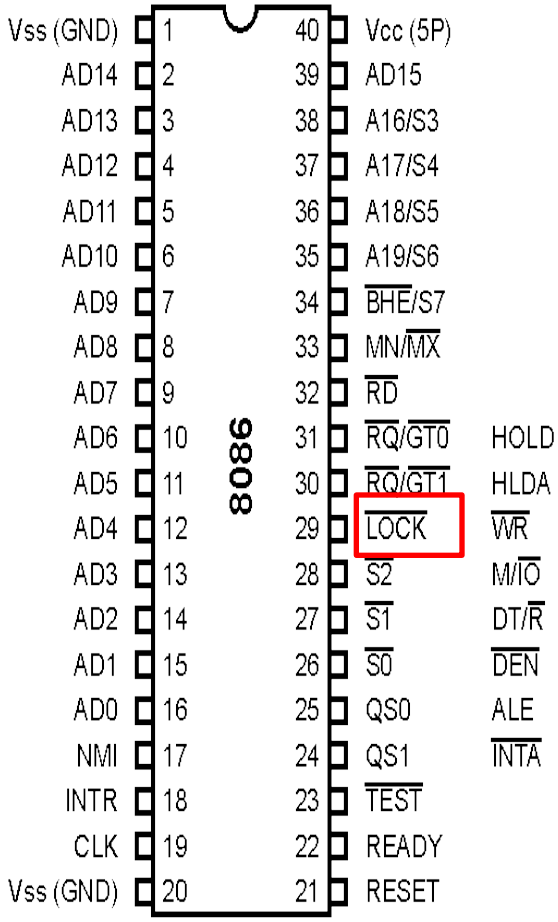
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Status
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Passive



LOCK

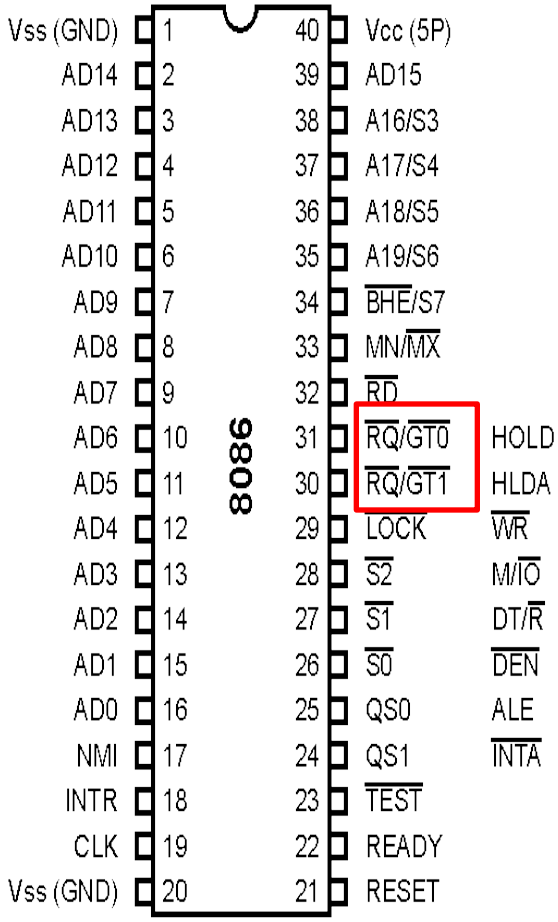
Pin 29 (Output)

- This signal indicates that other processors should not ask CPU to relinquish the system bus.
- When it goes low, all interrupts are masked and HOLD request is not granted.
- This pin is activated by using LOCK prefix on any instruction.



$\overline{RQ}/\overline{GT}_1$ and $\overline{RQ}/\overline{GT}_0$
Pin 30 and 31 (Bi-directional)

- These are Request/Grant pins.
- Other processors request the CPU through these lines to release the system bus.
- After receiving the request, CPU sends acknowledge signal on the same lines.
- $\overline{RQ}/\overline{GT}_0$ has higher priority than $\overline{RQ}/\overline{GT}_1$.



ADDRESSING MODES OF 8086

The addressing modes of 8086 are :-

i. register addressing mode

ii. immediate addressing mode

iii. direct addressing mode

iv. indirect addressing mode

v. base plus index addressing mode

vi. register relative addressing mode

vii. base relative plus index addressing mode

i. Register Addressing Mode :-

Data transfer using registers is called register addressing mode. Here operand value is present in register. For example :

MOV AL,BL; **//**Moves 8 bit contents of BL into AL
MOV DX,CX; //moves 16 bit content of CS into DX.

ii. Immediate Addressing mode :-

When data is stored in code segment instead of data segment immediate addressing mode is used. Here operand value is present in the instruction. For example :

MOV AX, 1234;
MOV CL, 03 H; **//**Moves the 8 bit data 03 H into CL

iii. Direct Addressing mode :-

When direct memory address is supplied as part of the instruction is called direct addressing mode. Operand offset value with respect to data segment is given in instruction. For example :

MOV AX, [1234];
ADD AX, [1234];

iv. Register Indirect Addressing mode :-

Here operand offset is given in a CPU register. Register used are BX, SI(source index), DI(destination index), or BP(base pointer). BP holds offset with respect to Stack segment, but SI,DI and BX refer to data segment. For example :

MOV [BX],AX;
ADD AX, [SI];

v. Indexed Addressing mode :-

Here operand offset is given by a sum of a value held in either SI, or DI register and a constant displacement specified as an operand. For example :
Lets take arrays as an example. This is very efficient way of accessing arrays.
My_array DB '1', '2', '3','4,'5';

MOV SI, 3;
MOV AL, My_array[3];

// So AL holds value 4.

vi. Base Relative Addressing mode :-

Operand offset given by a sum of a value held either in BP, or BX and a constant offset specified as an operand. For example

MOV AX,[BP+1];
JMP [BX+1];

vii. Base Indexed :-

Here operand offset is given by sum of either BX or BP with either SI or DI. For example :

MOV AX, [BX+SI]
JMP [BP+DI]