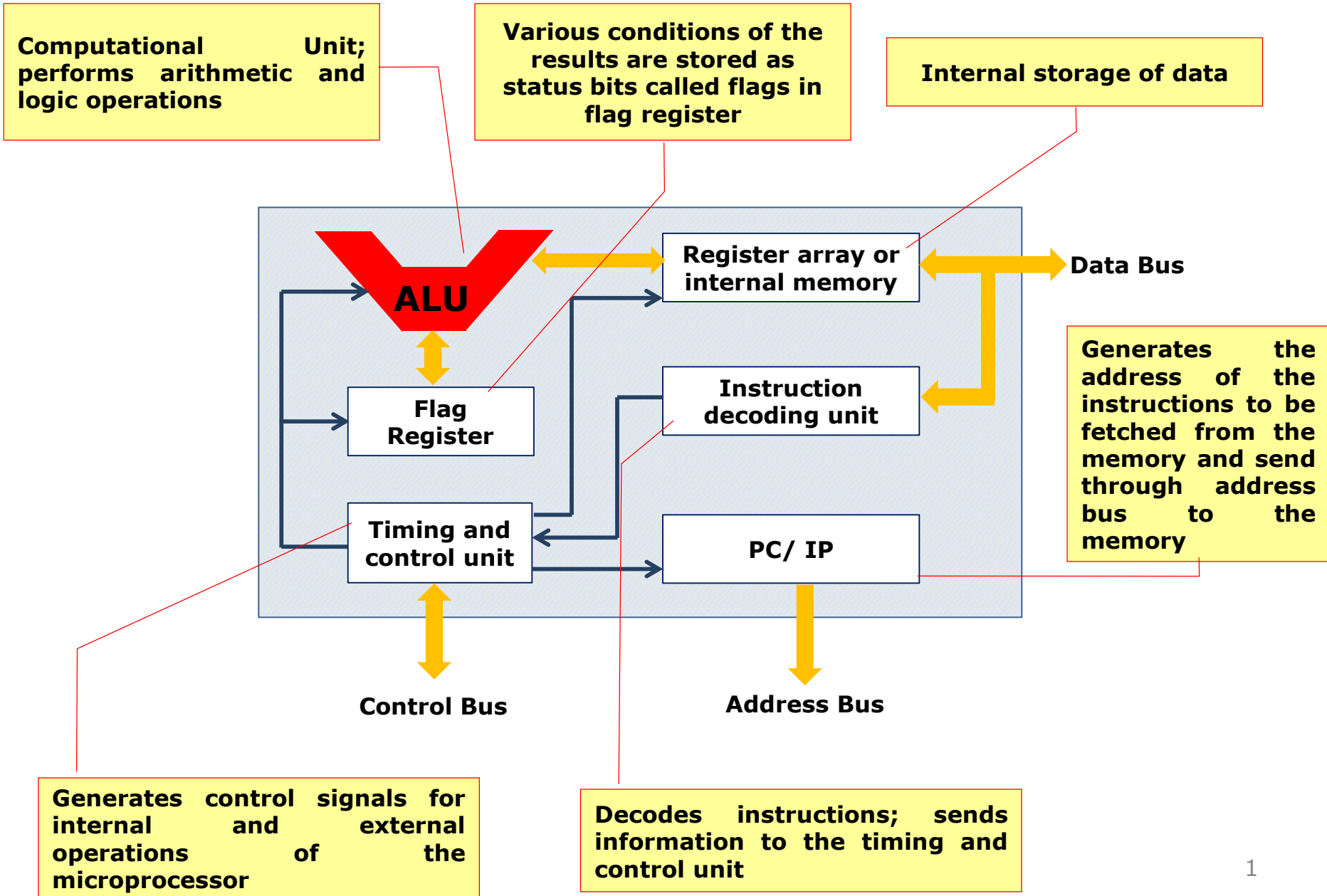


Microprocessor

Functional blocks



First 16-bit processor released by INTEL in the year 1978

Originally HMOS, now manufactured using HMOS III technique

Approximately 29,000 transistors, 40 pin DIP, 5V supply

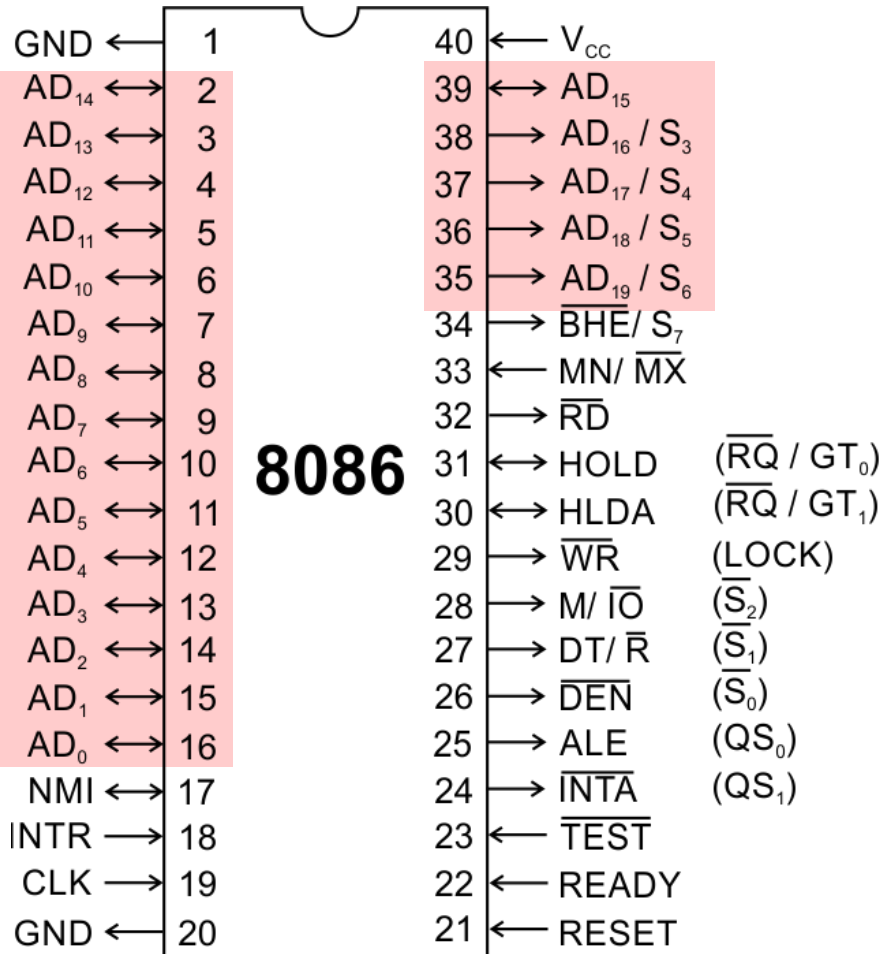
Does not have internal clock; external asymmetric clock source with 33% duty cycle

20-bit address to access memory \Rightarrow can address up to $2^{20} = 1$ megabytes of memory space.

Addressable memory space is organized into two banks of 512 kb each; **Even (or lower) bank and **Odd (or higher) bank**. Address line A_0 is used to select even bank and control signal \overline{BHE} is used to access odd bank**

Uses a separate 16-bit address for I/O mapped devices \Rightarrow can generate $2^{16} = 64$ k addresses.

Operates in two modes: **minimum mode and **maximum mode**, decided by the signal at \overline{MN} and \overline{MX} pins.**



AD₀-AD₁₅ (Bidirectional)

Address/Data bus

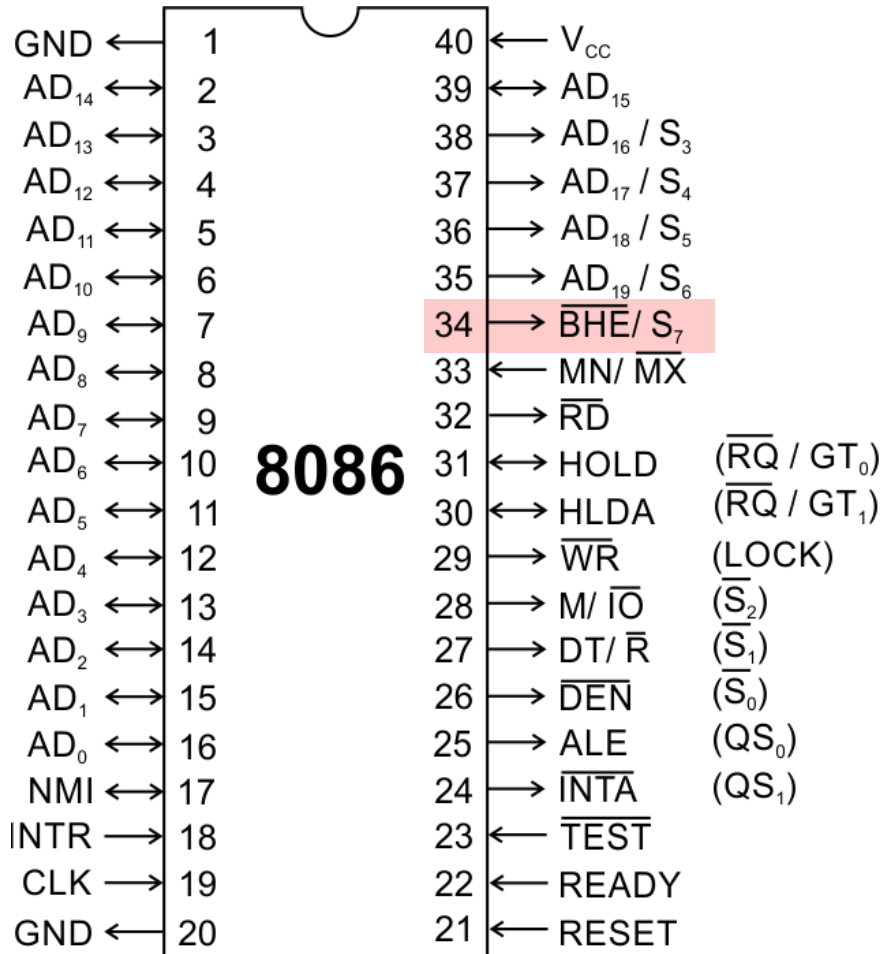
Low order address bus; these are multiplexed with data.

When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A₀-A₁₅.

When data are transmitted over AD lines the symbol D is used in place of AD, for example D₀-D₇, D₈-D₁₅ or D₀-D₁₅.

A₁₆/S₃, A₁₇/S₄, A₁₈/S₅, A₁₉/S₆

High order address bus. These are multiplexed with status signals



BHE (Active Low)/S₇ (Output)

Bus High Enable/Status

It is used to enable data onto the most significant half of data bus, D₈-D₁₅. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S₇.

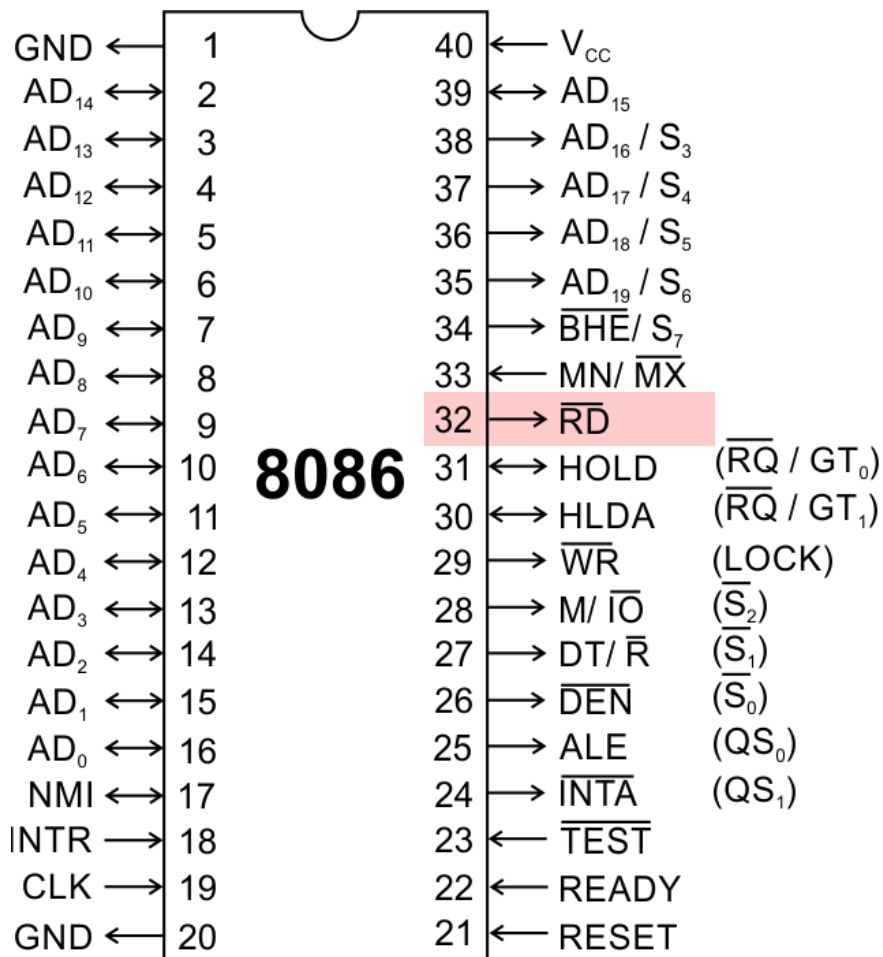
MN / MX

MINIMUM / MAXIMUM

This pin signal indicates what mode the processor is to operate in.

RD (Read) (Active Low)

The signal is used for read operation.
It is an output signal.
It is active when low.



TEST

$\overline{\text{TEST}}$ input is tested by the 'WAIT' instruction.

8086 will enter a wait state after execution of the WAIT instruction and will resume execution only when the $\overline{\text{TEST}}$ is made low by an active hardware.

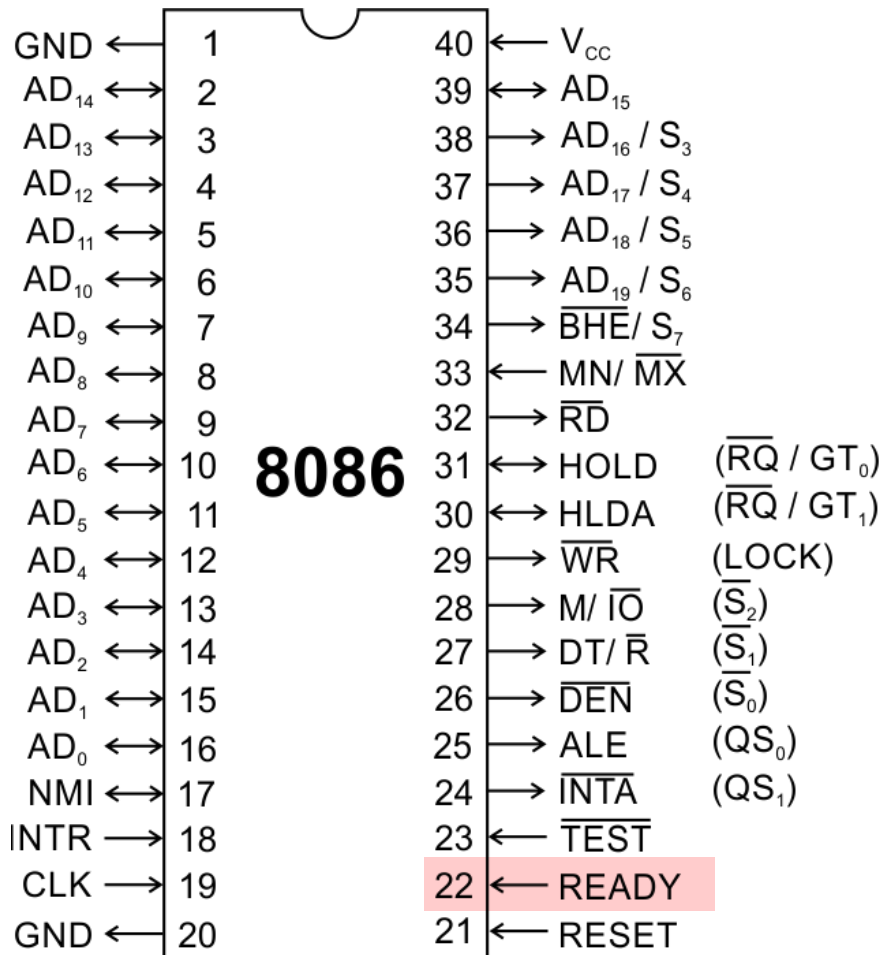
This is used to synchronize an external activity to the processor internal operation.

READY

This is the acknowledgement from the slow device or memory that they have completed the data transfer.

The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the **8086**.

The signal is active high.



RESET (Input)

Causes the processor to immediately terminate its present activity.

The signal must be active HIGH for at least four clock cycles.

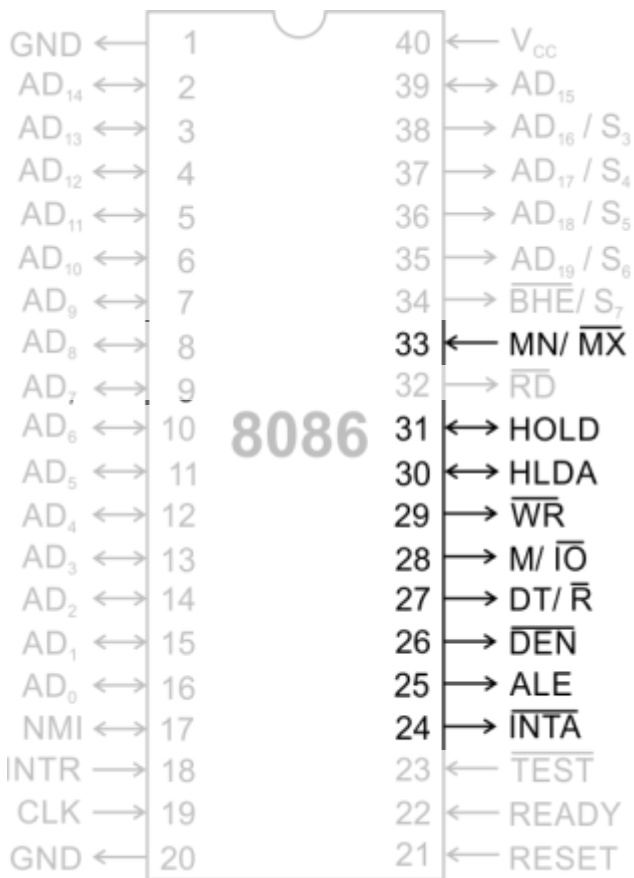
CLK

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

INTR Interrupt Request

This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

This signal is active high and internally synchronized.



The 8086 microprocessor can work in two modes of operations : **Minimum mode** and **Maximum mode**.

In the minimum mode of operation the microprocessor do not associate with any co-processors and can not be used for multiprocessor systems.

In the maximum mode the 8086 can work in multi-processor or co-processor configuration.

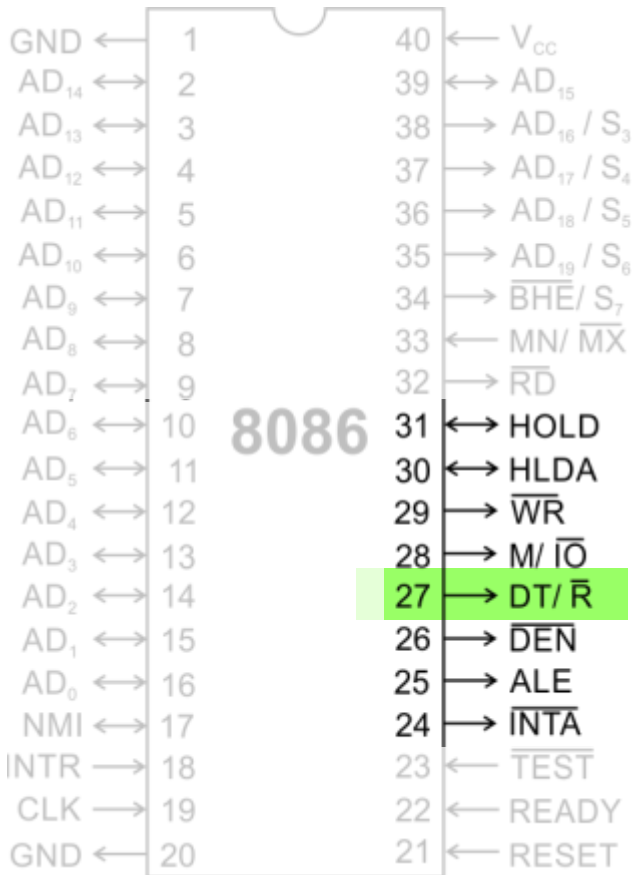
Minimum or maximum mode operations are decided by the pin MN/ MX(Active low).

When this pin is high 8086 operates in minimum mode otherwise it operates in Maximum mode.

Pins 24 -31

For minimum mode operation, the $\overline{MN}/\overline{MX}$ is tied to VCC (logic high)

8086 itself generates all the bus control signals

**DT/ \overline{R}**

(**Data Transmit/ Receive**) Output signal from the processor to control the direction of data flow through the data transceivers

 \overline{DEN}

(**Data Enable**) Output signal from the processor used as out put enable for the transceivers

ALE

(**Address Latch Enable**) Used to demultiplex the address and data lines using external latches

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

Used to differentiate memory access and I/O access. For memory reference instructions, it is **high**. For IN and OUT instructions, it is **low**.

 \overline{WR}

Write control signal; asserted **low** Whenever processor writes data to memory or I/O port

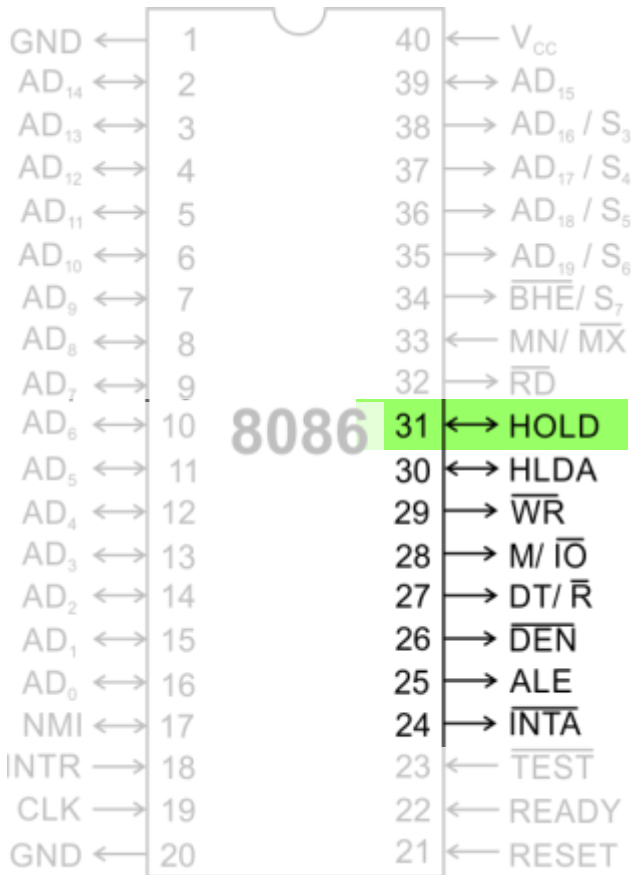
 \overline{INTA}

(**Interrupt Acknowledge**) When the interrupt request is accepted by the processor, the output is **low** on this line.

Pins 24 -31

For minimum mode operation, the MN/\overline{MX} is tied to VCC (logic high)

8086 itself generates all the bus control signals

**HOLD**

Input signal to the processor from the bus masters as a request to grant the control of the bus.

Usually used by the DMA controller to get the control of the bus.

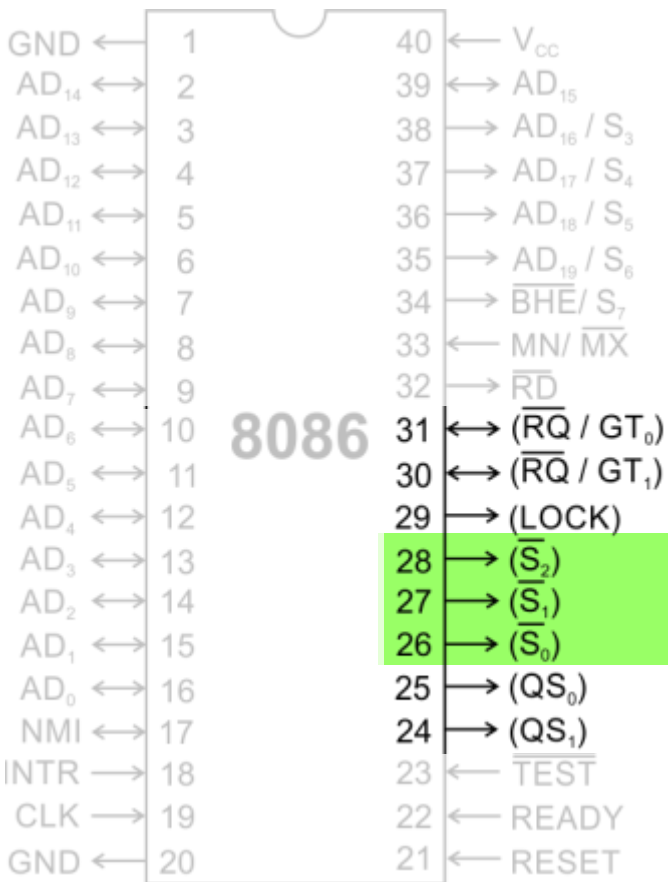
HLDA

(Hold Acknowledge) Acknowledge signal by the processor to the bus master requesting the control of the bus through HOLD.

The acknowledge is asserted high, when the processor accepts HOLD.

During maximum mode operation, the $\overline{MN}/\overline{MX}$ is grounded (logic low)

Pins 24 -31 are reassigned



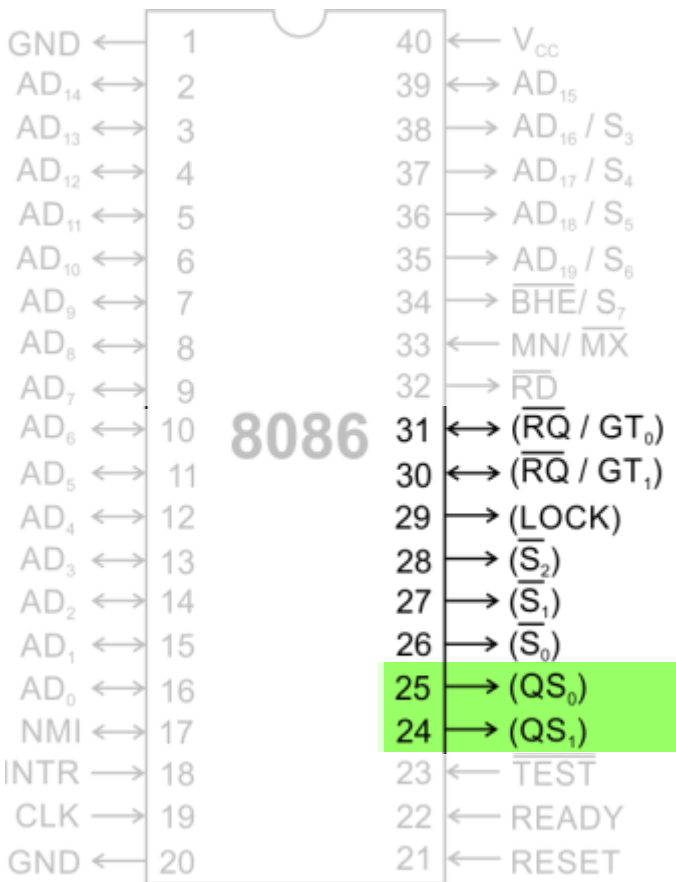
$\overline{S}_0, \overline{S}_1, \overline{S}_2$

Status signals; used by the 8086 bus controller to generate bus timing and control signals. These are decoded as shown.

Status Signal			Machine Cycle
\overline{S}_2	\overline{S}_1	\overline{S}_0	
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

During maximum mode operation, the $\overline{MN}/\overline{MX}$ is grounded (logic low)

Pins 24 -31 are reassigned



$\overline{QS}_0, \overline{QS}_1$

(Queue Status) The processor provides the status of queue in these lines.

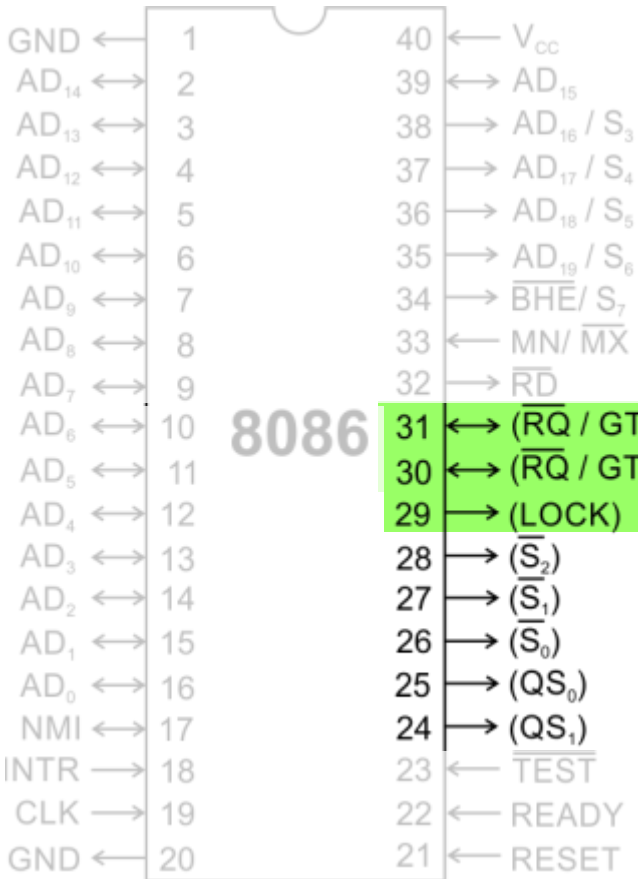
The queue status can be used by external device to track the internal status of the queue in 8086.

The output on QS_0 and QS_1 can be interpreted as shown in the table.

Queue status		Queue operation
QS_1	QS_0	
0	0	No operation
0	1	First byte of an opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

During maximum mode operation, the $\overline{MN}/\overline{MX}$ is grounded (logic low)

Pins 24 -31 are reassigned



$\overline{RQ}/\overline{GT_0}$,
 $\overline{RQ}/\overline{GT_1}$

(Bus Request/ Bus Grant) These requests are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle.

These pins are bidirectional.

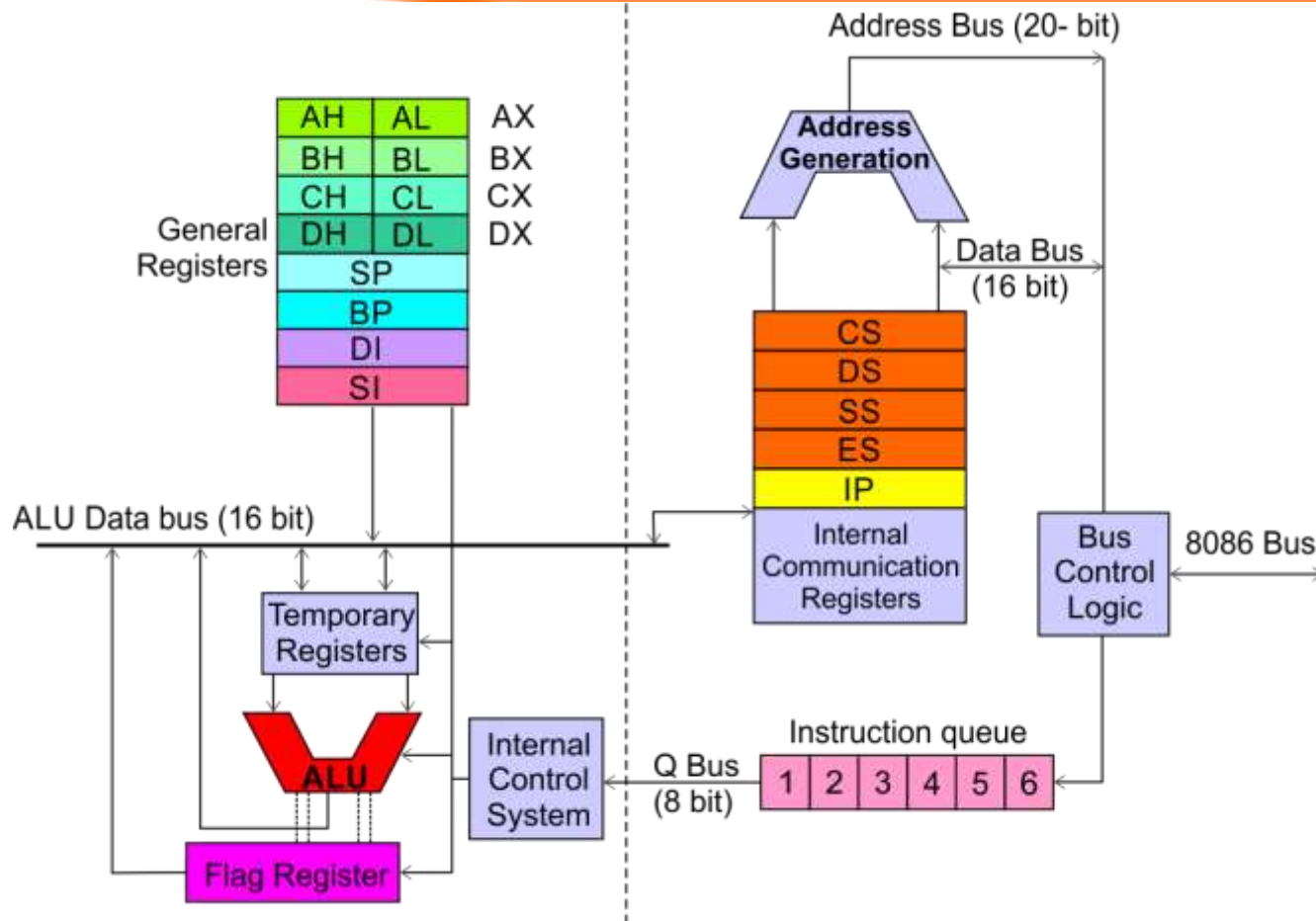
The request on $\overline{GT_0}$ will have higher priority than $\overline{GT_1}$

\overline{LOCK}

An output signal activated by the LOCK prefix instruction.

Remains active until the completion of the instruction prefixed by LOCK.

The 8086 output low on the \overline{LOCK} pin while executing an instruction prefixed by LOCK to prevent other bus masters from gaining control of the system bus.



Execution Unit (EU)

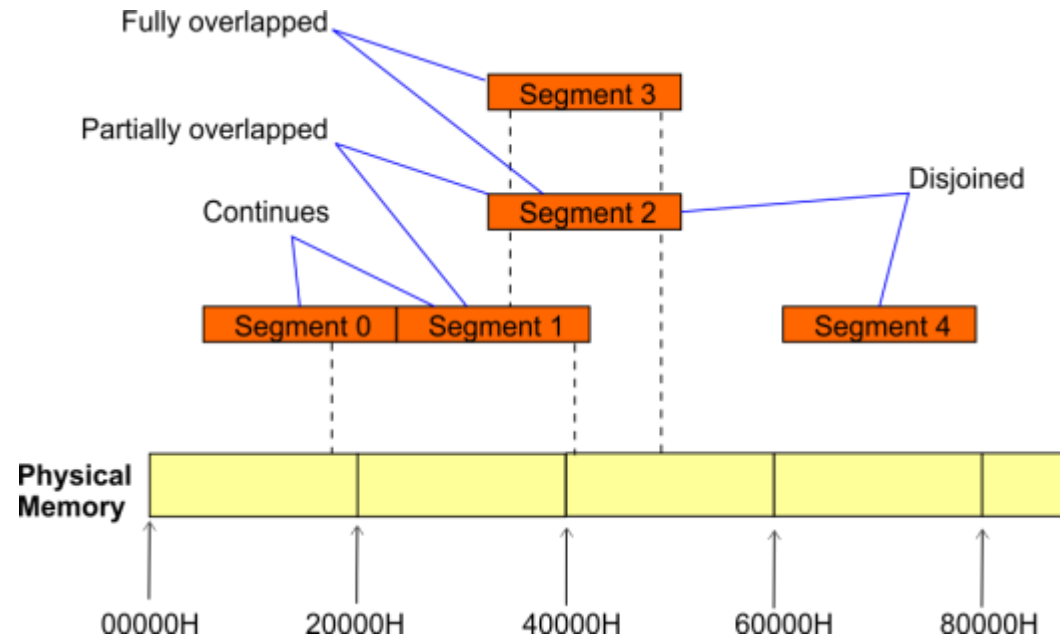
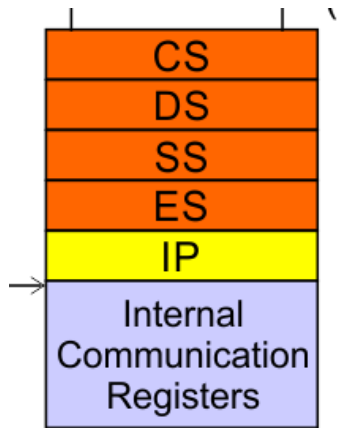
EU executes instructions that have already been fetched by the BIU.

BIU and EU functions separately.

Bus Interface Unit (BIU)

BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/O ports.

Segment Registers



- 8086's 1-megabyte memory is divided into segments of up to 64K bytes each.

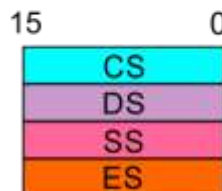
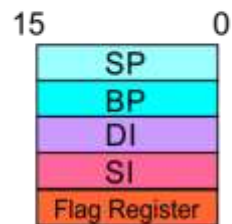
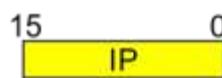
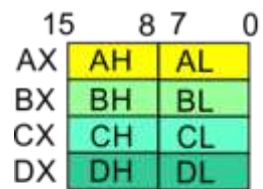
- The 8086 can directly address four segments (256 K bytes within the 1 M byte of memory) at a particular time.

- Programs obtain access to code and data in the segments by changing the segment register content to point to the desired segments.

Segment Registers

Code Segment Register

- 16-bit
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
- That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.



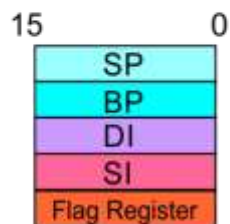
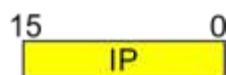
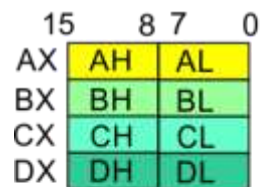
EU

BIU

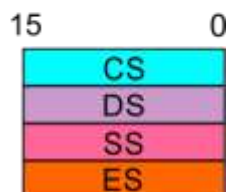
Segment Registers

Data Segment Register

- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.



EU

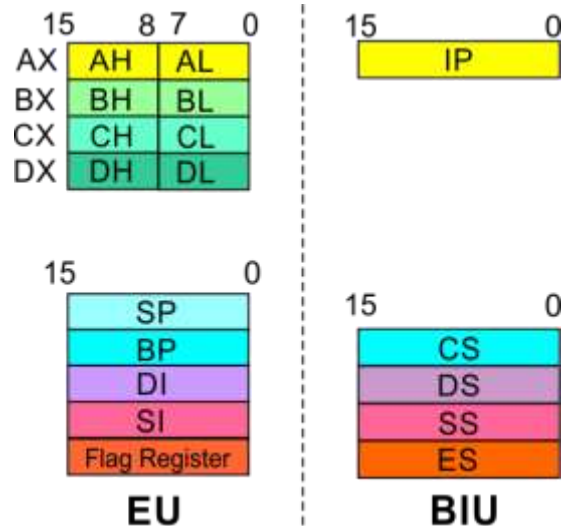


BIU

Segment Registers

Stack Segment Register

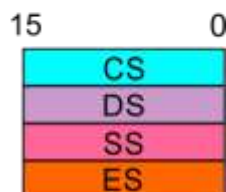
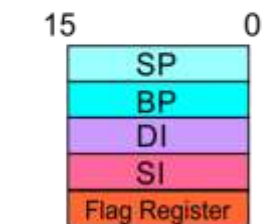
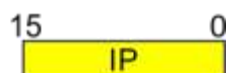
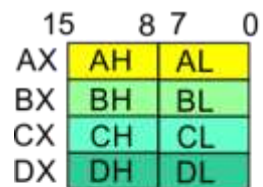
- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as **PUSH** and **POP**.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).



Segment Registers

Extra Segment Register

- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.



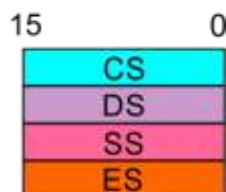
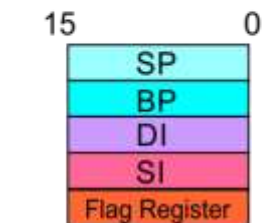
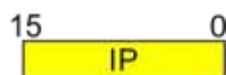
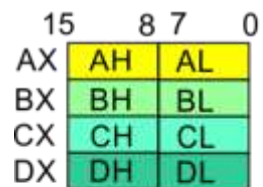
EU

BIU

Segment Registers

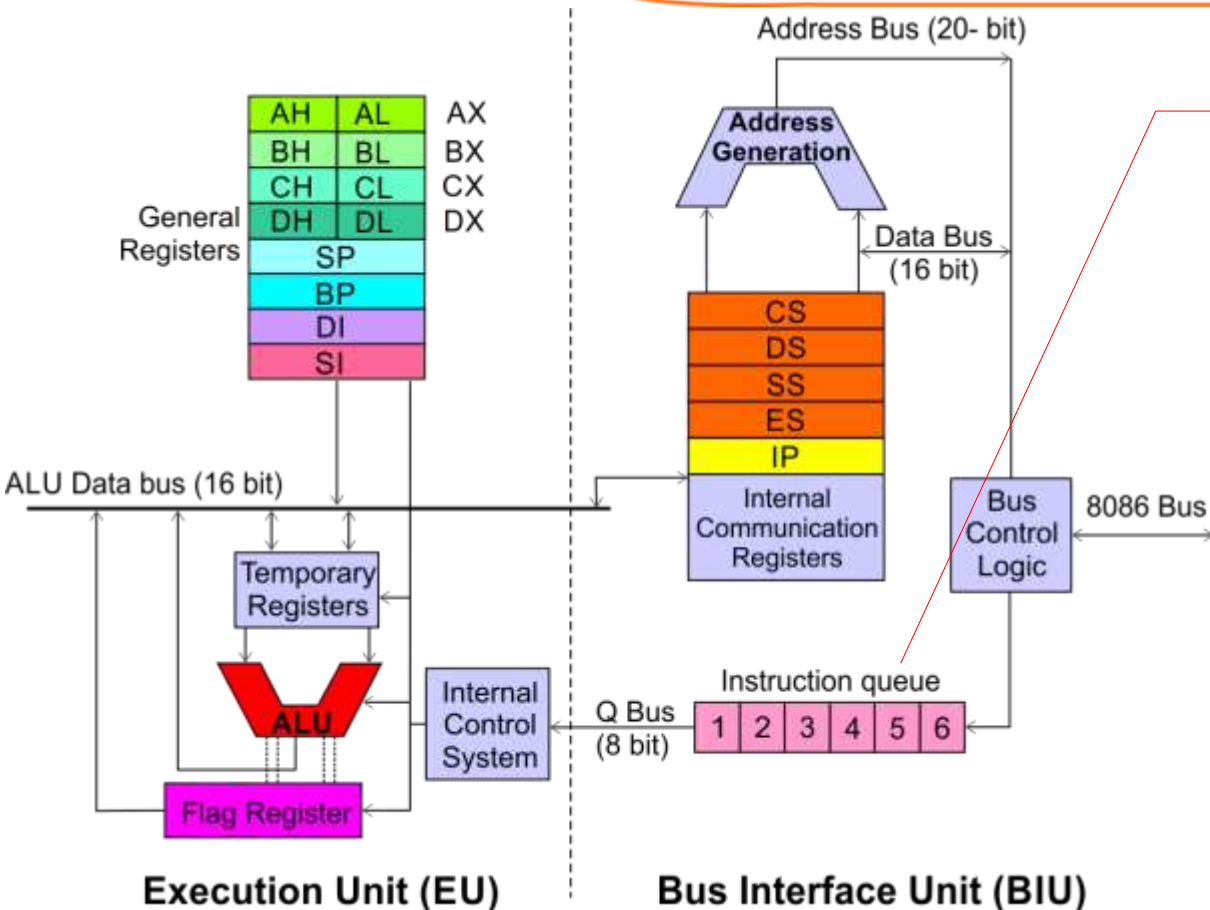
Instruction Pointer

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.



EU

BIU



Instruction queue

- A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.
- This is done in order to speed up the execution by overlapping instruction fetch with execution.
- This mechanism is known as pipelining.

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

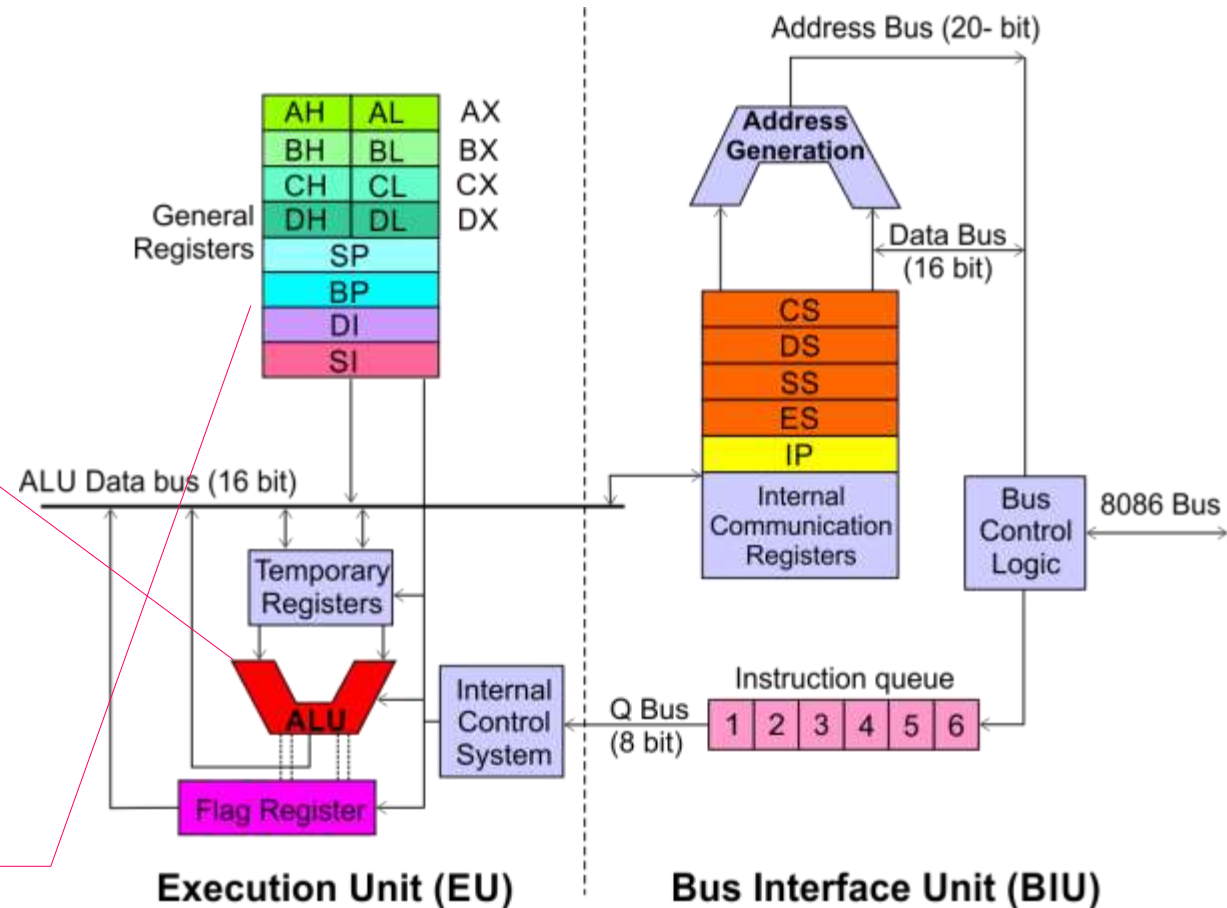
16-bit ALU for performing arithmetic and logic operation

Four general purpose registers (AX, BX, CX, DX);

Pointer registers (Stack Pointer, Base Pointer);

and

Index registers (Source Index, Destination Index) each of 16-bits



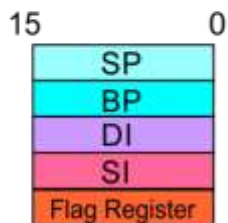
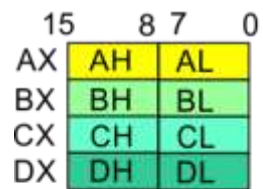
Some of the 16 bit registers can be used as two 8 bit registers as :

**AX can be used as AH and AL
 BX can be used as BH and BL
 CX can be used as CH and CL
 DX can be used as DH and DL**

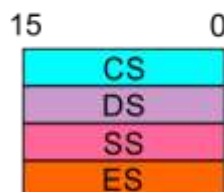
EU Registers

Accumulator Register (AX)

- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.



EU

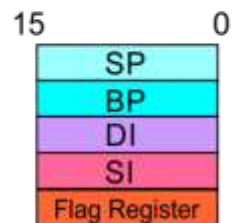
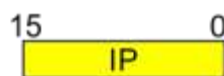
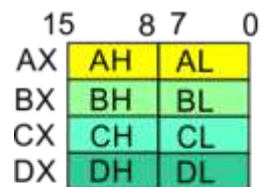


BIU

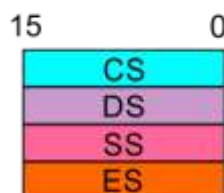
EU Registers

Base Register (BX)

- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.



EU



BIU

EU Registers

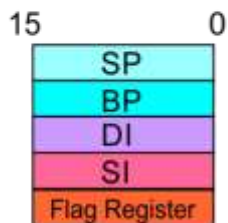
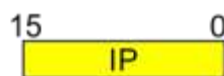
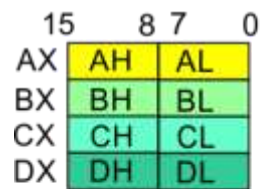
Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as **SHIFT**, **ROTATE** and **LOOP** use the contents of CX as a counter.

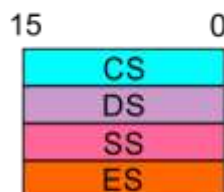
Example:

The instruction **LOOP START** automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

If it is zero, 8086 executes the next instruction; otherwise the 8086 branches to the label **START**.



EU

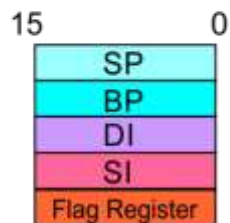
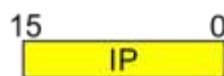
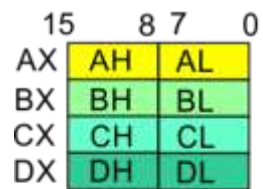


BIU

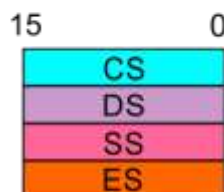
EU Registers

Data Register (DX)

- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit dividend (data) before a $32 \div 16$ division and the 16-bit remainder after division.



EU

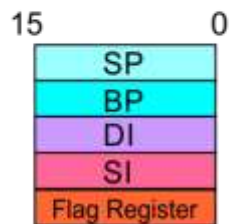
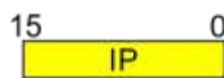
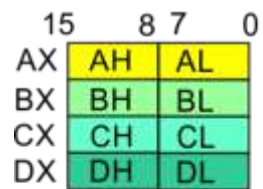


BIU

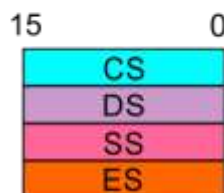
EU Registers

Stack Pointer (SP) and Base Pointer (BP)

- SP and BP are used to access data in the stack segment.
- SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.
- SP contents are automatically updated (incremented/decremented) due to execution of a POP or PUSH instruction.
- BP contains an offset address in the current SS, which is used by instructions utilizing the based addressing mode.



EU

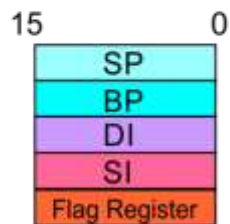
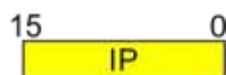
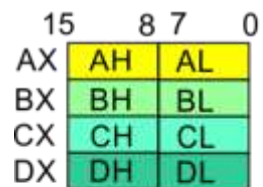


BIU

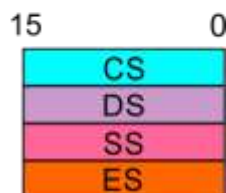
EU Registers

Source Index (SI) and Destination Index (DI)

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.



EU

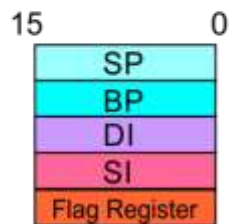
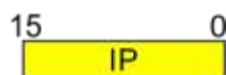
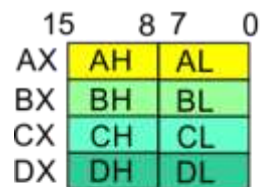


BIU

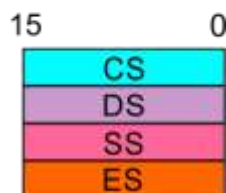
EU Registers

Source Index (SI) and Destination Index (DI)

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.



EU



BIU

Flag Register

Sign Flag

This flag is set, when the result of any computation is negative

Auxiliary Carry Flag

This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.

Carry Flag

This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

Zero Flag

This flag is set, if the result of the computation or comparison performed by an instruction is zero

Parity Flag

This flag is set to 1, if the lower byte of the result contains even number of 1's ; for odd number of 1's set to zero.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

**Over flow Flag**

This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then the overflow will be set.

Tarp Flag

If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction

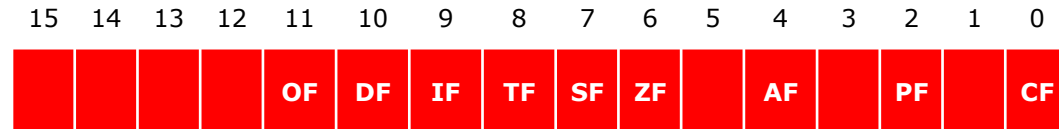
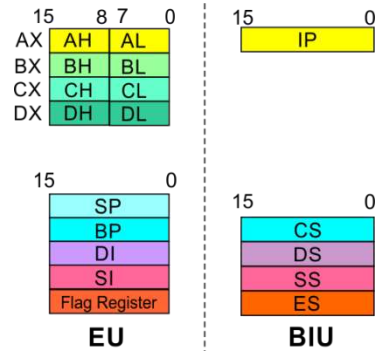
Direction Flag

This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

Interrupt Flag

Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

8086 registers categorized into 4 groups



Sl.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

Register	Name of the Register	Special Function
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
AL	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
BX	Base register	Used to hold 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 stack memory
BP	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
SI	Source Index	Used to hold index value of source operand (data) for string instructions
DI	Data Index	Used to hold the index value of destination operand (data) for string operations

ADDRESSING MODES & Instruction set


```
;PROGRAM TO ADD TWO 16-BIT DATA (METHOD-1)
```

```
DATA SEGMENT ;Assembler directive
    ORG 1104H ;Assembler directive
    SUM DW 0 ;Assembler directive
    CARRY DB 0 ;Assembler directive
```

```
DATA ENDS ;Assembler directive
```

```
CODE SEGMENT ;Assembler directive
    ASSUME CS:CODE ;Assembler directive
    ASSUME DS:DATA ;Assembler directive
    ORG 1000H ;Assembler directive
```

```
MOV AX,205AH ;Load the first data in AX register
MOV BX,40EDH ;Load the second data in BX register
MOV CL,00H ;Clear the CL register for carry
ADD AX,BX ;Add the two data, sum will be in AX
MOV SUM,AX ;Store the sum in memory location (1104H)
JNC AHEAD ;Check the status of carry flag
INC CL ;If carry flag is set,increment CL by one
AHEAD: MOV CARRY,CL ;Store the carry in memory location (1106H)
HLT
```

```
CODE ENDS ;Assembler directive
END ;Assembler directive
```

Program

A set of instructions written to solve a problem.

Instruction

Directions which a microprocessor follows to execute a task or part of a task.

Computer language

High Level

Low Level

Machine Language

Assembly Language

- Binary bits

- English Alphabets
- 'Mnemonics'
- Assembler
Mnemonics → Machine Language

ADDRESSING MODES

Addressing Modes

- Every instruction of a program has to operate on a data.
- The different ways in which a source operand is denoted in an instruction are known as addressing modes.

1. Register Addressing

2. Immediate Addressing

Group I : Addressing modes for register and immediate data

3. Direct Addressing

4. Register Indirect Addressing

5. Based Addressing

6. Indexed Addressing

7. Based Index Addressing

8. String Addressing

Group II : Addressing modes for memory data

9. Direct I/O port Addressing

10. Indirect I/O port Addressing

Group III : Addressing modes for I/O ports

11. Relative Addressing

Group IV : Relative Addressing mode

12. Implied Addressing

Group V : Implied Addressing mode

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

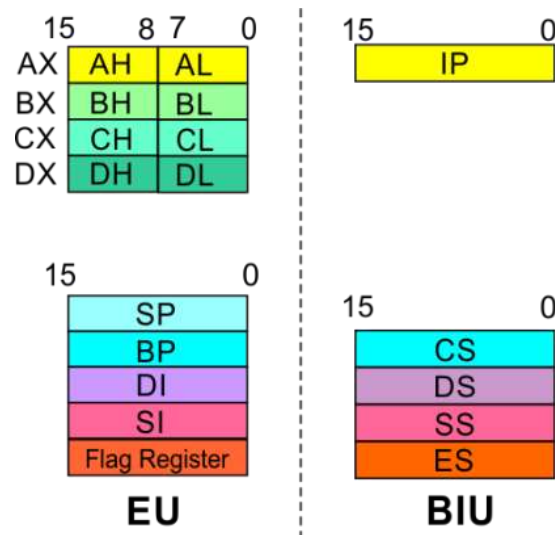
The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

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

$(CL) \leftarrow (DH)$



1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

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

Example:

MOV DL, 08H

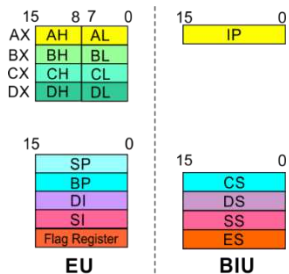
The 8-bit data (08_H) given in the instruction is moved to DL

(DL) ← 08_H

MOV AX, 0A9FH

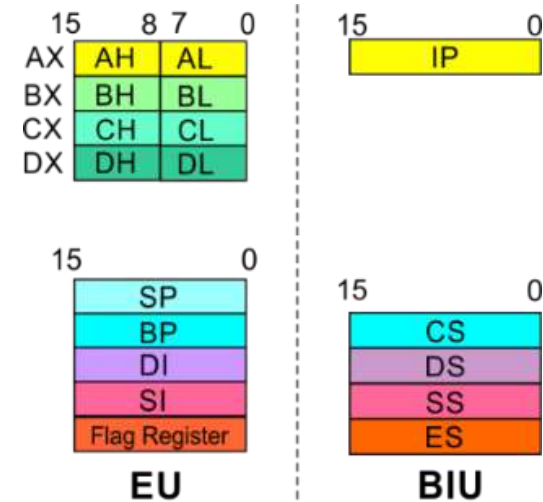
The 16-bit data (0A9F_H) given in the instruction is moved to AX register

(AX) ← 0A9F_H



Addressing Modes : Memory Access

- 20 Address lines \Rightarrow 8086 can address up to $2^{20} = 1\text{M}$ bytes of memory
- However, the largest register is only 16 bits
- Physical Address will have to be calculated
Physical Address : Actual address of a byte in memory. i.e. the value which goes out onto the address bus.
- Memory Address represented in the form –
Seg : Offset (Eg - 89AB:F012)
- Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20-bit address



16 bytes of contiguous memory

89AB : F012 \rightarrow 89AB \rightarrow 89AB0 (Paragraph to byte $\rightarrow 89AB \times 10 = 89AB0$)
 F012 \rightarrow 0F012 (Offset is already in byte unit)
 + -----
 98AC2 (The absolute address)

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

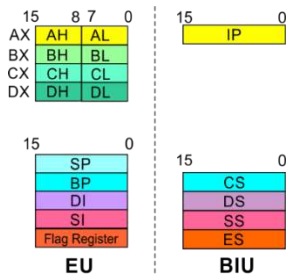
The effective address is just a 16-bit number written directly in the instruction.

Example:

```
MOV BX, [1354H]
MOV BL, [0400H]
```

The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.



1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers:

BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example:

MOV CX, [BX]

Operations:

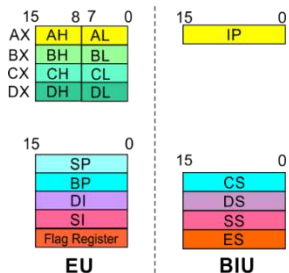
$$\begin{aligned} EA &= (BX) \\ BA &= (DS) \times 16_{10} \\ MA &= BA + EA \end{aligned}$$

$$(CX) \leftarrow (MA) \text{ or,}$$

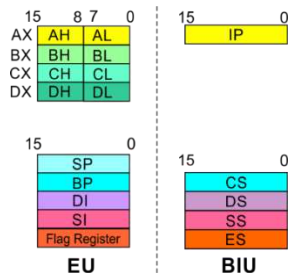
$$(CL) \leftarrow (MA)$$

$$(CH) \leftarrow (MA + 1)$$

Note : Register/ memory enclosed in brackets refer to content of register/ memory



1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In Based Addressing, **BX** or **BP** is used to hold the base value for effective address and a **signed 8-bit** or **unsigned 16-bit** displacement will be specified in the instruction.

In case of 8-bit displacement, it is **sign extended** to 16-bit before adding to the base value.

When **BX** holds the base value of EA, 20-bit physical address is calculated from **BX** and **DS**.

When **BP** holds the base value of EA, **BP** and **SS** is used.

Example:

MOV AX, [BX + 08H]

Operations:

$0008_H \leftarrow 08_H$ (Sign extended)

$EA = (BX) + 0008_H$

$BA = (DS) \times 16_{10}$

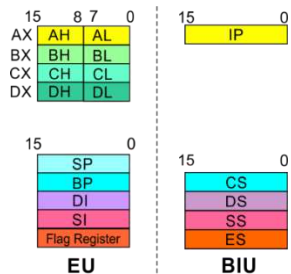
$MA = BA + EA$

$(AX) \leftarrow (MA)$ or,

$(AL) \leftarrow (MA)$

$(AH) \leftarrow (MA + 1)$

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:

MOV CX, [SI + 0A2H]

Operations:

$FFA2_H \leftarrow A2_H$ (Sign extended)

$EA = (SI) + FFA2_H$

$BA = (DS) \times 16_{10}$

$MA = BA + EA$

$(CX) \leftarrow (MA)$ or,

$(CL) \leftarrow (MA)$

$(CH) \leftarrow (MA + 1)$

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example:

MOV DX, [BX + SI + 0AH]

Operations:

$000A_H \leftarrow 0A_H$ (Sign extended)

$EA = (BX) + (SI) + 000A_H$

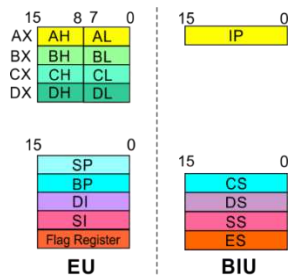
$BA = (DS) \times 16_{10}$

$MA = BA + EA$

$(DX) \leftarrow (MA)$ or,

$(DL) \leftarrow (MA)$

$(DH) \leftarrow (MA + 1)$



1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

Note : Effective address of the Extra segment register

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in **SI register** and the EA of destination is stored in **DI register**.

Segment register for calculating base address of source data is **DS** and that of the destination data is **ES**

Example: MOVSB

Operations:

Calculation of source memory location:

$$EA = (SI) \quad BA = (DS) \times 16_{10} \quad MA = BA + EA$$

Calculation of destination memory location:

$$EA_E = (DI) \quad BA_E = (ES) \times 16_{10} \quad MA_E = BA_E + EA_E$$

$$(MAE) \leftarrow (MA)$$

If $DF = 1$, then $(SI) \leftarrow (SI) - 1$ and $(DI) = (DI) - 1$

If $DF = 0$, then $(SI) \leftarrow (SI) + 1$ and $(DI) = (DI) + 1$

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

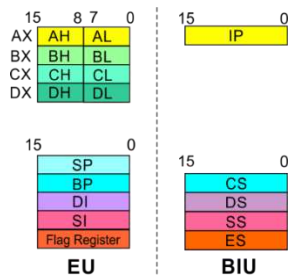
These addressing modes are used to access data from standard I/O mapped devices or ports.

In **direct port addressing mode**, an 8-bit port address is directly specified in the instruction.

Example: `IN AL, [09H]`

Operations: $PORT_{addr} = 09_H$
 $(AL) \leftarrow (PORT)$

Content of port with address 09_H is moved to AL register



1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

Operations:

$000A_H \leftarrow 0A_H$ (sign extend)

If $ZF = 1$, then

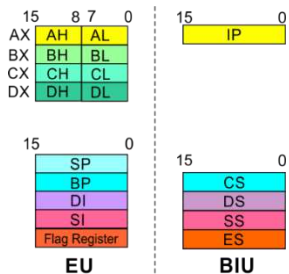
$EA = (IP) + 000A_H$

$BA = (CS) \times 16_{10}$

$MA = BA + EA$

If $ZF = 1$, then the program control jumps to new address calculated above.

If $ZF = 0$, then next instruction of the program is executed.

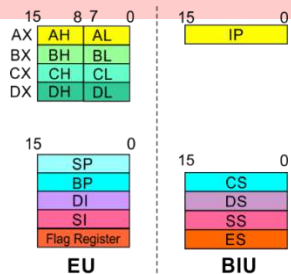


1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.

Example: CLC

This clears the carry flag to zero.



Interfacing memory and i/o ports

Multiprocessor system

- **A microprocessor system comprising of two or more processors**
- **Distributed processing: Entire task is divided in to subtasks**

Advantages

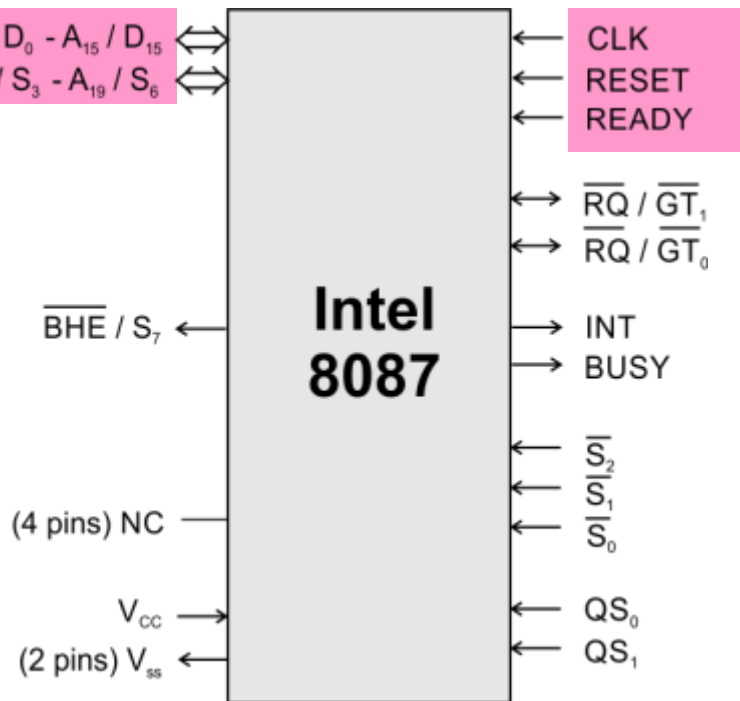
- **Better system throughput by having more than one processor**
- **Each processor have a local bus to access local memory or I/O devices so that a greater degree of parallel processing can be achieved**
- **System structure is more flexible.**
One can easily add or remove modules to change the system configuration without affecting the other modules in the system

8087 coprocessor

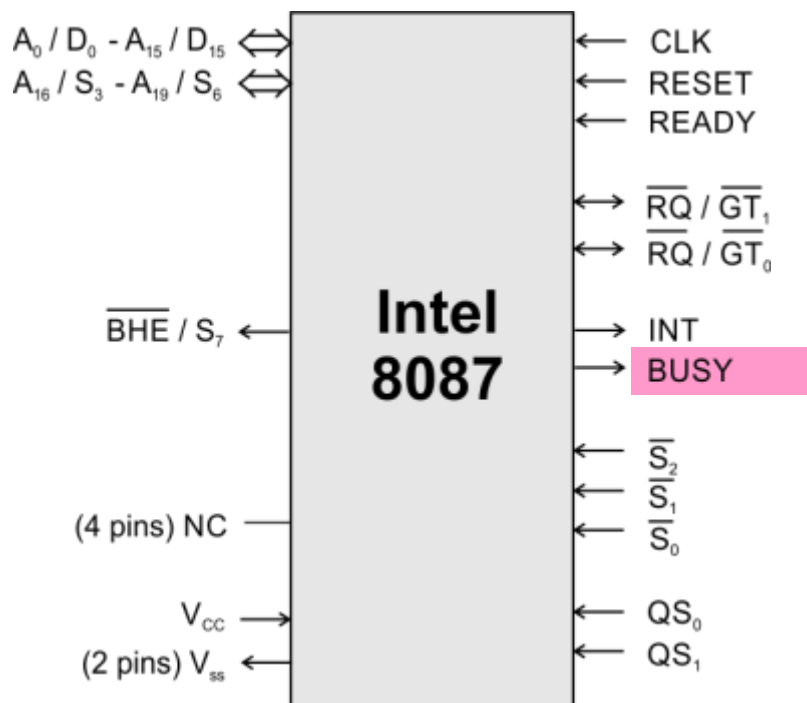
- Specially designed to take care of mathematical calculations involving integer and floating point data
- “Math coprocessor” or “Numeric Data Processor (NDP)”
- Works in parallel with a 8086 in the maximum mode

Features

- 1) Can operate on data of the integer, decimal and real types with lengths ranging from 2 to 10 bytes
- 2) Instruction set involves square root, exponential, tangent etc. in addition to addition, subtraction, multiplication and division.
- 3) High performance numeric data processor \Rightarrow it can multiply two 64-bit real numbers in about $27\mu\text{s}$ and calculate square root in about $36\mu\text{s}$
- 4) Follows IEEE floating point standard
- 5) It is multi bus compatible

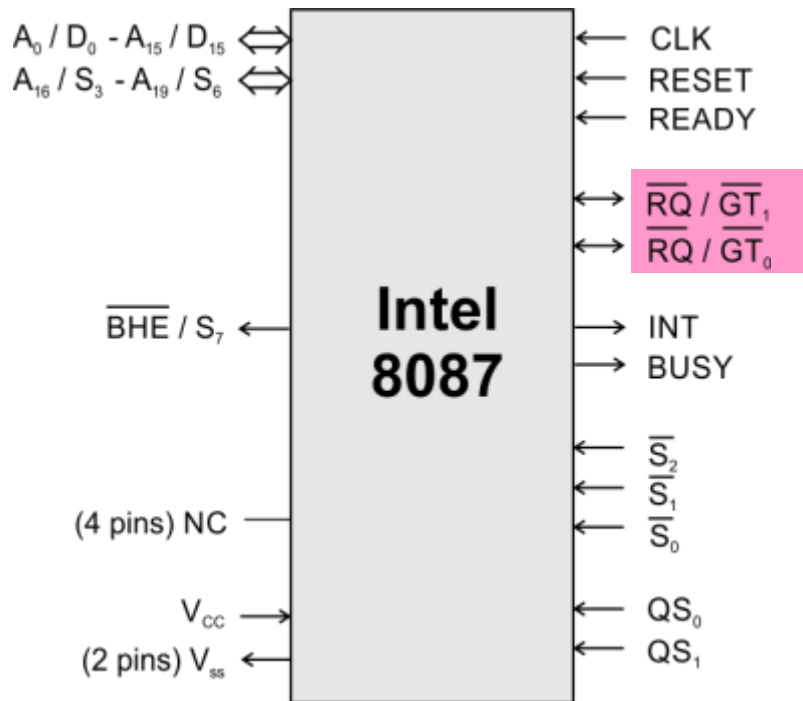


- **16 multiplexed address / data pins and 4 multiplexed address / status pins**
- **Hence it can have 16-bit external data bus and 20-bit external address bus like 8086**
- **Processor clock, ready and reset signals are applied as clock, ready and reset signals for coprocessor**



BUSY

- **BUSY signal from 8087 is connected to the \overline{TEST} input of 8086**
- **If the 8086 needs the result of some computation that the 8087 is doing before it can execute the next instruction in the program, a user can tell 8086 with a WAIT instruction to keep looking at its \overline{TEST} pin until it finds the pin low**
- **A low on the BUSY output indicates that the 8087 has completed the computation**

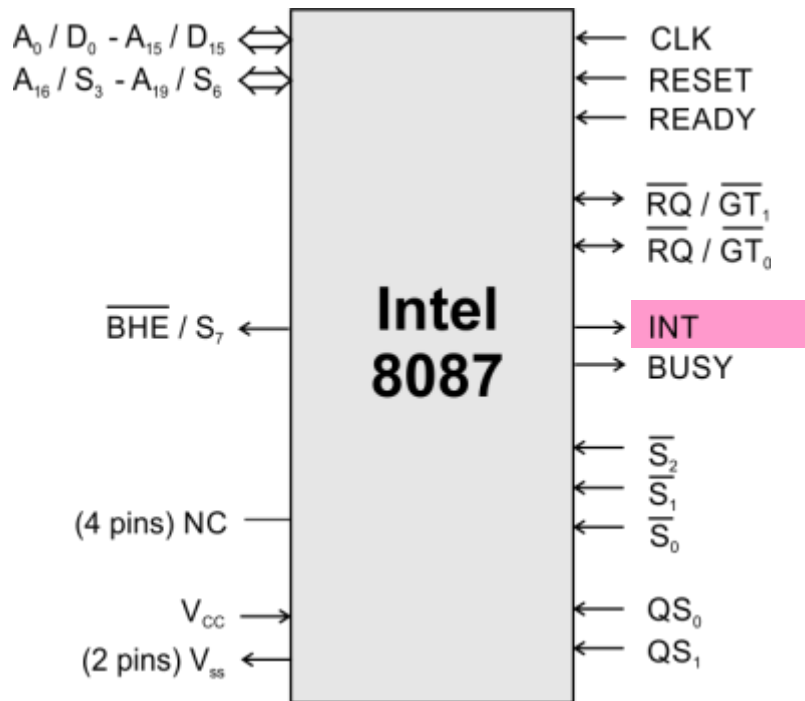


$$\overline{RQ} / \overline{GT}_0$$

- The request / grant signal from the 8087 is usually connected to the request / grant ($\overline{RQ} / \overline{GT}_0$ or $\overline{RQ} / \overline{GT}_1$) pin of the 8086

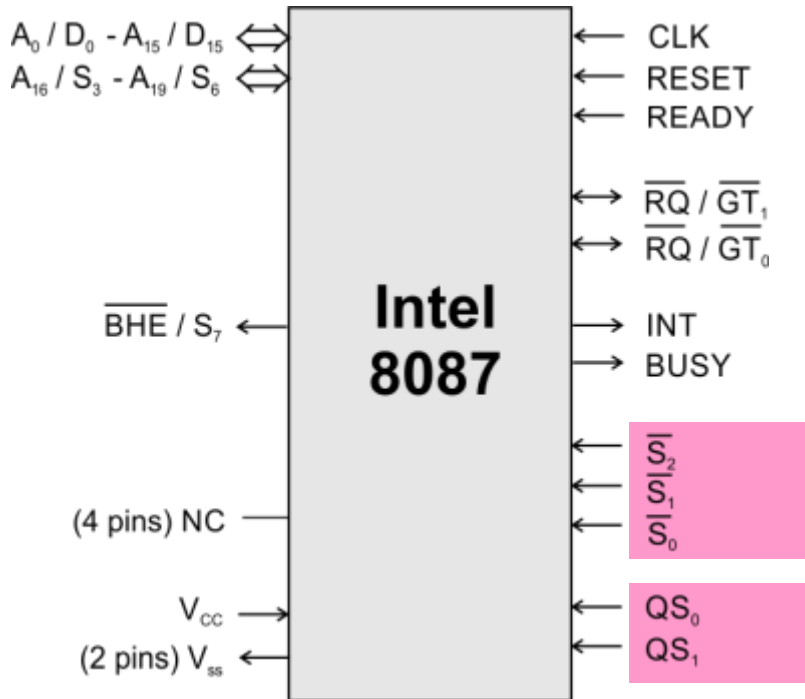
$$\overline{RQ} / \overline{GT}_1$$

- The request / grant signal from the 8087 is usually connected to the request / grant pin of the independent processor such as 8089



INT

- The interrupt pin is connected to the interrupt management logic.
- The 8087 can interrupt the 8086 through this interrupt management logic at the time error condition exists

 $\bar{S}_0 - \bar{S}_2$

\bar{S}_2	\bar{S}_1	\bar{S}_0	Status
1	0	0	Unused
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

 $QS_0 - QS_1$

QS_0	QS_1	Status
0	0	No operation
0	1	First byte of opcode from queue
1	0	Queue empty
1	1	Subsequent byte of opcode from queue

- 8087 instructions are inserted in the 8086 program



- 8086 and 8087 reads instruction bytes and puts them in the respective queues
- NOP
- 8087 instructions have 11011 as the MSB of their first code byte



- 8087 keeps track for ESC instruction by monitoring $\overline{S_2} - \overline{S_0}$ and $AD_0 - AD_{15}$ of 8086.
- Also keeps track of $QS_0 - QS_1$.
- Q status 00; does nothing
- Q status 01; 8087 compares the five MSB bits with 11011
- If there is a match, then the ESC instruction is fetched and executed by 8087
- If there is error during decoding an ESC instruction, 8087 sends an interrupt request



- Memory read/ write
- Additional words : $\overline{RQ} - \overline{GT}_0$
- 8087 BUSY pin high
- \overline{TEST}
- WAIT

