

**UNIT**

**1**

# THE 8086 MICROPROCESSOR

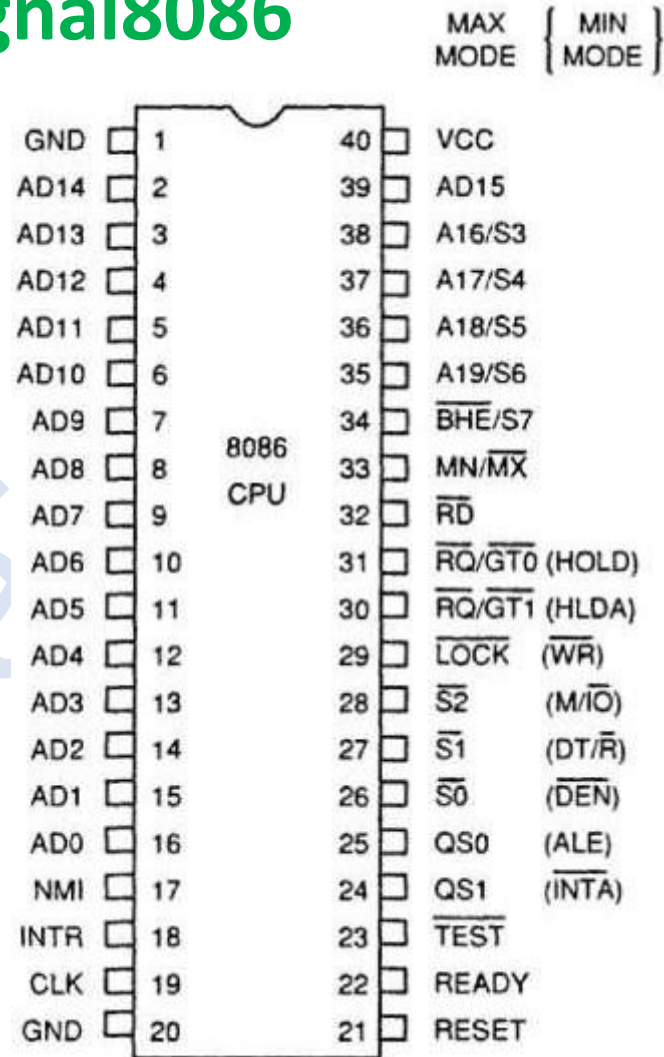
# 8086 Microprocessor-introduction

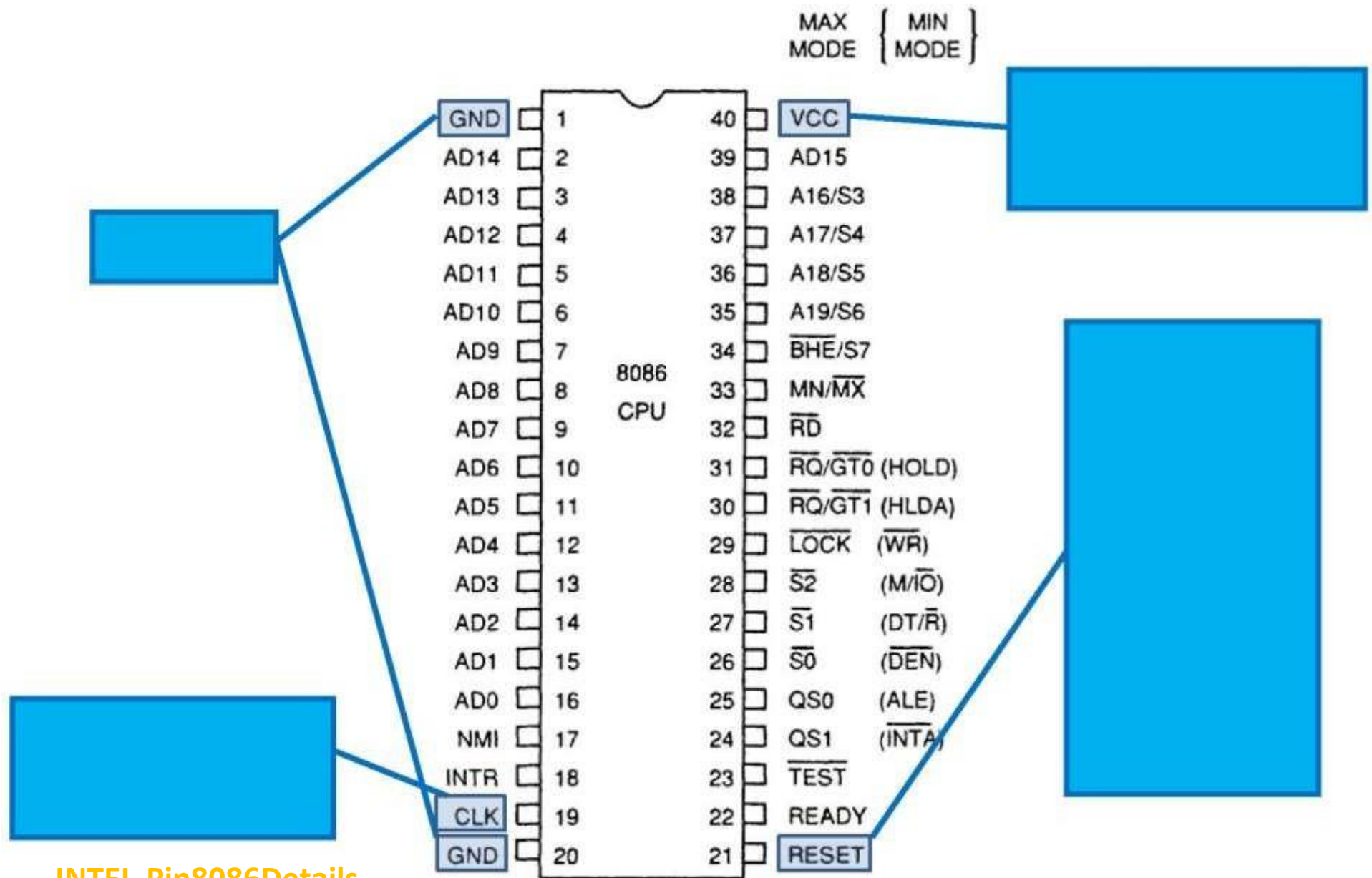
- INTEL launched 8086 in 1978
- 8086 is a 16-bit microprocessor with
  - 16-bit Data Bus **{D<sub>0</sub>-D<sub>15</sub>}**
  - 20-bit Address Bus **{A<sub>0</sub>-A<sub>19</sub>}** [can access upto  $2^{20} = \mathbf{1\ MB}$  memory locations] .
- It has multiplexed address and data bus **AD<sub>0</sub>-AD<sub>15</sub>** and **A<sub>16</sub>-A<sub>19</sub>**.
- It can support upto **64K** I/O ports

# 8086 Microprocessor

- It provides 14, 16-bit registers.
- 8086 requires one phase clock with a 33% duty cycle to provide optimized internal timing.
  - Range of clock:
    - 5 MHz for 8086
    - 8Mhz for 8086-2
    - 10Mhz for 8086-1

## INTEL-PinDiagram/Signal8086





INTEL-Pin8086Details

## Power S

5V $\pm$ 10%

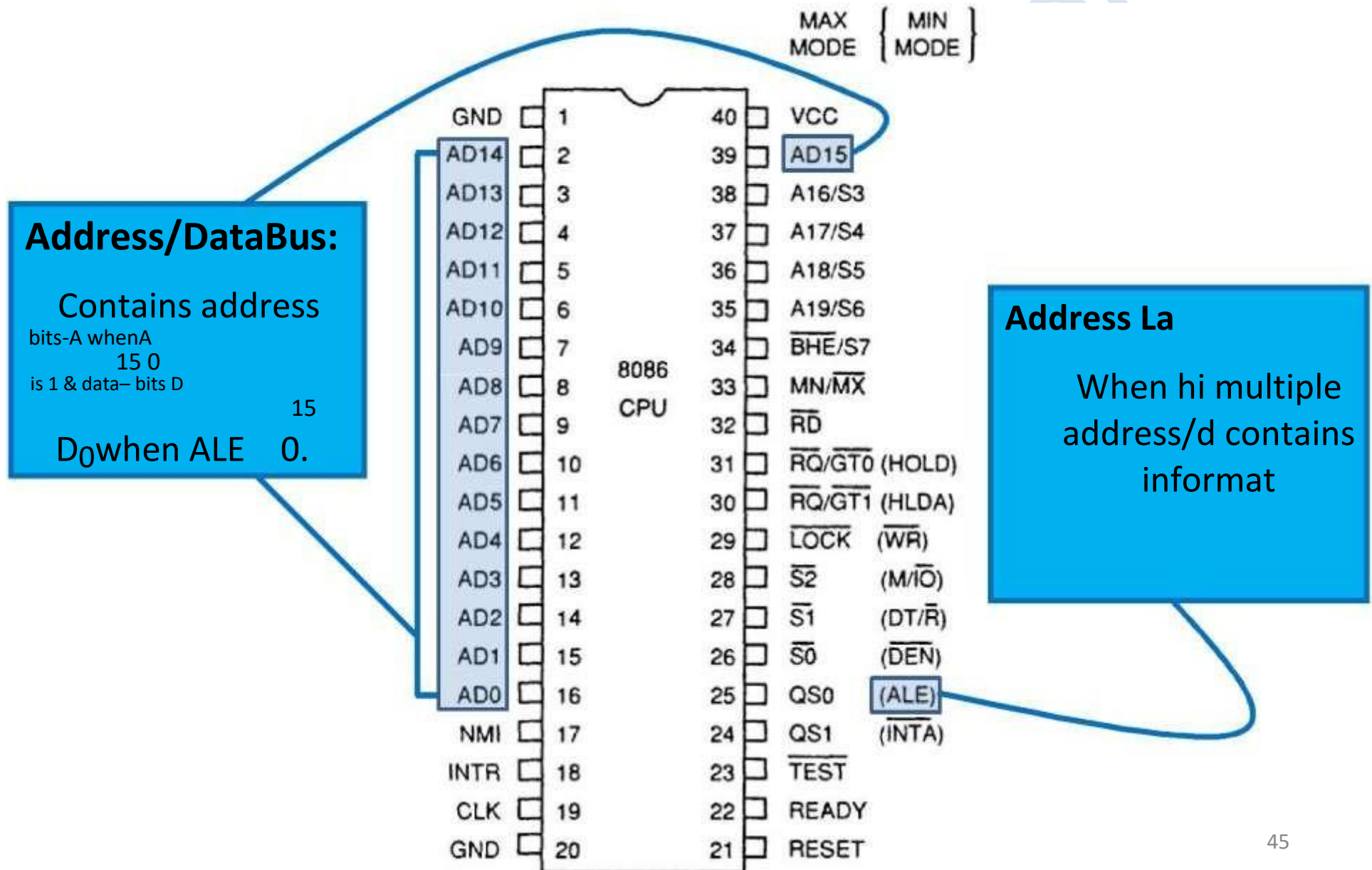
## Reset

Register regs, f

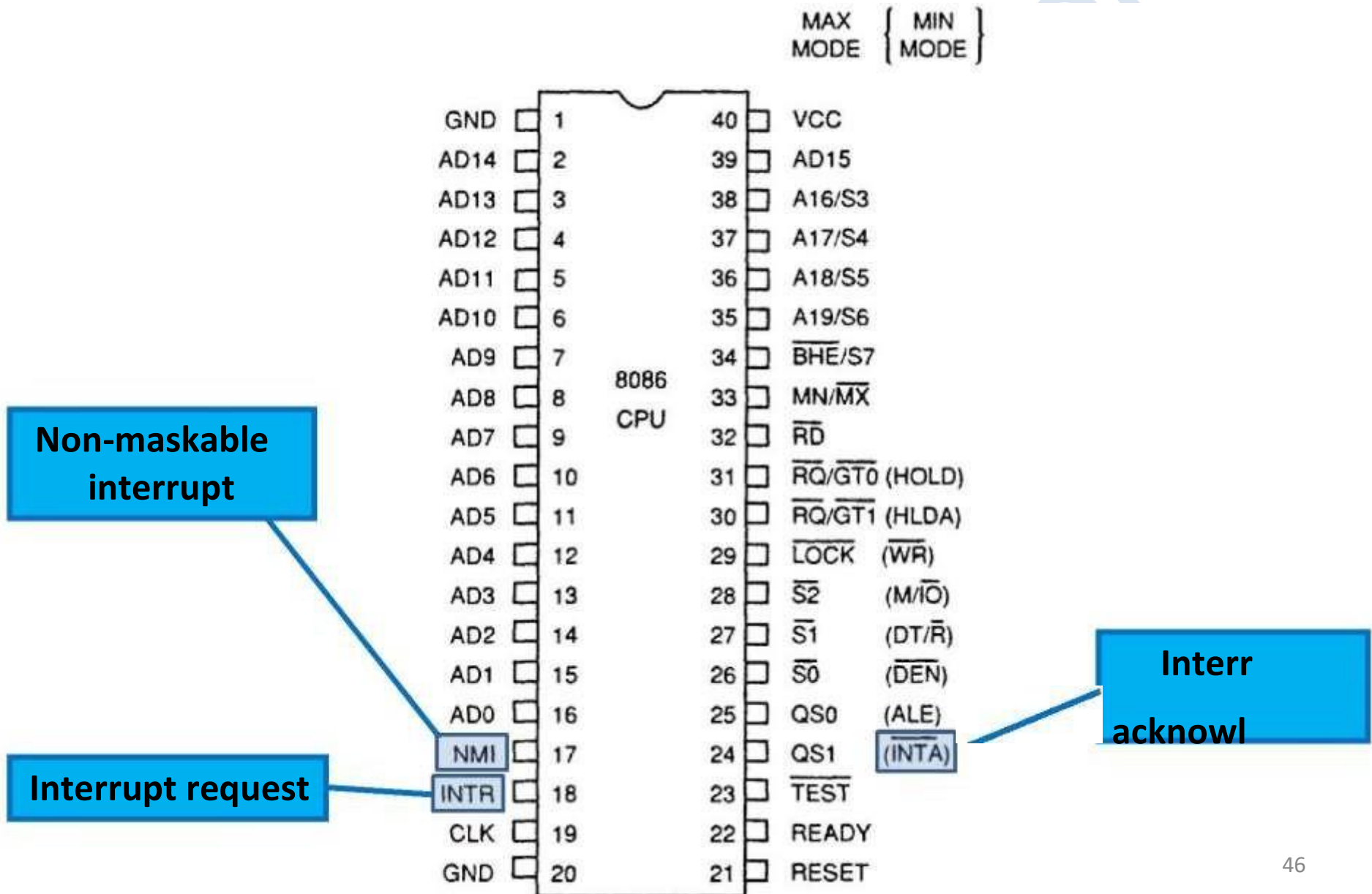
CS: FFFF 0000H

If high minimum clks

## INTEL-Pin8086Details

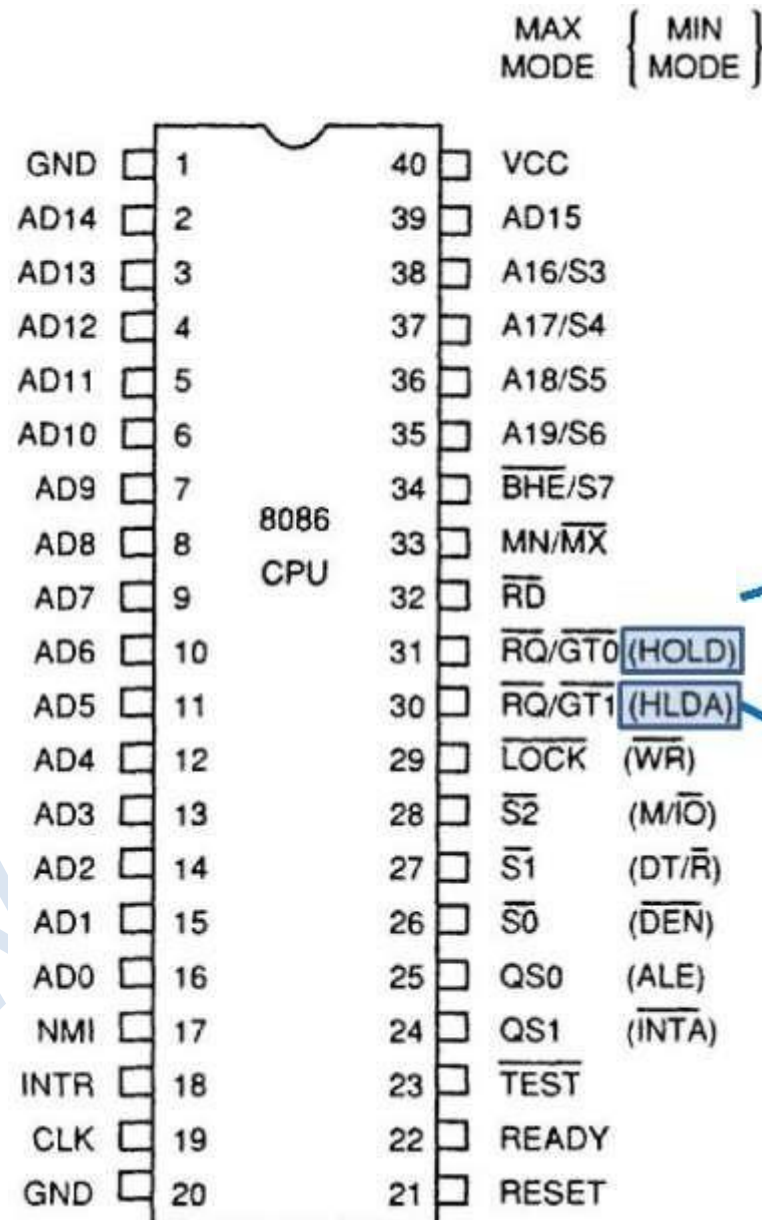


## INTEL-Pin8086Details





## INTEL-Pin8086Details



Hold

Hold  
acknowl

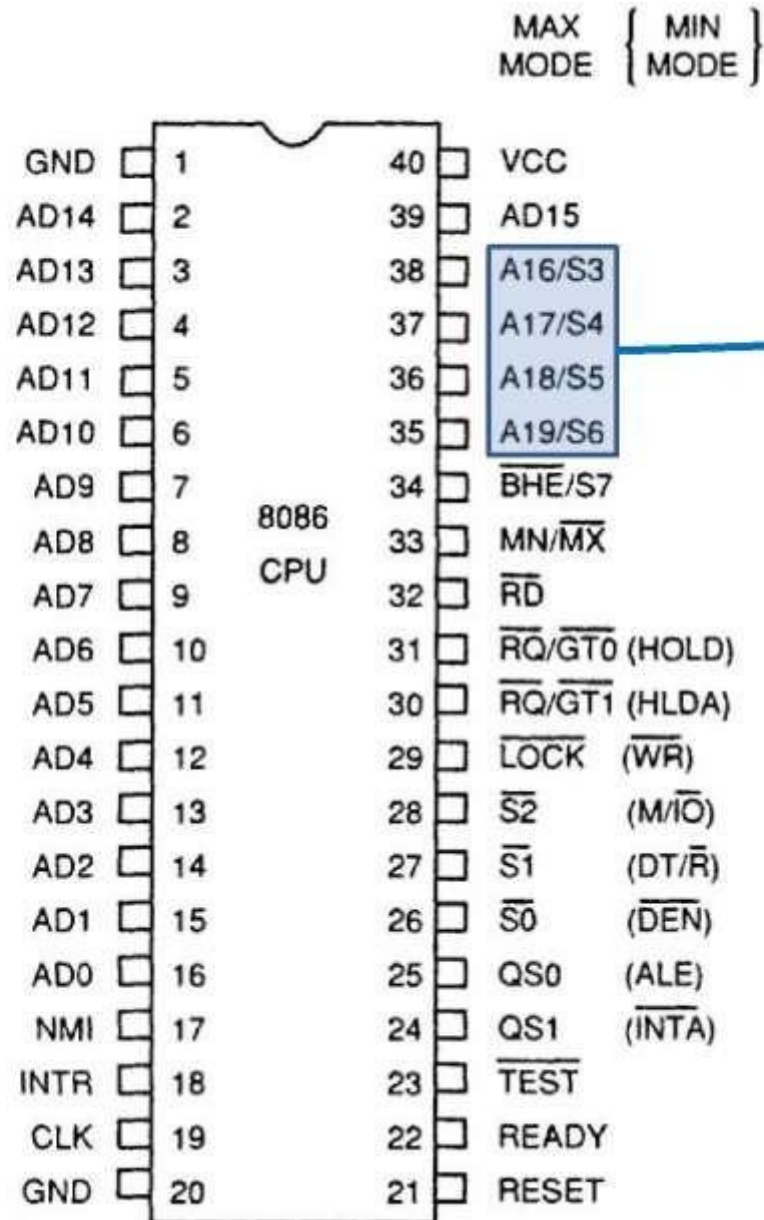
## INTEL-Pin8086Details

**S6:** Logic 0.

**S5:** Indicates condition of IF flag bits.

**S4-S3:** Indicate which segment is accessed during current bus cycle:

S4	S3	Function
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment



**Address/S**

Address—b  
19

A & Statu

16—S 6

3

## INTEL-Pin8086Details

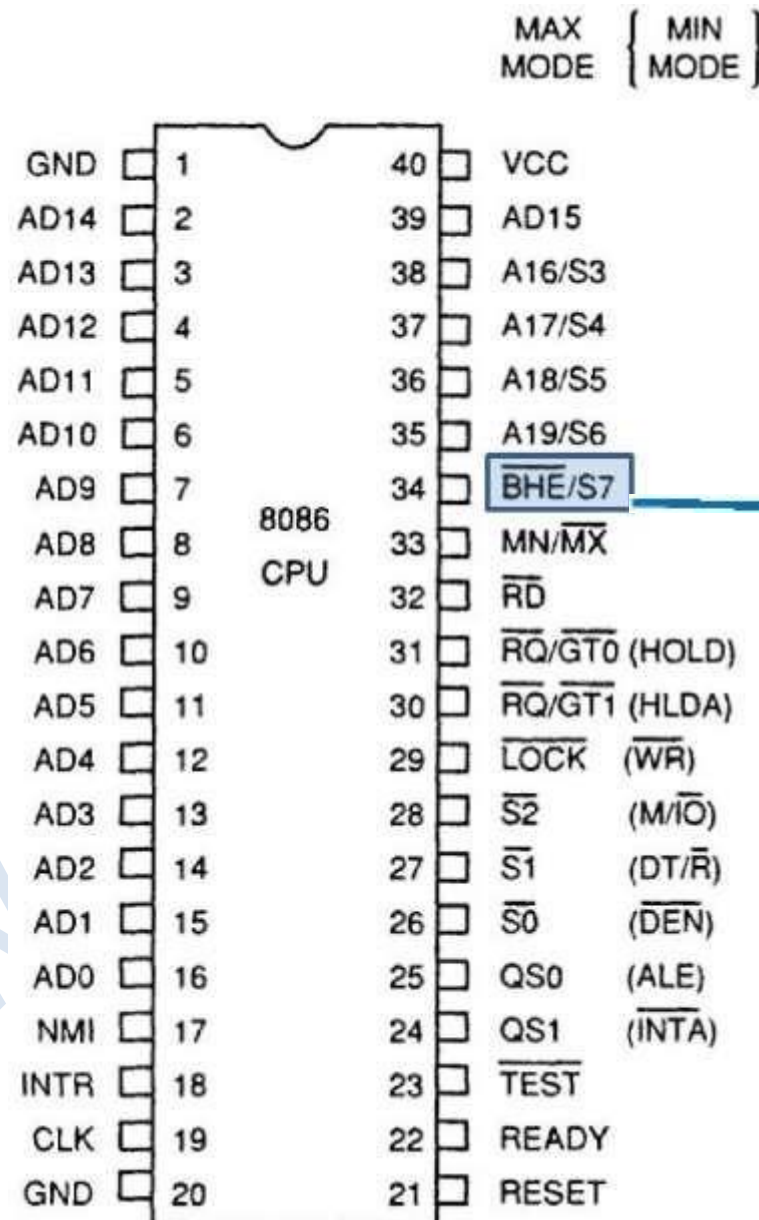
### BHE#, A<sub>0</sub>:

**0,0:** Whole word (16-bits)

**0,1:** High byte to/from odd address

**1,0:** Low byte to/from even address

**1,1:** No selection



Bus High

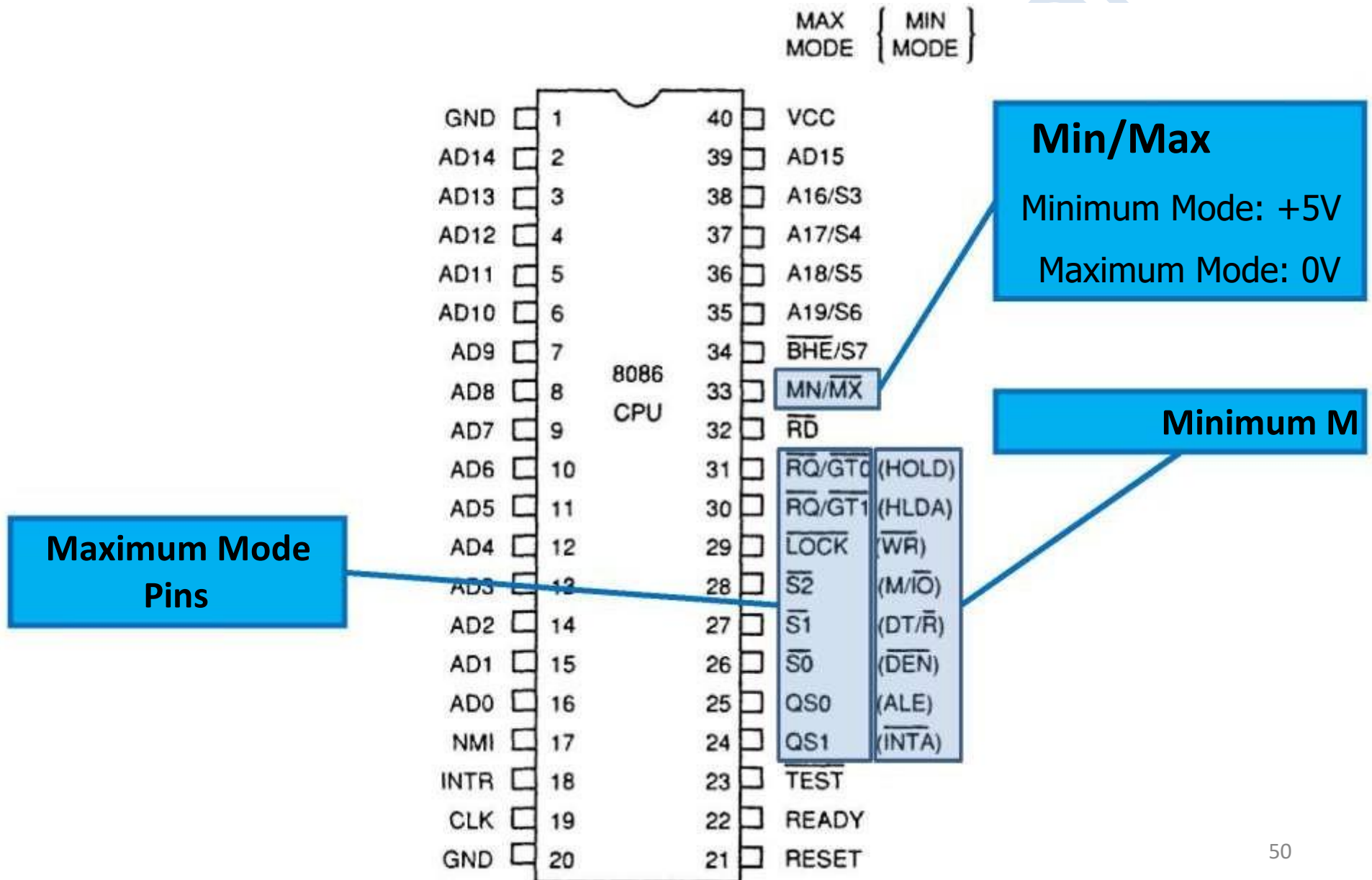
Enables signifi

D –Dduring  
15 8  
or write

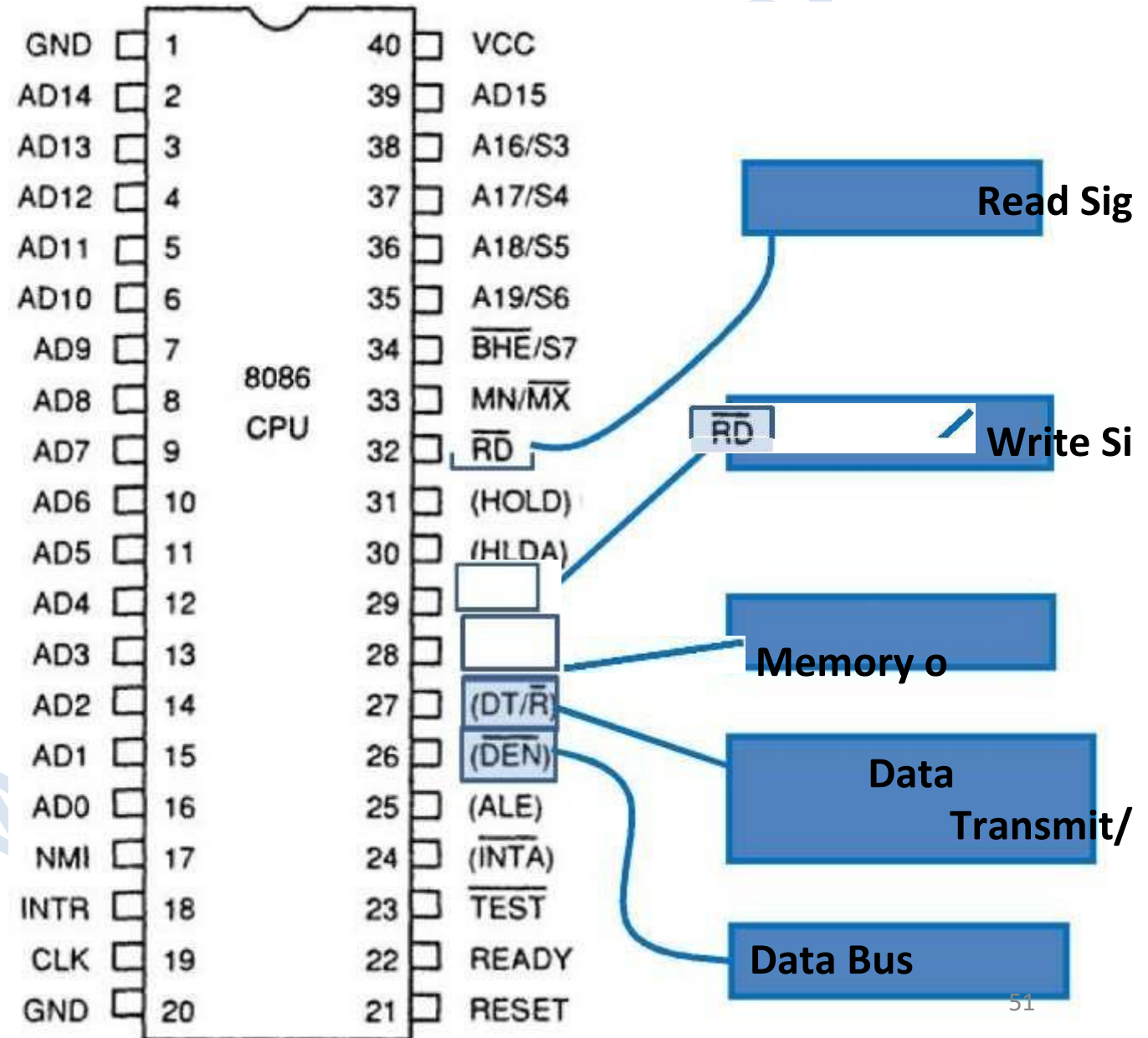
S: Alway

7

## INTEL-Pin8086Details



## Minimum-Pin Mode Details

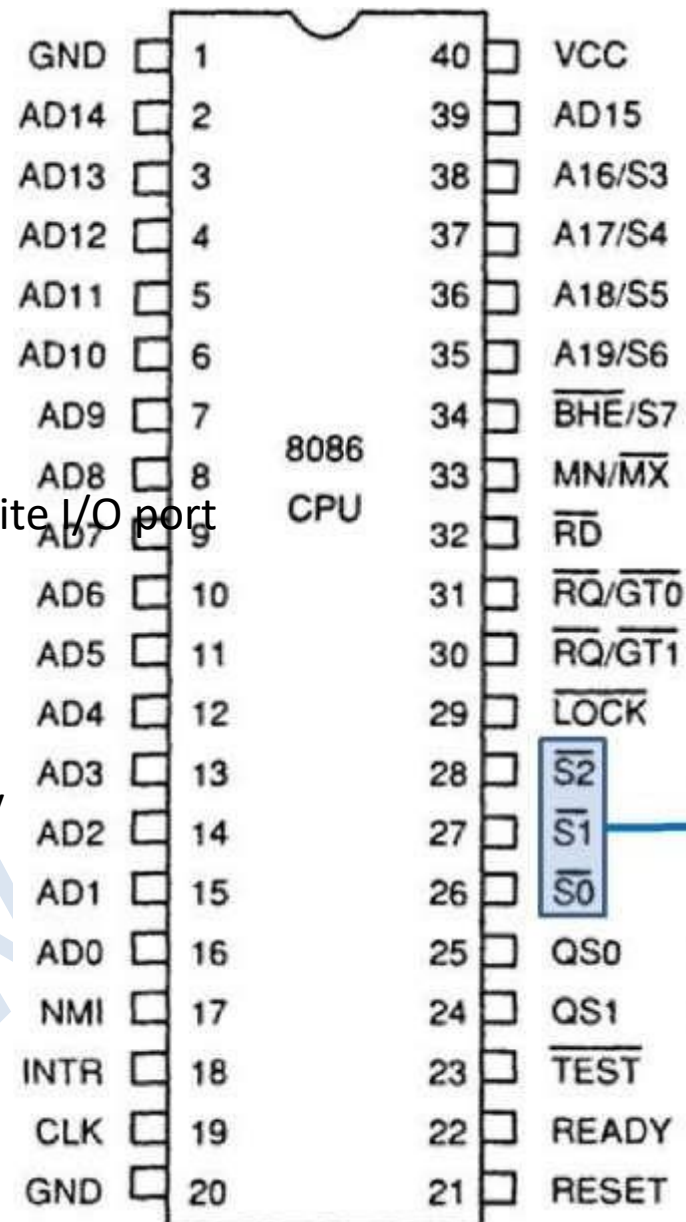




## Maximum -PinModeDetails

**S2 S1 S0**

0: INTA  
 001: read I/O port 010: write I/O port  
 11: halt  
 100: code access  
 101: read memory  
 110: write memory  
 111: -passivenone



### Status

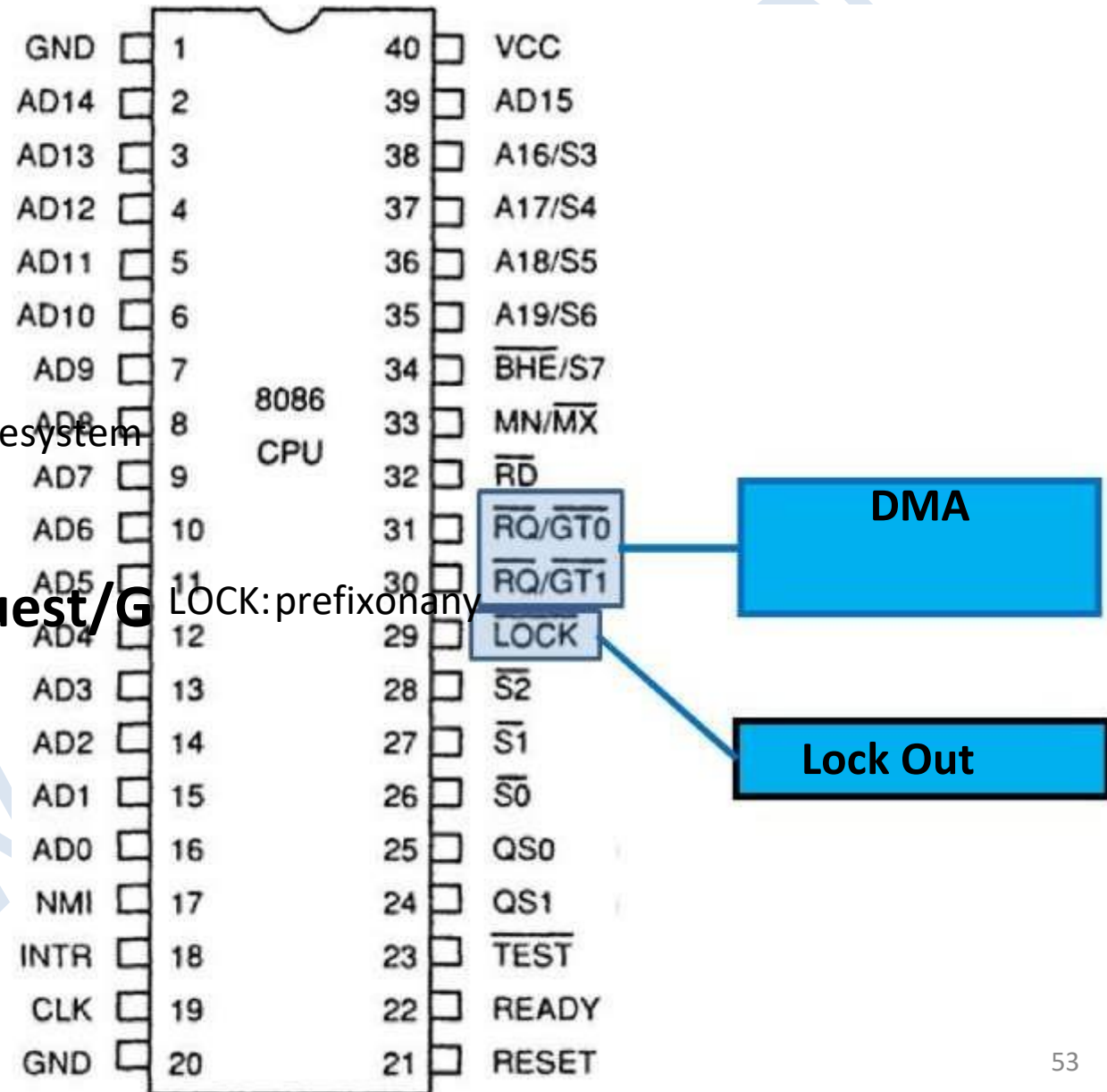
Inputs to generate e signals du mode.

## Maximum -PinModeDetails

### LockOutput

Used to lock peripherals off the system

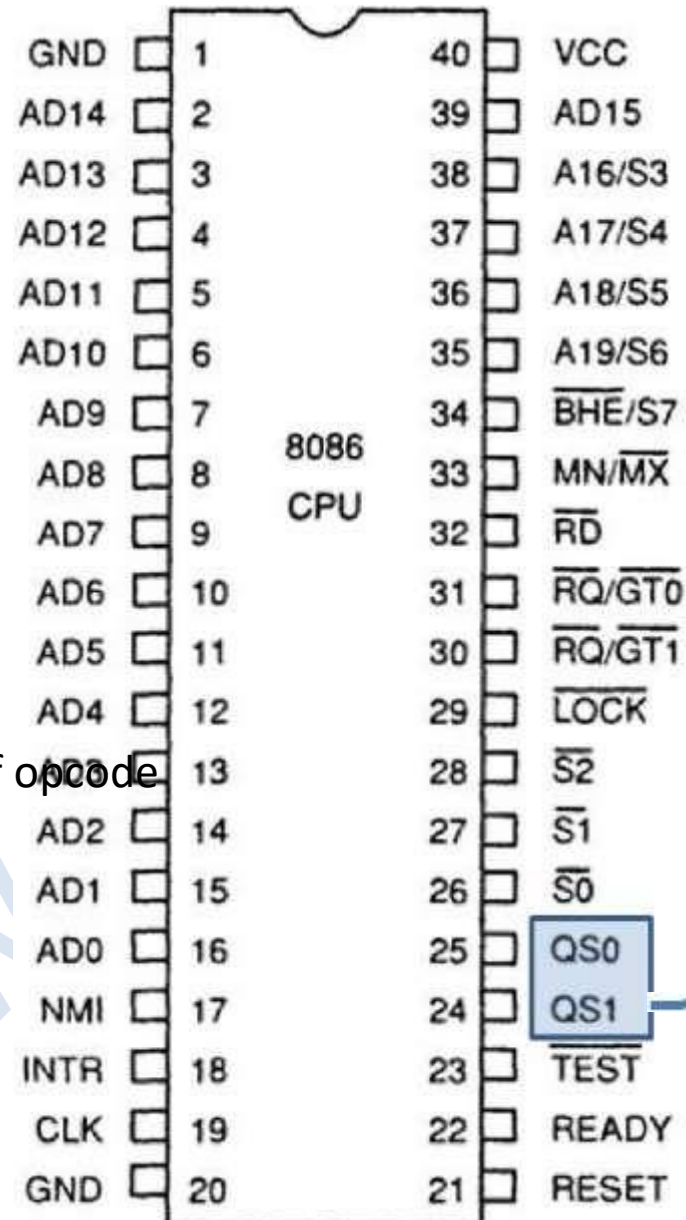
Activated by using the **Request/G** LOCK: prefix on any instruction



## Maximum -PinModeDetails

### QS1 QS0

- 0: Queue is idle
- 01: First byte of opcode
- 10: Queue is empty
- 11: Subsequent byte of opcode



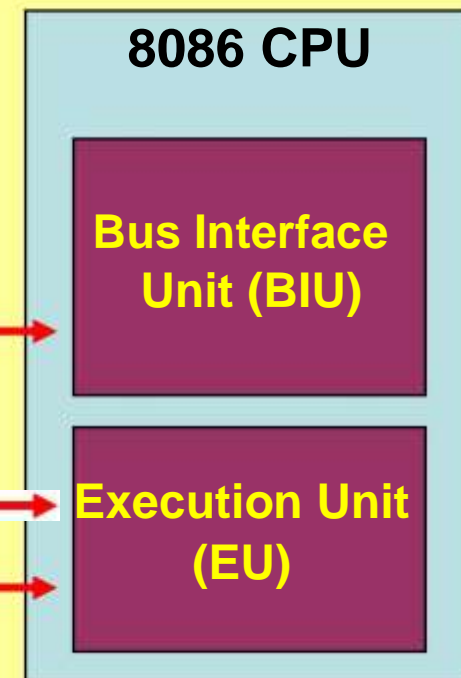
**Queue S**

Used by n coprocesso



# 8086 Internal Architecture

- 8086 employs parallel processing
- 8086 CPU has two parts which operate at the same time
  - Bus Interface Unit
  - Execution Unit
- CPU functions
  1. Fetch
  2. Decode
  3. Execute



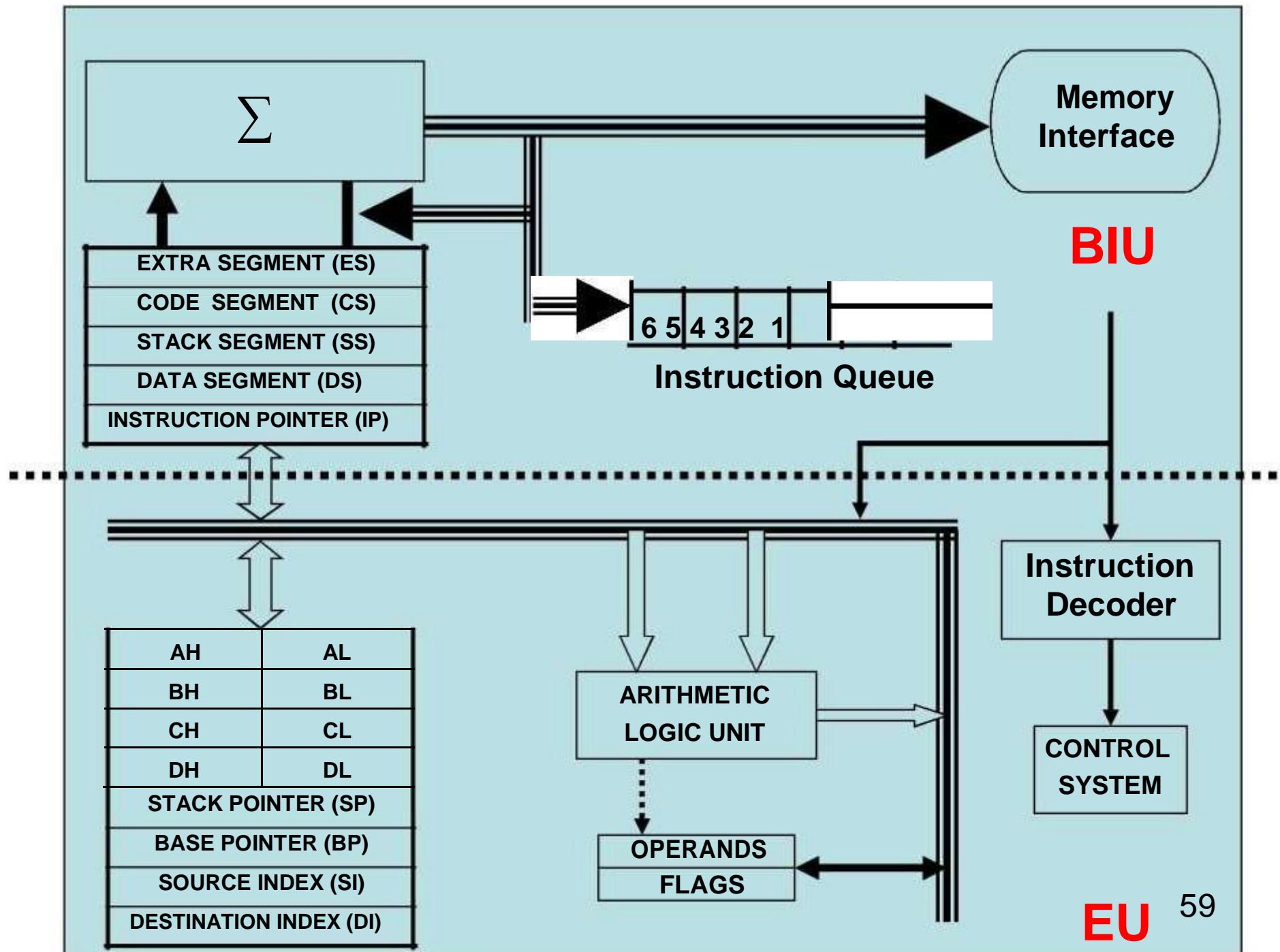
# Bus Interface Unit

- Sends out addresses for memory locations
- Fetches Instructions from memory
- Reads/Writes data to memory
- Sends out addresses for I/O ports
- Reads/Writes data to Input/Output ports

# Execution Unit

- Tells BIU (addresses) where to fetch instructions or data
- Decodes & Executes instructions
- Dividing the work between BIU & EU speeds up processing

# Architecture Diagram of 8086



# Execution Unit

- Main components are
  - **Instruction Decoder**
  - **Control System**
  - **Arithmetic Logic Unit**
  - **General Purpose Registers**
  - **Flag Register**
  - **Pointer & Index registers**

## Instruction Decoder

- **Translates instructions fetched from memory into a series of actions which EU carries out**

## Control System

- **Generates timing and control signals to perform the internal operations of the microprocessor**

## Arithmetic Logic Unit

- **EU has a 16-bit ALU which can ADD,**

# **SUBTRACT, AND, OR, increment, decrement, complement or shift binary numbers**

www.BrainKart.com



# General Purpose Registers

- EU has 8 general purpose registers
- Can be individually used for storing 8-bit data
- AL register is also called Accumulator
- Two registers can also be combined to form 16-bit registers
- The valid register pairs are – AX, BX, CX, DX

AH	AL
BH	BL
CH	CL
DH	DL

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

# Flag Register

- 8086 has a 16-bit flag register
- Contains 9 active flags
- There are two types of flags in 8086
  - **Conditional** flags – six flags, set or reset by EU on the basis of results of some arithmetic operations
  - **Control** flags – three flags, used to control certain operations of the processor

# Flag Register

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF				
---	---	---	---	----	----	----	----	----	----	---	----	---	----	---	----	--	--	--	--

1.	CF	CARRY FLAG	<b>Conditional Flags</b> (Compatible with 8085, except OF)
2.	PF	PARITY FLAG	
3.	AF	AUXILIARY CARRY	
4.	ZF	ZERO FLAG	
5.	SF	SIGN FLAG	
6.	OF	OVERFLOW FLAG	<b>Control Flags</b>
7.	TF	TRAP FLAG	
8.	IF	INTERRUPT FLAG	

## 9. **DF**    **DIRECTION FLAG**

[www.BrainKart.com](http://www.BrainKart.com)

# Flag Register

## 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.

## Sign Flag

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

## 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 9 8 7 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

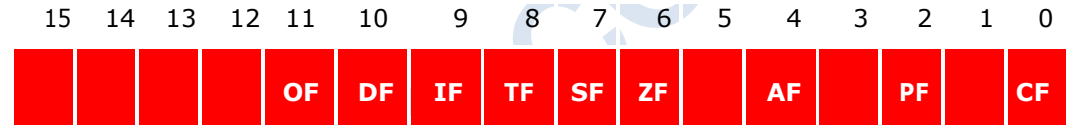
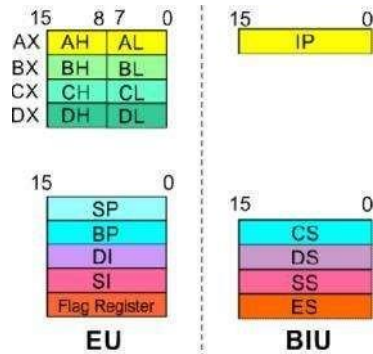
This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

## Interrupt Flag

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

## Registers, Flag

**8086 registers categorized into 4 groups**



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

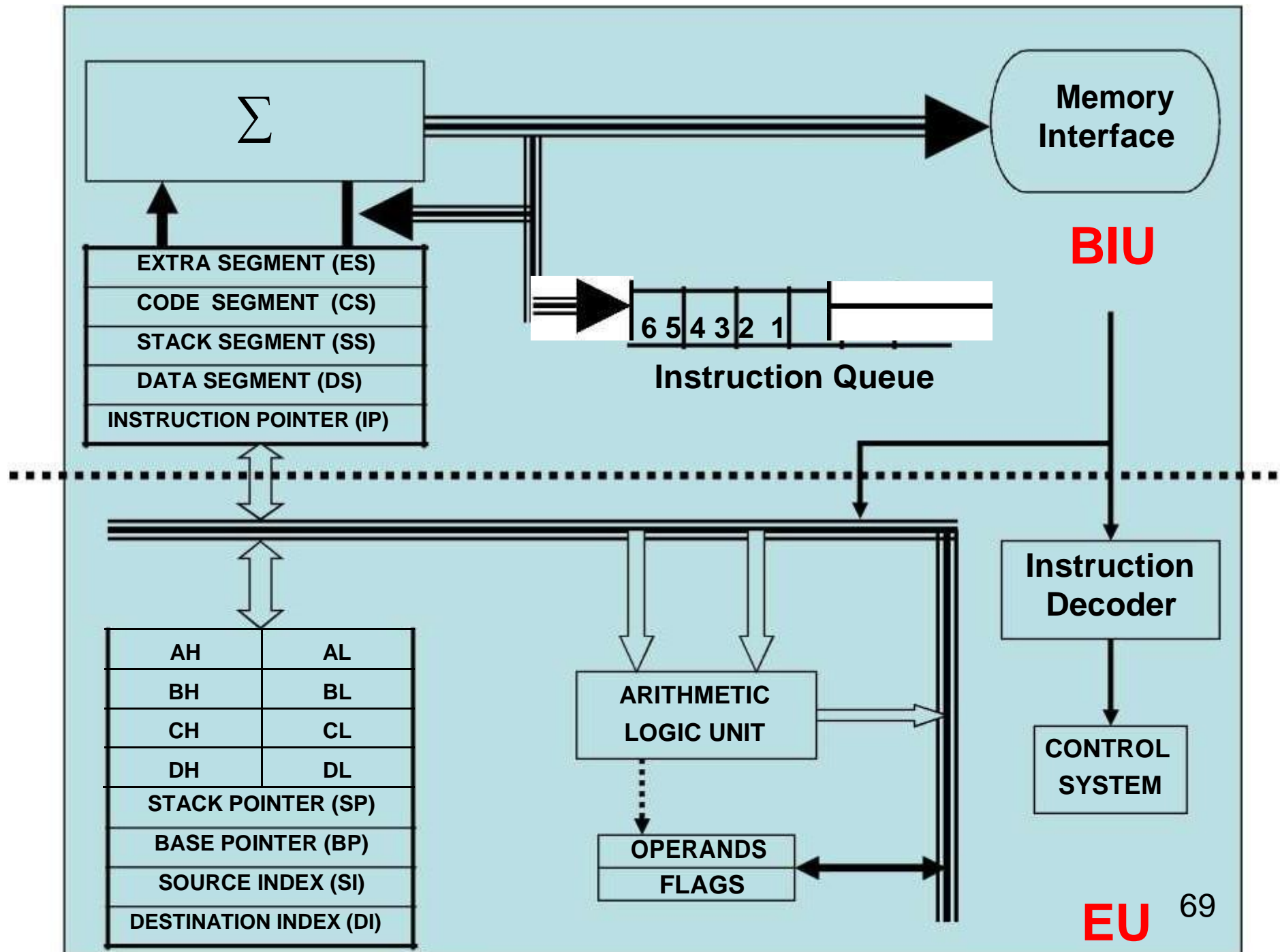
## Registers and Special Functions

Register	Name of the Register	Special Function
<b>AX</b>	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
<b>AL</b>	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
<b>BX</b>	Base register	Used to hold base value in base addressing mode to access memory data
<b>CX</b>	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
<b>DX</b>	Data Register	Used to hold data for multiplication and division operations
<b>SP</b>	Stack Pointer	Used to hold the offset address of top stack memory
<b>BP</b>	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
<b>SI</b>	Source Index	Used to hold index value of source operand (data) for string instructions
<b>DI</b>	Data Index	Used to hold the index value of destination operand (data) for string operations

# Bus Interface Unit

- Main Components are
  - **Instruction Queue**
  - **Segment Registers**
  - **Instruction Pointer**





# Instruction Queue

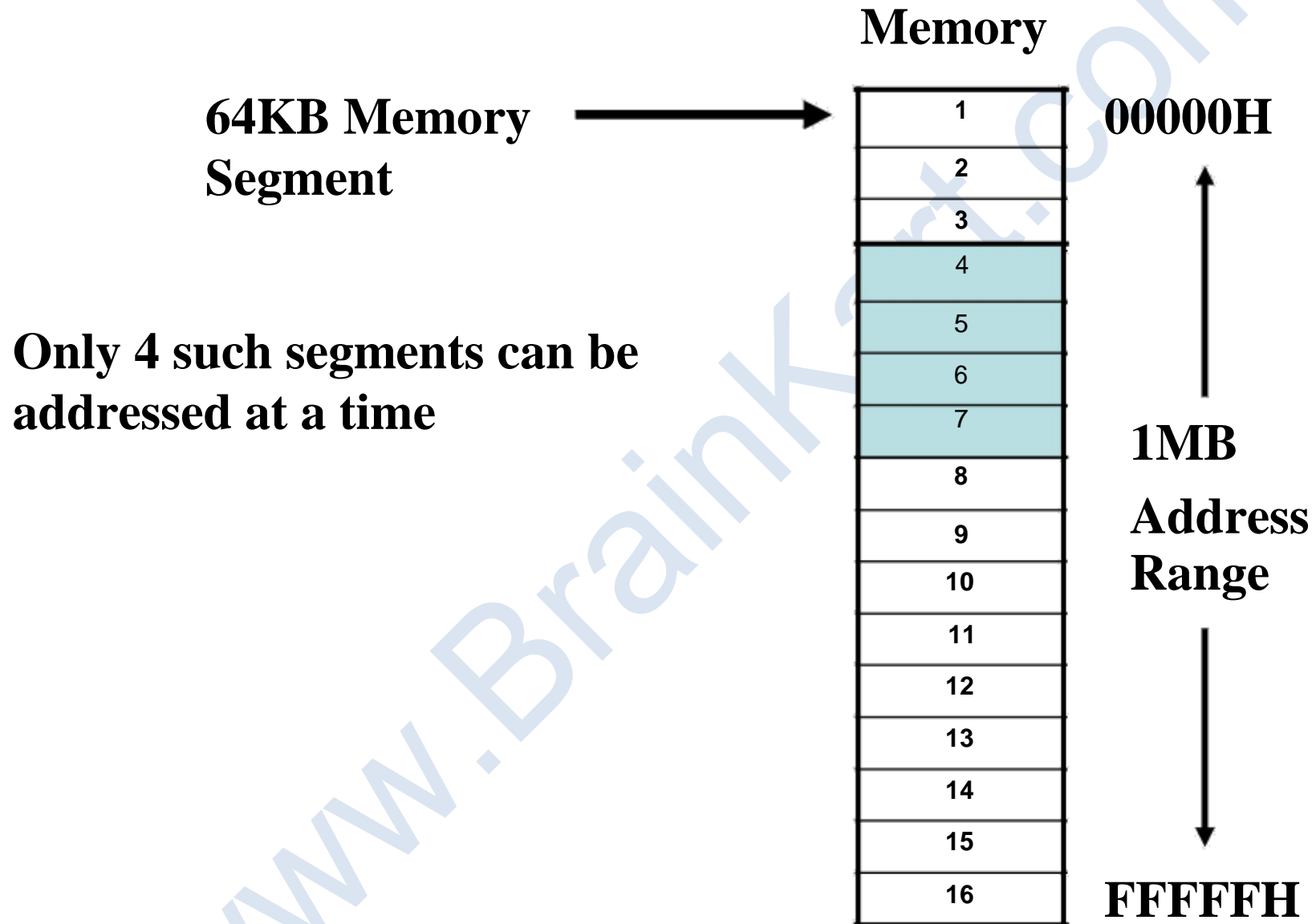
- 8086 employs parallel processing
- When EU is busy decoding or executing current instruction, the buses of 8086 **may not be** in use.
- At that time, BIU can use buses to fetch upto six instruction bytes for the following instructions
- BIU stores these pre-fetched bytes in a **FIFO** register called Instruction **Queue**
- When EU is ready for its next instruction, it simply reads the instruction from the queue in BIU

# Pipelining

- **EU of 8086 does not have to wait in between for BIU to fetch next instruction byte from memory**
- **So the presence of a queue in 8086 speeds up the processing**
- **Fetching the next instruction while the current instruction executes is called pipelining**

# Memory Segmentation

- 8086 has a **20-bit** address bus
- So it can address a maximum of **1MB** of memory
- 8086 can work with only **four 64KB** segments at a time within this 1MB range
- These four memory segments are called
  - **Code** segment
  - **Stack** segment
  - **Data** segment
  - **Extra** segment



## Code Segment

- That part of memory from where BIU is currently fetching instruction code bytes

## Stack Segment

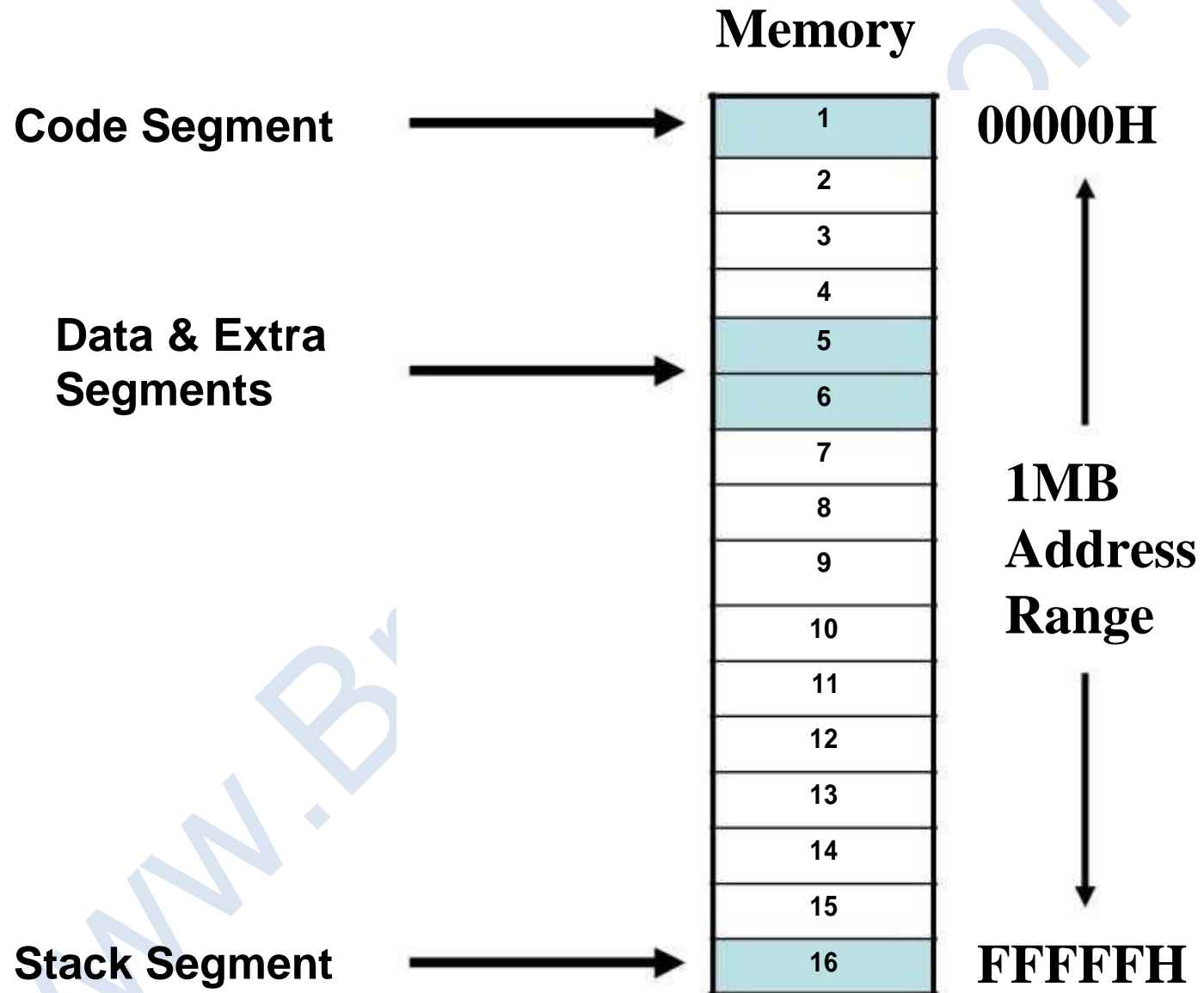
- A section of memory set aside to store addresses and data while a subprogram executes

## Data & Extra Segments

- Used for storing data values to be used

**in the program**

www.BrainKart.com

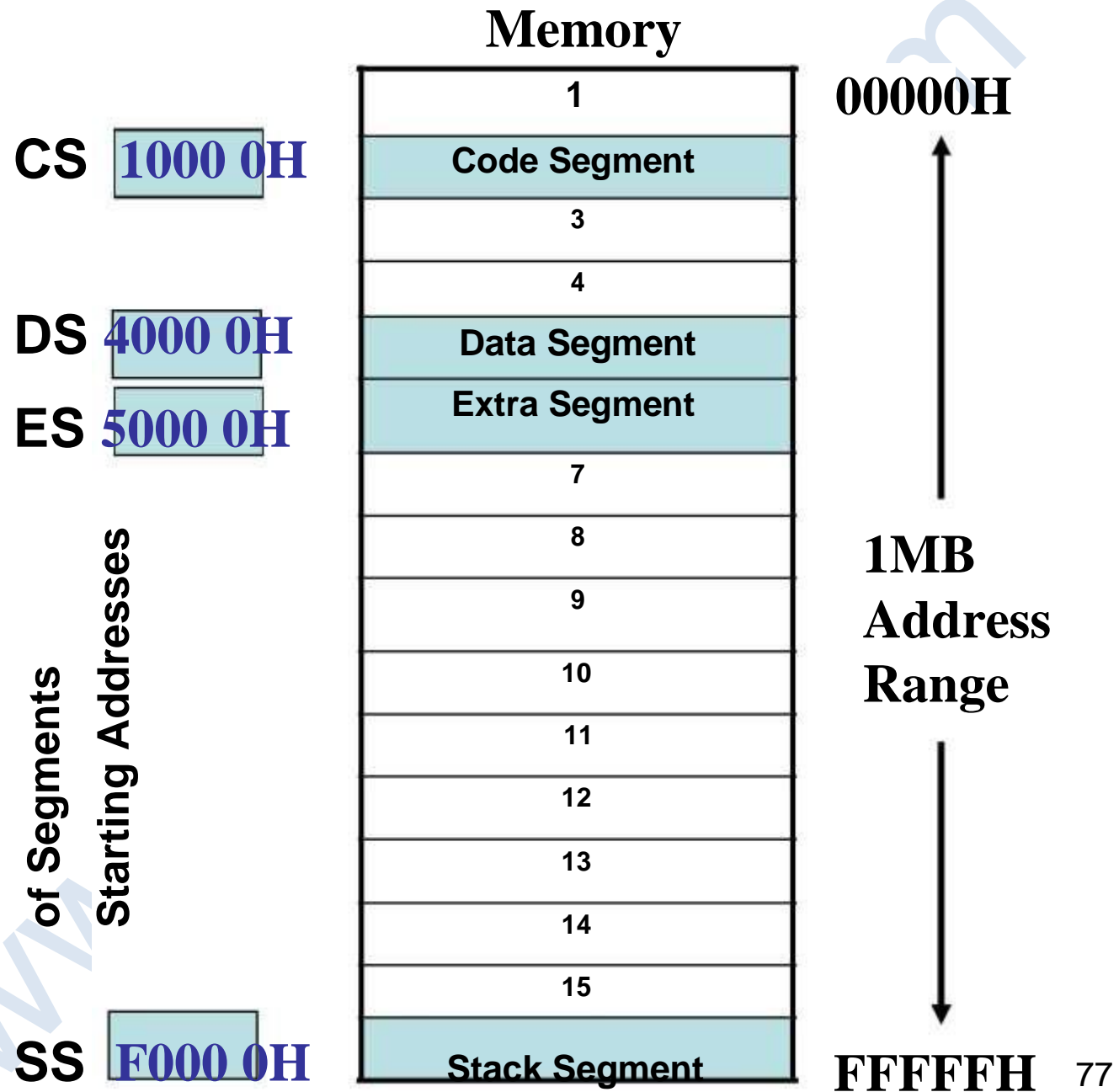




# Segment Registers

- hold the upper 16-bits of the starting address for each of the segments
- The four segment registers are
  - **CS (Code Segment register)**
  - **DS (Data Segment register)**
  - **SS (Stack Segment register)**
  - **ES (Extra Segment register)**

www.BrainKart.com



- Address of a segment is of 20-bits
- A segment register stores only upper 16-bits
- BIU always inserts zeros for the lowest 4-bits of the 20-bit starting address.
- E.g. if CS = 348AH, then the code segment will start at 348A0H
- A 64-KB segment can be located anywhere in the memory, but will start at an address with zeros in the

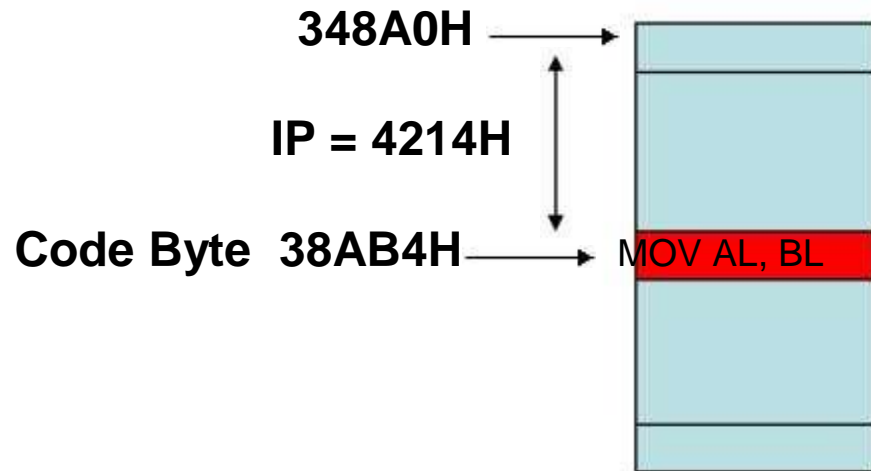
lowest 4-bits

# Instruction Pointer (IP) Register

- a 16-bit register
- Holds 16-bit **offset**, of the next instruction byte in the **code segment**
- BIU uses **IP** and **CS** registers to generate the **20-bit address** of the instruction to be fetched from memory

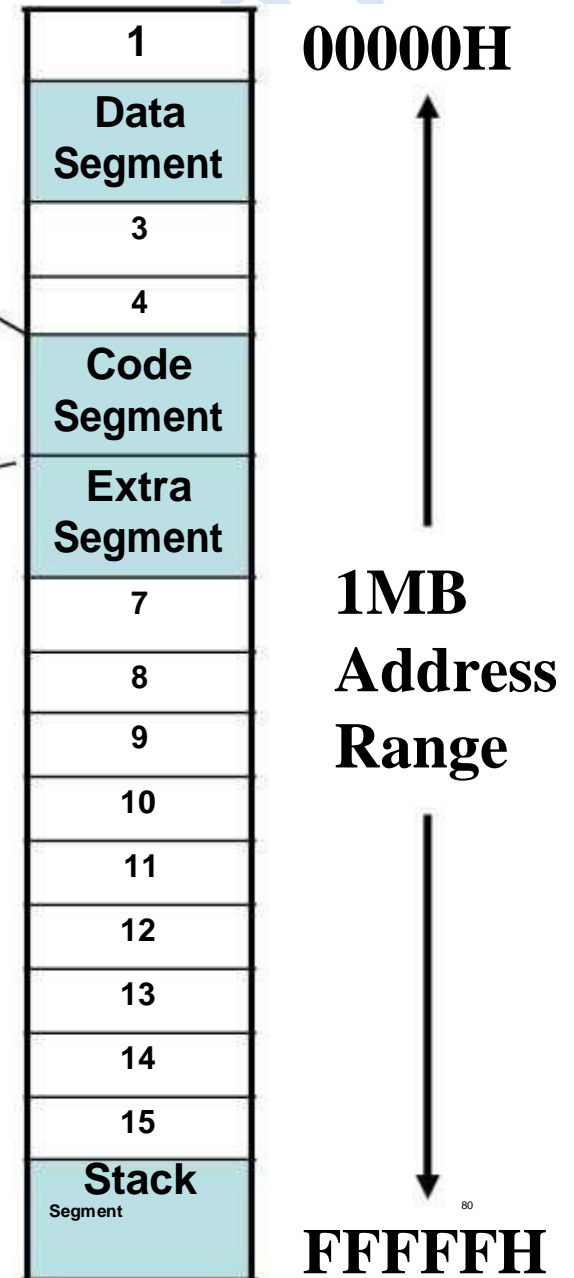
# Physical Address Calculation

Start of Code Segment



$$\begin{array}{rcl} \text{CS} & 348A0 \text{ H} & \\ \text{IP} & + 4214 \text{ H} & \\ \hline \text{Physical Address} & 38AB4 \text{ H} & \end{array}$$

Memory



## Stack Segment (SS) Register Stack Pointer (SP) Register

- Upper 16-bits of the starting address of stack segment is stored in SS register
- It is located in BIU
- SP register holds a 16-bit offset from the start of stack segment to the top of the stack
- It is located in EU



www.BrainKart.com

# Other Pointer & Index Registers

➤ Base Pointer (BP)

register ➤ Source Index (SI)

register ➤ Destination Index  
(DI) register

➤ Can be used for temporary storage of data

➤ Main use is to hold a 16-bit offset of a  
data word in one of the segments

# ADDRESSING MODES OF 8086

# Various Addr

- 1.Immediate Addressing    2.Register Addressing    M
- 3.Direct Addressing Mod    4.Register Indirect Add
- 5.Index Addressing Mode
- 6.Based Addressing Mode    7.Based & Indexed Addre
- 8.Based & Indexed with Mode
- 9.Strings Addressing Mo

# 1. IMMEDIATE ADD

- The instruction will specify the name of the register which holds the data to be operated by the instruction.
- Source data is instruction
- Ex:  $\text{MOV}_{\text{H}} \text{AX}, 10\text{AB}$   
 $\rightarrow \text{AL} = \text{AB}_{\text{H}}, \text{AH} = 10$

## 2. REGISTER ADDR

- In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction
- **Ex:**  $\text{MOV}_{\text{H}} \text{AX}, \text{BL}$   
 $\text{MOV} \text{AX}, \text{BL}_{\text{H}}$

### 3. DIRECT ADDRE

- Memory address is supplied with in the instruction
- Mnemonic: MOV AH,[MEMBDS]

AH ← [1000<sub>H</sub>]

- But the memory address is

**not** index or pointer register

www.BrainKart.com



## 4. REGISTER INDIRECT

- Memory address is supplied in an index or pointer register
- **EX:**

MOV AX,[SI] ;       $AL \leftarrow [SI] ; AH \leftarrow [SI+1]$

JMP [DI] ;       $IP \leftarrow [DI+1: DI]$

INC BYTE PTR [BP] ;       $[BP] \leftarrow [BP]+1$

DEC WORD PTR [BX] ;  
                          $[BX+1:BX] \leftarrow [BX+1:BX]-1$

## 5.Indexed Addr

- Memory address is register plus disp

MOV AX,[SI+2] **AL**  $\leftarrow$  **[SI+2];**

JMP [DI+2] **IP** **[BX+3];**

## 6. Based Addre

- Memory address is th base register plus a instruction

- Ex:

MOV AX,[BP+2] **AL** ←

← **[BP+2]**

JMP [BX+2] **IP** ←

**[BX+2]**

## 7.BASED & INDEX ADD

- Memory address is the sum of the index register & base register

Ex:

MOV AX,[BX+SI] ;  $AL \leftarrow [BX+SI]$  ;  $AH \leftarrow [BX+SI+1]$

JMP [BX+DI] ;  $IP \leftarrow [BX+DI+1 : BX+DI]$

INC BYTE PTR [BP+SI] ;  $[BP] \leftarrow [BP]+1$

DEC WORD PTR [BP+DI] ;

$[BX+1:BX] \leftarrow [BX+1:BX]-1$

## 8. BASED & INDEXED WITH DISP

- Memory address is the sum of an index register , base register and displacement within instruction

MOV AX,[BX+SI+6] ;  $AL \leftarrow [BX+SI+6]$  ;  $AH \leftarrow [BX+SI+7]$

JMP [BX+DI+6] ;  $IP \leftarrow [BX+DI+7 : BX+DI+6]$

INC BYTE PTR [BP+SI+5] ;

DEC WORD PTR [BP+DI+5] ;

## 9. Strings Addr

- The memory source addr data segment, and the address is register DI
- Ex: `MOVSB` ← `[ES:DI]`
- If `DF=0`                      `SI, DI` ← `SI+1 DI+1`  
    `DF=1` `SI-1 SI`, ← `DI-1`