ज्ञानेन प्रकाशते जगत्
INDUS
UNIVERSITY

ELECTRONICS & COMMUNICATION ENGINEERING
IITE Ahmedabad

MICROPROCESSORS

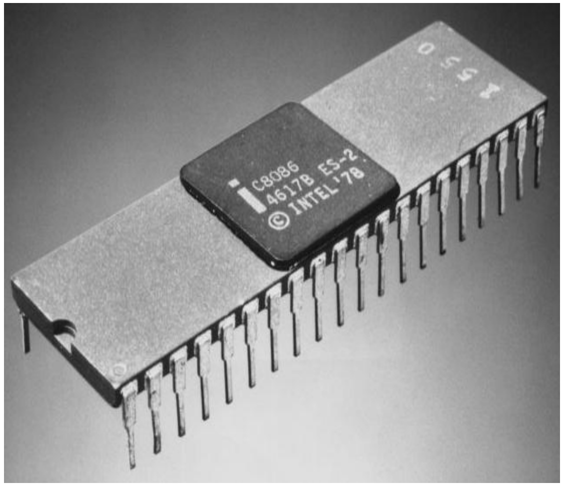
(SUBJECT CODE-CE0506/CS0506)
UNIT-2&3
B.TECH (CE/CSE)
SEMESTER-V

Shikha Singh

Academic Year 2019-2020

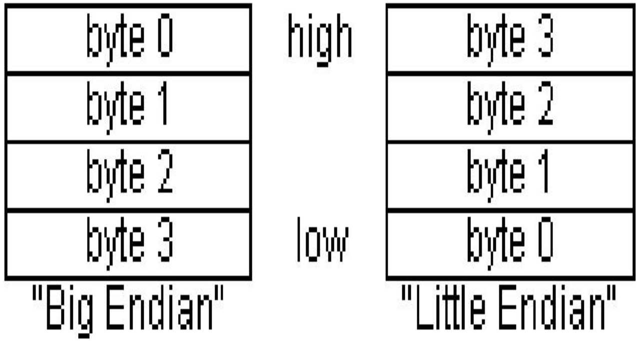
Features of 8086 Microprocessor

1. Intel 8086 was launched in 1978.
2. It was the first 16-bit microprocessor, 40-pin Dual-Inline-Package (DIP).
3. This microprocessor had major improvement over the execution speed of 8085.
4. It is available in three versions:
    - a. 8086 (5 MHz)
    - b. 8086 (8 MHz)
    - c. 8086 (10 MHz)
5. It consists of 29,000 transistors & had Multiply and Divide instructions.



Data Structure

- Most Significant byte at lowest address ("Big Endian"), OR
- Least Significant byte at lowest address ("Little Endian")
- 8086 has a little Endian data structure



byte 0	high	byte 3
byte 1		byte 2
byte 2		byte 1
byte 3	low	byte 0
"Big Endian"		"Little Endian"

Block Diagram of Intel 8086

The 8086 CPU is divided into two functional units:

1. **Bus Interface Unit (BIU)**
2. **Execution Unit (EU)**

Internal (Block Diagram) Architecture of 8086 CPU

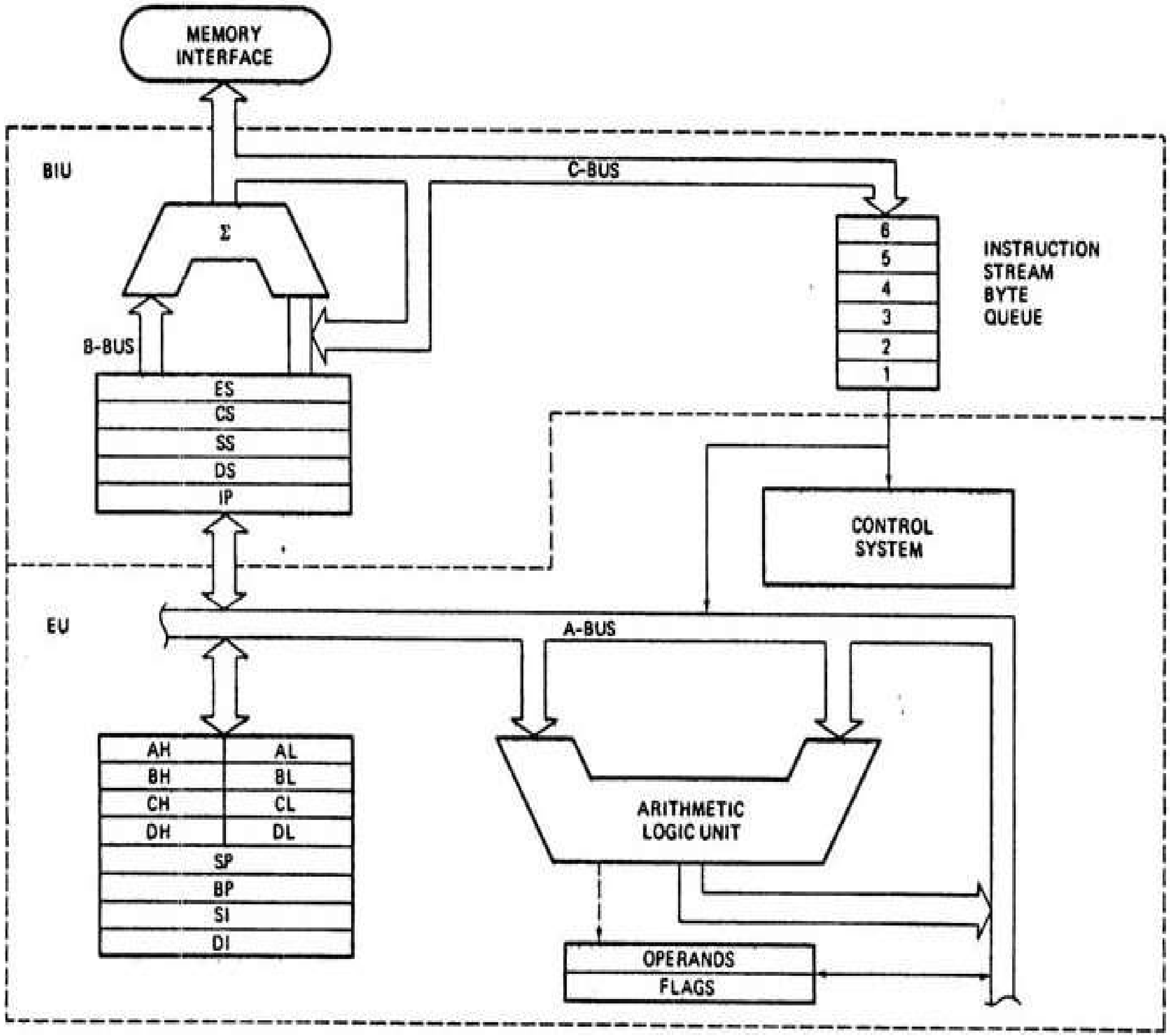


Fig. 1: Block Diagram of Intel

Bus Interface Unit (BIU)

The BIU provides hardware functions. Including generation of the memory and I/0 addresses for the transfer of data between itself and the outside world.It also Fetches the instruction or data from memory and Writes the data to memory.

Instruction Queue

To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory and stored in a register called instruction queue.Then all bytes have to be given to EU one by one.

This pre fetching operation of BIU may be in parallel with execution operation of EU, which improves the speed execution of the instruction.

Execution Unit (EU)

The functions of execution unit are:

To decode the instructions.

To execute the instructions.

EU receives the program instruction codes and data from the BIU, executes these instructions and store the result in general purpose registers. By passing the data back to the BIU data can be stored in a memory location or written to an output device

It receives and outputs all its data through BIU.

Fetch and Execute cycle

- The BIU outputs the contents of the instruction pointer register (IP) onto the Address bus, causing the selected byte or word in memory to be read into the BIU.
- Register IP is incremented by one to prepare for the next instruction fetch.
- Once inside the BIU, the instruction is passed to the queue: a first-in/first-out storage register sometimes likened to a pipeline.
- Assuming that the queue is initially empty, the EU immediately draws this instruction from the queue and begins execution.

Fetch and Execute cycle

- While the EU is executing this instruction, the BIU proceeds to fetch a new instruction. Depending on the execution time of the first instruction, the BIU may fill the queue with several new instructions before the EU is ready to draw its next instruction.

- The cycle continues, with the BIU filling the queue with instructions and the EU fetching and executing these instructions.

Fetch and Execute cycle

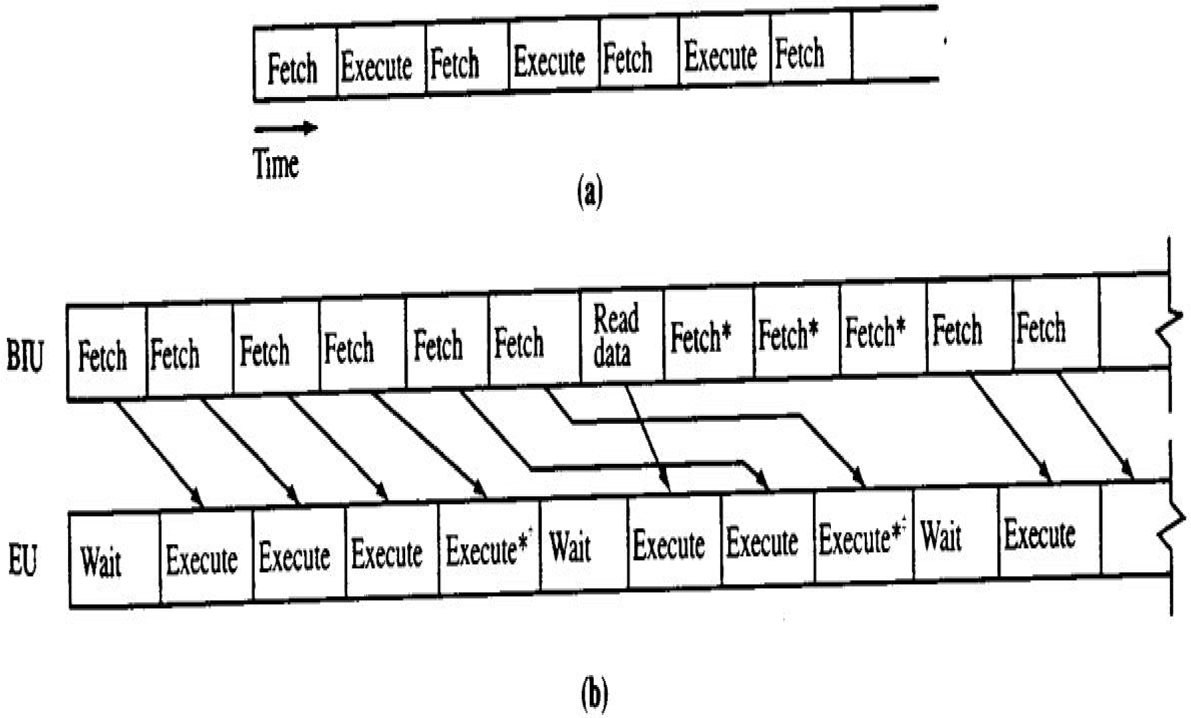
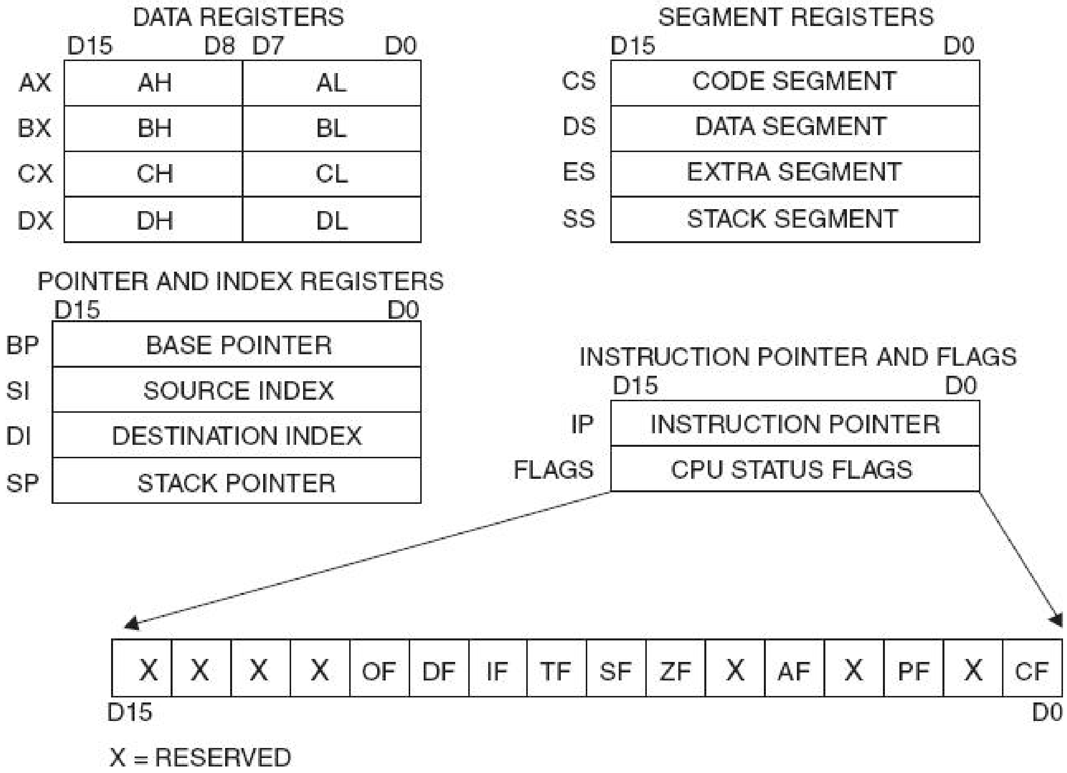


Figure 3.2 (a) The nonpipelined microprocessor follows a sequential fetch and execute cycle. (b) The 8086's pipelined architecture allows the EU to execute instructions without the delays associated with instruction fetching.

Programming model of 8086

- The programming model for a microprocessor shows the various internal registers that are accessible to the programmer. The Following Figure is a model for the 8086.Ingeneral, each register has a special function.



DATA REGISTERS

	D15 D8	D7 D0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

SEGMENT REGISTERS

	D15 D0
CS	CODE SEGMENT
DS	DATA SEGMENT
ES	EXTRA SEGMENT
SS	STACK SEGMENT

POINTER AND INDEX REGISTERS

	D15 D0
BP	BASE POINTER
SI	SOURCE INDEX
DI	DESTINATION INDEX
SP	STACK POINTER

INSTRUCTION POINTER AND FLAGS

	D15 D0
IP	INSTRUCTION POINTER
FLAGS	CPU STATUS FLAGS

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

D15 D0

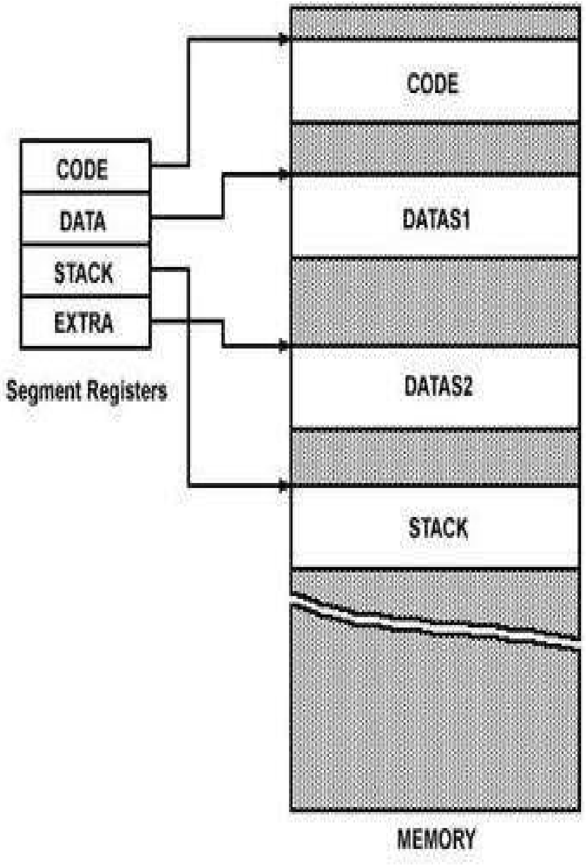
X = RESERVED

Registers

- General Purpose Registers:
  - These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX,BX, CX, and DX.
- Pointer Group Registers:
  - The pointer and index group are all 16 bit registers. These registers are used as memory pointers.
    - **MOV AH,[SI]**
  - Register IP is included into memory pointer but this register has only one function to point the next instruction to be fetched to the BIU.

Registers

- Segment Register
- Additional registers called segment registers generate memory address when combined with other in the microprocessor. In 8086 microprocessor, memory is divided into 4 segments as follow:



CONT…

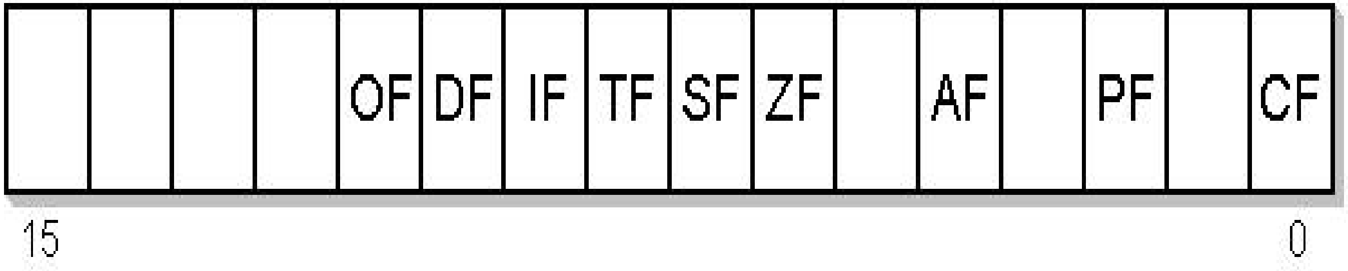
- 1. **Code Segment** (**CS**): The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.
- 2. **Data Segment** (**DS**): The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.
- 3. **Stack Segment** (**SS**): SS defined the area of memory used for the stack.
- 4. **Extra Segment** (**ES**): ES is additional data segment that is used by some of the string to hold the destination data.

Segment registers

- Calculate the beginning and ending address for the data segment assuming that register
- DS = E000H
    - Solution
        - Base address can be found by appending four 0's :
        - Base address E0000H
        - Ending address can be found by adding 64K ..
        - E0000H + FFFFH = EFFFFH

Flag Registers of 8086

Flag register in EU is of 16-bit and is shown in fig. :



Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.

CONT...

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

- 1. **Conditional Flags**
- 2. **Control Flags**

Conditional Flags

- Conditional flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:
  - ***Carry Flag (CF)***
  - ***Auxiliary Flag (AF)***
  - ***Parity Flag (PF)***
  - ***Zero Flag (ZF)***
  - ***Sign Flag (SF)***
  - ***Overflow Flag (OF)***

Conditional flag

- **Carry Flag (CF)** - this flag is set to 1 when there is an carry out from MSB.
- **Parity Flag (PF)** - this flag is set to 1 when there is even number of one bits in result, and to 0 when there is odd number of one bits.
- **Auxiliary Flag (AF)** - set to 1 when there is a carry from low nibble to upper nibble(4 bits).
- **Zero Flag (ZF)** - set to 1 when result is **zero**. For non-zero result this flag is set to 0.
- **Sign Flag (SF)** - set to 1 when result is **negative**. When result is **positive** it is set to 0. (This flag takes the value of the most significant bit.)

Control Flags

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

1. ***Trap Flag (TP):***
2. ***Interrupt Flag (IF):***
3. ***Direction Flag (DF):***

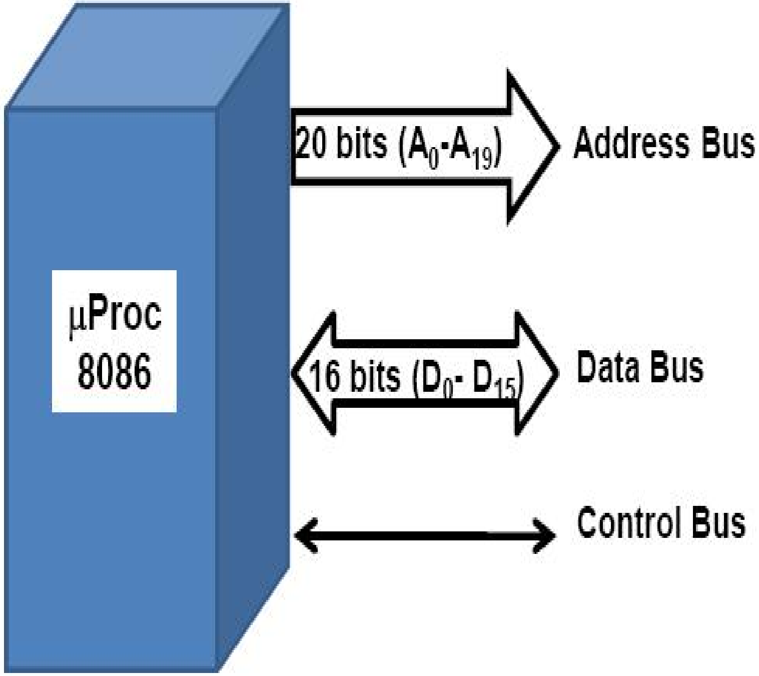
Control Flags

- **Trap Flag (TF)** - Used for on-chip debugging.

- **Interrupt enable Flag (IF)** - when this flag is set to **1** CPU reacts to interrupts from external devices.

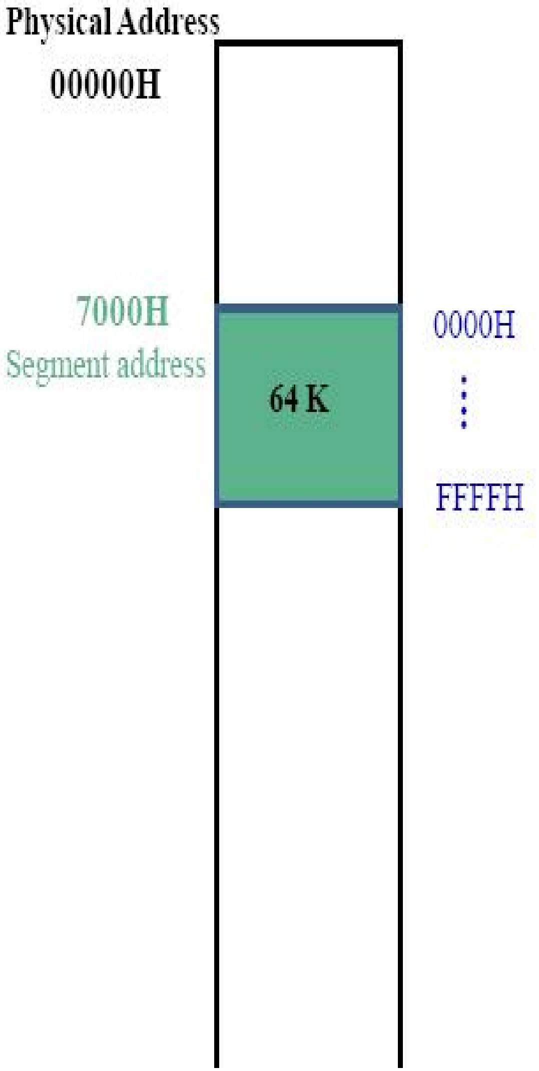
- **Direction Flag (DF)** –Causing the string instruction to auto decrement the index register when set and Clearing DF causes the Auto increment.

Physical And Logical Address



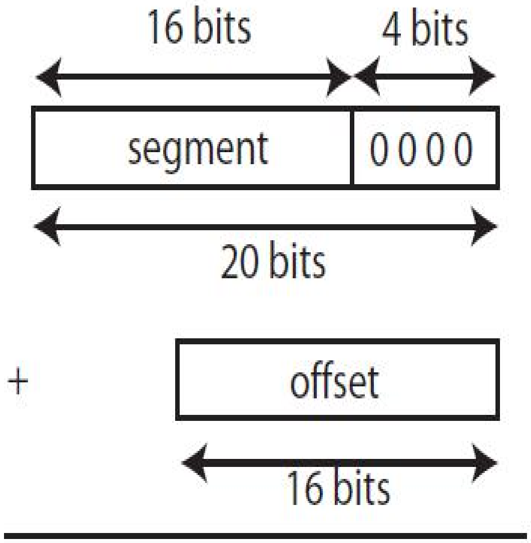
IP Register

- The Instruction Pointer register(IP) contains the offset address of the next sequential instruction to be executed. Thus, the IP register can not be directly modified.
- These register descriptions have slowly been introducing us to a new way of addressing memory, called:
- segment-offset addressing.
- The segment register is used to point to the beginning of any



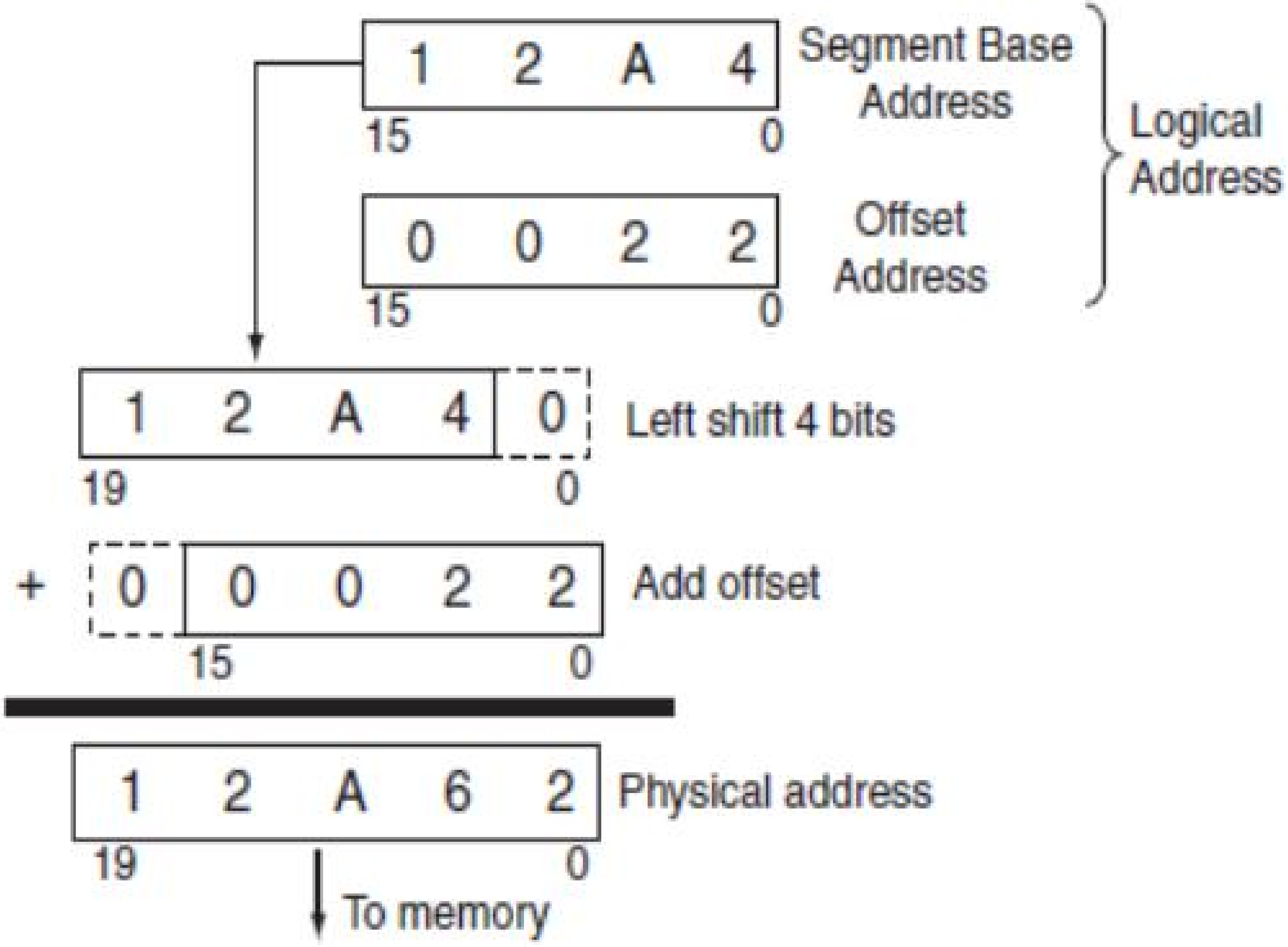
Physical Address

- Each segment register is 16 bit wide while the address bus is 20 bits wide. The BIU takes care of this by appending four 0 s to the lower order bits of the segment regi



Physical Address
20 bit

Converting Logical Address to Physical address



Segment Register and Default offset registers in 8086

Segment Registers	**Default offset Register**
CS	IP
DS	BX,SI,DI
SS	SP or BP
ES	DI for String Instruction

Ex1

- Let us Assume that the CS register has the value 3000H and IP register has value 2000h. To fetch an instruction from the next memory location.

CS x 10H = 30000H Base address of the code Segment

+IP = 02000H Offset Address

32000H Memory Address from where the next Instruction to be taken

Ex:2

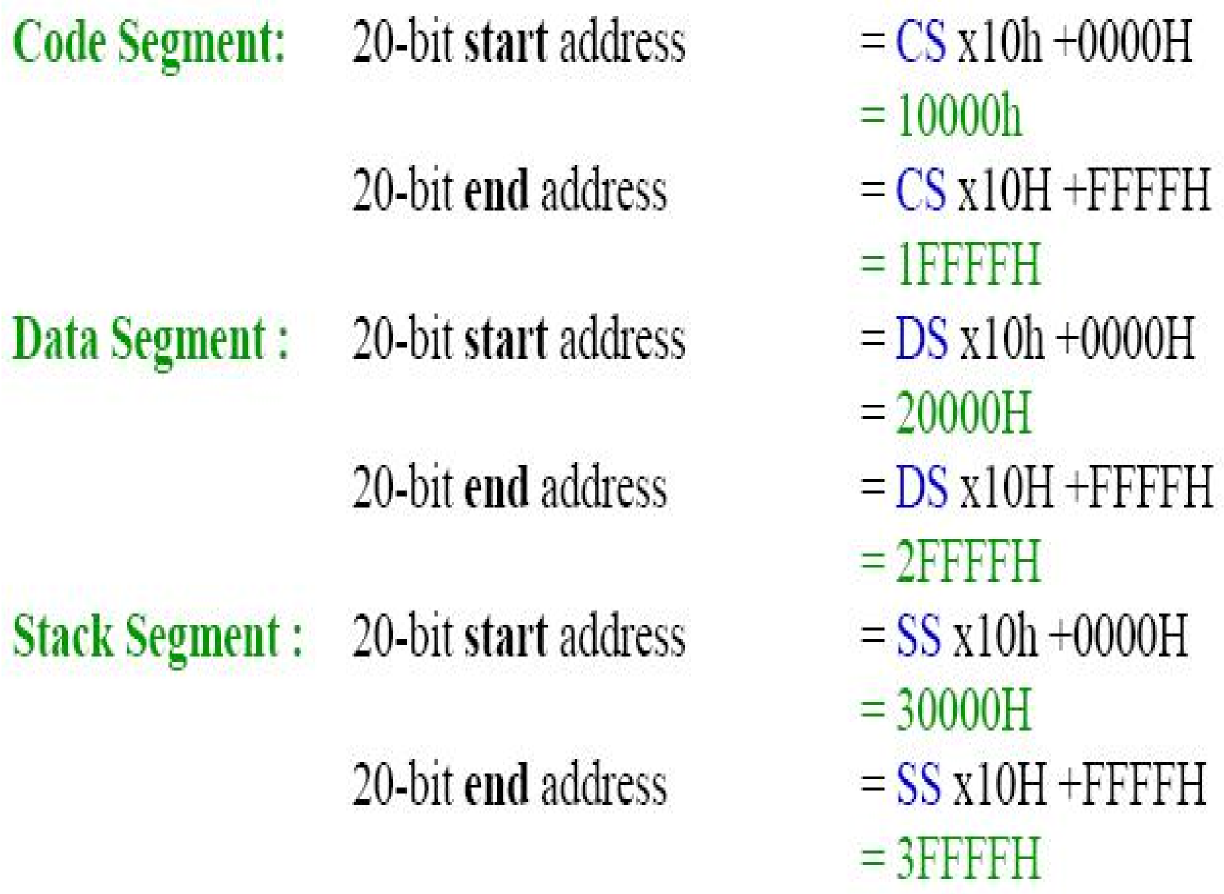
- Consider the Execution of the instruction MOV AX,[BX]
- Let us assume that DS and BX have the values 1000H and 3000H respectively.
- To calculate the address from where the data has to be taken

DS X 10 H = 10000H Base Address of Data Segment

+ BX   = 03000H Offset Address

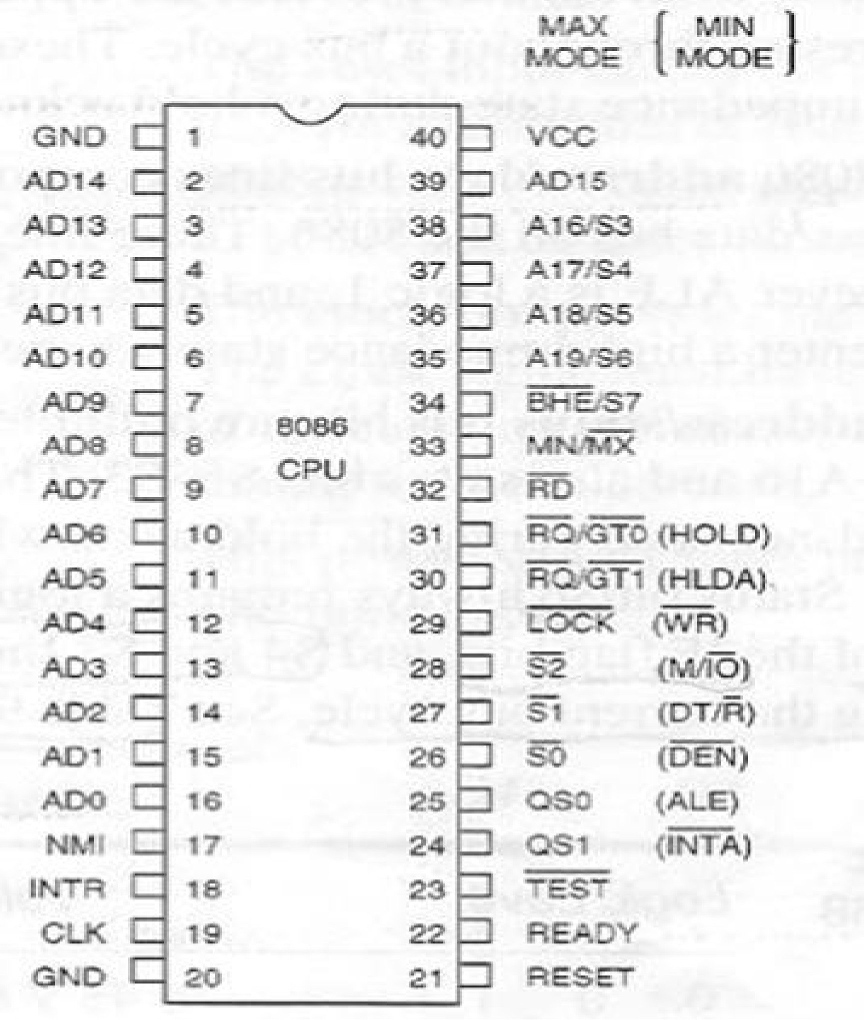
13000H Memory address from where data is to be taken

- If the segment registers CS,DS and SS have values 1000H , 2000H, 3000H respectively. What will be the 20 bit start and end address of the code , data and Stack Segments？



Code Segment:	20-bit **start** address	= CS x10h +0000H
		= 10000h
	20-bit **end** address	= CS x10H +FFFFH
		= 1FFFFH
Data Segment :	20-bit **start** address	= DS x10h +0000H
		= 20000H
	20-bit **end** address	= DS x10H +FFFFH
		= 2FFFFH
Stack Segment :	20-bit **start** address	= SS x10h +0000H
		= 30000H
	20-bit **end** address	= SS x10H +FFFFH
		= 3FFFFH

8086 Pin Diagram

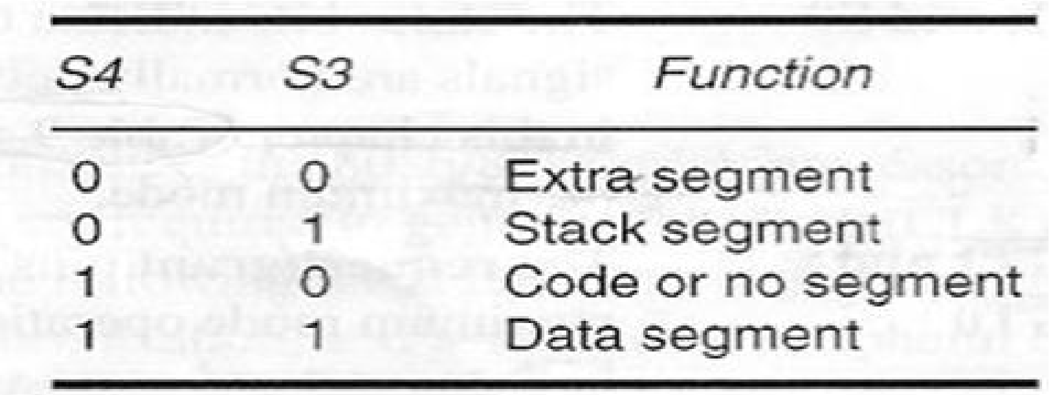


Pin Description

- Data Bus (AD0 – AD15): D0-D15
  - These 16 pins are used as Address and data bus. During T1 state this lines provides address and data lines are valid only during T2 to T4.
  - Whenever the ALE pin is high these pins works as a address bus and when ALE is low these pins carry the data.

Pin Description

- Address bus (AD0 – AD 15, A16/S3 – A19/S6):
  - This 20 lines are correspond to the CPU’s 20-bit address. These lines are valid only during T1 state.
  - S6 : always remains at logic 0
  - S5 : indicate condition of IF flag bit
  - S4 and S3 indicate the segment addressed by 8086 during the cu



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

- **NMI :** Non- Maskable interrupt is a hardware interrupt.
  - similar to INTR except that no check IF flag bit

- **RESET :**
  - This input causes the 8086 to reset, if it is held at logic 1 for at least 4 clock cycle. Whenever the 8086 is RESET, CS and IP are initialized to FFFFh and 0000H, respectively and all other registers are initialized to 0000h.

- **VCC(power supply) :** +5.0V, ±10%

- **INTR-Interrupt Request :**
  - This is a level 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.
  - When IF = 1 and if INTR is held high the 8086 gets interrupted . When IF = 0 , INTR is disabled.

- Ready:
  - This input signal is used to insert wait state in to the timing cycle of 8086. if the READY pin is at logic 1, it has no effect on operation of the microprocessor. If it is at logic 0, the 8086 enters the wait state and remains idle. This signal is used to interface the slowly interfacing device with the 8086.
- MN/MX:
  - This pin is used to select either the minimum mode or maximum mode operation for the 8086. this is achieved by connecting this pin to either +5V (Minimum mode) or to the

- $\overline{\text{TEST}}$:
  - This pin is an input pin that is tested by wait instruction. If this pin is at logic 0 , the wait instruction function as NOP. This pin is often connected to the BUSY pin of the 8087 to perform the floating point operations.

- Read (RD):
  - This active low output signal indicates that the direction of data flow from memory or IO to CPU. It can be combined with M/IO to generate MEMR and IOR signals.

Pin Description

- Clock (CLK):
  - All events in the microprocessor are synchronizes to the system clock applied to CLK pin. The clock signal must ~~have~~ duty cycle for 33%.

- BHE / S7:
  - This signal is multiplexed with the S7 status indicator. It is output only during the T1 state. BHE and A0 are typically used to select even or odd memory banks.

BHE	A0	Action
0	0	Access 16-bit word
0	1	Access odd byte to D8- D15
1	0	Access even byte to D0-D7
1	1	No action

Function of Pins used in Minimum mode

- $\overline{\text{INTA}}$ –Interrupt Acknowledge :
  - This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

- DT/R –Data Transmit/Receive:
  - This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

Pin Description

- ALE (Address Latch Enable):
  - Signal output on this pin can be used to Demultiplex the address , data bus and status lines. ALE pulse is high during T1 state.

- M/IO :
  - The 8086 does not output separate memory and I/O signals. Instead , the M/IO signal output during early in T1 state. So if M/IO = 1 then it is memory operation or M/IO = 0 then IO operation.

Pin Description

- Write($\overline{WR}$):
  - This active low output signal indicates that the direction of data flow from CPU to memory or IO. It can be combined with M/IO to generate MEMW and IOW signals.

- $\overline{\text{DEN}}$ –Data Enable :
  - This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle ofT4. This is tristated during ‘ hold acknowledge’ cycle.

- HOLD, HLDA-Acknowledge :
  - When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus cycle.

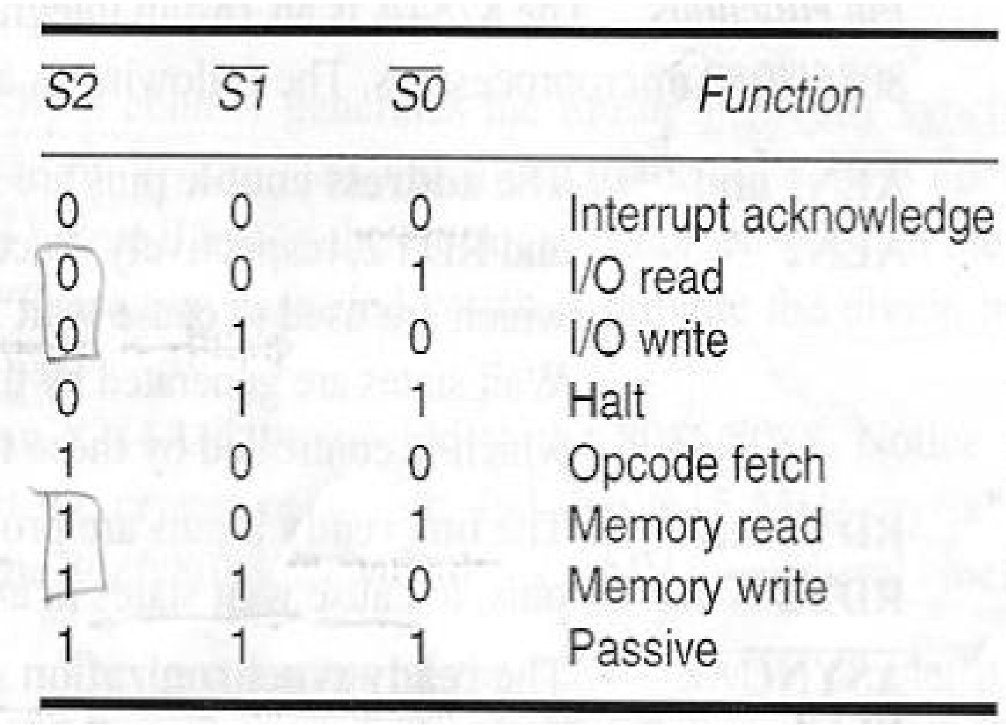
- GND(Ground) : GND

- Minimum mode Pins

- M/IO
- WR
- INTA
- HOLD
- HLDA
- DEN
- ALE

Maximum Mode Pins

- MN/$\overline{MX}$ = 0(ground)
- S2’,S1’,S0’:indicate function of current bus cycle
  - these signal : normally decoded by 8288 bus controller



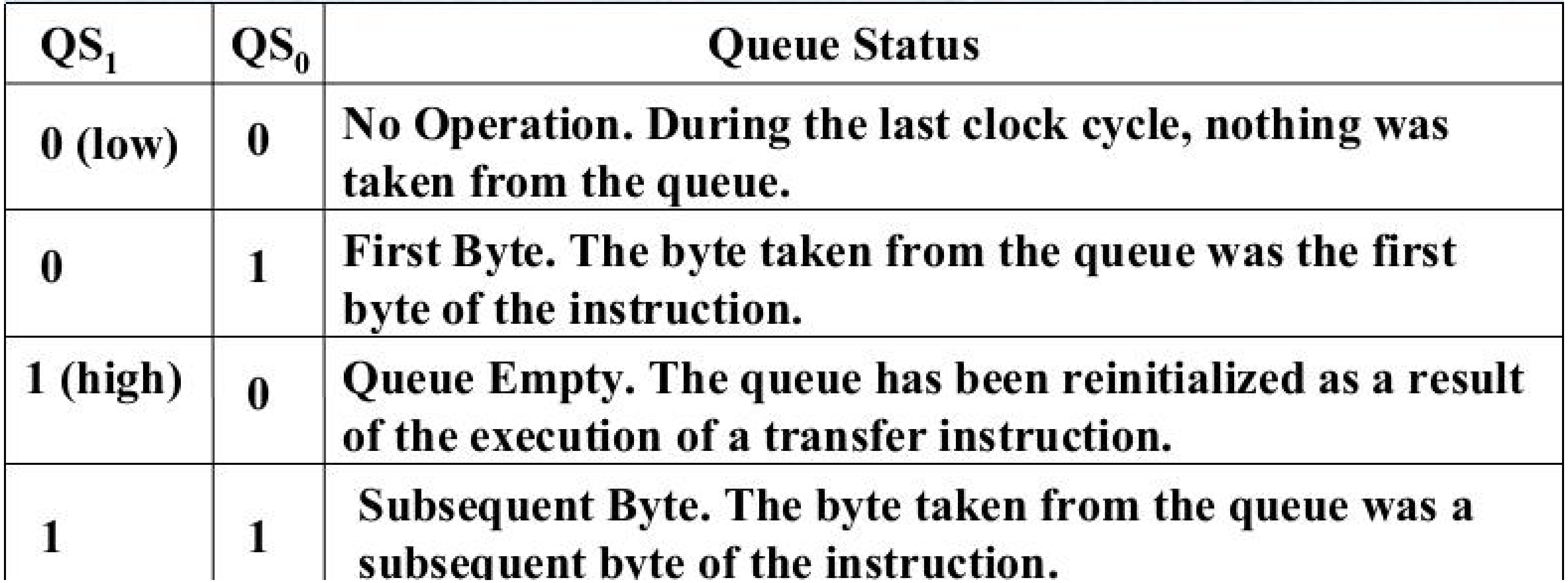
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Function
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

- Request / Grant ($\overline{RQ0}/\overline{GT0}$, $\overline{RQ1}/\overline{GT1}$) :

  o These two pins are bidirectional, allowing a coprocessor to request control of the system buses . The 8086 respond by disconnecting itself from the system buses and pulsating the RQ/GT line in acknowledgement. These lines are bidirectional and are used to request and grant DMA operation.

- $\overline{\text{LOCK}}$(lock output):
  - lock is an output signal intended for use in a bus arbitration scheme with another processor. Arbitration refers to the process of determining which processor should have the control of the system buses at any given time. The LOCK signal is output low during the execution of any instruction with LOCK prefix. **This signal is meant to be output whenever processor wants to lock out other processor from using the bus.**

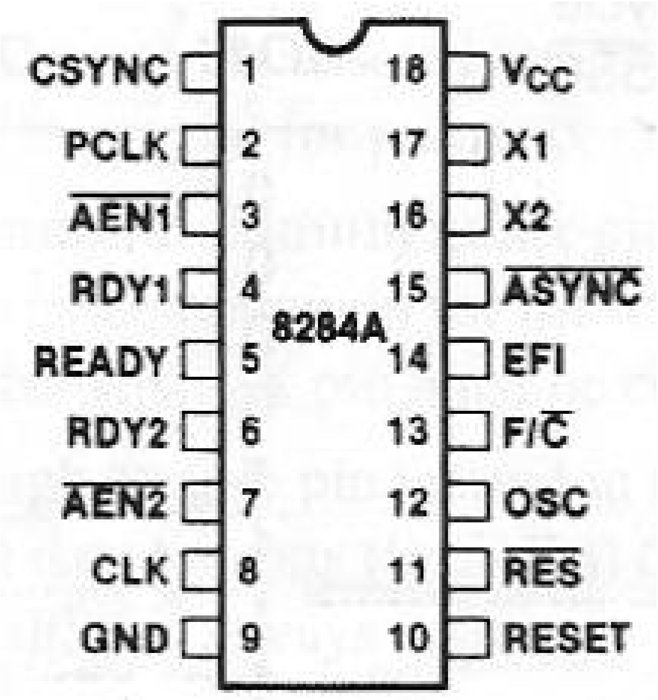
- QS1, QS0(queue status) :
  - show status of internal instruction queue
  - provided for access by the numeric coprocessor(8087)
  - They allow the coprocessor to track the progress of an instruction through the queue



QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

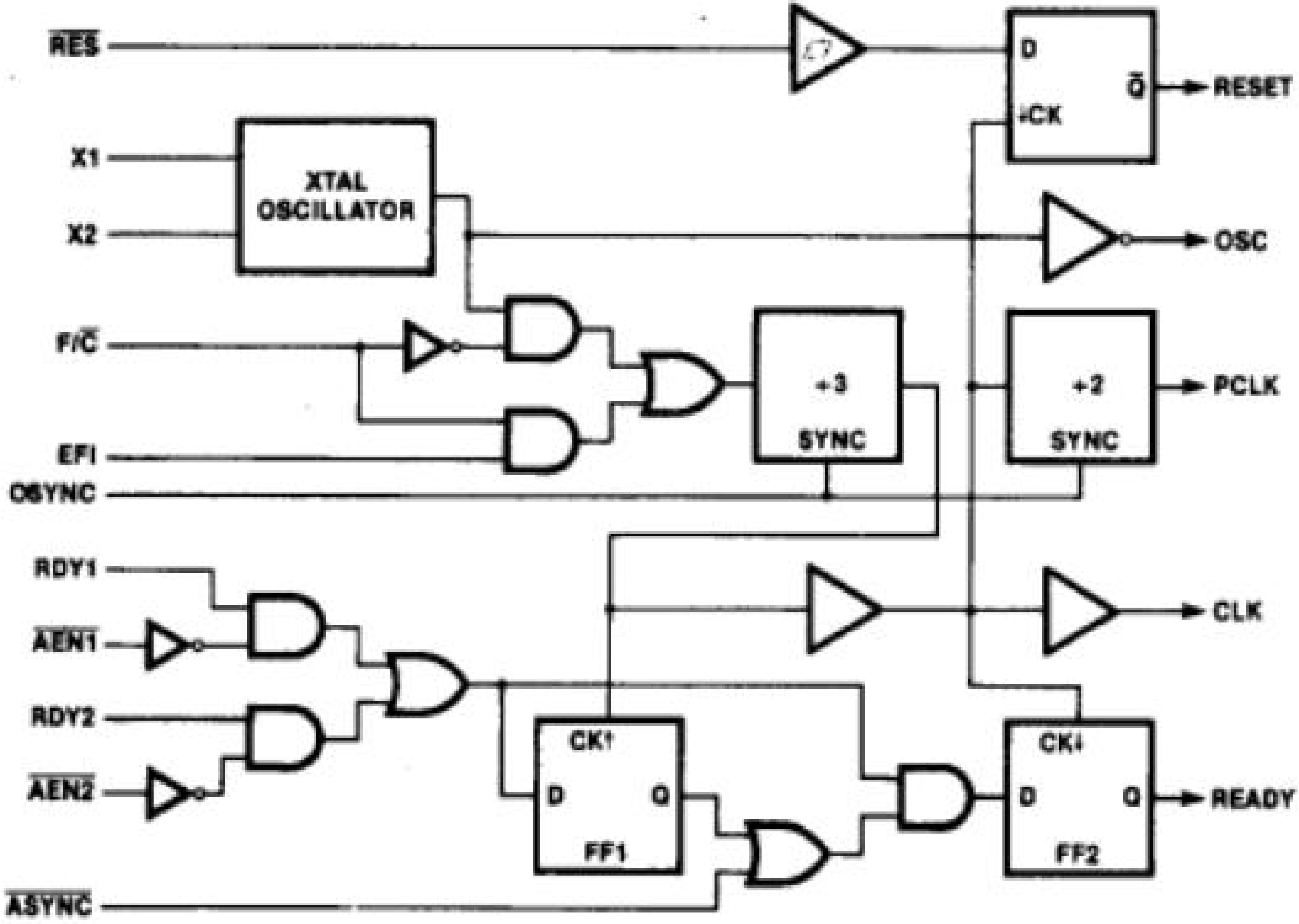
8284A

clock generation, RESET synchronization, READY synchronization, and TTL-level peripheral clock signal

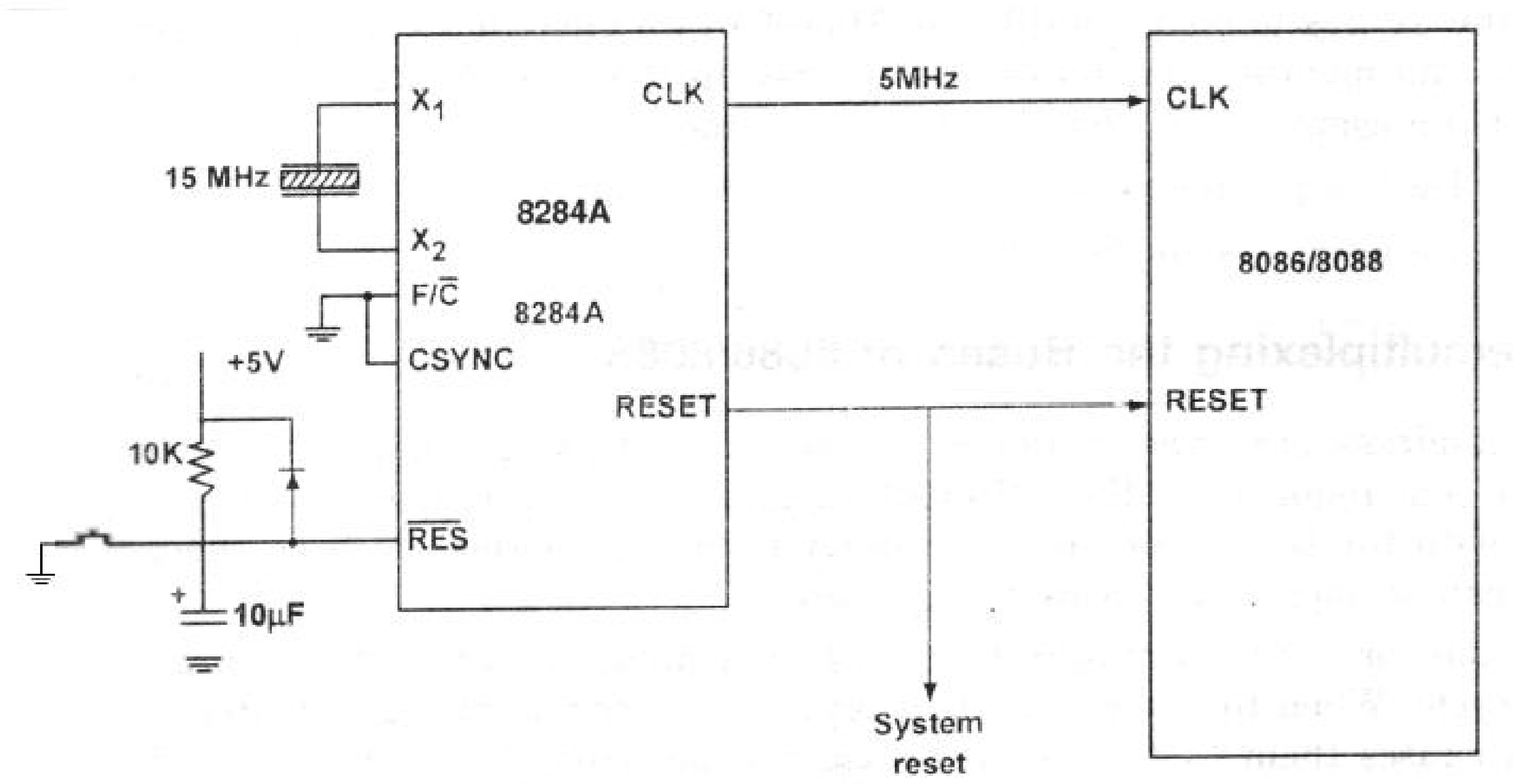


Pin Diagram of 8284A Clock generator

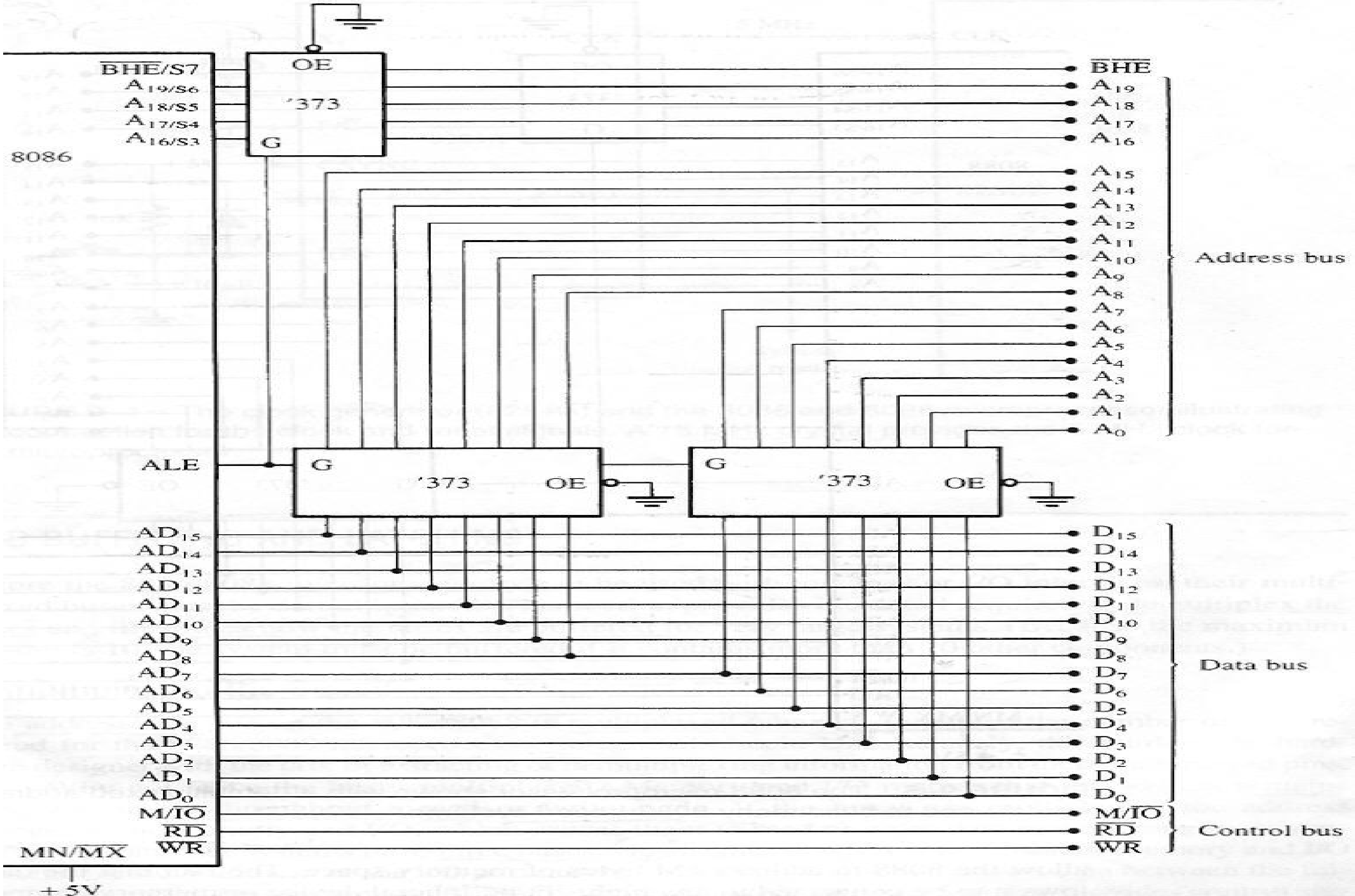
8284A


RES
X1
X2
XTAL OSCILLATOR
F/C
EFI
OSYNC
RDY1
AEN1
RDY2
AEN2
ASYNC
D
Q
RESET
CK
OSC
÷3 SYNC
÷2 SYNC
PCLK
CLK
CK↑
D
Q
FF1
CK↓
D
Q
FF2
READY

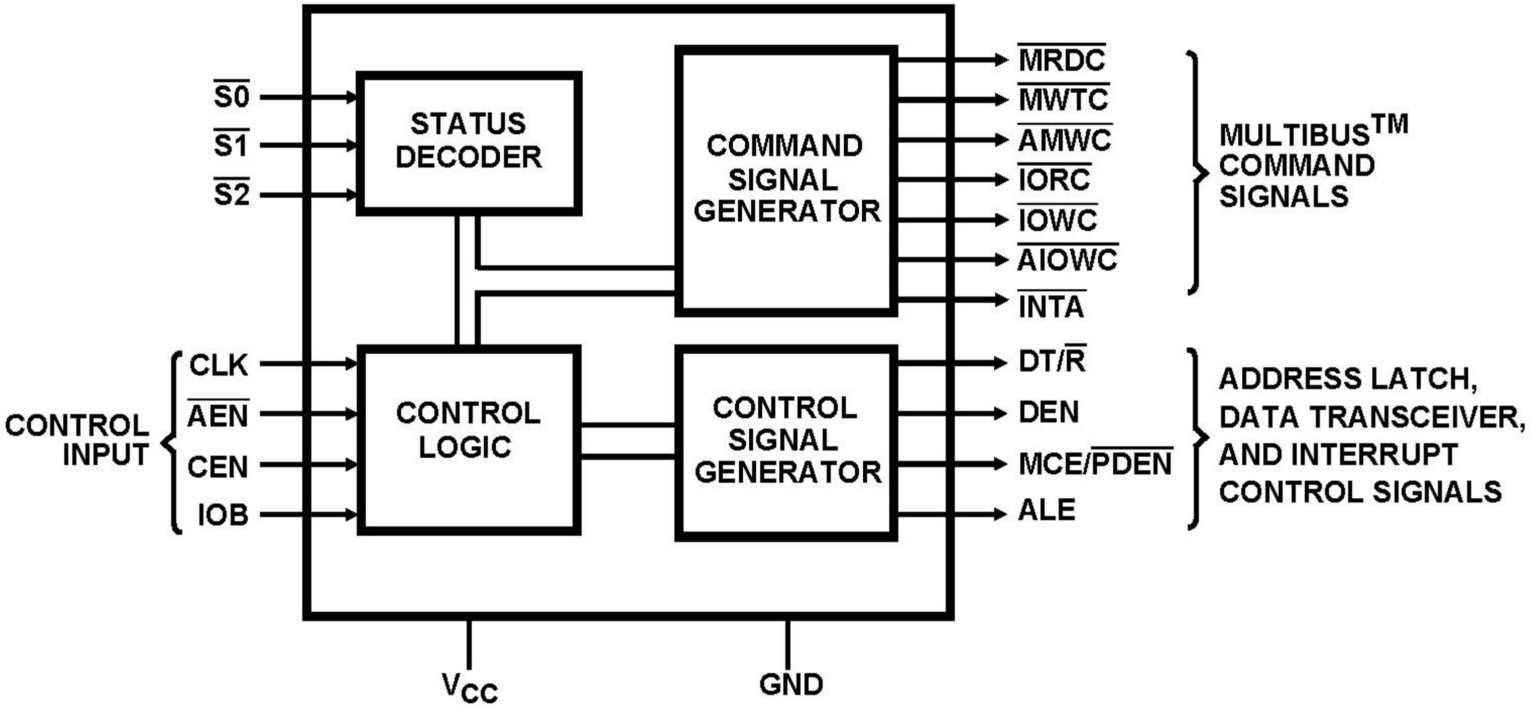
8284A connection with 8086



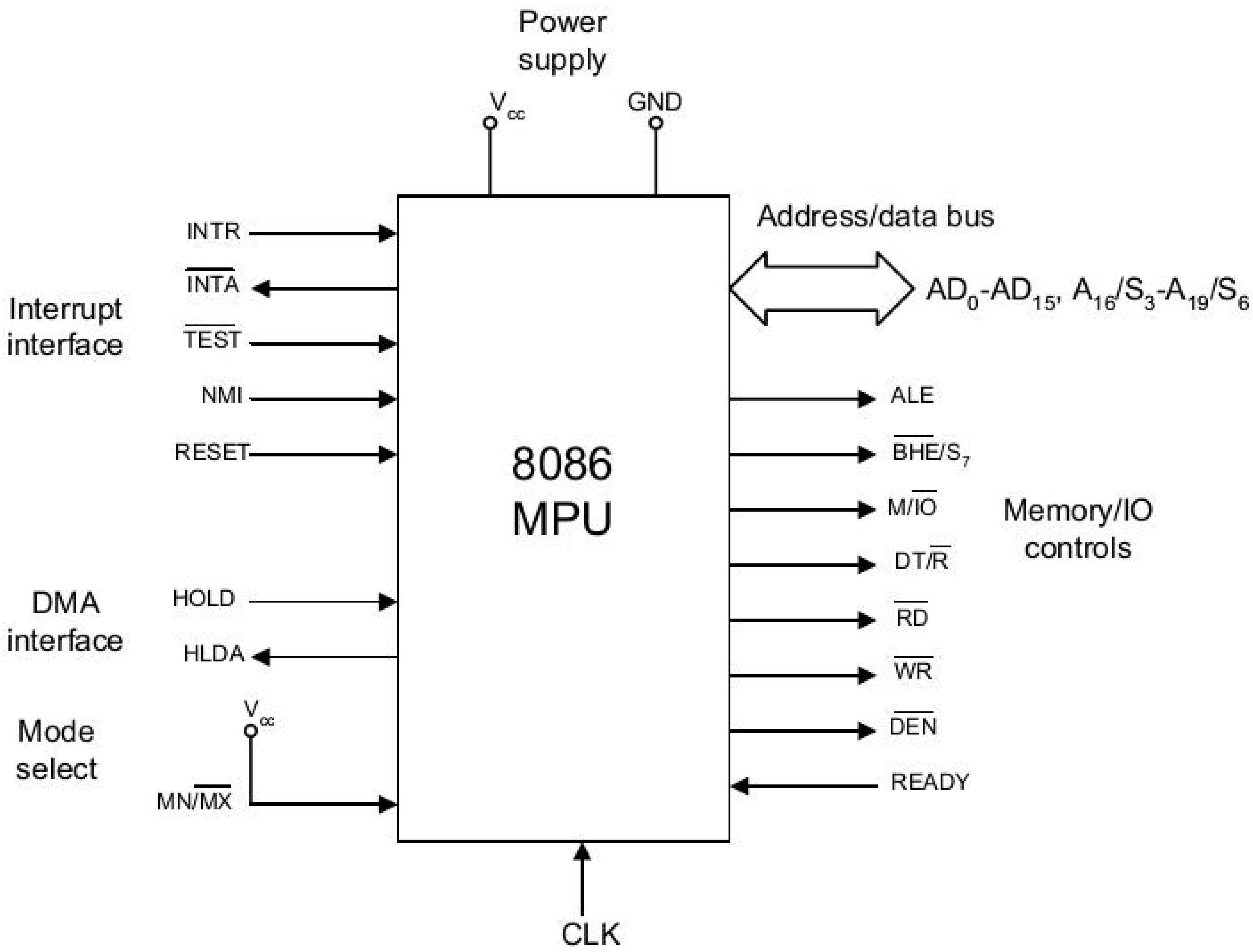
Bus Buffering and Latching



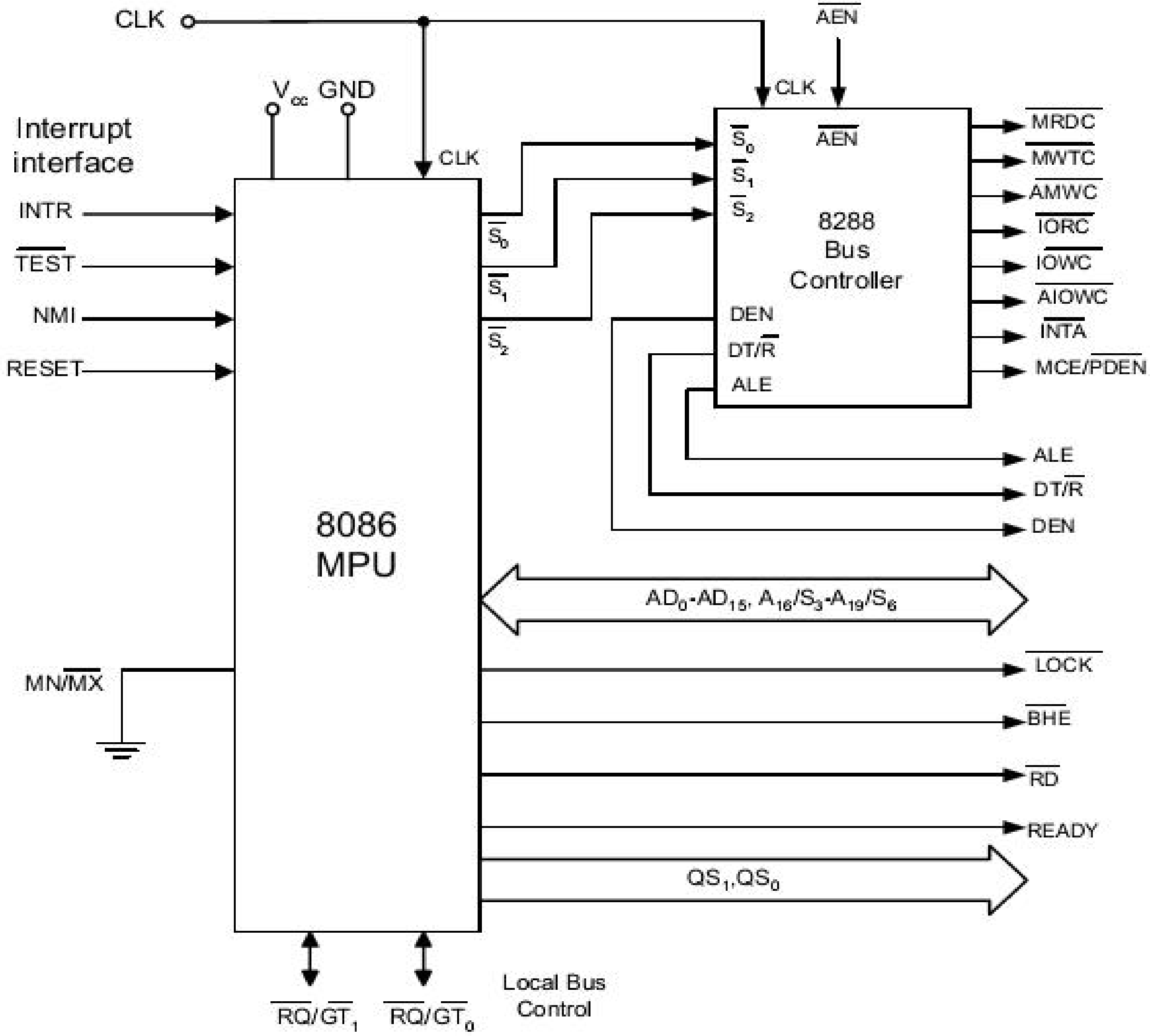
The 8288 Bus Controller



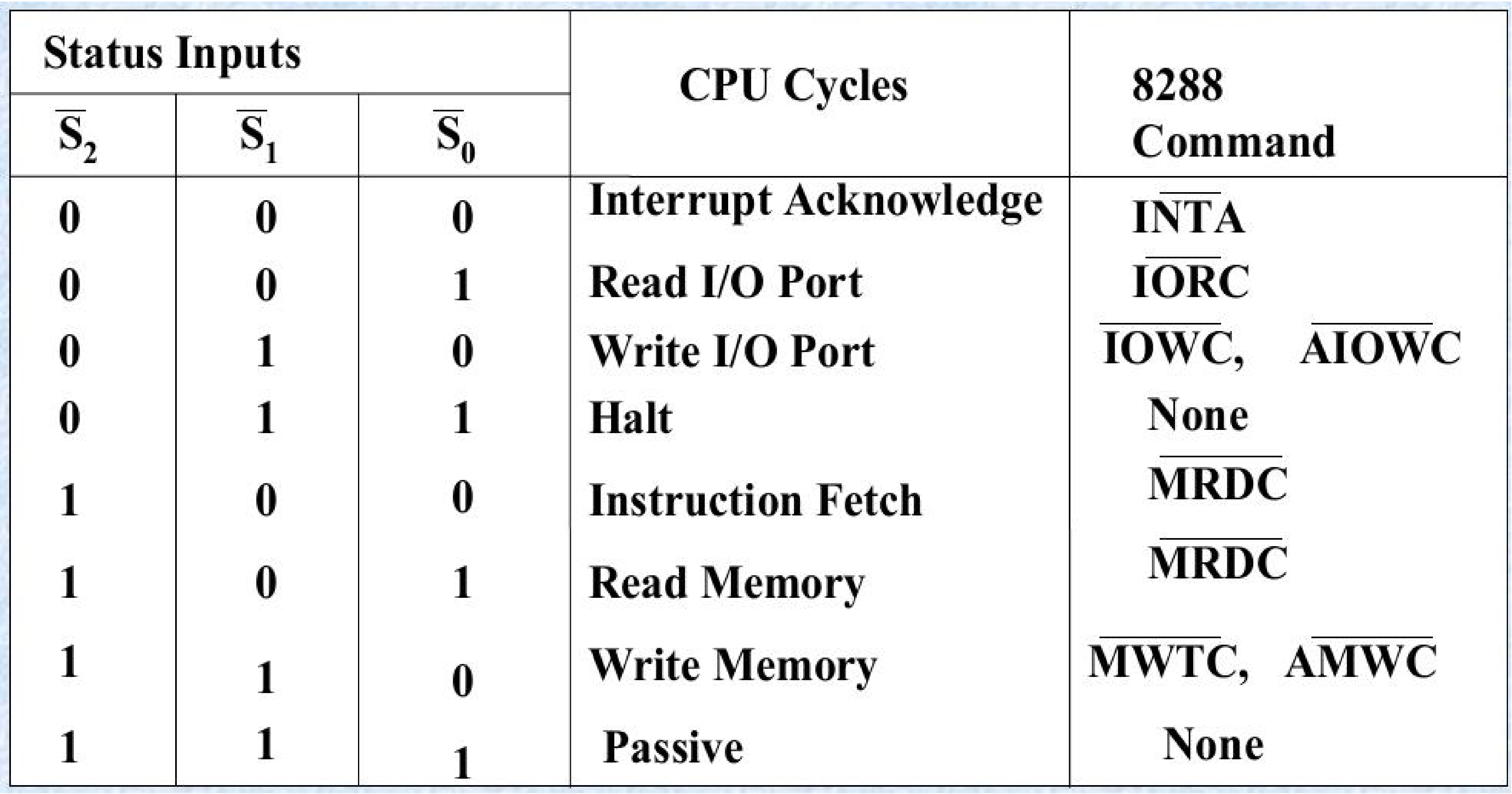
Minimum mode Interface



Maximum mode Interface

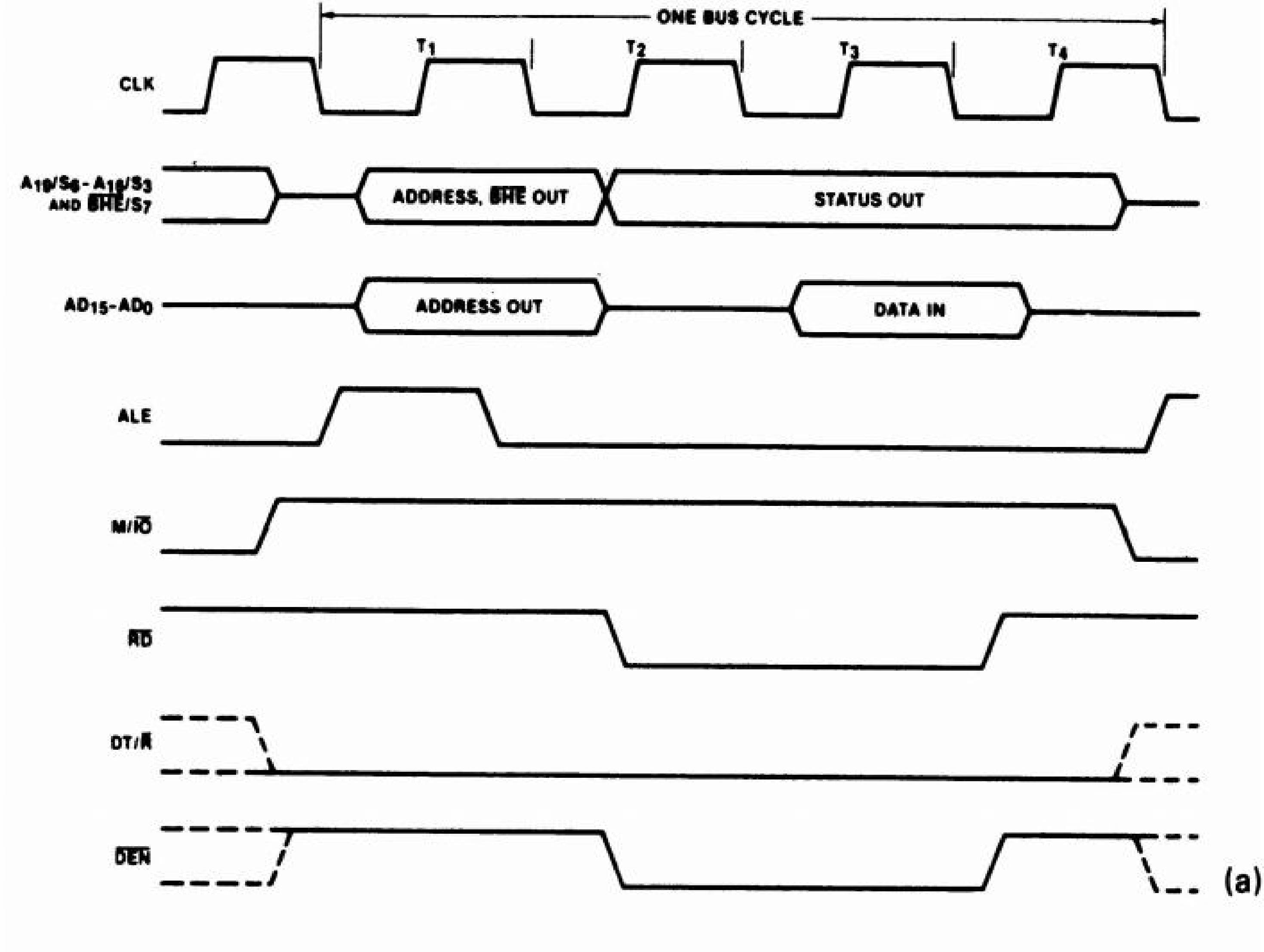


8288 Commands

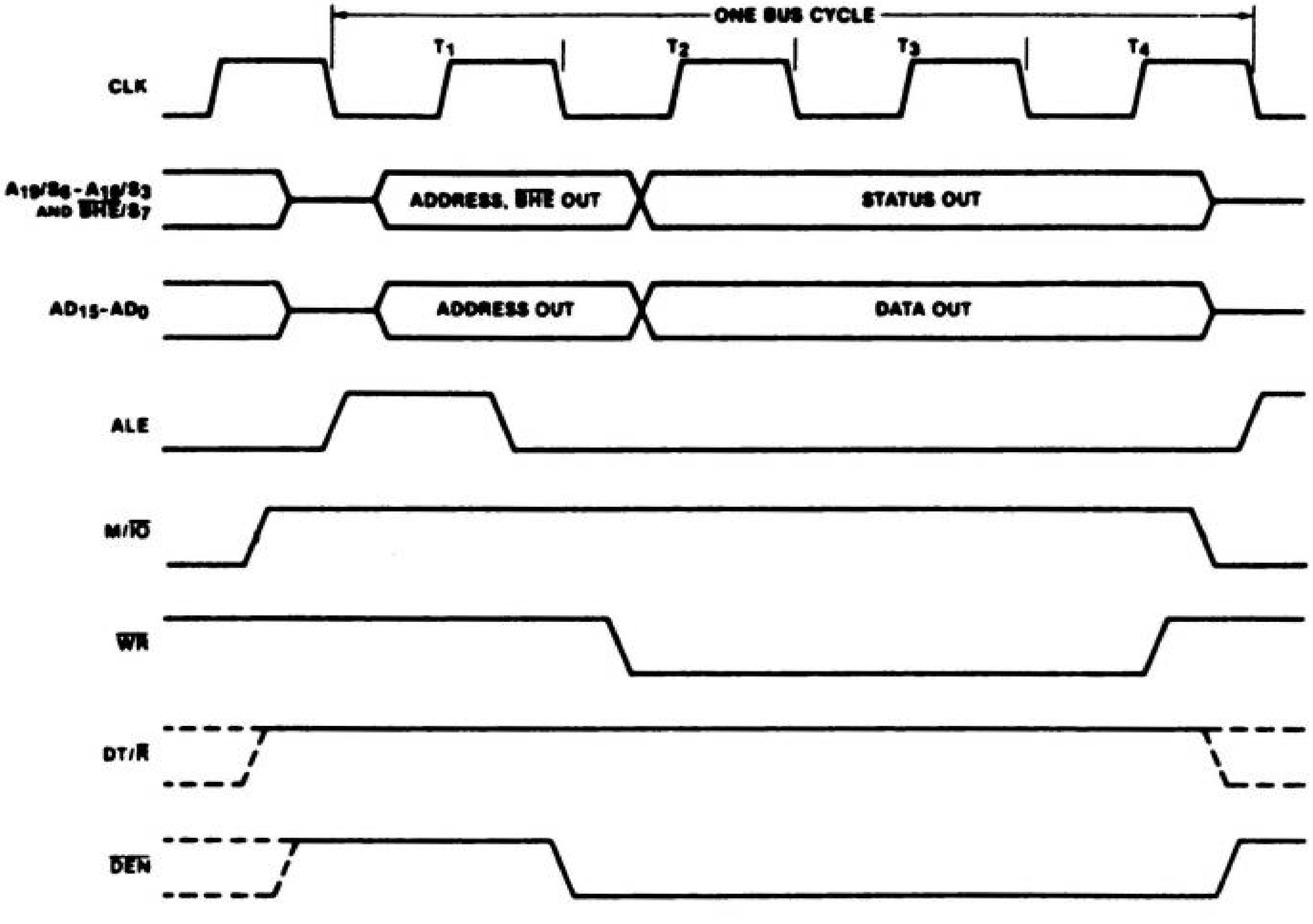


Status Inputs			CPU Cycles	8288 Command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

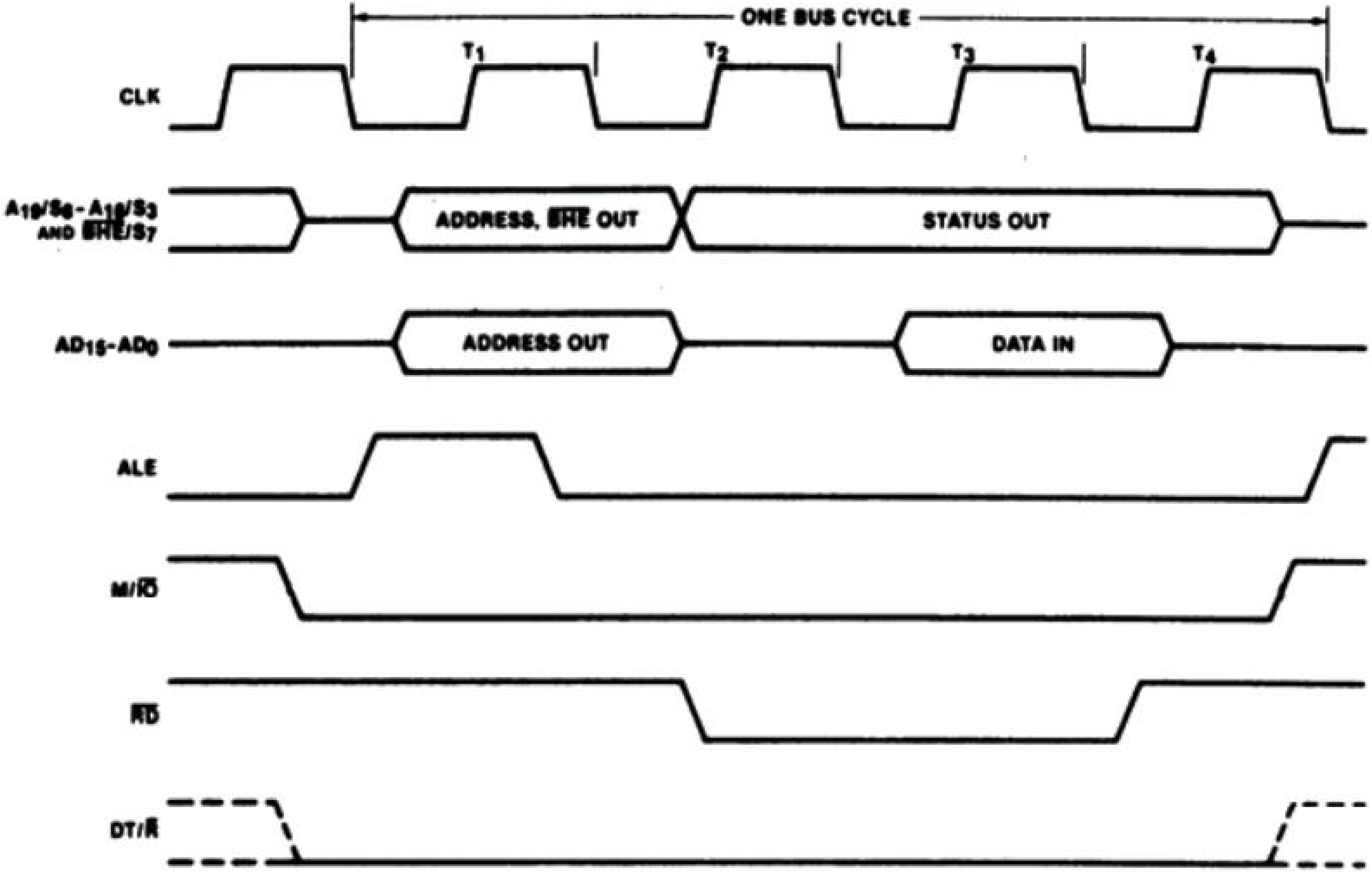
Memory Read cycle Minimum mode



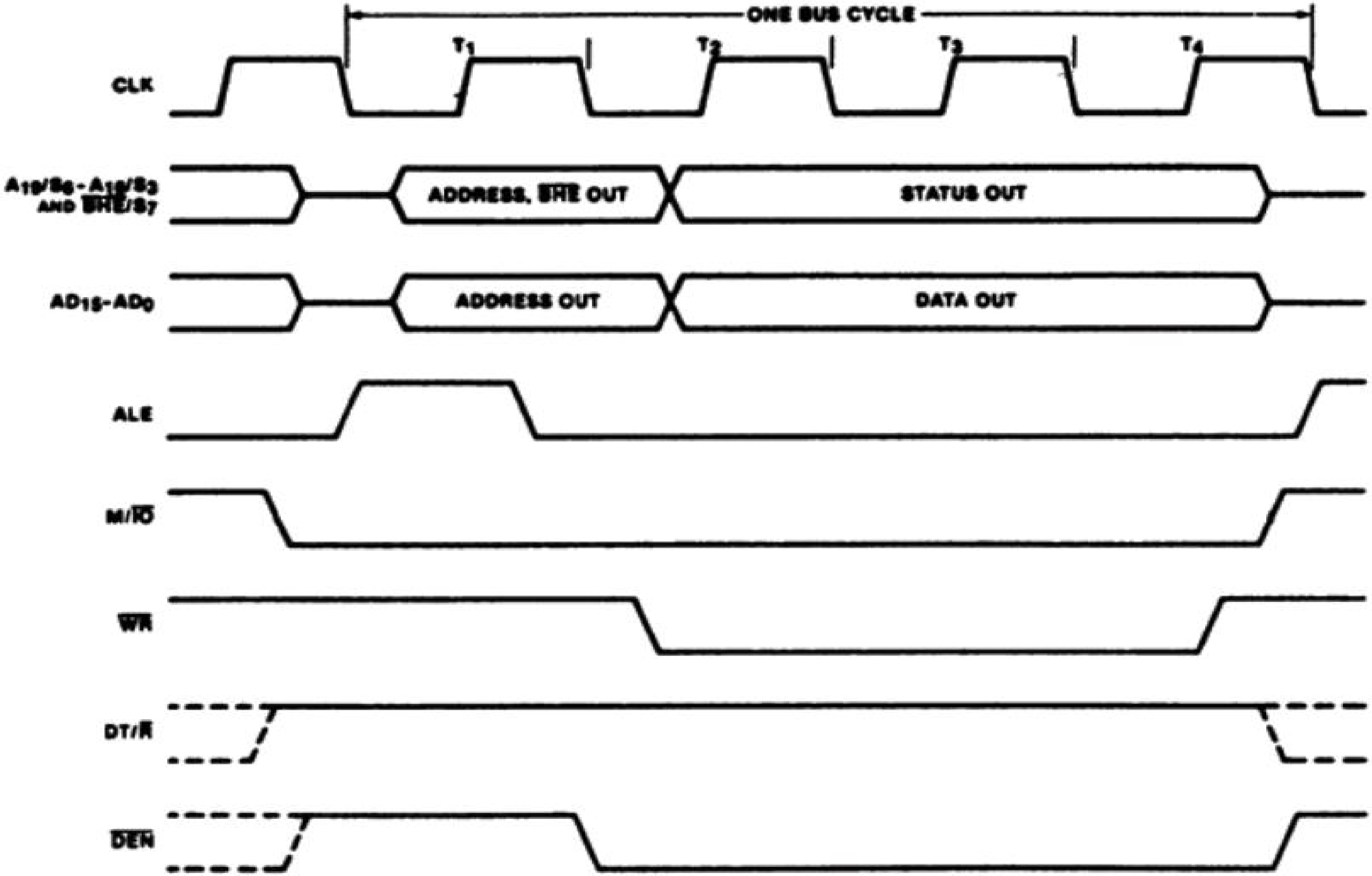
Memory write cycle Minimum mode



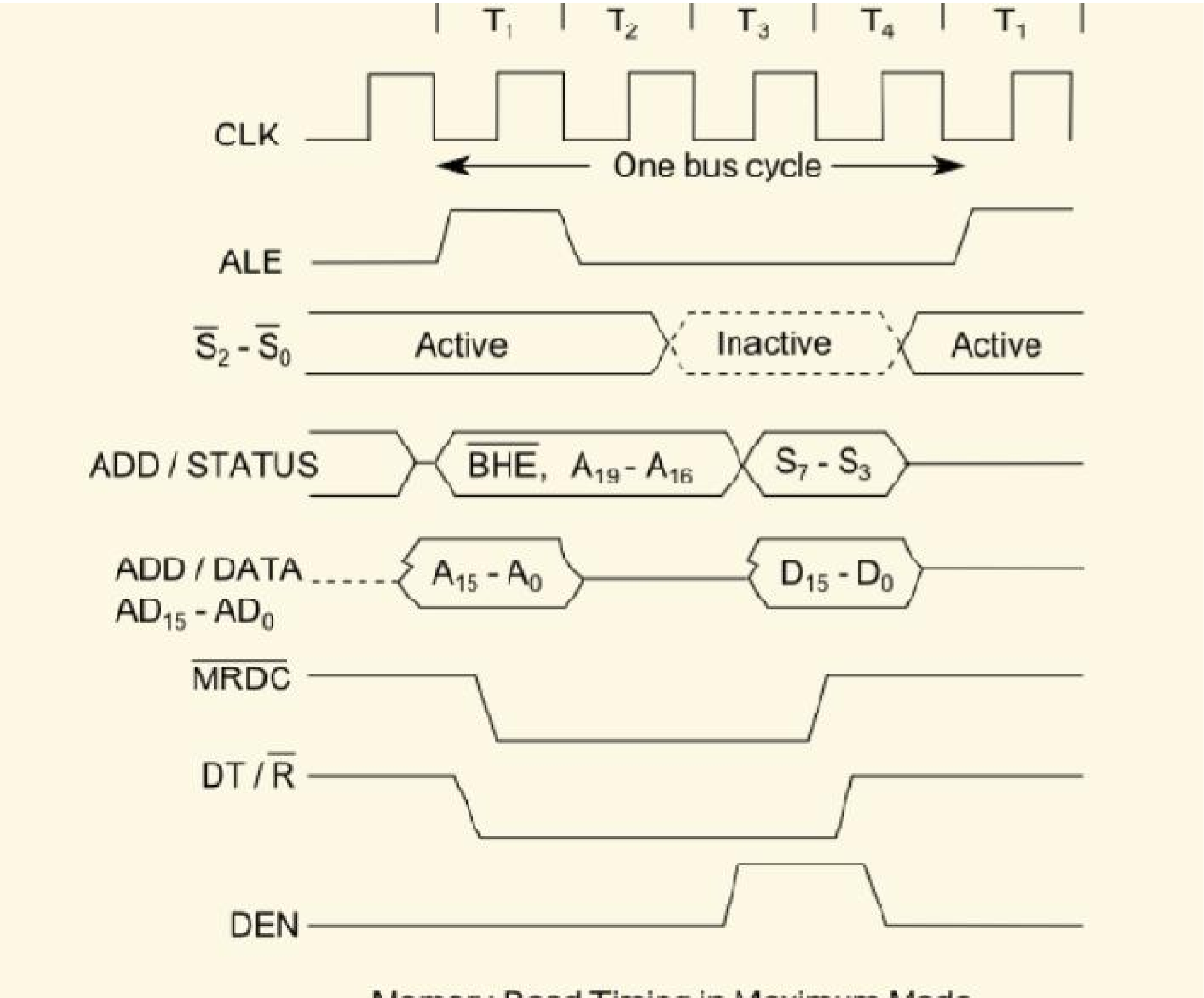
Minimum-mode I/O read cycle



Minimum-mode I/O write cycle

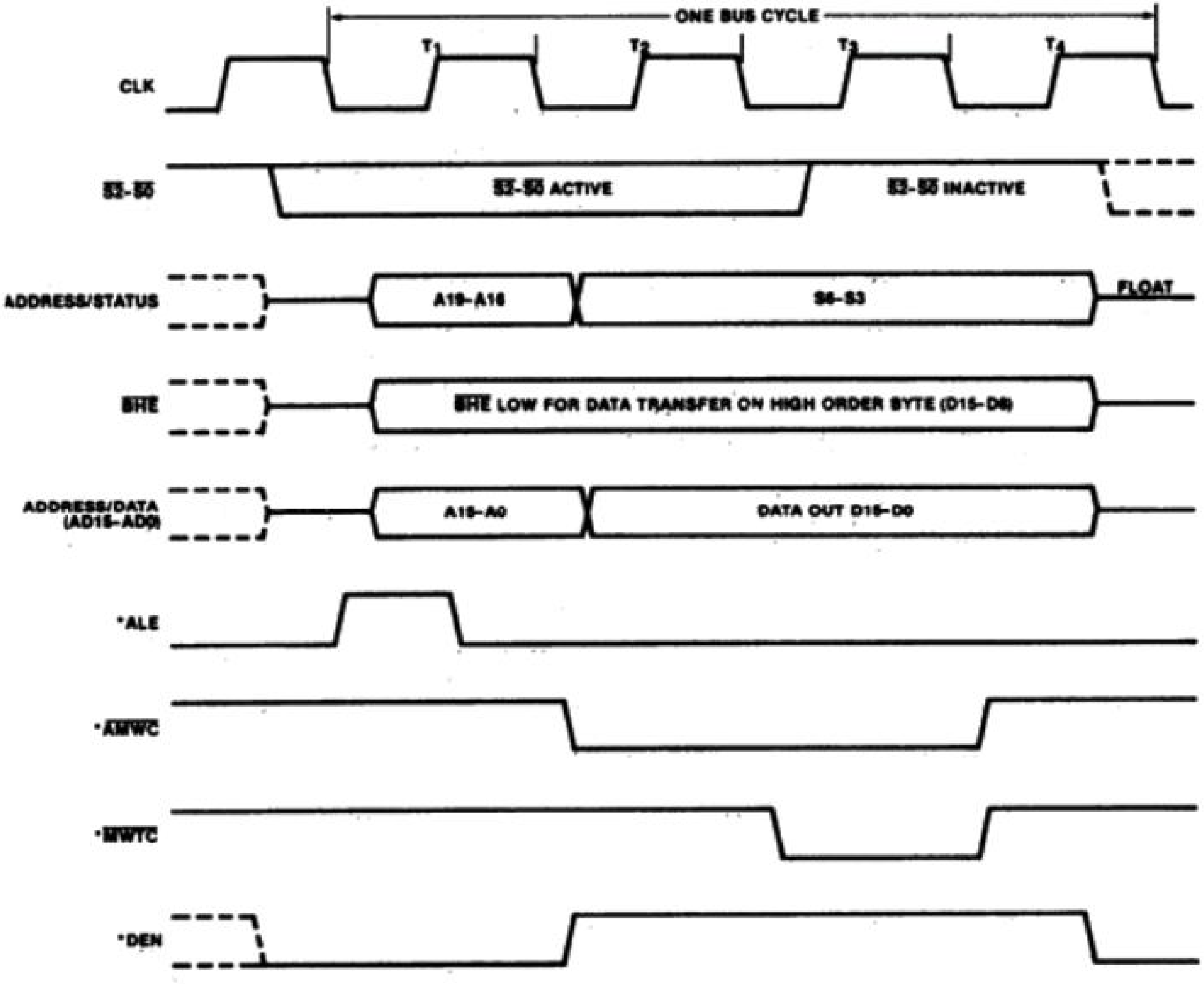


Maximum mode memory read cycle

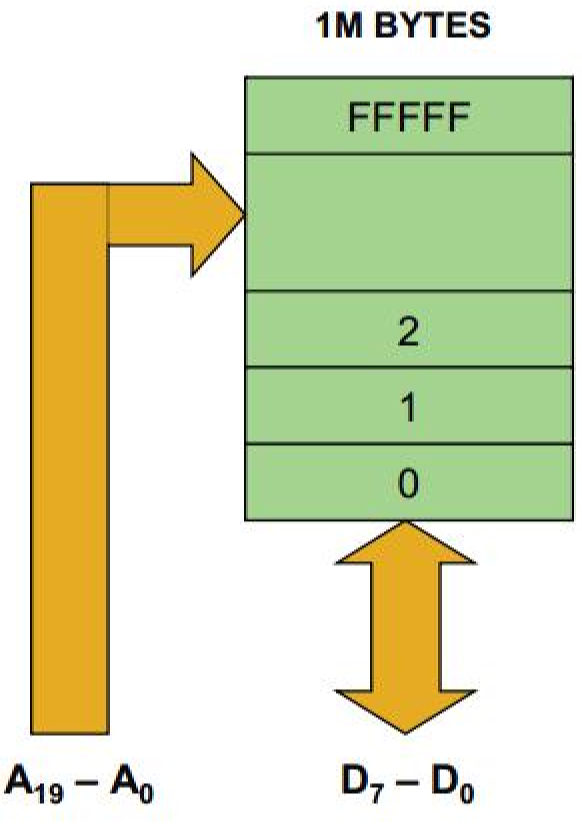


Memory Read Timing in Maximum Mode

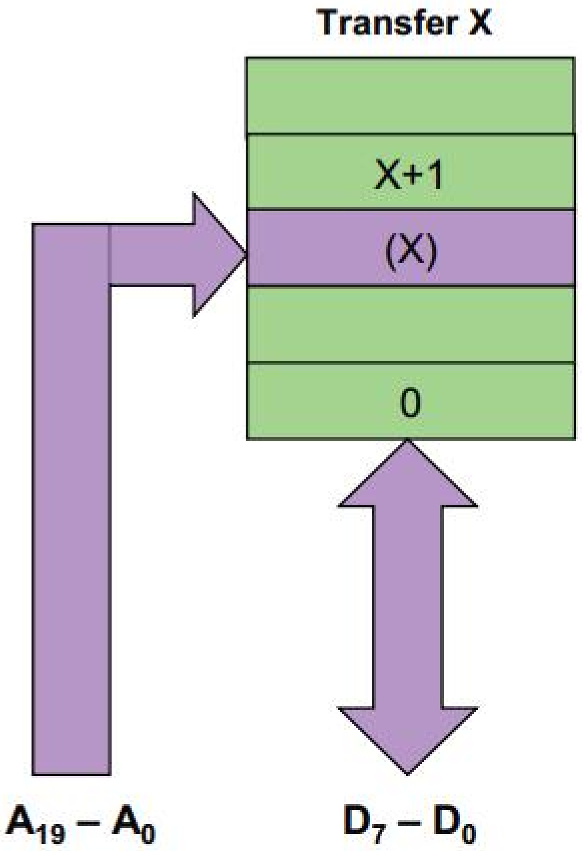
Maximum mode Memory Write Cycle



Memory Bank

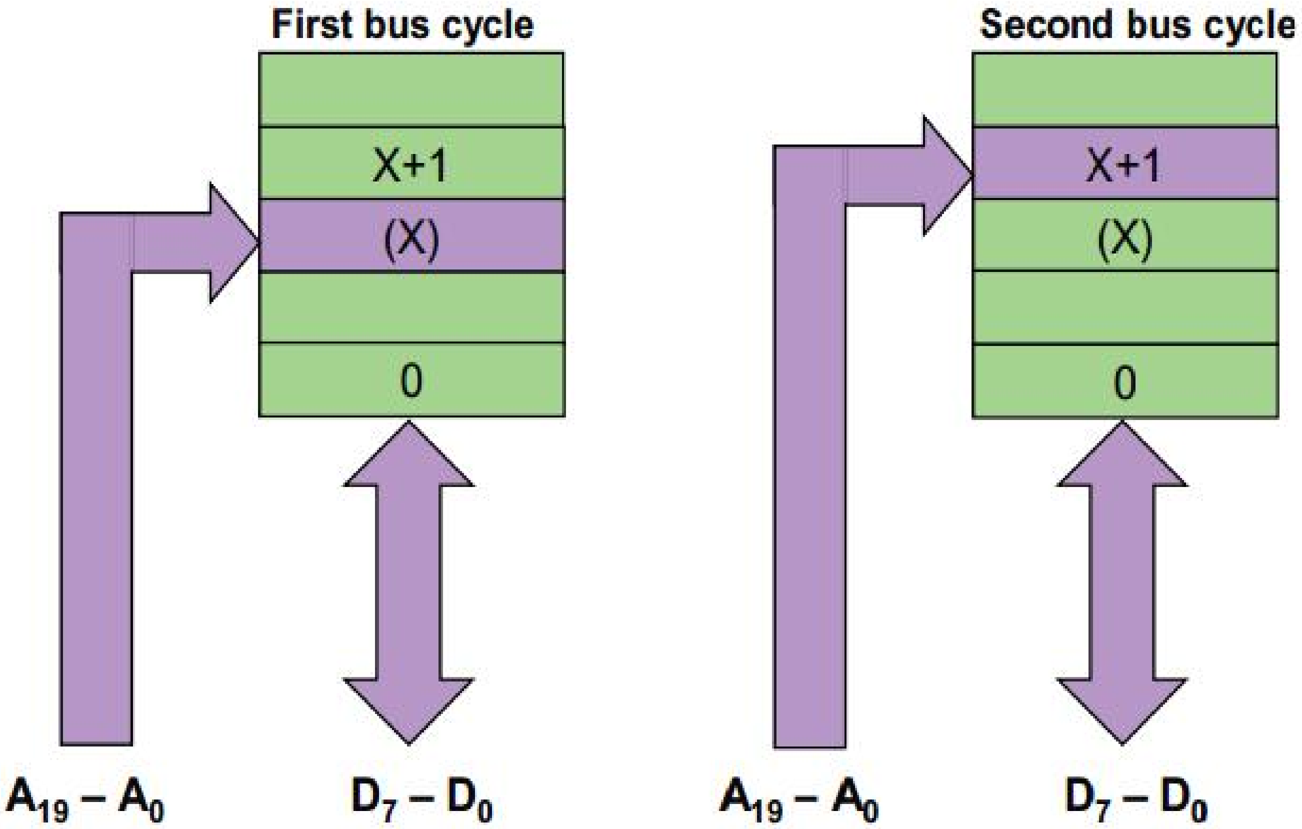


1Mx 8 Memory bank of 8088



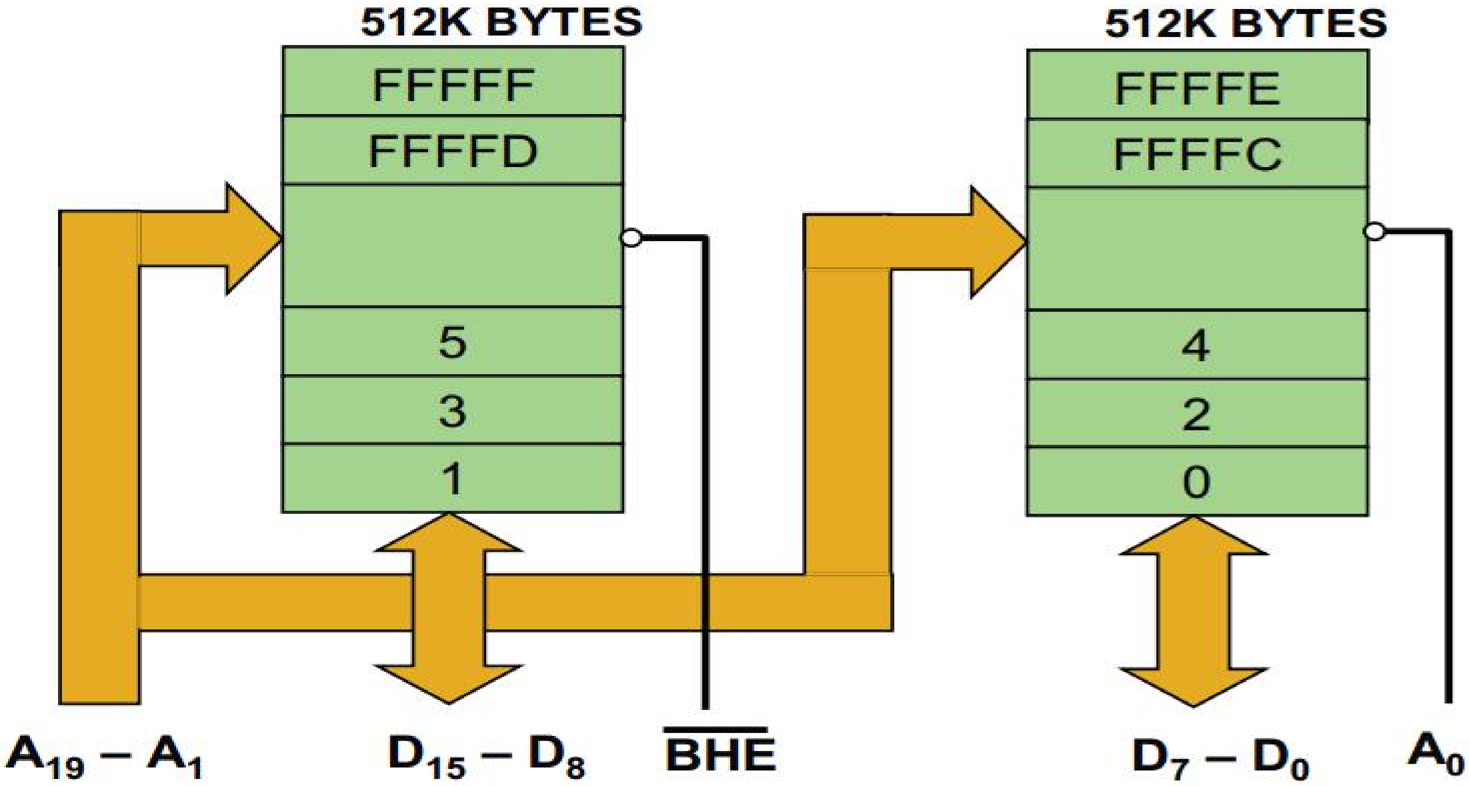
Byte transfer by 8088

Memory Bank



Word transfer by the 8088

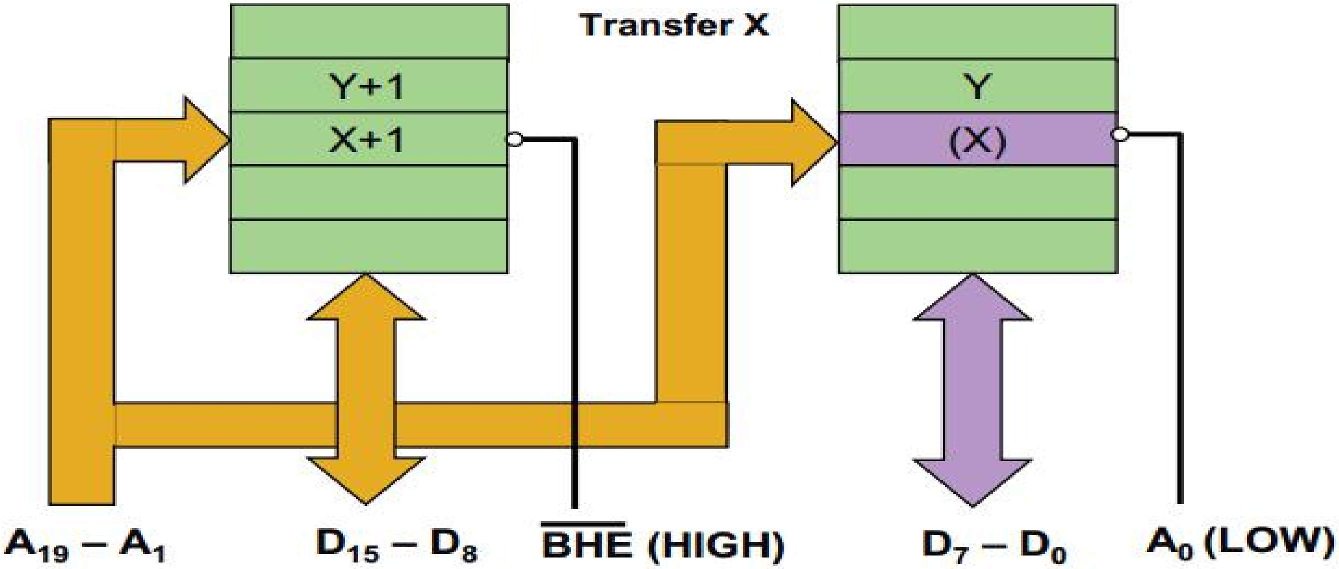
Memory Bank



High and low memory banks of the 8086

Memory Bank

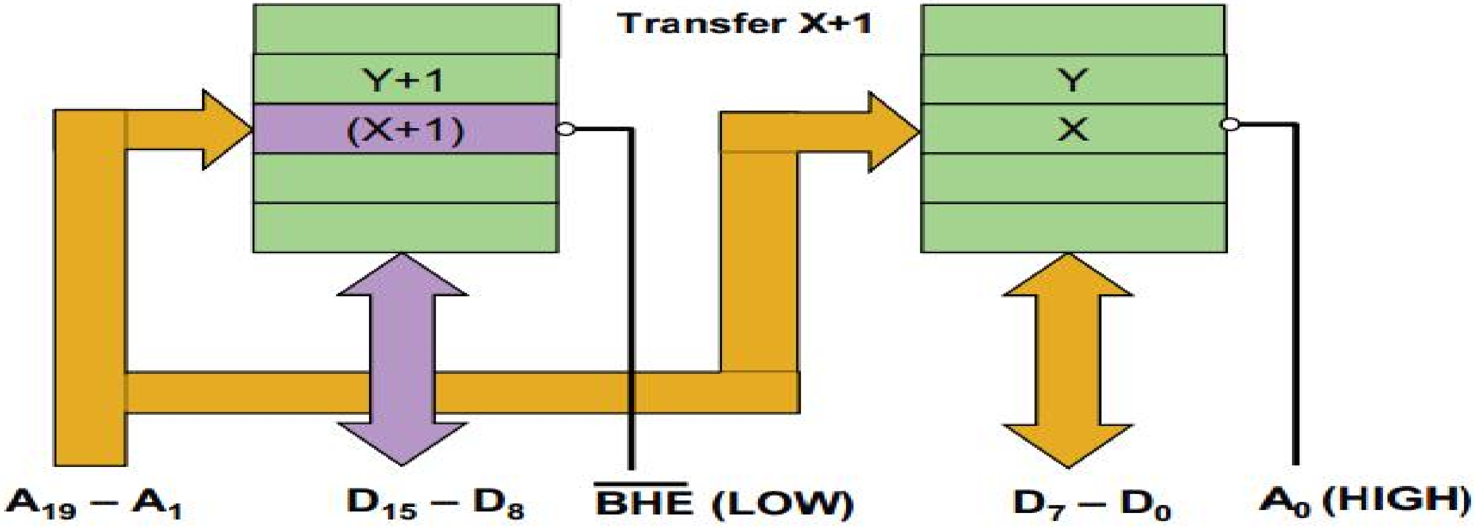
BHE	A0	Action
0	0	Access 16-bit word
0	1	Access odd byte to D8- D15
1	0	Access even byte to D0-D7
1	1	No action



Even address byte transfer by the 8086

Memory Bank

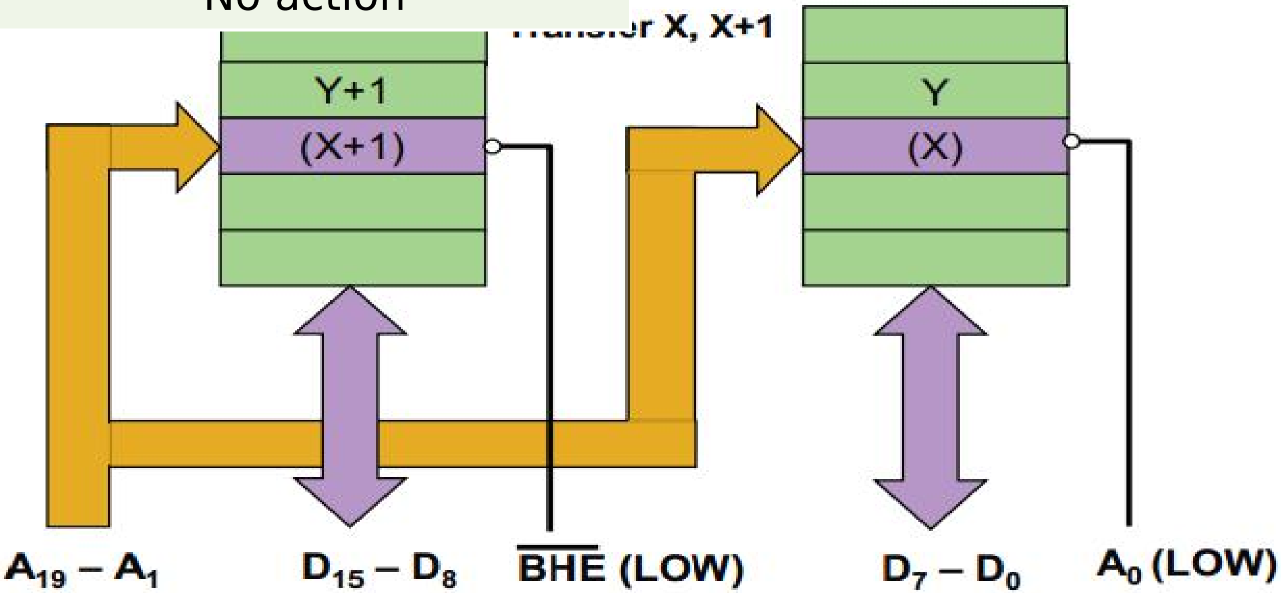
BHE	A0	Action
0	0	Access 16-bit word
0	1	Access odd byte to D8- D15
1	0	Access even byte to D0-D7
1	1	No action



Odd address byte transfer by the 8086

Memory Bank

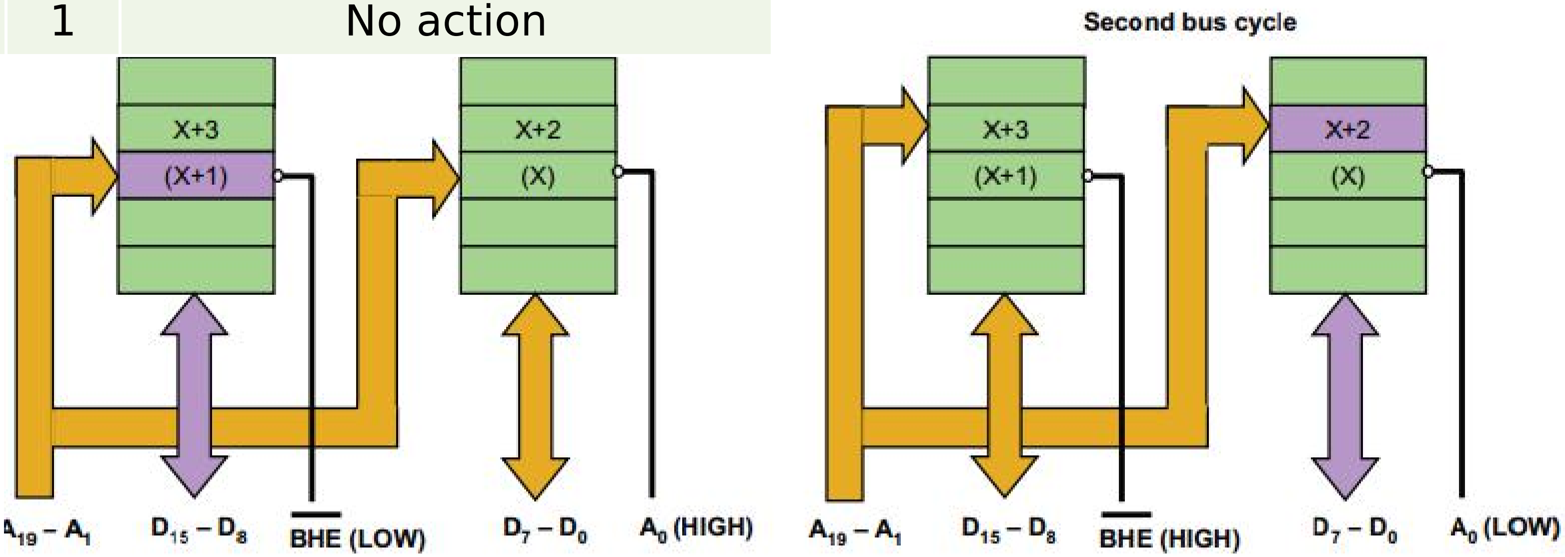
BHE	A0	Action
0	0	Access 16-bit word
0	1	Access odd byte to D8- D15
1	0	Access even byte to D0-D7
1	1	No action



Even address word transfer by the 8086

Memory bank

BHE	A0	Action
0	0	Access 16-bit word
0	1	Access odd byte to D8- D15
1	0	Access even byte to D0-D7
1	1	No action



Odd-address word transfer by the 8086

Addressing Modes

- Method of Accessing data from memory or Registers.
- Types of Addressing Modes
  - Register Addressing mode
  - Immediate Addressing mode
  - Direct Addressing Mode
  - Register Indirect Addressing Mode
  - Based-Relative Addressing mode
  - Indexed Relative Addressing mode
  - Based-indexed relative addressing mode

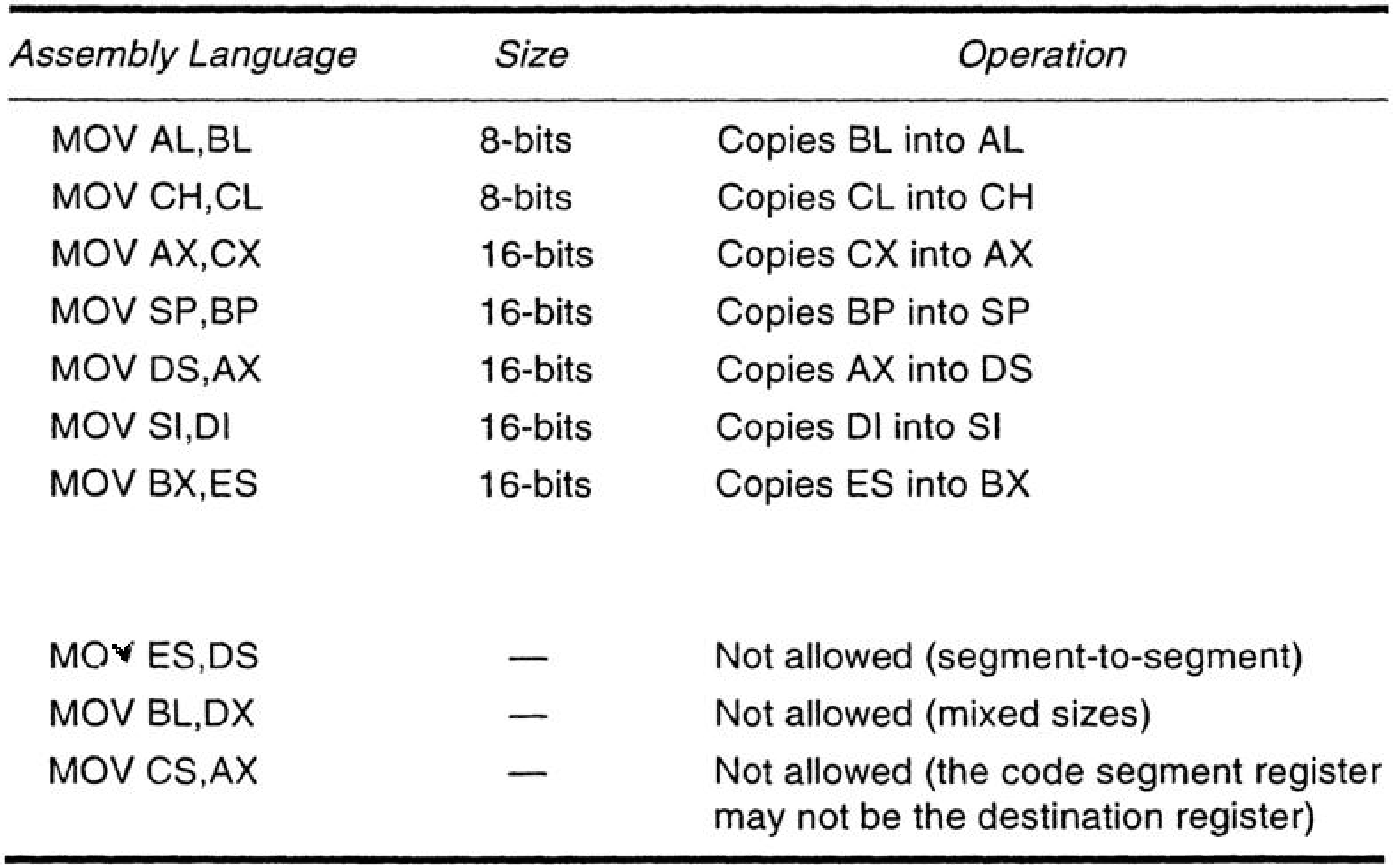
Register Addressing mode

- Transfers a copy of a byte or word from the source register or memory location to the destination register or memory location.

```
MOV BX, DX   ; copy the contents of DX into BX
MOV ES,AX    ; copy the contents of AX into ES
ADD AL,BH    ; add the contents of BH to contents of AL
```

Source and destination registers must have the same size

Register Addressing mode



Assembly Language	*Size*	*Operation*
MOV AL,BL	8-bits	Copies BL into AL
MOV CH,CL	8-bits	Copies CL into CH
MOV AX,CX	16-bits	Copies CX into AX
MOV SP,BP	16-bits	Copies BP into SP
MOV DS,AX	16-bits	Copies AX into DS
MOV SI,DI	16-bits	Copies DI into SI
MOV BX,ES	16-bits	Copies ES into BX
MOV ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

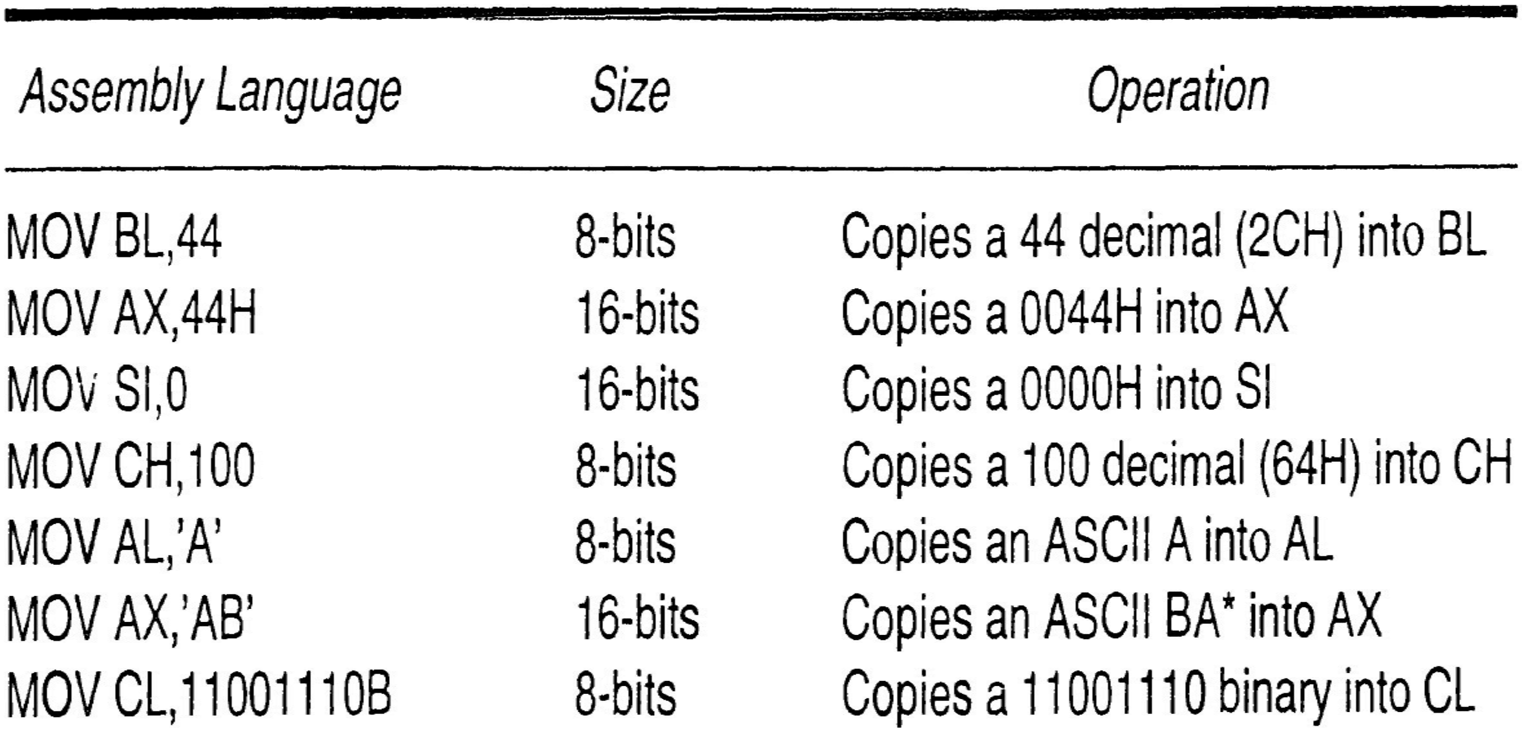
Immediate Addressing mode

- Transfers the source, an immediate byte or word of data, into the destination register or memory location.
- The source operand is a constant
- Immediate addressing mode can be used to load information into any of the registers except the segment registers and flag registers.

```
MOV AX,2500H           ; move 2500H into AX
MOV CX,600       ; load the decimal value 600 into CX
MOV BL, 80H      ; load 80H into BL

MOV AX,2500H
MOV DS, AX
MOV DS, 0133H ; illegal instruction!
```

Immediate Addressing mode



Assembly Language	*Size*	*Operation*
MOV BL,44	8-bits	Copies a 44 decimal (2CH) into BL
MOV AX,44H	16-bits	Copies a 0044H into AX
MOV SI,0	16-bits	Copies a 0000H into SI
MOV CH,100	8-bits	Copies a 100 decimal (64H) into CH
MOV AL,’A’	8-bits	Copies an ASCII A into AL
MOV AX,’AB’	16-bits	Copies an ASCII BA* into AX
MOV CL,11001110B	8-bits	Copies a 11001110 binary into CL

Direct Addressing mode

- Moves a byte or word between a memory location and a register.
- This address is the offset address.

MOV AX, [2500] ; move contents of DS:2500H into AX

The physical address is calculated by combining the contents of offset location 2500 with DS.

Example::

Find the physical address off the memory location and its contents after the execution off the following, assuming that DS = 1412H.

```
        MOV AL, 3BH
        MOV [2518], AL
```

Solution:

First 3BH is copied into AL,

Then in line two, the contents off AL are moved to logical address DS:3518 which is 1412:2518.

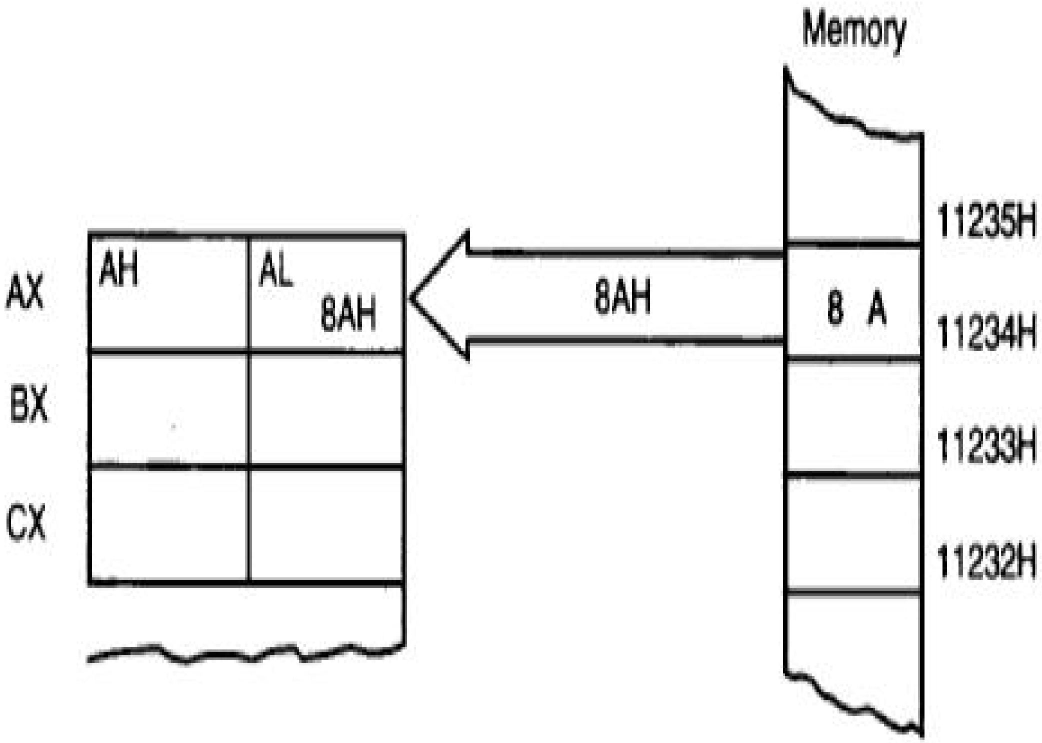
Shifting DS left and adding it to the offset gives the physical

address off 18638H **(14120H + 2518H = 16638H).**

After the execution off the second instruction, the memory

location with address 18638H will contain tithe value 3BH..

- Example for the instruction MOV AL,[1234H] and DS = 1000H



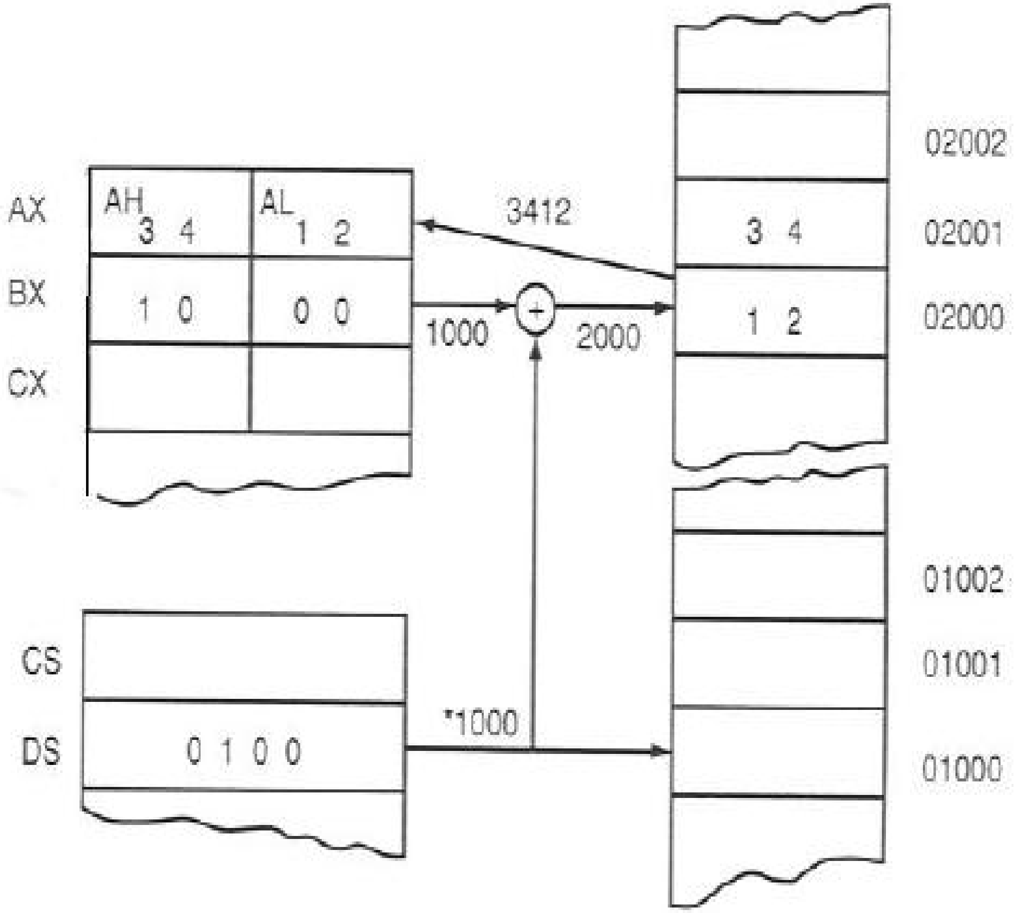
Examples of Direct Addressing modes

MOV AL,[2C00h]	Copy the content of data segment memory location 2C00h in to AL register (8 Bit)
MOV AX,[3400h]	Copies the word content of data segment memory location 3400h in to AX (16 bit)
MOV [100H], AL	Copies BL in to Data Segment Memory Location 100h (8 bit)
MOV [72C2h], CX	Copies CX in to data segment memory location 72C2H (16 bit)
MOV ES:[2000H],AL	Copies Al in to extra segment memory location 2000H (8 Bit)
MOV SP,[4E2h]	Copies the word contents of data segment memory location 4E2h in SP (16 bit)

Register Indirect Addressing mode

- Instruction specifies an address where data is located. This addressing mode works with SI,DI,BX,BP registers.
- Example1 : Write value 0065h at the address pointed by DS:BX
  - MOV BX,1200h
  - MOV [BX],65H
- Example1 : transfer the byte from AL to the address pointed by DS: 1202H
  - MOV [1202h],AL

The operation of MOV AX,[BX] instruction when BX = 1000H and DS = 0100H.

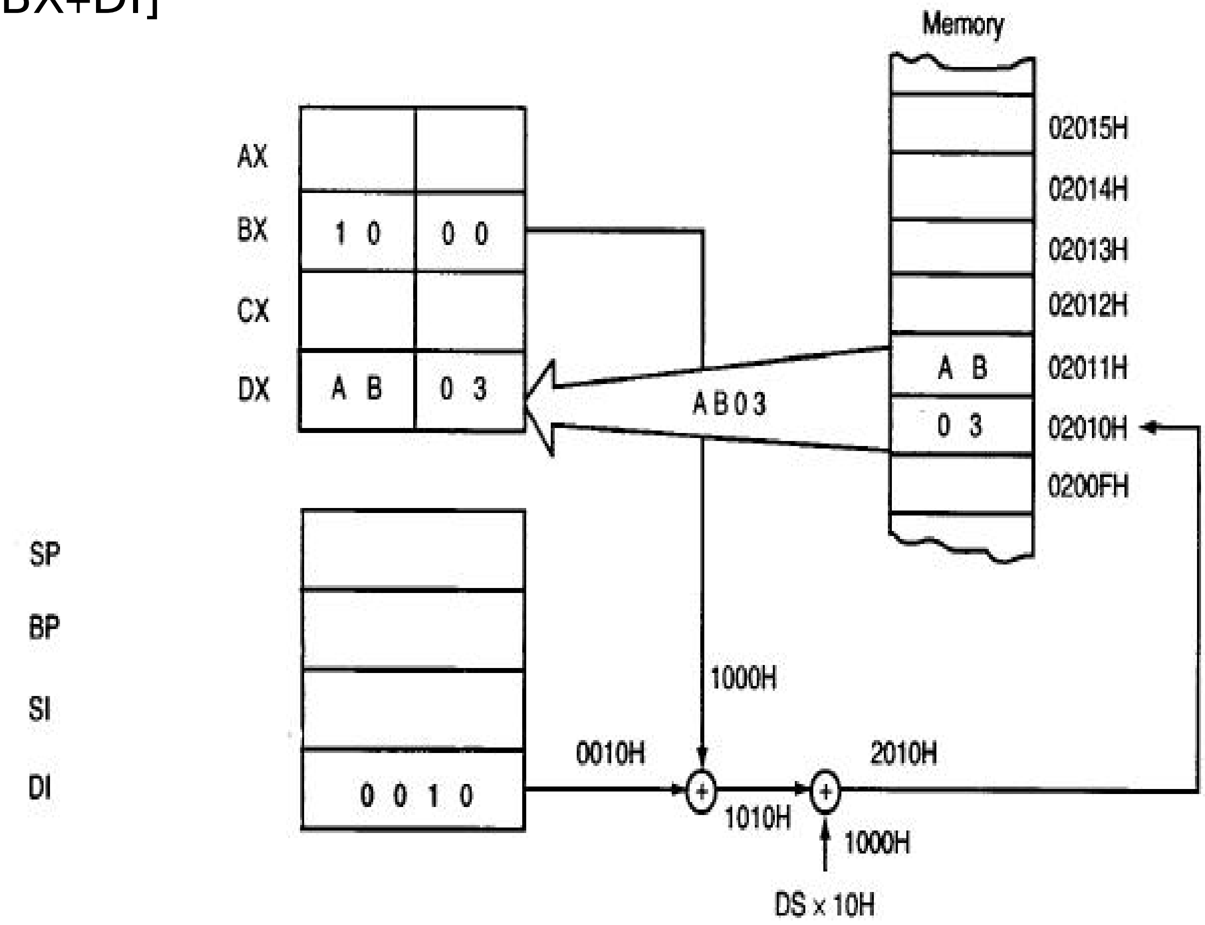


Base Plus index addressing mode

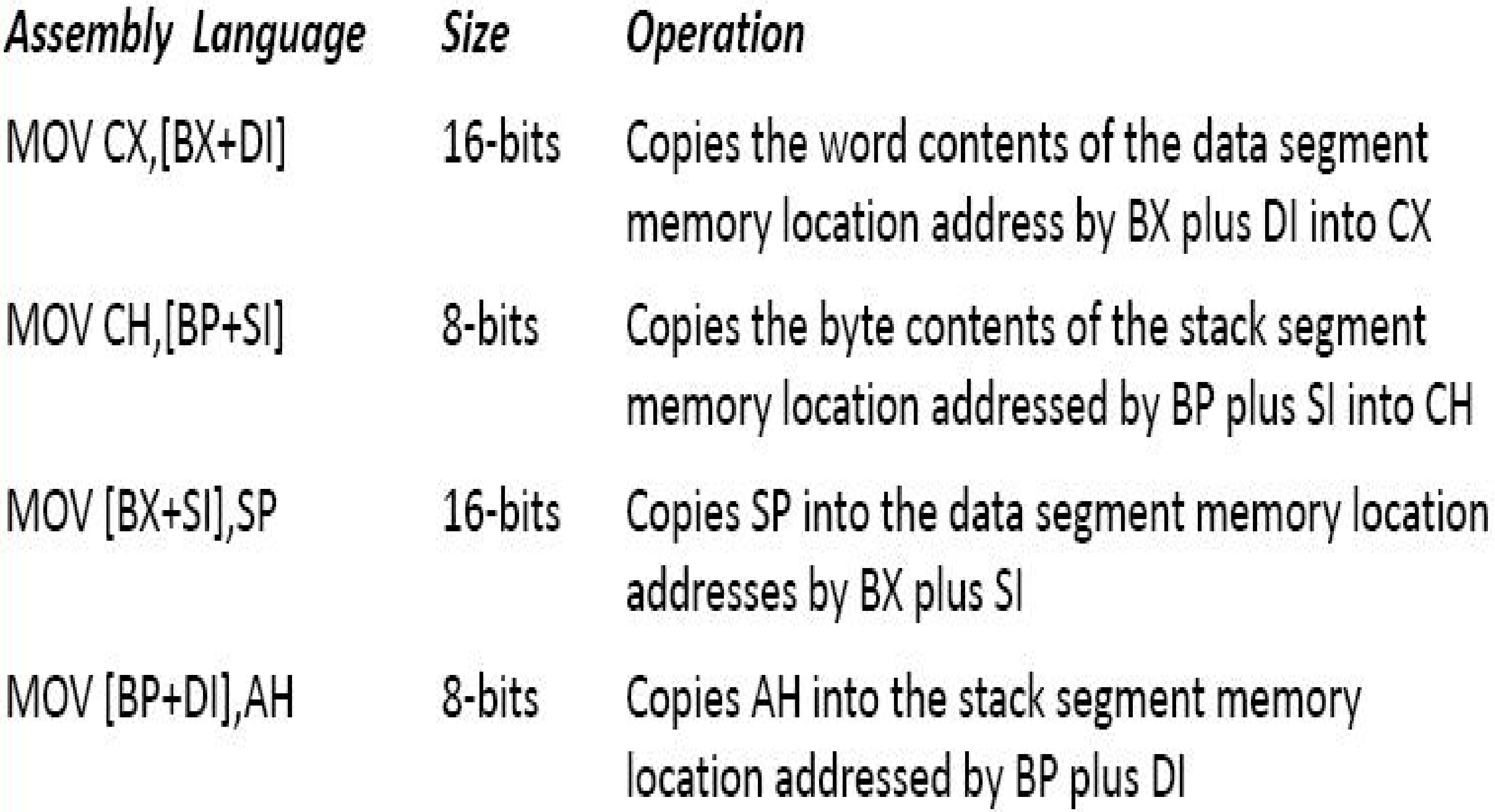
- This mode is similar to indirect addressing mode because it indirectly addresses the memory data. This type of addressing uses one base register and one index register to indirectly address the memory
- MOV DX,[BX+DI]
- Suppose BX = 1000h , DI = 0010h , and DS = 0100h, which translate in to memory location 2010H. This instruction transfers a copy of the word from location 2010H in to DX register.

Base Plus Index Mode

MOV DX,[BX+DI]



Examples



Assembly Language	*Size*	*Operation*
MOV CX,[BX+DI]	16-bits	Copies the word contents of the data segment memory location address by BX plus DI into CX
MOV CH,[BP+SI]	8-bits	Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH
MOV [BX+SI],SP	16-bits	Copies SP into the data segment memory location addresses by BX plus SI
MOV [BP+DI],AH	8-bits	Copies AH into the stack segment memory location addressed by BP plus DI

Base Plus index Plus offset addressing mode

- Examples:
  - MOV AL,DISP[BX][SI]
  - MOV AL, DISP[BX+DI]
  - MOV AL,[BP+SI+DISP]

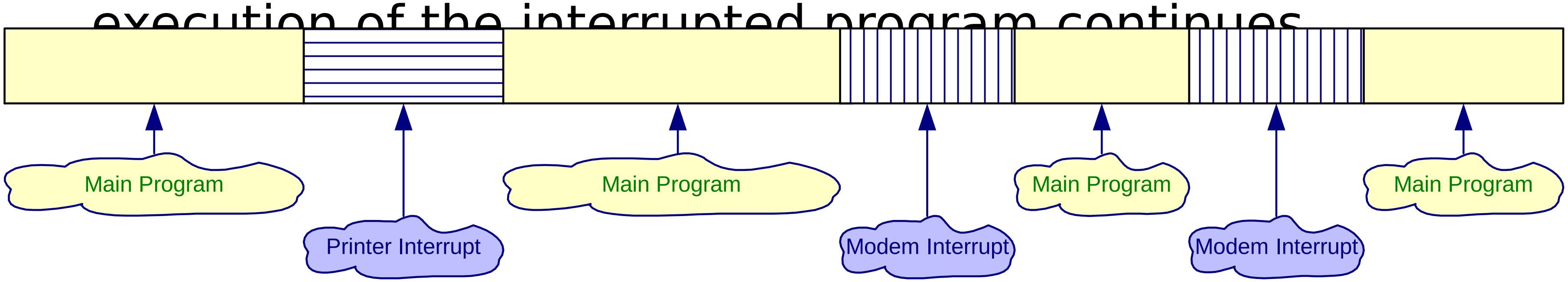
- MOV AX,[BX+DI+15]

Interrupts

- Interrupt Types
  - Hardware Interrupts: External event
  - Software Interrupts: Internal event (Software generated)
  - Maskable and non-maskable interrupts
- Interrupt priority
- Interrupt Vectors and Interrupt Handlers

Purpose of Interrupts

- Interrupts are useful when interfacing I/O devices with low data-transfer rates, like a keyboard or a mouse, in which case polling the device wastes valuable processing time
- The peripheral interrupts the normal application execution, requesting to send or receive data.
- The processor jumps to a special program called *Interrupt Service Routine* to service the peripheral
- After the processor services the peripheral, the execution of the interrupted program continues

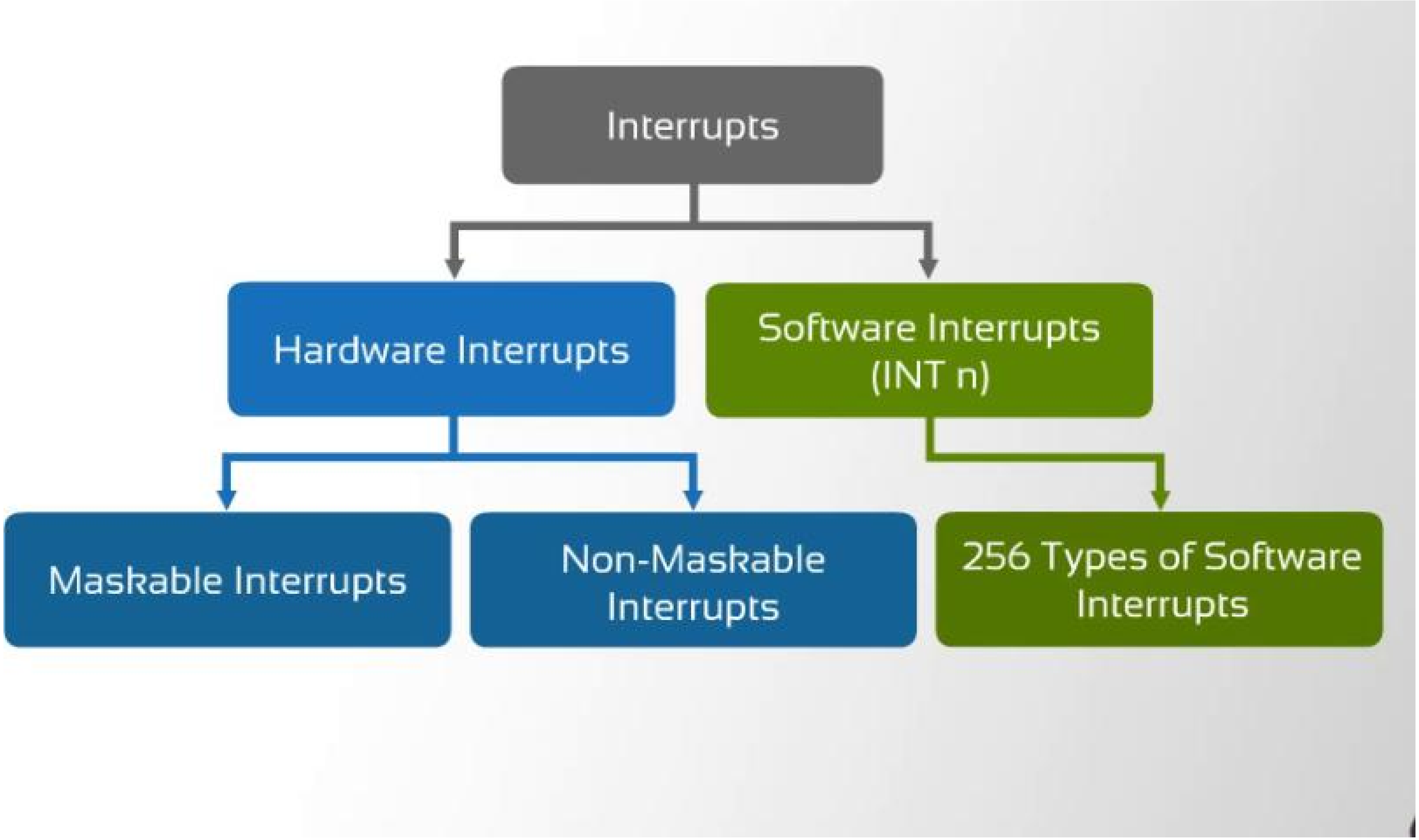


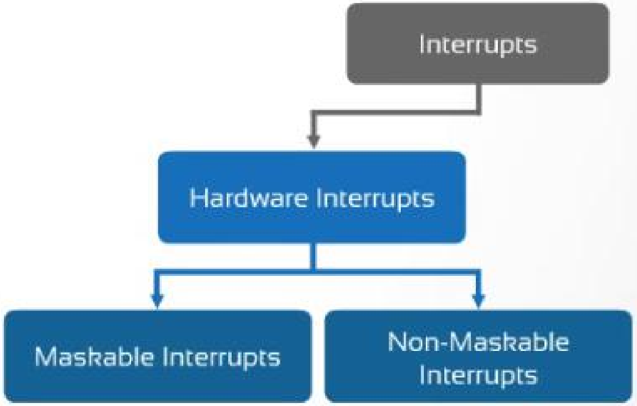
BASIC INTERRUPT TERMINOLOGY

- ***Interrupt Service Routine (ISR) or Interrupt handler***: code used for handling a specific interrupt
- ***Interrupt priority*****:** In systems with more than one interrupt inputs, some interrupts have a higher priority than other
  - They are serviced first if multiple interrupts are triggered simultaneously
- ***Interrupt vector*****:** Code loaded on the bus by the interrupting device that contains the Address (segment and offset) of specific interrupt service routine
- ***Interrupt Masking*****:** Ignoring (disabling) an interrupt

- **Interrupt V/S Polling**

Types of Interrupts



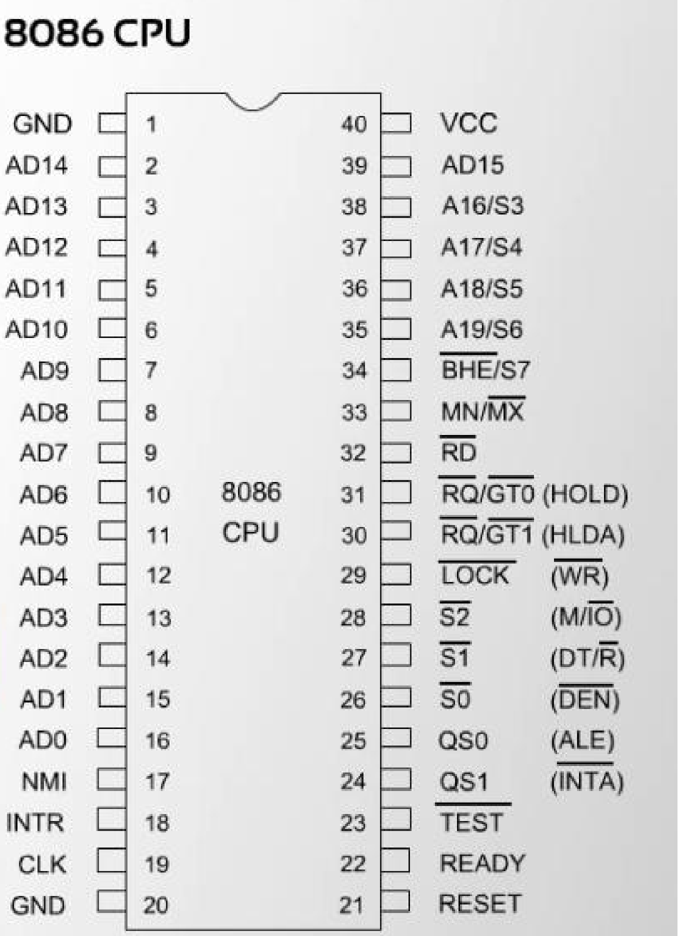


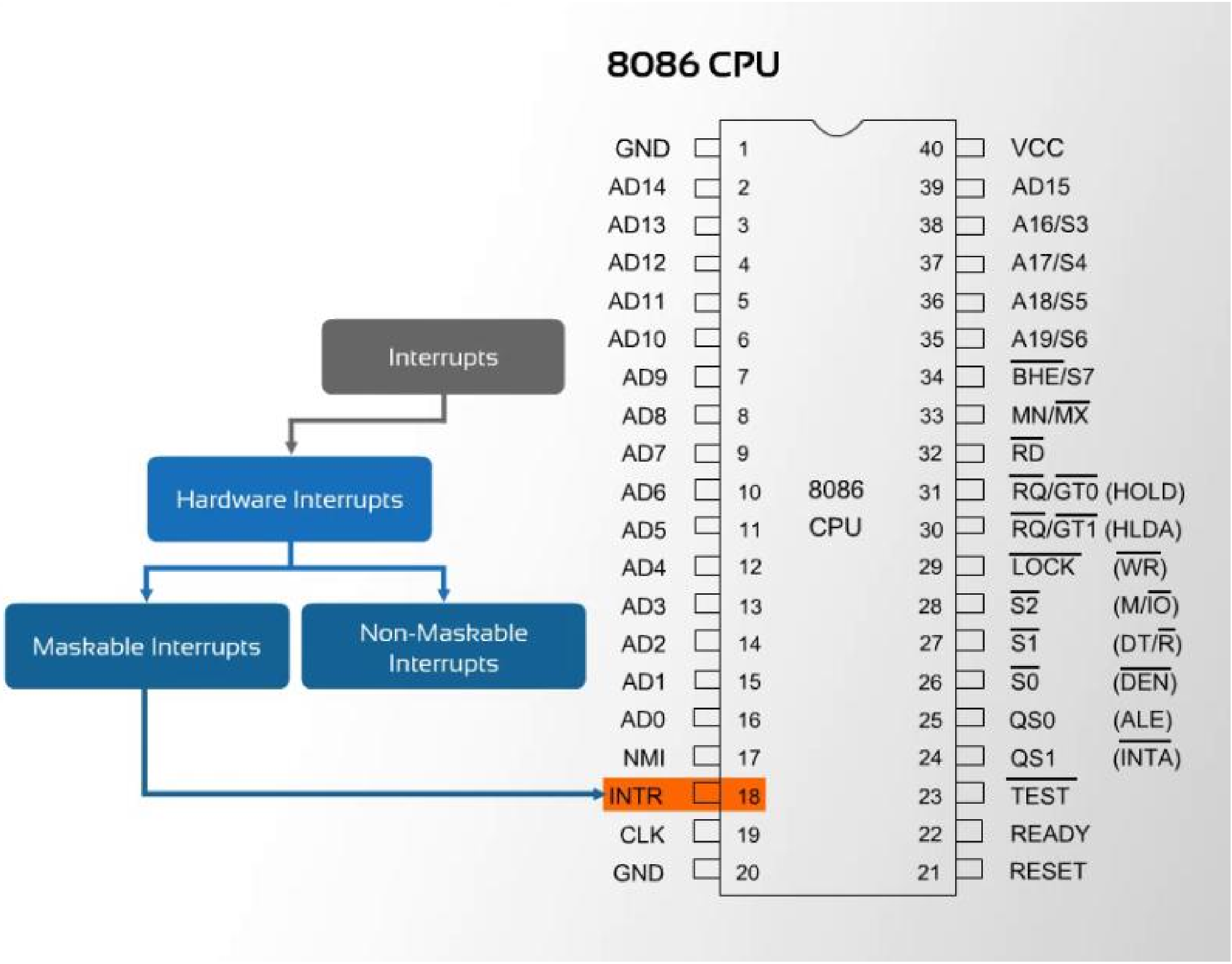
Interrupts

Hardware Interrupts

Maskable Interrupts

Non-Maskable Interrupts





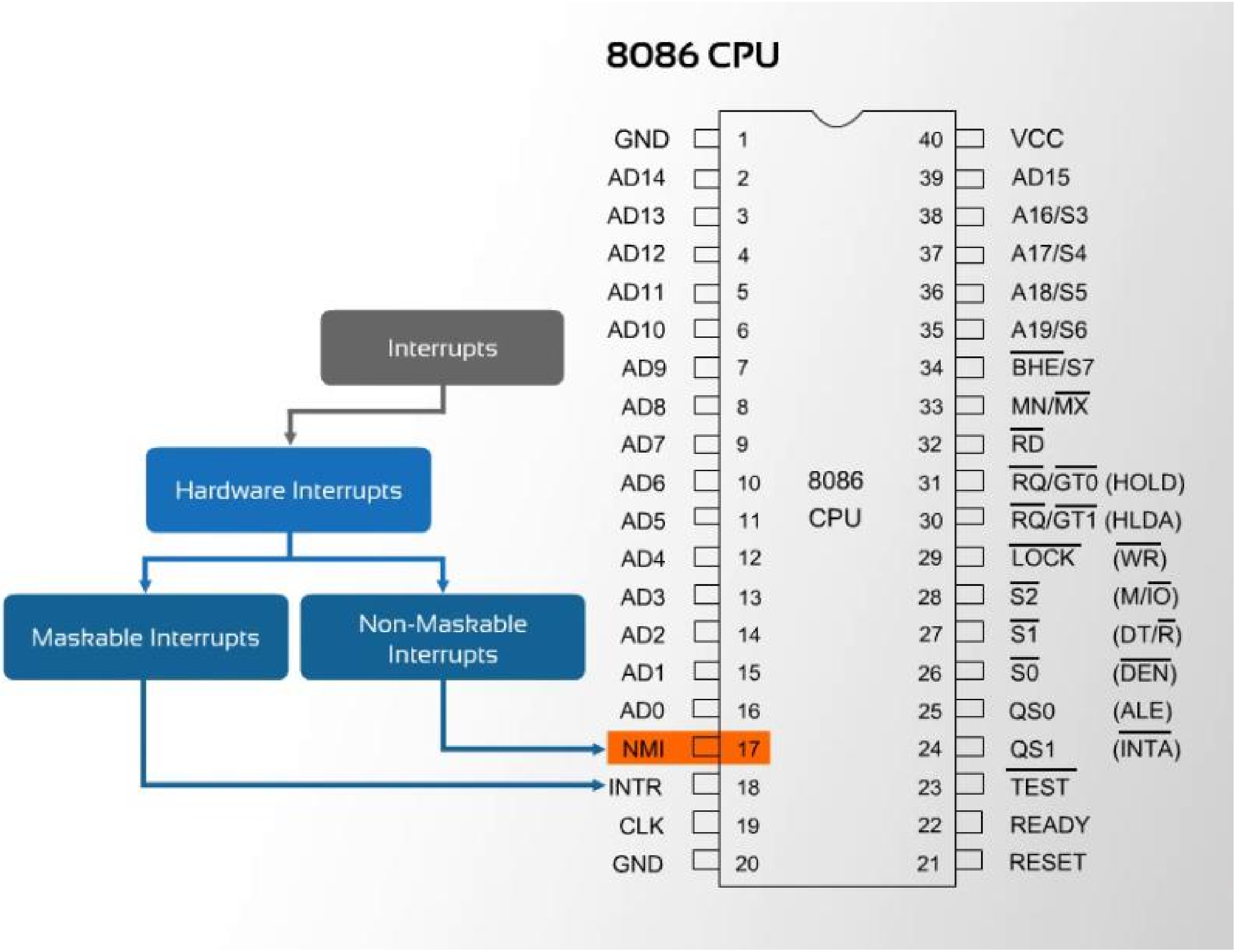
8086 CPU

Interrupts

Hardware Interrupts

Maskable Interrupts

Non-Maskable Interrupts



8086 CPU

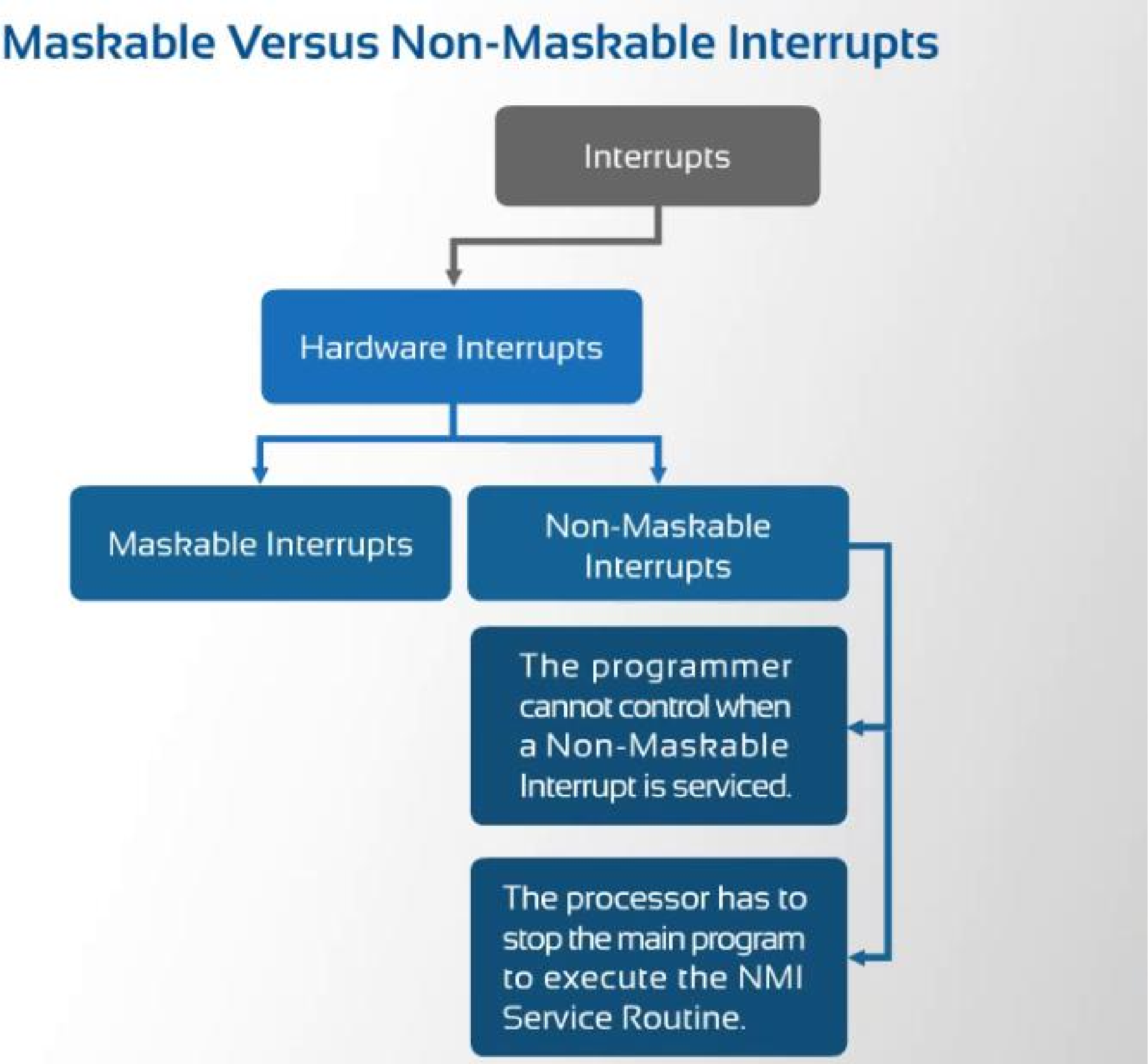
Interrupts

Hardware Interrupts

Maskable Interrupts

Non-Maskable Interrupts

GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086
CPU
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 $\overline{\text{BHE}}$/S7
33 MN/$\overline{\text{MX}}$
32 $\overline{\text{RD}}$
31 $\overline{\text{RQ}}$/$\overline{\text{GT0}}$ (HOLD)
30 $\overline{\text{RQ}}$/$\overline{\text{GT1}}$ (HLDA)
29 $\overline{\text{LOCK}}$ ($\overline{\text{WR}}$)
28 $\overline{\text{S2}}$ (M/$\overline{\text{IO}}$)
27 $\overline{\text{S1}}$ (DT/$\overline{\text{R}}$)
26 $\overline{\text{S0}}$ ($\overline{\text{DEN}}$)
25 QS0 (ALE)
24 QS1 ($\overline{\text{INTA}}$)
23 $\overline{\text{TEST}}$
22 READY
21 RESET



Maskable Versus Non-Maskable Interrupts

Interrupts

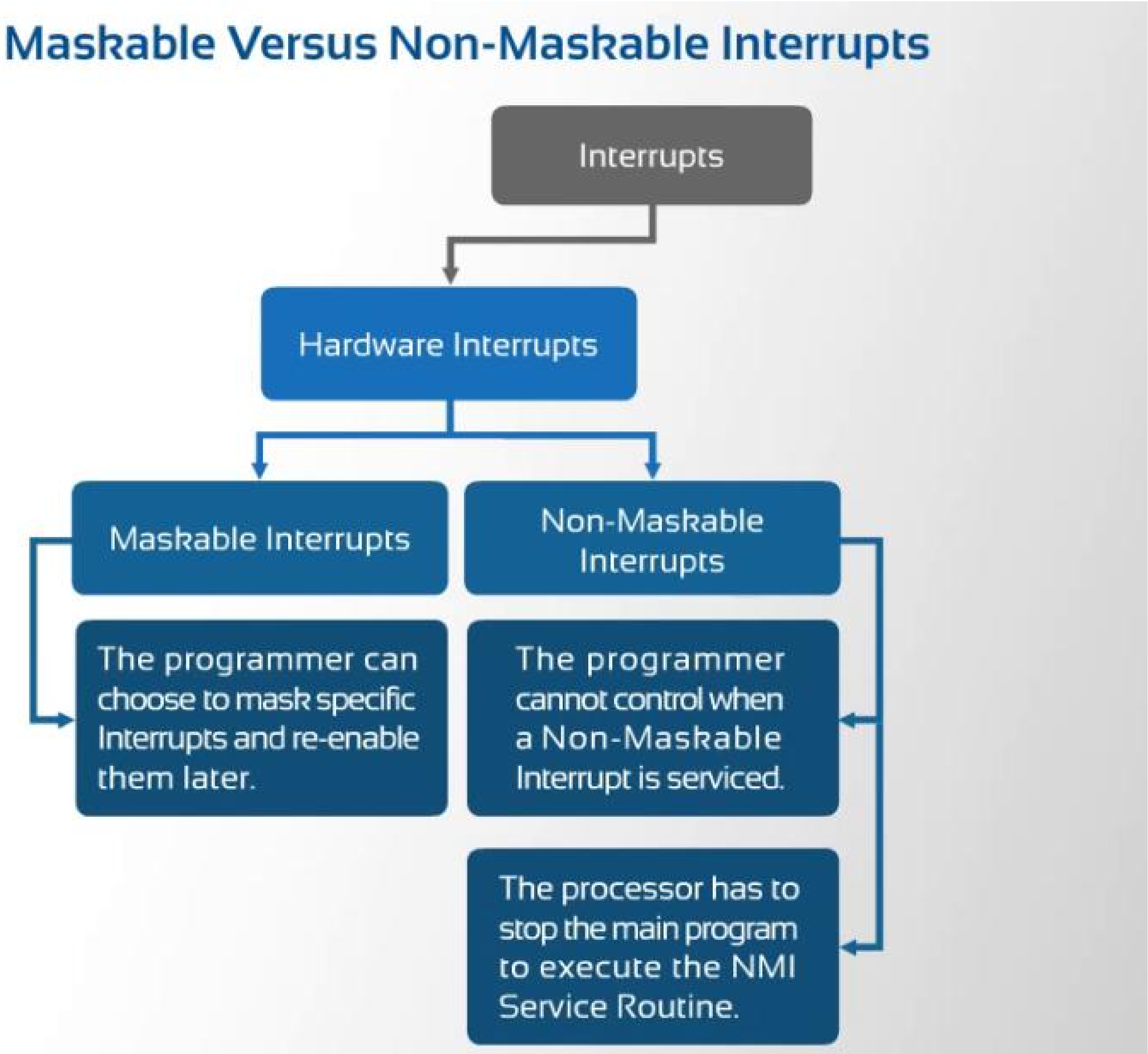
Hardware Interrupts

Maskable Interrupts

Non-Maskable Interrupts

The programmer cannot control when a Non-Maskable Interrupt is serviced.

The processor has to stop the main program to execute the NMI Service Routine.



Maskable Versus Non-Maskable Interrupts

Interrupts

Hardware Interrupts

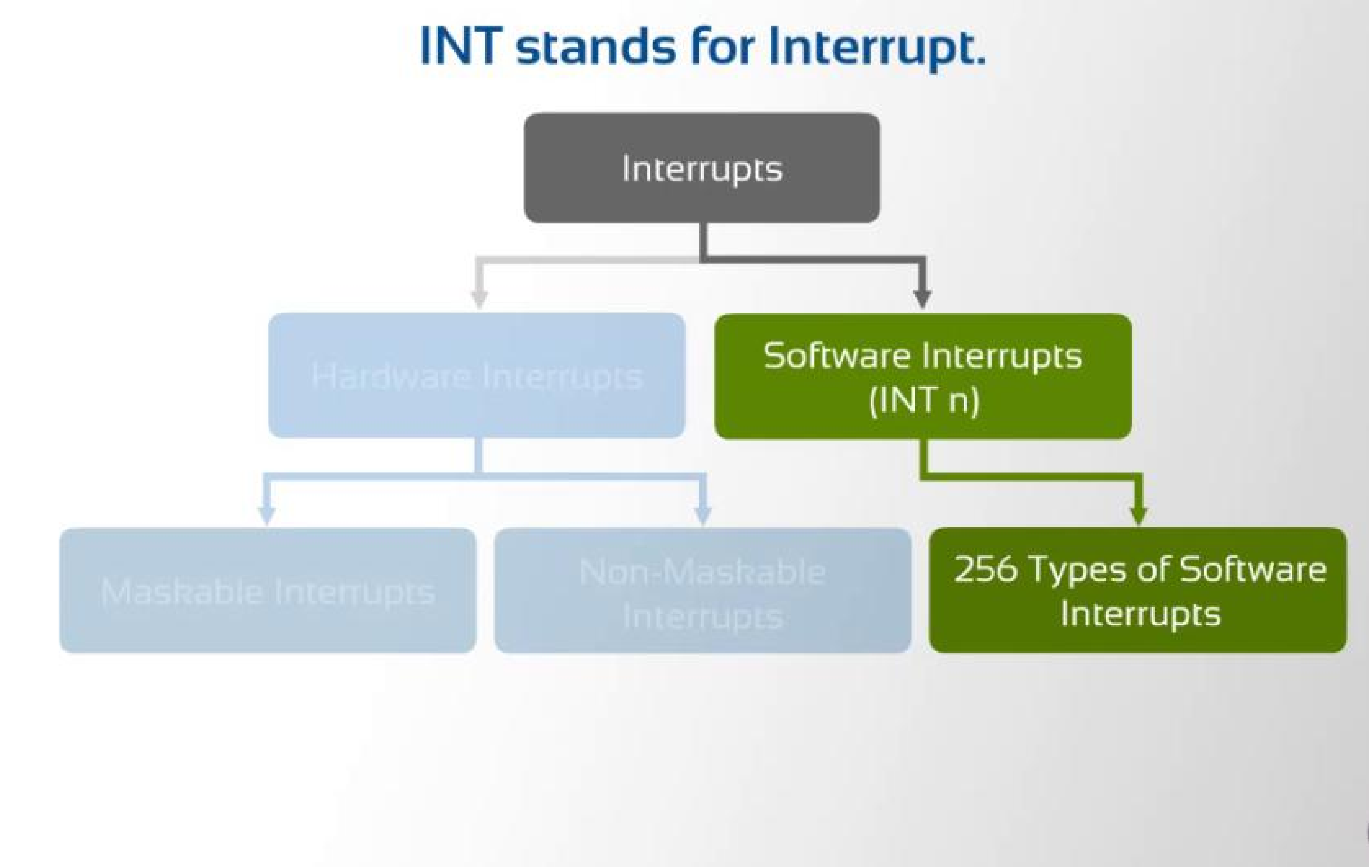
Maskable Interrupts

Non-Maskable Interrupts

The programmer can choose to mask specific Interrupts and re-enable them later.

The programmer cannot control when a Non-Maskable Interrupt is serviced.

The processor has to stop the main program to execute the NMI Service Routine.



INT stands for Interrupt.

Interrupts

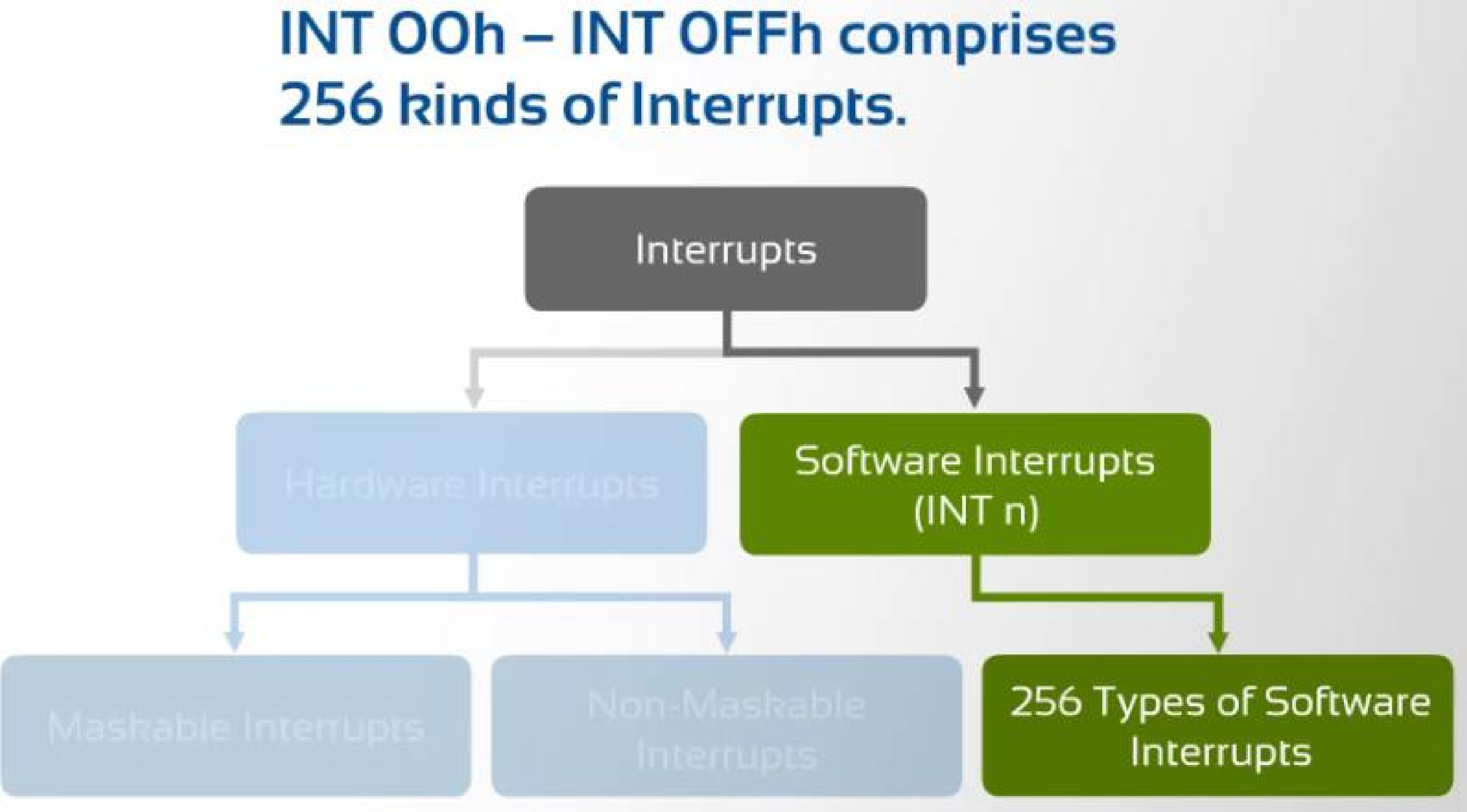
Hardware Interrupts

Software Interrupts (INT n)

Maskable Interrupts

Non-Maskable Interrupts

256 Types of Software Interrupts



INT 00h – INT 0FFh comprises 256 kinds of Interrupts.

Interrupts

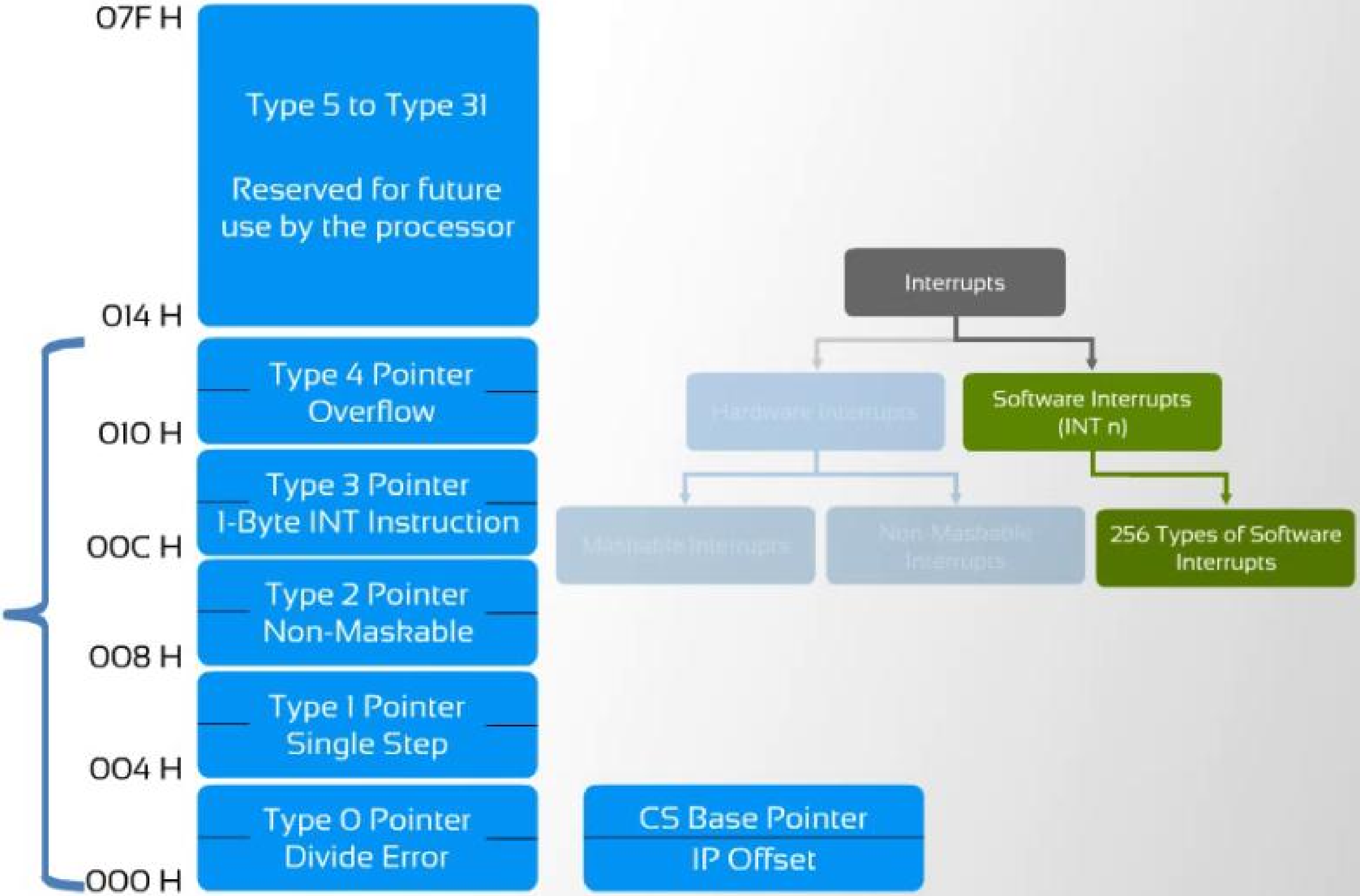
Hardware Interrupts

Software Interrupts (INT n)

Maskable Interrupts

Non-Maskable Interrupts

256 Types of Software Interrupts



07F H

Type 5 to Type 31

Reserved for future use by the processor

014 H

Type 4 Pointer
Overflow

010 H

Type 3 Pointer
1-Byte INT Instruction

00C H

Type 2 Pointer
Non-Maskable

008 H

Type 1 Pointer
Single Step

004 H

Type 0 Pointer
Divide Error

000 H

CS Base Pointer
IP Offset

Interrupts

Hardware Interrupts

Software Interrupts (INT n)

Maskable Interrupts

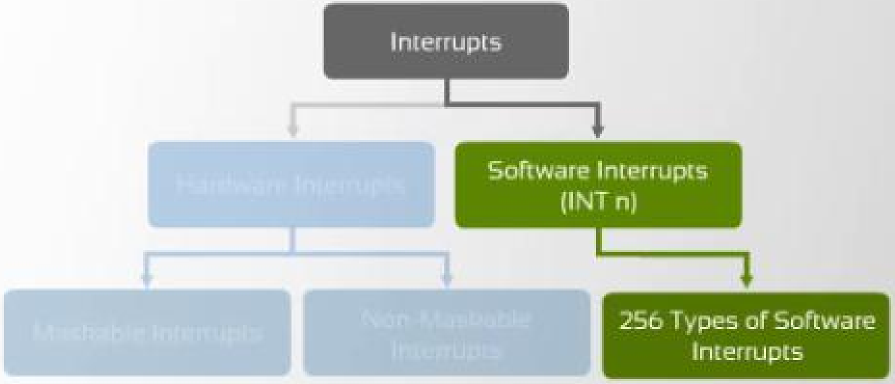
Non-Maskable Interrupts

256 Types of Software Interrupts

3FF H

Type 32 to Type 255
Free for User

080 H



Interrupts

Hardware Interrupts

Software Interrupts
(INT n)

Maskable Interrupts

Non-Maskable Interrupts

256 Types of Software Interrupts

Procedure when interrupt arrives

1. It decremented SP by 2 and pushes ***Flag register on the stack.***

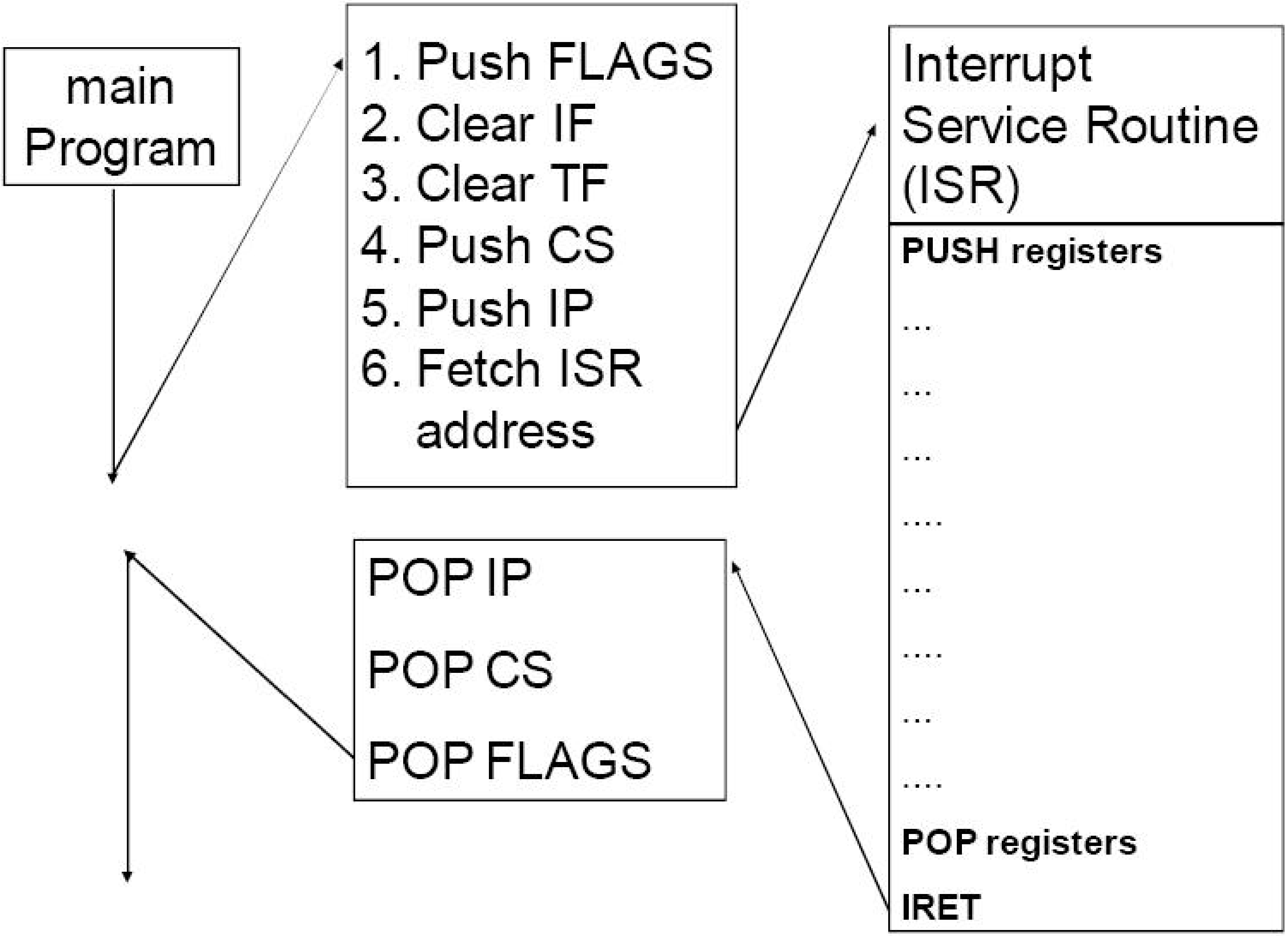
2. It disables 8086 **INTR input by clearing IF flag** in Flag register

3. It resets the **TF (trap) flag in Flag register**

4. It decremented SP again by 2 and pushes current **CS contents on the stack.**

5. It decremented SP again by 2 and pushes current **IP contents on the stack.**

6. It does an indirect far jump to the start of the



main Program

1. Push FLAGS
2. Clear IF
3. Clear TF
4. Push CS
5. Push IP
6. Fetch ISR address

Interrupt Service Routine (ISR)

PUSH registers

...

...

...

....

...

....

...

....

POP registers

IRET

POP IP

POP CS

POP FLAGS

How does 8086 get the address of a particular ISR**?**

In an 8086 system, each “interrupter” has an id#

–8086 treat this id# as interruption type#

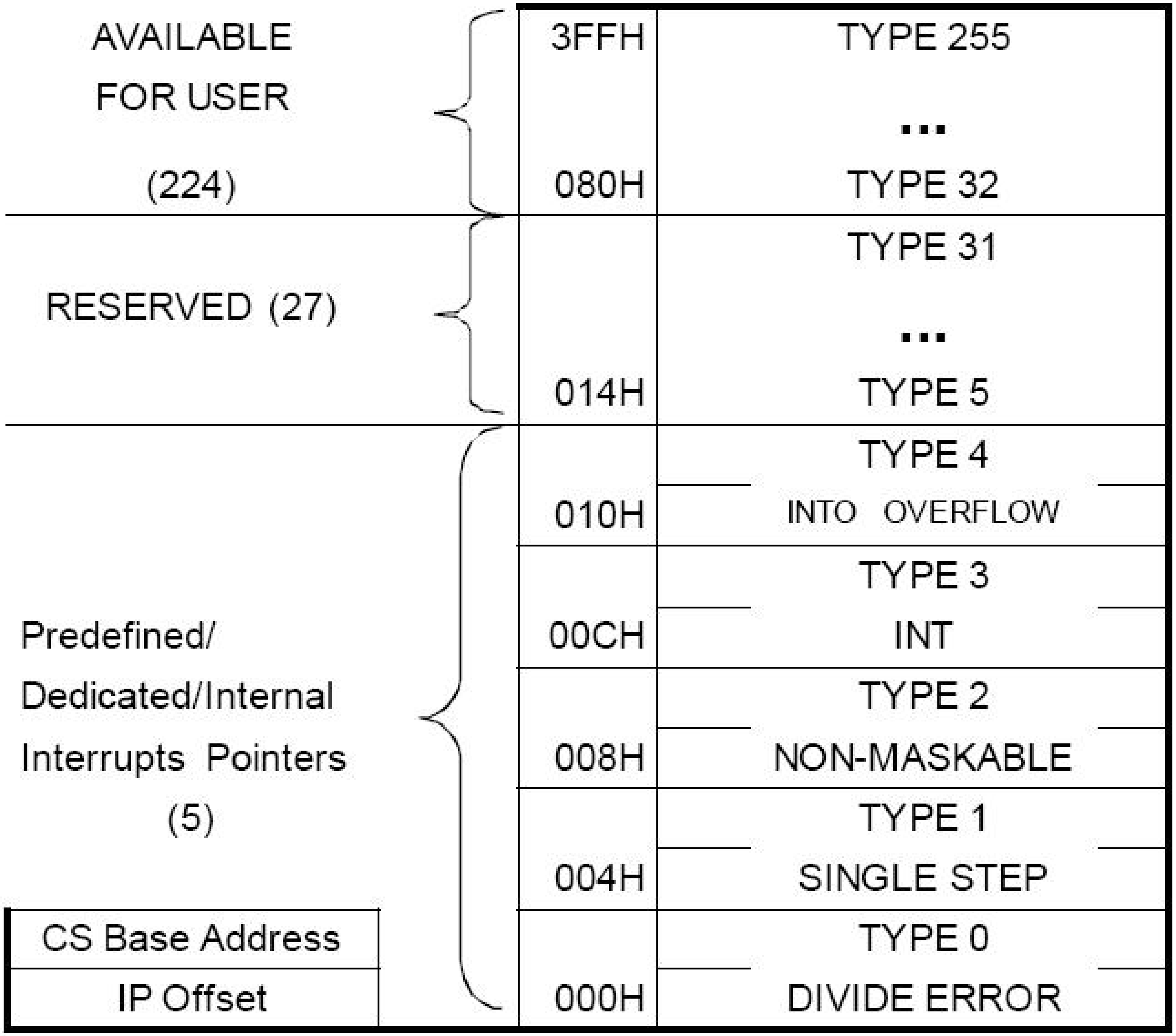
–after receiving INTR signal, 8086 sends an INTA signal

–after receiving INTA signal, interrupter releases it’s id#, i.e., type# of the interruption.

8086 multiplies this id# or type# by 4 to produced the desired address in the ***vector table***

8086 reads 4 bytes of memory starting from this address to get the starting address of ISR

- lower 2 byte is loaded in to IP
- higher 2 bytes to CS



AVAILABLE FOR USER (224)

RESERVED (27)

Predefined/ Dedicated/Internal Interrupts Pointers (5)

CS Base Address

IP Offset

3FFH	TYPE 255
	...
080H	TYPE 32
	TYPE 31
	...
014H	TYPE 5
	TYPE 4
010H	INTO OVERFLOW
	TYPE 3
00CH	INT
	TYPE 2
008H	NON-MASKABLE
	TYPE 1
004H	SINGLE STEP
	TYPE 0
000H	DIVIDE ERROR