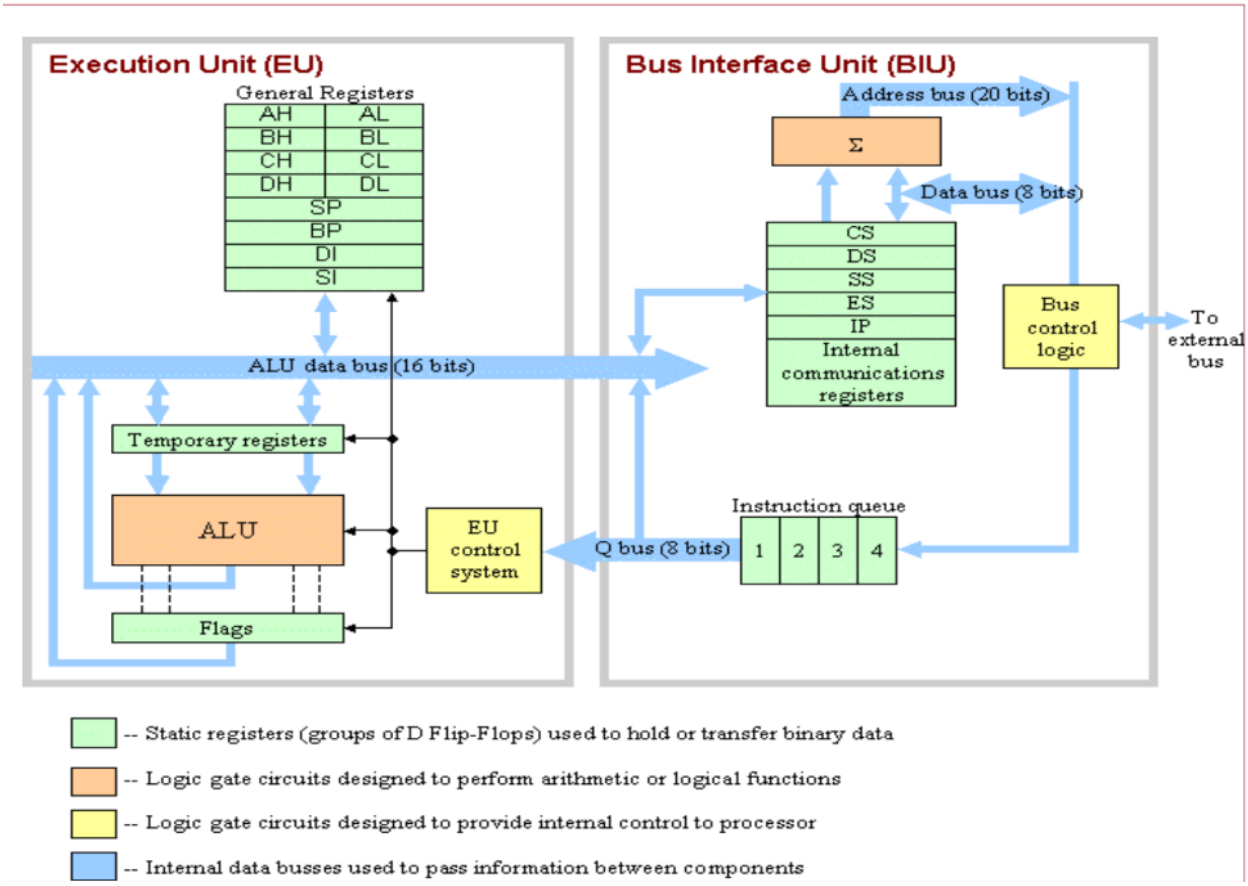
9th Batch

1.

a)Microprocessor vs microcontroller:

Microprocessor	Microcomputer
1. Computer system	1. Embedded system
2. Memory, IP are external	2. Internal
3. Large circuit	3. Small
4. High power consumption.	4. Low
5. PC	5. Washing mateiry, music player
6. Less no. of registers	6. more
7. Follows von Freumann model	7. Harvard model. archi.
8. Ram Rom, absent	7. Present.
9. External buses.	9. Internal bus
10. spread high	10. Low
11. complex and expensive	11. simple and affordable.
12. Require more instructions	Less

B) internal architecture of 8086:



BUS INTERFACE UNIT:

- The BIU has four 16-bit segment registers. These are the Code Segment (CS) register, the Data Segment (DS) register, the Stack Segment (SS) register, and the Extra Segment (ES) register.
- The BIU computes the 20-bit physical address internally using the programmer-provided logical address(16-bit contents of CS and IP) by logically shifting the contents of CS four bits to left and then adding the 16-bit contents of IP. For example, if [CS] = (456A)16 and [IP] = (1620)16, then the 20-bit physical address is generated by the BIU as follows:

Four times logically shifted [CS] to left = (456A0) 16

+ [IP] as offset = (1620)16

20-bit physical address = (46CC0)16

- The SS register points to the current stack. The 20-bit physical stack address is calculated from SS and SP for stack instruction such as PUSH and POP.
- The DS register points to the current data segment; operands for most instructions are fetched from this segment. The 16-bit contents of Source Index (SI) or Destination Index (DI) are used as offset for computing the 20-bit physical address.
- The ES register points to the extra segment in which data (in excess of 64k pointed to by DS) is stored. String instructions always use ES and DI to determine the 20-bit physical address for the destination.

c)

ADDRESSING MODES OF 8086

1. Immediate: In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

   Example: MOV AX, 0005H In the above example, 0005H is the immediate data.

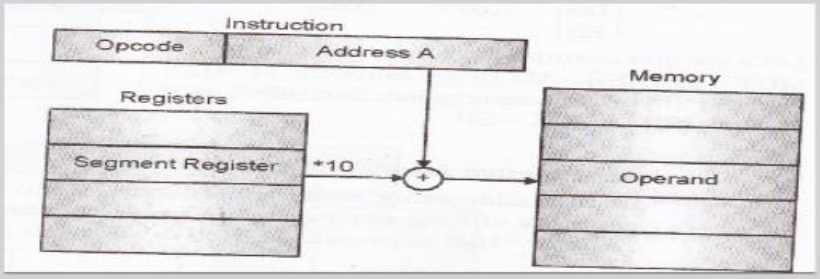
   The immediate data may be 8-bit or 16-bit in size.

Instruction

Opcode	Operand

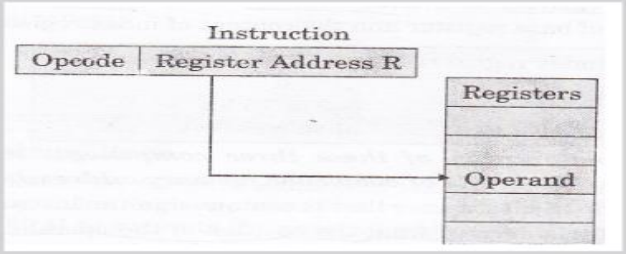
2. Direct: In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

   Example: MOV AX, [5000H]. Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is 10H*DS+5000H.



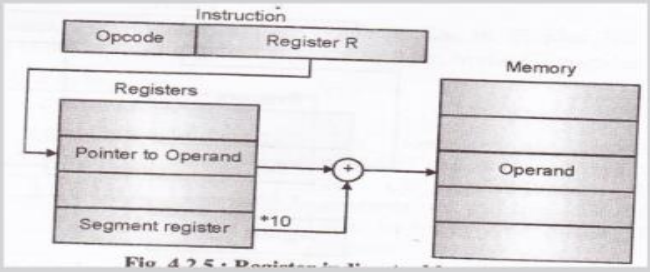
3. Register: In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

Example: MOV BX, AX.



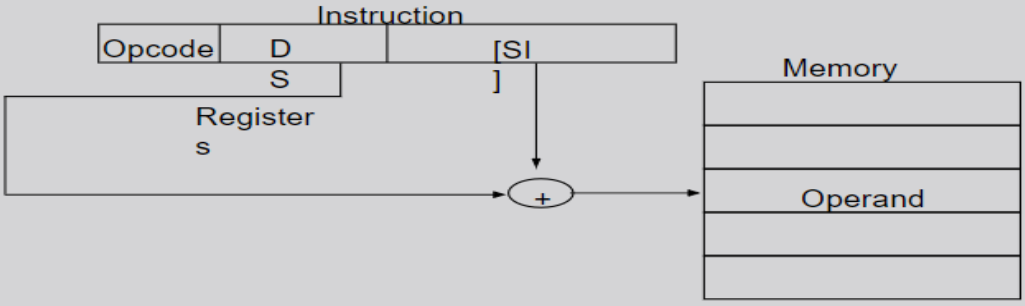
4. Register Indirect: In this addressing mode, the offset address of data is in either BX or SI or DI registers. The default segment is either DS or ES.

Example: MOV AX, [BX]. Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+ [BX].



5. Indexed: In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers SI and DI respectively. This mode is a special case of the above discussed register indirect addressing mode.

Example: MOV AX, [SI]. Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+ [SI].

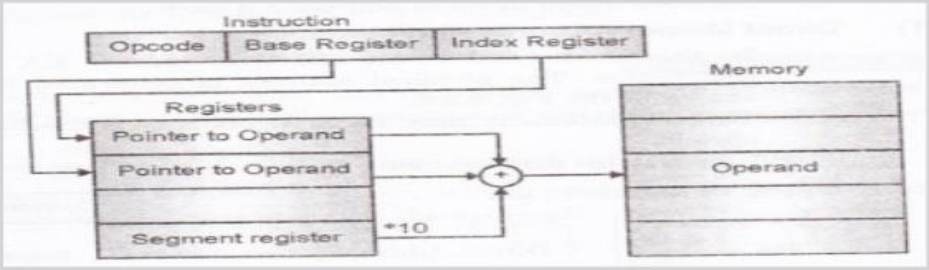


6. Register Relative: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given before explains this mode.

Example: MOV Ax, 50H [BX]. Here, effective address is given as 10H*DS+50H+ [BX].

7. Based Indexed: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI).

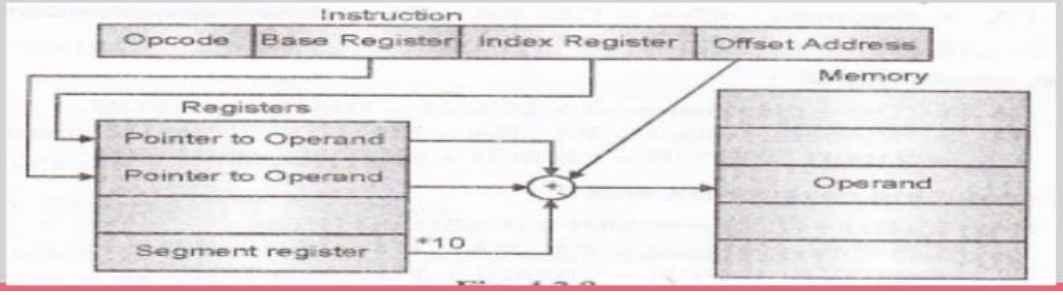
Example: MOV AX, [BX] [SI]. Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+ [BX] + [SI].



8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the bases registers (BX or BP) and any one of the index registers, in a default segment.

Example: MOV AX, 50H [BX] [SI]

Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 10H*DS+ [BX] + [SI] + 50H.



2.a)

I) NMI: The Non Maskable Interrupt input is similar to INTR except that the NMI interrupt does not check to see if the IFflag bit is at logic 1.

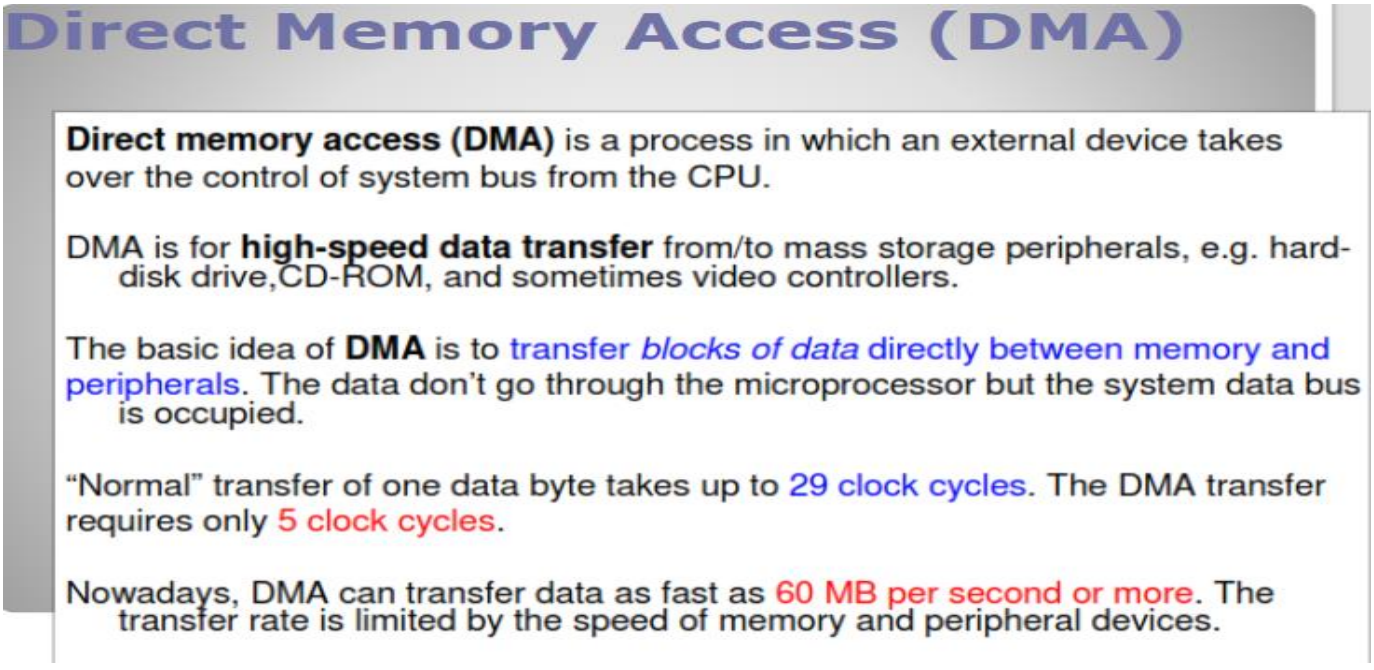
This interrupt cannot be masked (ordisabled) and no acknowledgment is required.

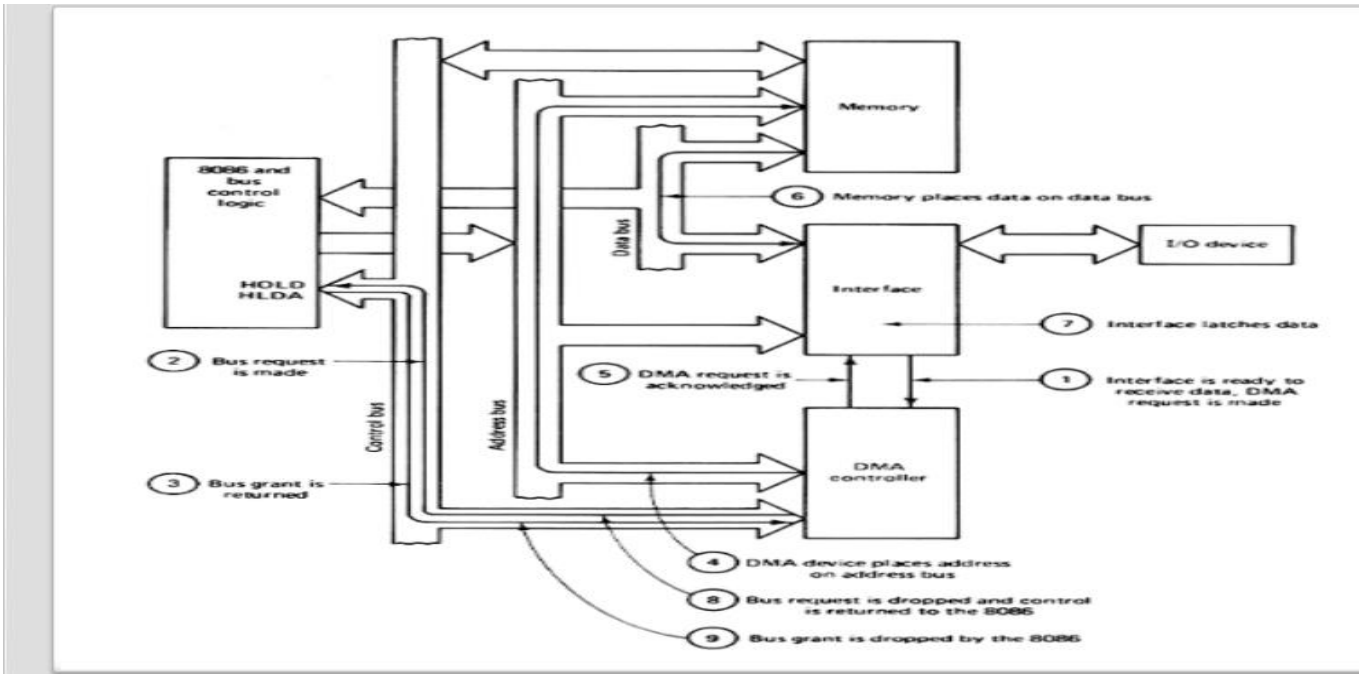
It should be reserved for “catastrophic”events such as power failure or memory errors.

ii)HLDA: Hold acknowledge is made high to indicate,to the DMA controller that the processor has entered hold state and it can take control over the system bus for DMA operation.

iii)ALE: Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high(1) pulse during T1 of any bus cycle.

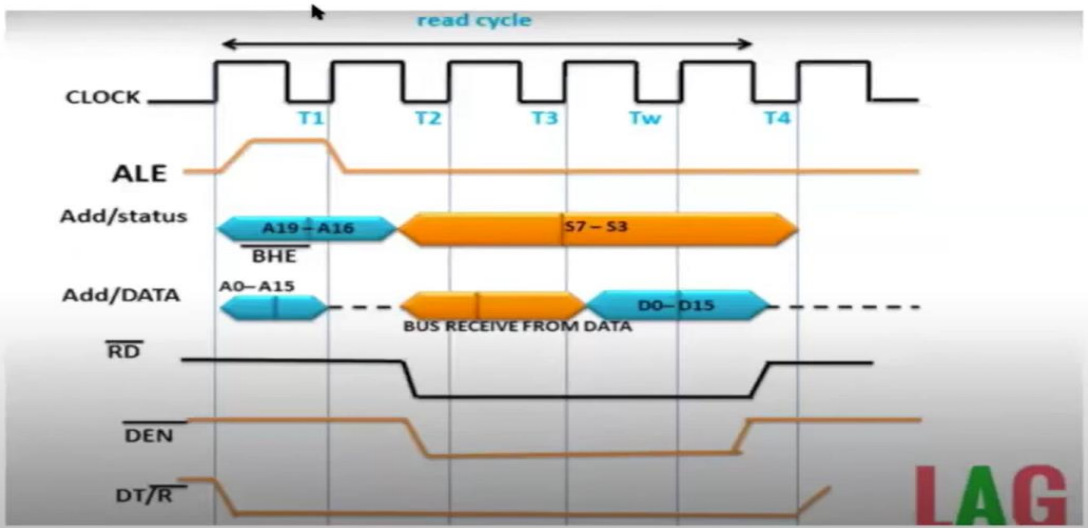
b)

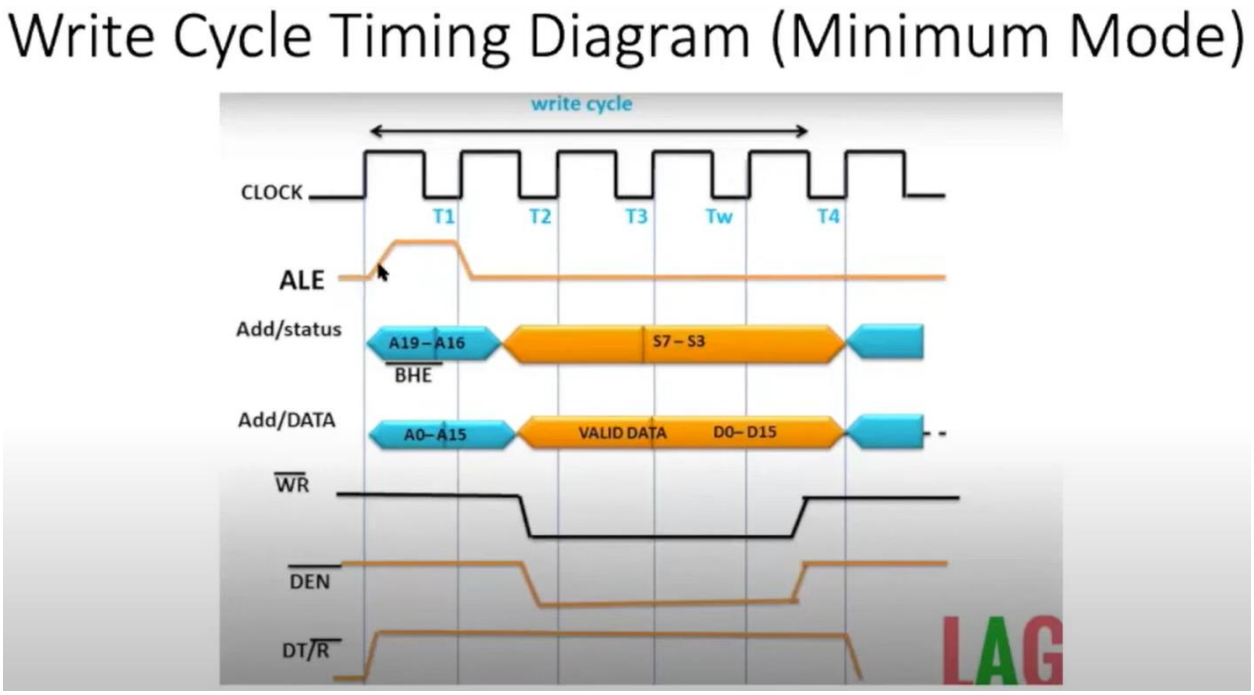




c)

Read Cycle Timing Diagram (Minimum Mode





3)

a)flags(SF,ZF,PF,OF,CF)

Initial register values:

- AX = 9FFFH
- BX = FFFE
- CX = FFEF
- DX = FA29H

Initial flag settings:

- SF = 0 (Sign Flag)
- ZF = 0 (Zero Flag)
- PF = 0 (Parity Flag)
- OF = 0 (Overflow Flag)
- CF = 0 (Carry Flag)

i) ADD AX, BX:

- This instruction adds the value of BX to AX.

Result: AX = AX + BX = 9FFFH + FFFE = A9FFH

Flag updates:

- SF = 1
- ZF = 0
- PF = 0
- OF = 1
- CF = 0

ii) SUB AX, CX:

- This instruction subtracts the value of CX from AX.

Result: AX = AX - CX = A9FFH - FFEFH = A900H

Flag updates:

- SF = 1
- ZF = 0
- PF = 1
- OF = 0
- CF = 1

iii) INC DX:

- This instruction increments the value of DX by 1.

Result: DX = DX + 1 = FA29H + 1 = FA2AHFlag updates:

- SF = 0
- ZF = 0
- PF = 1
- OF = 0
- CF = 0

iv) MOV AX, DX:

- This instruction moves the value of DX into AX.

Result: AX = DX = FA2AH

Final flag values after all the instructions:

- SF = 0

- ZF = 0
- PF = 1
- OF = 0
- CF = 0

b)

i)TRAP FLAG: Trap Flag (T) – This flag is used for on-chip debugging. Setting trap flag puts the microprocessor into single step mode for debugging. In single stepping, the microprocessor executes a instruction and enters into single step ISR.

ii)INTERRUPT FLAG: Interrupt Flag (I) – This flag is for interrupts. If interrupt flag is set (1), the microprocessor will recognize interrupt requests from the peripherals. If interrupt flag is reset (0), the microprocessor will not recognize any interrupt requests and will ignore them.

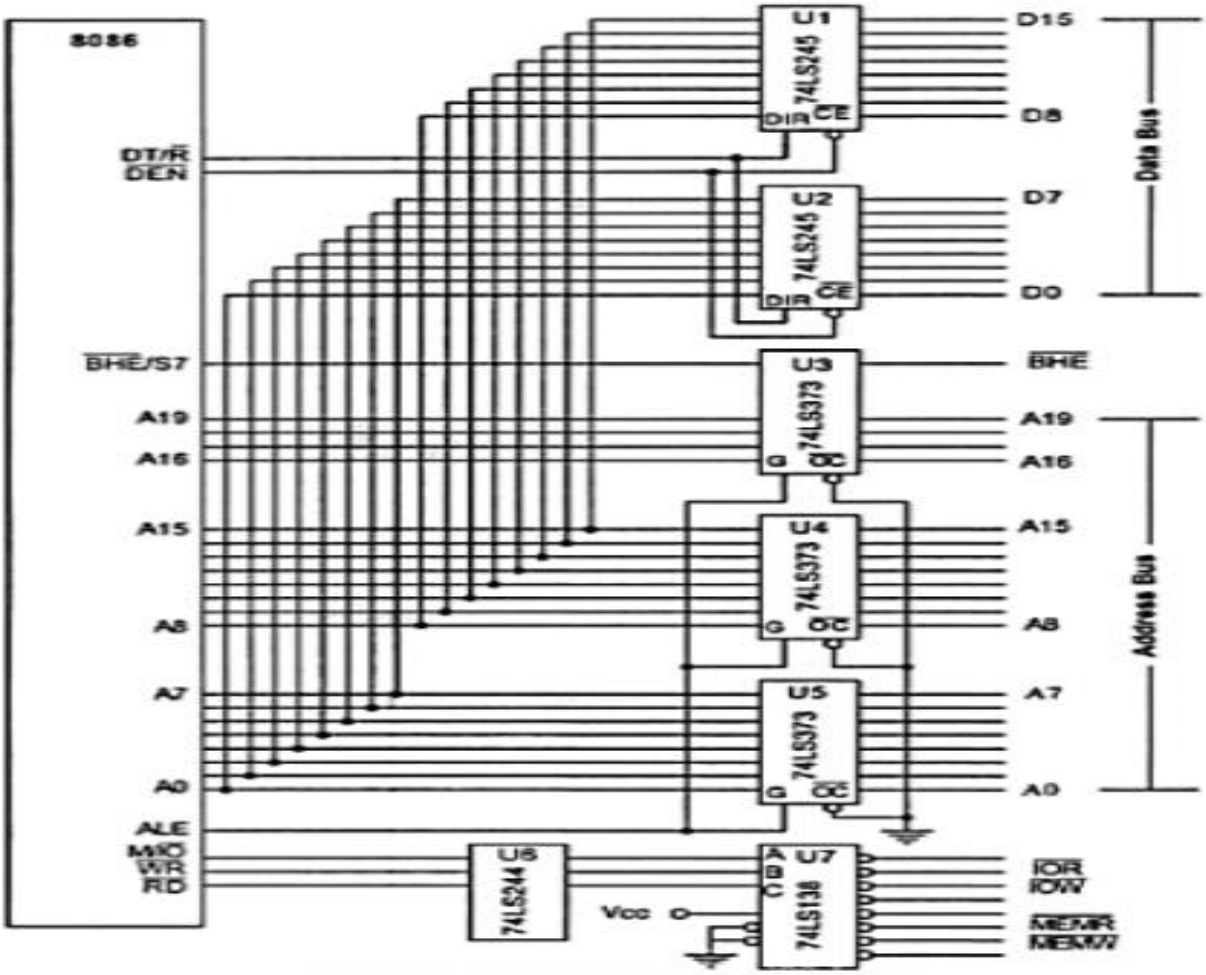
iii)DIRECTION FULG: Directional Flag (D) – This flag is specifically used in string instructions. If directional flag is set (1), then access the string data from higher memory location towards lower memory location. If directional flag is reset (0), then access the string data from lower memory location towards higher memory location.

c) 8288 clock generator:

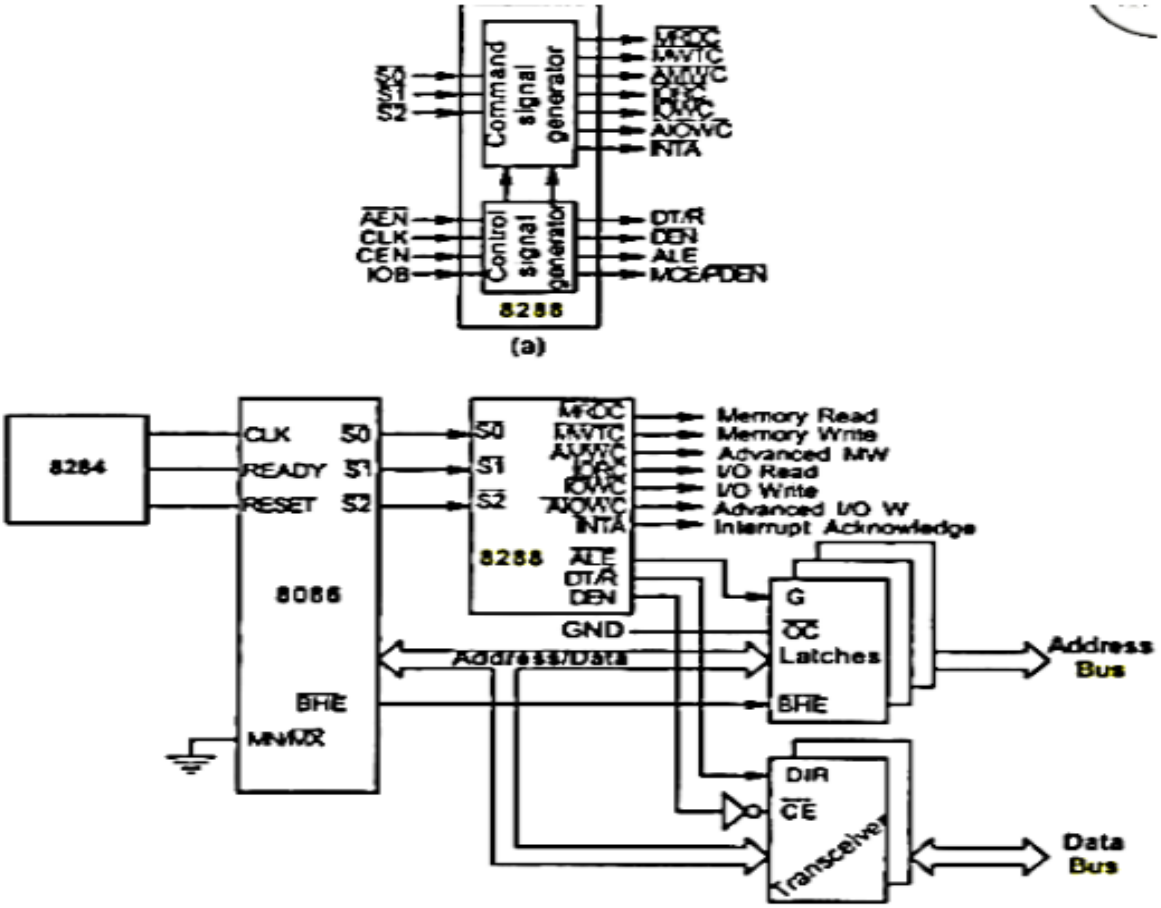
The 8288 input and output signals

- $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$. The inputs (8086 Status outputs) are decoded to generate command signals.
- $\overline{AEN}$. A low Address Enable signal activates the memory control signals.
- CEN. The Control Enable signal enables the 8288 command outputs.
- IOB. High on the I/O Bus input operates the 8288 in the I/O bus mode in systems where there are separate system bus and I/O bus.
- CLK. The Clock input
- DEN. The Data bus Enable signal controls the data bus buffers in the system. This signal is active-high in contrast to the $\overline{DEN}$ signal in the minimum mode.

- **ALE**. The Address Latch Enable signal is used to demuliplex address and data lins signals.
- **DT/$\overline{R}$**. The Data Transmit Receive signal controls bidirectional data bus buffer.
- **$\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$ and $\overline{IOWC}$**. The 8288 generates the normal Memory Read, Memory Write, I/O Read, I O Write Control signals.
- **$\overline{AMWC}$, and $\overline{AIOWC}$**. These are Advanced Memory and Advanced I/O Write Control signals.
- **$\overline{INTA}$**. The Interrupt Acknowledge output.
- **MCE/PDEN**. The Master Cascade Enable/Peripheral Data Enable output serves dual function. If IOB input is low it selects cascading of interrupt controllers, and if high enables the I/O bus transceivers.



Bus buffering and de-multiplexing



4.

a) Initially, the stack segment (SS) is 3000H, and the stack pointer (SP) is 4FFOH. After each PUSH operation, the stack pointer decrements by the size of the pushed data.

Let's see how the physical addresses change after executing each instruction consecutively:

1. PUSH BX:

   - Data to be pushed: Contents of BX register (16 bits).

   - Current SP: 4FFOH

   - SP after PUSH: 4FFEH (SP decrements by 2 bytes as BX is a 16-bit register).

   - Physical address: SS:SP = 3000H:4FFEH

2. PUSH AX:

   - Data to be pushed: Contents of AX register (16 bits).

- Current SP: 4FFEH
- SP after PUSH: 4FFCH (SP decrements by 2 bytes as AX is a 16-bit register).
- Physical address: SS:SP = 3000H:4FFCH

3. PUSH DX:

- Data to be pushed: Contents of DX register (16 bits).
- Current SP: 4FFCH
- SP after PUSH: 4FFAH (SP decrements by 2 bytes as DX is a 16-bit register).
- Physical address: SS:SP = 3000H:4FFAH

4. PUSH AX:

- Data to be pushed: Contents of AX register (16 bits).
- Current SP: 4FFAH
- SP after PUSH: 4FF8H (SP decrements by 2 bytes as AX is a 16-bit register).
- Physical address: SS:SP = 3000H:4FF8H

So, after executing all the instructions consecutively, the physical addresses on the stack will be as follows:

- SS:SP = 3000H:4FF8H

- The top of the stack (address pointed by SS:SP) contains the last pushed data (contents of AX). The stack grows downwards from this address.

b)SHL,SHR,RCL,RCR,ROR,ROL:

1. SHL (Shift Left):

- SHL (Shift Logical Left) is used to perform a left shift operation on the specified operand.
- The bits are shifted to the left, and zeros are brought in from the right side.

- SHL can also be referred to as SAL (Shift Arithmetic Left), and in x86 assembly language, they are synonymous.

Example:

Suppose we have the value 10 (in binary: 0000 1010) in the AL register, and we execute the instruction `SHL AL, 1` (shift AL left by 1 bit).

Before: AL = 0000 1010 (10 in decimal)

After: AL = 0001 0100 (20 in decimal)

2. SHR (Shift Right):

- SHR (Shift Logical Right) is used to perform a right shift operation on the specified operand.

- The bits are shifted to the right, and zeros are brought in from the left side.

Example:

Suppose we have the value 40 (in binary: 0010 1000) in the AL register, and we execute the instruction `SHR AL, 2` (shift AL right by 2 bits).

Before: AL = 0010 1000 (40 in decimal)

After: AL = 0000 1010 (10 in decimal)

3. RCL (Rotate through Carry Left):

- RCL (Rotate through Carry Left) is used to perform a left rotation on the specified operand through the carry flag.

- The most significant bit (MSB) is shifted into the carry flag, and the carry flag is shifted into the least significant bit (LSB).

Example:

Suppose we have the value 129 (in binary: 1000 0001) in the AL register, and the carry flag is set to 1. When we execute the instruction `RCL AL, 1`:

Before: AL = 1000 0001 (129 in decimal), CF = 1

After: AL = 0000 0011 (3 in decimal), CF = 1

4. RCR (Rotate through Carry Right):

- RCR (Rotate through Carry Right) is used to perform a right rotation on the specified operand through the carry flag.

- The least significant bit (LSB) is shifted into the carry flag, and the carry flag is shifted into the most significant bit (MSB).

Example:

Suppose we have the value 2 (in binary: 0000 0010) in the AL register, and the carry flag is set to 1. When we execute the instruction `RCR AL, 1`:

Before: AL = 0000 0010 (2 in decimal), CF = 1

After: AL = 1000 0001 (129 in decimal), CF = 0

5. ROR (Rotate Right):

- ROR (Rotate Right) is used to perform a right rotation on the specified operand.

- The bits are shifted to the right, and the MSB is brought in from the left side (similar to SHR). However, the bit that goes off the right end is brought back in from the left side.

Example:

Suppose we have the value 100 (in binary: 0110 0100) in the AL register, and we execute the instruction `ROR AL, 3` (rotate AL right by 3 bits).

Before: AL = 0110 0100 (100 in decimal)

After: AL = 1000 0010 (130 in decimal)

6. ROL (Rotate Left):

- The bits are shifted to the left, and the LSB is brought in from the right side (similar to SHL). However, the bit that goes off the left end is brought back in from the right side.

Example:

Suppose we have the value 5 (in binary: 0000 0101) in the AL register, and we execute the instruction `ROL AL, 2` (rotate AL left by 2 bits).

Before: AL = 0000 0101 (5 in decimal)

After: AL = 0001 0100 (20 in decimal)

Key differences between these instructions:

- SHL and SHR perform logical shifts, i.e., they fill the shifted bits with zeros.

- RCL and RCR perform rotations through the carry flag, which allows circular shift operations.

- ROL and ROR also perform rotations, but they bring in the bits that go off the opposite end. This makes them suitable for creating circular shift effects.

c)Difference:

1**. MOV BX, A536H**: Directly loads the value `A536H` into the BX register.

2. **MOV BX, [A536H]**: Indirectly loads the value stored at memory address `A536H` into the BX register.

5.

a) 1^2+2^2+3^2……+n^2

section .data

```
    n equ 10
section .bss
    result resd 1
section .text
    global _start
start:
    mov eax, 0
    mov ecx, 1
sum_loop:
    cmp ecx, n
    jg end_program
 mov ebx, ecx
    imul ebx, ecx
    add eax, ebx
     inc ecx
    jmp sum_loop
end_program:
    mov [result], eax
   mov eax, 1
    xor ebx, ebx
    int 0x80
```

b) INTERRUPT VECTOR:

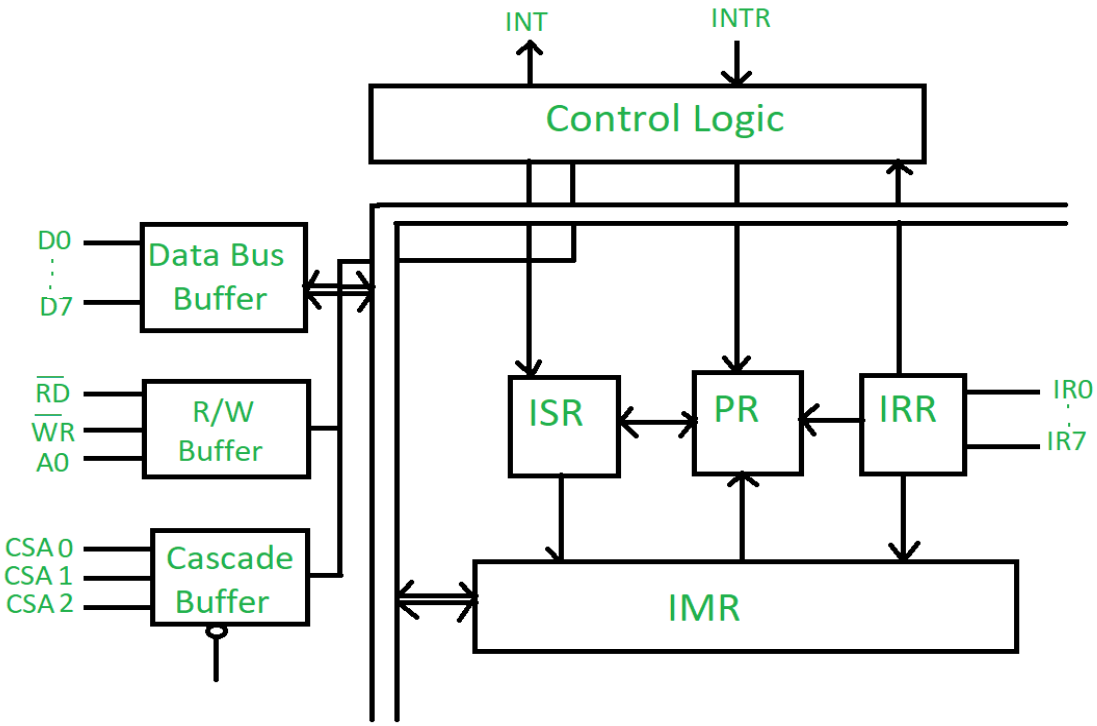
i) TYPE 0: This could represent an interrupt vector associated with a specific "type 0" event. In traditional interrupt handling, type 0 interrupts are often reserved for processor-specific purposes, like division by zero or other exceptional conditions.

ii) TYPE 1: The text "TYPE 1" appears to be a mix of Greek letters and numbers. It's not a standard representation for an interrupt vector. If we assume it's a typo and should be "TYPE 1", then it could represent an interrupt vector associated with a "type 1" event. In some systems, type 1 interrupts are used for hardware-related exceptions or traps.

iii) TYPE4: This could represent an interrupt vector associated with a "type 4" event. Like the previous examples, the exact meaning would depend on the specific system and its interrupt handling scheme.

iv) TYPE 5: Similarly, this could represent an interrupt vector associated with a "type 5" event. The exact purpose would be determined by the system's design and configuration.

c)8259A



It contains 3 registers commonly known as ISR, IRR, IMR & there is 1 priority resolver (PR).

1. **Interrupt Request Register (IRR):** It stores those bits which are requested for their interrupt services.

2. **Interrupt Service Register (ISR):** It stores the interrupt levels which is currently being served.
3. **Interrupt Mask Register (IMR):** It stores interrupt levels that have to be masked. These interrupt levels are already accepted by the 8259 microprocessor.

Priority Resolver (PR): It examines all the 3 registers and sets the priority of interrupts and sets the interrupt levels in ISR which has the highest priority and the rest of the interrupt bit is IRR which is already accepted.
SP/EN (low active pin): If its value is 1 it works in master mode & if its value=e is 0 then it works in slave mode.

6.

a) To solve this address mismatch problem and interface the 2716 EPROM with the 8086 microprocessor, we need to use memory decoding techniques. The basic idea is to use additional logic to enable the EPROM only for specific address ranges that the 2716 can handle. The rest of the address space is left unconnected or mapped to other memory or I/O devices.

Example:

Assume that we want to connect the 2716 EPROM, which stores some program or data, to the 8086 microprocessor. The 2716 EPROM has a capacity of 4 KB, and we will use memory addresses in the range 0x00000 to 0x00FFF (0 to 4095) for the EPROM.

Step 1: Address Decoding

First, we need to design a circuit that decodes the appropriate address range (0x00000 to 0x00FFF) from the 8086's address bus and enables the EPROM for that range. One common way to achieve this is by using a combination of logic gates (AND, OR, NOT) and a chip called a decoder.

Step 2: Enable Signal Generation

We'll use the lower 12 bits of the 8086's address bus as inputs to the decoder circuit. The decoder will produce an output signal when the input address falls within the range of the EPROM (0x00000 to 0x00FFF). Let's call this signal "EPROM_CS" (EPROM Chip Select).

Step 3: EPROM Connection

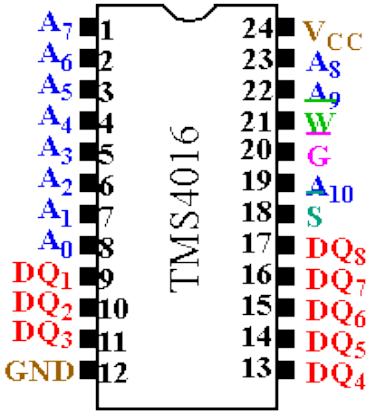
Connect the EPROM's Chip Select (CS) pin to the "EPROM_CS" signal generated by the decoder. This means the EPROM will be enabled only when "EPROM_CS" is active (low or high, depending on the EPROM's specifications).

Step 4: Address Mapping for Unconnected Memory

For the remaining address space that the EPROM doesn't cover (0x01000 to 0xFFFFF), you can either leave it unconnected or map it to other memory or I/O devices as required for your system design.

By following these steps, you can successfully interface the 2716 EPROM with the 8086 microprocessor, solving the address mismatch problem.

b)4016 SRAM



2K x 8 SRAM

Pin(s)	Function
A_0-A_{10}	Address
DQ_0-DQ_7	Data In/Data Out
$\overline{S}$ ($\overline{CS}$)	Chip Select
$\overline{G}$ ($\overline{OE}$)	Read Enable
$\overline{W}$ ($\overline{WE}$)	Write Enable

c)Limitations of Real and Protected mode:

Real Mode Limitations:

- Limited to 1 MB memory access.

- No memory protection or multitasking support.

- Cumbersome segment:offset addressing.

Protected Mode Features:

- 32-bit memory access (up to 4 GB).

- Memory protection for stability and security.

- Multitasking support with efficient context switching.

- Flat memory model and advanced instructions.

- Virtual memory and paging for efficient memory management.
- I/O privilege levels (rings) for security.

d) 80286 vs 80386:

80286 microprocessor	80386 microprocessor
Intel 80286 was developed in 1983 by Intel and it is the improved version of 80186.	Intel 80386 was developed in 1985 usi CHMOS III technology and it is the improved version of 80286.
80286 is a 16-bit processor.	80386 is the first 32-bit processor.
It is available with 134K transistors in a 68 Pin PGA package.	It is available with 275K transistors in a 132-pin PGA package.
80286 has operating frequency 8 MHz to 12.5 MHz.	80386 operates at clock speed of 16 MHz to 33 MHz.
The 80286 has a 16-bit bus and a 24-bit address bus.	The 80386 has a 32-bit data bus and a 32-bit address bus.
The 80286 has 24-bit address lines and can able to access 224 = 16 MB of physical memory.	The 80386 has 32-bit address lines and is al toaddress up to 232 = 4 GB physical memory.
The 80286 processor supports Intel 80287 numeric data processor.	The 80386 processor supports Intel 80387 numeric data processor.
The 80286 processor has a 6-byte prefetch queue.	The 80386 processor has a 16-byte prefetch queue.

7.

a) Storable I/O:

- Microprocessor transfers the data only when the peripheral is ready.

• Slow devices generate a control signal called strobe along with the valid data.

• Strobe signal indicates that valid data is present on the data line.

• The readiness of the peripheral is communicated to the microprocessor in two ways, by polling or interrupt

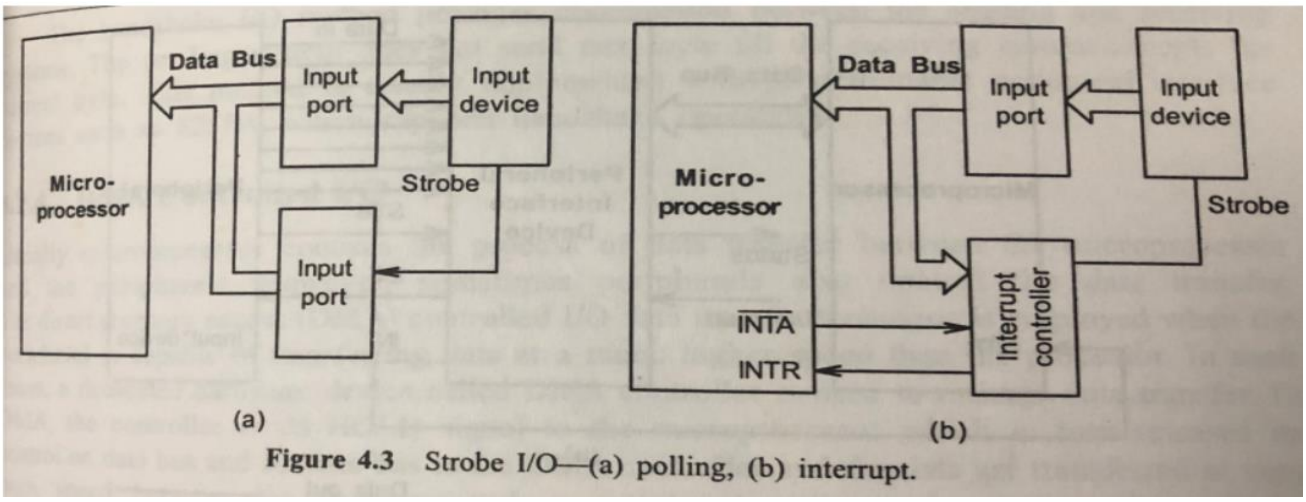


Figure 4.3 Strobe I/O—(a) polling, (b) interrupt.

Handshake I/O:

Microprocessor and peripheral devices exchange handshake signals to indicate the readiness of the peripherals and synchronize the timing of data transfer

• An interface device assists the exchange of data and handshake signals.

Handshake I/O

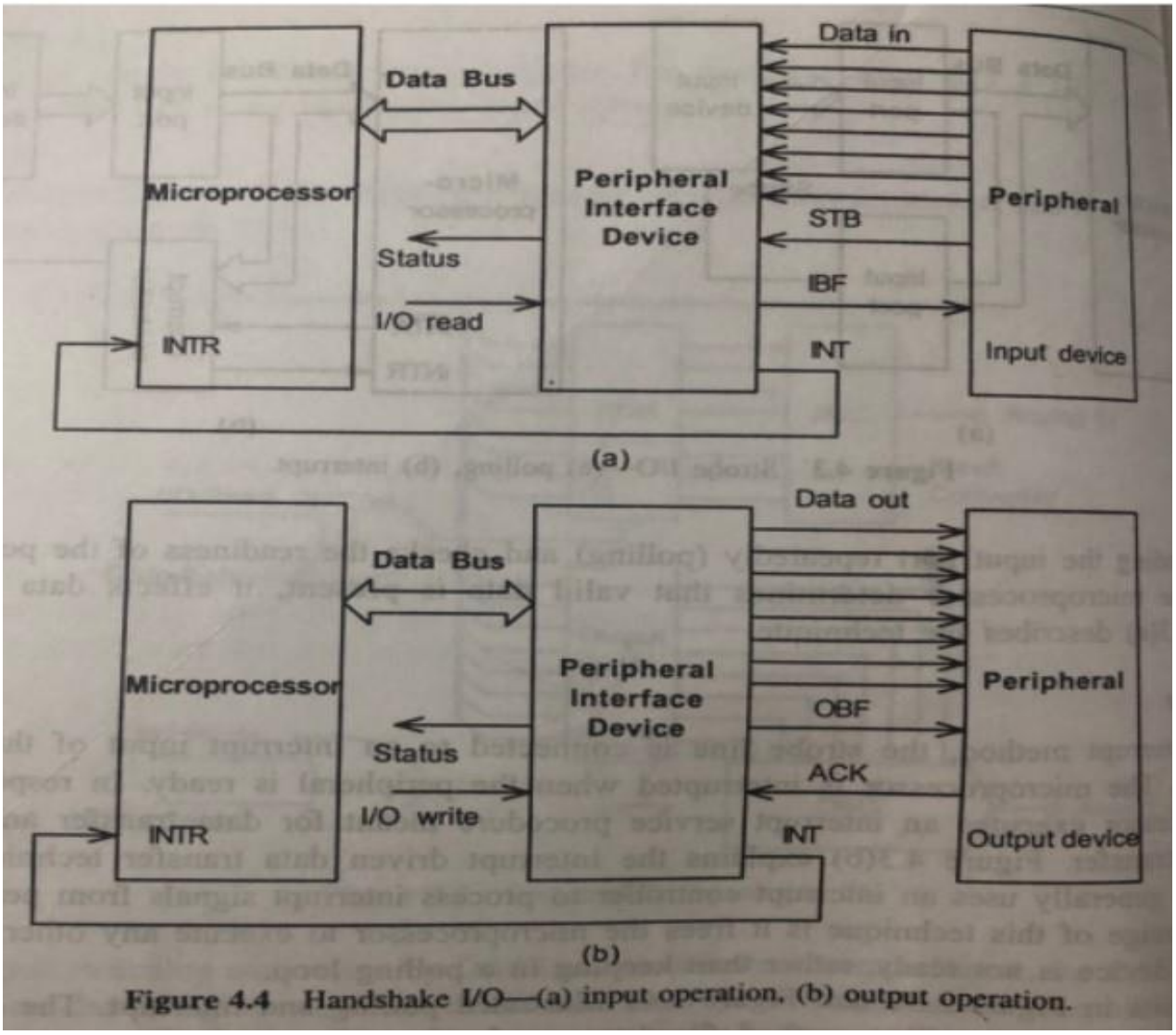
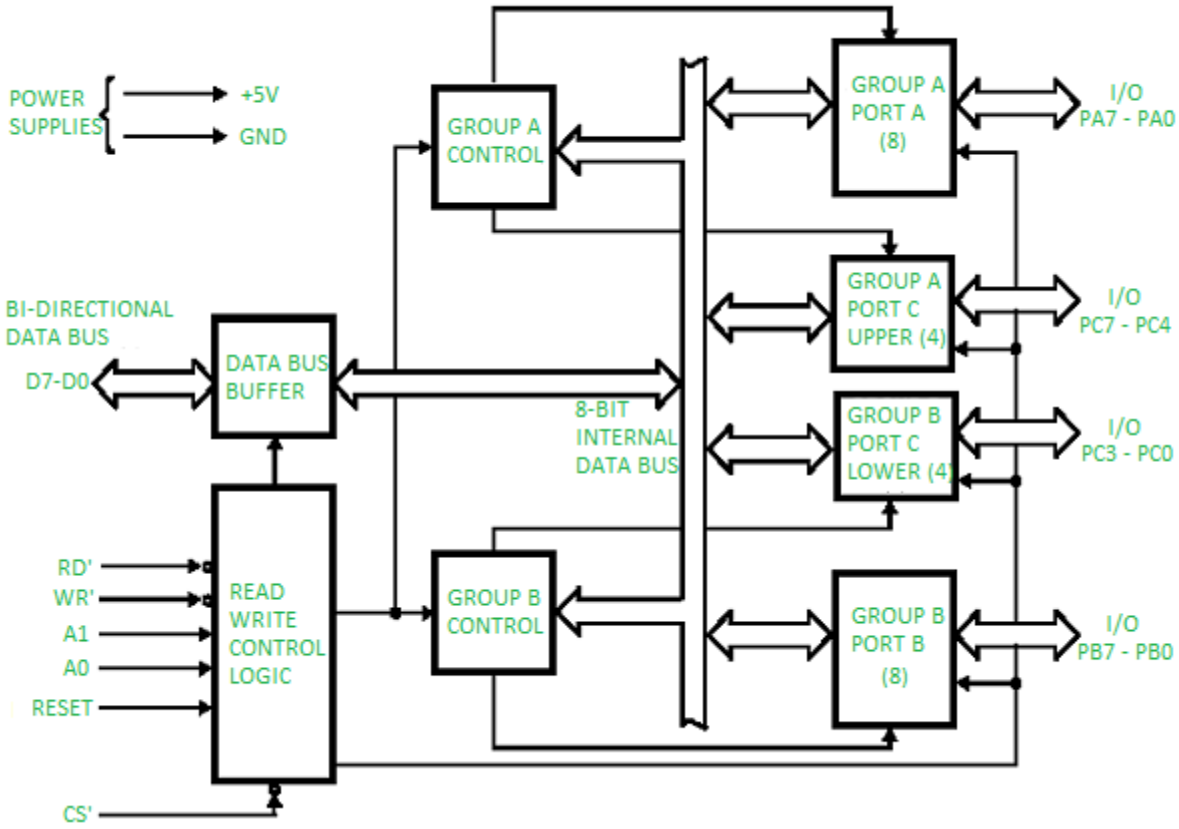
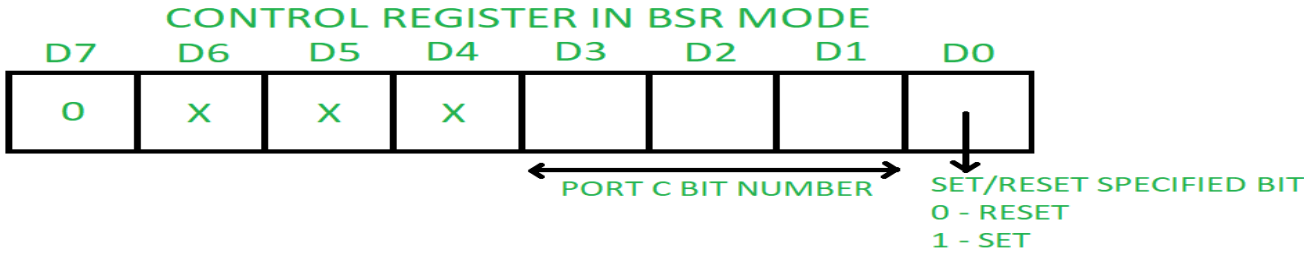


Figure 4.4 Handshake I/O—(a) input operation, (b) output operation.

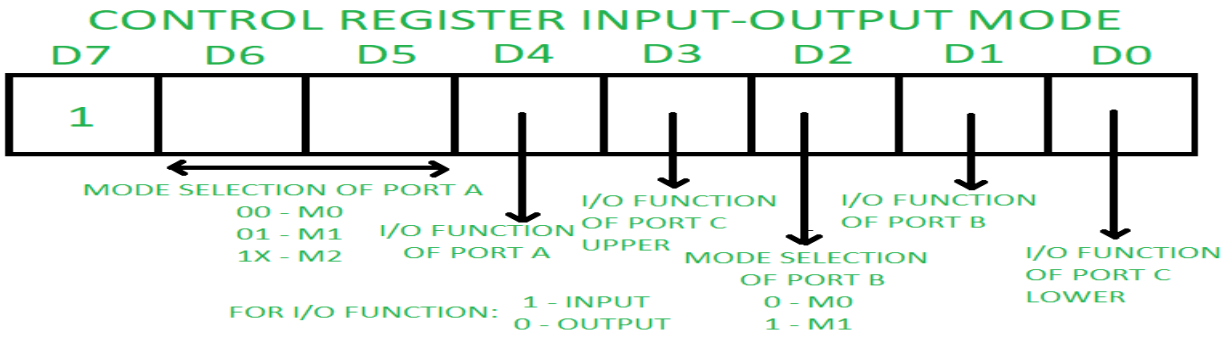
b)**82c55 modes operations:**



1. Bit set reset (BSR) mode – If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.



2. Input-Output mode – If MSB of control word (D7) is 1, PPI works in input-output mode. This is further divided into three modes:



- Mode 0 –In this mode all the three ports (port A, B, C) can work as simple input function or simple output function. In this mode there is no interrupt handling capacity.
- Mode 1 – Handshake I/O mode or strobed I/O mode. In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission. It has interrupt handling capacity and input and output are latched. Example: A CPU wants to transfer data to a printer. In this case since speed of processor is very fast as compared to relatively slow printer, so before actual data transfer it will send handshake signals to the printer for synchronization of the speed of the CPU and the peripherals.
- Mode 2 – Bi-directional data bus mode. In this mode only port A works, and port B can work either in mode 0 or mode 1. 6 bits port C are used as handshake signals. It also has interrupt handling capacity.

c)command byte of 82c55 PPI:

Each bit of the command byte corresponds to a specific configuration option or control setting. Here's a brief overview of the bits in the command byte:

Bit 7 (Mode Set/Reset, M):

- 0: Mode set. This allows you to program the 82C55 in various modes (mode selection).

- 1: Mode reset. When this bit is set to 1, the 82C55 enters the mode set state, and the next byte written to the chip is interpreted as a command byte to configure the mode and other settings.

Bit 6 (Group A Mode, G):

- 0: Group A is in mode 0 (basic input/output mode).

- 1: Group A is in mode 1 (strobed input/output mode).

Bit 5 (Group B Mode, C):

- 0: Group B is in mode 0 (basic input/output mode).

- 1: Group B is in mode 1 (strobed input/output mode).

Bit 4 (Port A Control, A):

- 0: Port A operates in input mode.

- 1: Port A operates in output mode.

Bit 3 (Port B Control, B):

- 0: Port B operates in input mode.

- 1: Port B operates in output mode.

Bit 2 (Port C Control, CH2):

- 0: Port C is in mode 0 (basic input/output mode).

- 1: Port C is in mode 1 (handshake output mode).

Bit 1 (Port C Control, CH1):

- 0: Port C is in mode 0 (basic input/output mode).

- 1: Port C is in mode 2 (strobed input/output mode).

Bit 0 (Port C Control, CH0):

- 0: Port C is in mode 0 (basic input/output mode).

- 1: Port C is in mode 3 (strobe input/output mode).

8.

A)

i) STOSB: Stores a byte from AL into memory pointed by DI and increments/decrements DI based on the direction flag (DF).

ii) MOVSB: Moves a byte from memory pointed by SI to memory pointed by DI and increments/decrements both SI and DI based on the direction flag (DF).

iii) SHL: Shifts the bits of a register/memory operand to the left by a specified number of positions, filling empty positions with zeros.

iv) RCR: Rotates the bits of a register/memory operand to the right through the carry flag, preserving the bit that is rotated out.

v) AAD: Adjusts the binary value in AL to valid unpacked BCD before division.

vi) LEA: Loads the effective memory address of the source operand into the destination register without accessing the memory content.

b)

String Reverse:

reverse a string

```
Data Segment
  str1 db 'This is a assembly language','$'
      strlen1 dw $-str1
  strrev db 20 dup(' ')
Data Ends
Code Segment
  Assume cs:code, ds:data
```

```
  Begin:
    mov ax, data
    mov ds, ax
    mov es, ax
    mov cx, strlen1
    add cx, -2
    lea si, str1
    lea di, strrev
    add si, strlen1
    add si, -2
    L1:
       mov al, [si]
       mov [di], al
       dec si
       inc di
       loop L1
       mov al, [si]
       mov [di], al
       inc di
       mov dl, '$'
       mov [di], dl
    Print:
       mov ah, 09h
       lea dx, strrev
       int 21h
Exit:
       mov ax, 4c00h
       int 21h
Code Ends
End Begin
```

c) Pentium processor:

5.0

Pentium processors have 64-bit data bus and 32-bit address bus. The Pentium processors can access a maximum of 4 GB of physical memory and a maximum of 64 TB of virtual memory. There are several versions of Pentium microprocessors operating at frequencies 66, 100, 133, 166, and 200 MHz. Features of a Pentium processor over a 486 processor are:

(a) Pentium processor has dual data pipelines. The two pipelines are designated as U-pipeline and V-pipeline. Each pipeline has its own ALU, address generators, data cache, etc. The two pipelines enable the processor to execute two instructions at a time. It is equivalent to having two 486 on a single chip. The microprocessor architecture which incorporates more than one execution unit is called *superscalar architecture*. The use of superscalar technology to execute more than one instruction at the same time is called *multithreading*. Multithreading is different from multitasking in the sense that in multithreading the processor actually performs two processes at the same instant, whereas in multitasking the processor performs only one process at an instant and presents an illusion of performing more than one process. The Pentium processor has another pipeline to execute floating-point instructions.

(b) The Pentium processor has two separate 8 KB caches, one for code and another for data. The data cache can be operated as either a write-through or a write-back cache.

(c) Pentium processors support the external L2 cache up to 512 KB.

(d) The Pentium processor has a new mode of operation called the System Memory Management (SMM) mode. The mode is accessed via the system memory management interrupt applied to the $\overline{\text{SMI}}$ input pin to the processor. In response to the interrupt, the microprocessor executes an interrupt service procedure at memory location 38000H

(e) The Pentium processor has Branch Target Buffer (BTB) and provided with branch prediction ability. It predicts the data required in future and gets it from memory or hard disk and keeps ready in a prefetch buffer. It enables the processor to keep both pipelines operating at full speed.

(f) The paging unit of Pentium allows 4 MB pages instead of 4 KB pages.

(g) It includes on-board power management features.

(h) The Pentium processor also has built-in math co-processor.