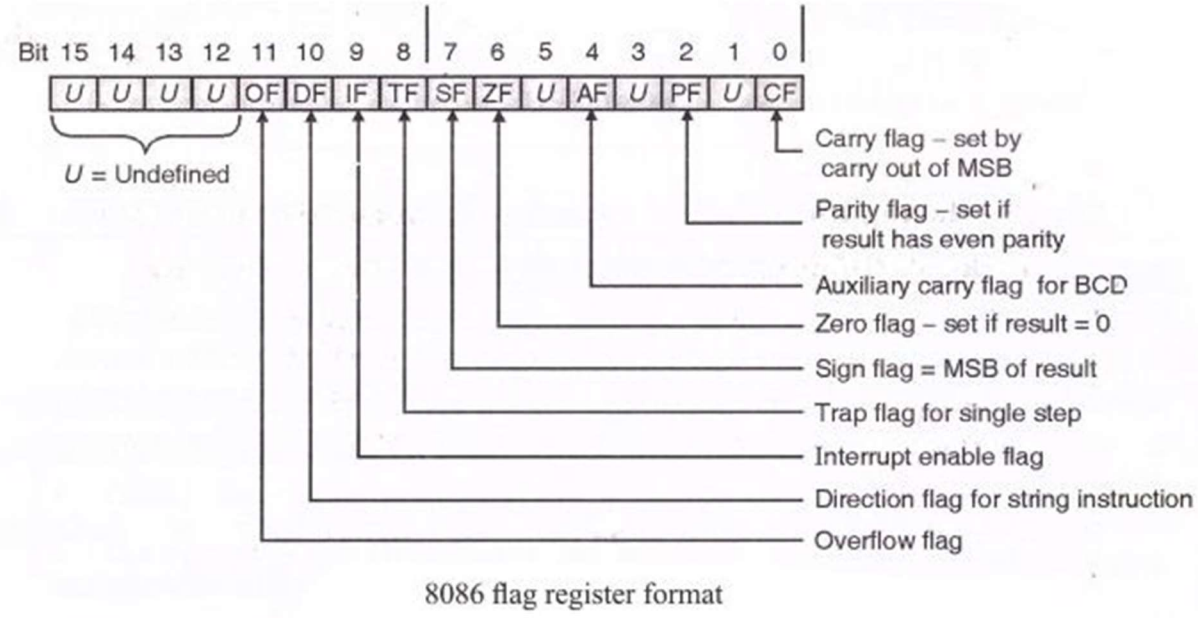
MICROPROCESSOR

MODULE 1. The Intel Microprocessor 8086 Architecture.

1. Explain Flag Register of 8086 Architecture.

1. A 16 bit flag register is used in 8086. It is divided into two parts. 1. Condition code or status flags AND 2. Machine or control flags.
2. The condition code flag register is the lower byte of the 16-bit flag register. The condition code flag register is identical to 8085 flag register, with an additional overflow flag.
3. The control flag register is the higher byte of the flag register. It contains three flags namely direction flag (D), interrupt flag (I) and trap flag (T).

Figure below shows the details of the 16 bit flag register of 8086 CPU.



8086 flag register format

It consists of 9 active flags out of 16. The remaining 7 flags marked 'U' are undefined flags. These 9 flags are of two types:

- 6 Status flags
- 3 Control flags

Status flags:

1. Carry flag (CY)-

- It is set whenever there is a carry or borrow out of the MSB (most significant bit) of a result. D7 bit for an 8 bit operation and D15 bit for a 16 bit operation.

2. Parity flag (PF)-

- It is set if the result has even parity. If parity is odd, PF is reset.
- This flag is normally used for data transmission errors.

3. Auxiliary carry flag (AC)-

- It is set if a carry is generated out of the lower nibble.
- It is used only in 8 bit operations like DAA and DAS.

4. Zero flag (ZF)-

- It is set if the result is zero.

5. Sign flag (SF)-

- It is set if the MSB of the result is 1. For signed operations such a number is treated as negative.

6. Overflow flag (OF)-

- It will be set if the result of a signed operation is too large to fit in the number of bits available to represent it.
- It can be checked using the instruction INTO (Interrupt on Overflow).

Control flags:

1. Trap flag (TF)-

- It is used to set the trace mode i.e. start single stepping mode.
- Here the microprocessor is interrupted after every instruction so that the program can be debugged.

2. Interrupt enable flag (IF)-

- It is used to mask (disable) or unmask (enable) the INTR interrupt.
- If user sets IF flag, the CPU will recognize external interrupt requests. Clearing IF disables these interrupts.

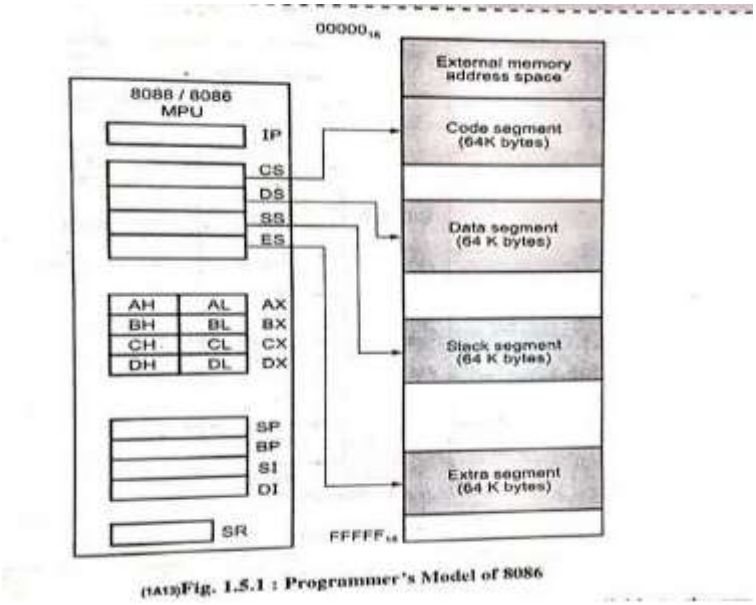
3. Direction flag (DF)-

- If this flag is set, SI and DI are in auto-decrementing mode in string operations.

2. Explain 8086 programming model.

The programming model of the 8086 is considered to be program visible because its registers are used during application programming and are specified by the instructions. below Figure illustrates the programming model of 8086 microprocessor.

Some registers are general-purpose or multipurpose registers, while some have special purposes. The multipurpose registers include AX, BX, CX, DX, BP, DI, and SI. These registers hold various data sizes (bytes, words, or double words) and are used for almost any purpose, as dictated by a program.



(DATA)Fig. 1.5.1 : Programmer's Model of 8086

Segment Registers

Additional registers, called segment registers, generate memory addresses when combined with other registers in the microprocessor. Following is a list of each segment register, along with its function in the system:

1. CS (code)

- The code segment is a section of memory that holds the code used by the microprocessor. The code segment register defines the starting address of the section of memory holding code.

2. DS (data)

- The data segment is a section of memory that contains most data used by a program.

3. ES (extra)

- The extra segment is an additional data segment that is used by some of the string instructions to hold destination data.

4. SS (stack)

- The stack segment defines the area of memory used for the stack.

Multipurpose Registers

1. AX (accumulator)

- AX is referenced as a 32-bit register (AX), as a 16-bit register (AX), or as either of two 8-bit registers (AH and AL).

2. BX (base index)

- BX is addressable as EBX, BX, BH, or BL. The BX register sometimes holds the offset address of a location in the memory system in all versions of the microprocessor. In the 80386 and above, BX also can address memory data.

3. CX (count)

- CX is a general-purpose register that also holds the count for various instructions.

4. DX (data)

- DX is a general-purpose register that holds a part of the result from a multiplication or part of the dividend before a division. In the 80386 and above, this register can also address memory data.

5. BP (base pointer)

- BP points to a memory location in all versions of the microprocessor for memory data transfers.

6. DI (destination index)

  - DI often addresses string destination data for the string instructions. It also functions as 16-bit (DI) general-purpose register.

7. SI (source index)

- The source index register often addresses source string data for the string instructions.

Special-purpose Registers.

The special-purpose registers include IP, SP; and the segment registers CS, DS, ES, SS.

1. IP (instruction pointer)

- IP addresses the next instruction in a section of memory defined as a code segment.

2. SP (stack pointer)

- SP addresses an area of memory called the stack. The stack memory stores data through this pointer.

3. Write short note on RESET signal in 8086 Architecture.

1. Reset causes the processor to immediately terminate its present activity.
2. To be recognized, the signal must be active high for at least four clock cycles, except after power-on which requires a 50 Micro Sec. pulse.
3. It causes the 8086 to initialize registers DS, SS, ES, IP and flags to all zeros. It also initializes CS to FFFF H. Upon removal of the RESET signal from the RESET pin, the 8086 will fetch its next instruction from the 20 bit physical address FFFF0H.
4. The reset signal to 8086 can be generated by the 8284. (Clock generation chip).
5. To guarantee reset from power-up, the reset input must remain below 1.5 volts for 50 Micro sec. after Vcc has reached the minimum supply voltage of 4.5V.

4. Explain the Memory Segmentation of 8086 microprocessor. Give its Advantages

Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast.

1. Overlapping Segment – A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts along with this 64kilobytes location of the first segment, then the two are said to be *Overlapping Segment*.
2. Non-Overlapped Segment – A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts before this 64kilobytes location of the first segment, then the two segments are said to be *Non-Overlapped Segment*.

    - It provides a powerful memory management mechanism.
    - Data related or stack related operations can be performed in different segments.
    - Code related operation can be done in separate code segments.
    - It allows to processes to easily share data.
    - It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.
    - It is possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

5. Explain memory banking in 8086.

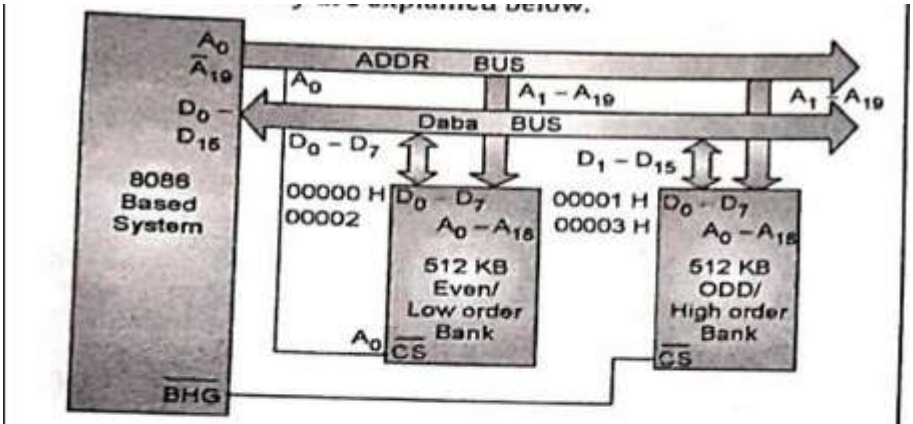
• The 8086 processor provides a 16-bit data bus. So, It is capable of transferring 16 bits in one cycle, but each the memory location is only of a byte(8 bits), therefore we need two cycles to access 16 bits(8 bit each) from two different memory locations.

• The solution to this problem is Memory Banking. Through Memory banking, our goal is to access two consecutive memory locations in one cycle(transfer 16 bits).

• The 1MB memory chip is equally divided into two parts(banks). One of the banks contains even addresses called Even bank and the other contains odd addresses called Odd bank.

• Even bank always gives lower byte So Even bank is also called Lower bank(LB) and Odd bank is also called a Higher bank(HB).

• This banking scheme allows accessing two aligned memory locations from both banks simultaneously and process 16-bit data transfer.



(a)

Memory location	Address	Data Type	BHE	A0	Bus Cycles	Data Lines Used
Even	00000	Byte	1	0	One	$D_0 - D_7$
Even	00000	Word	0	0	One	$D_0 - D_{15}$
Odd	00001	Byte	0	1	One	$D_8 - D_{15}$
Odd	00001	Word	0	1	First	$D_0 - D_7$
			1	0	Second	$D_8 - D_{15}$

6. Explain Minimum mode of 8086 microprocessor.

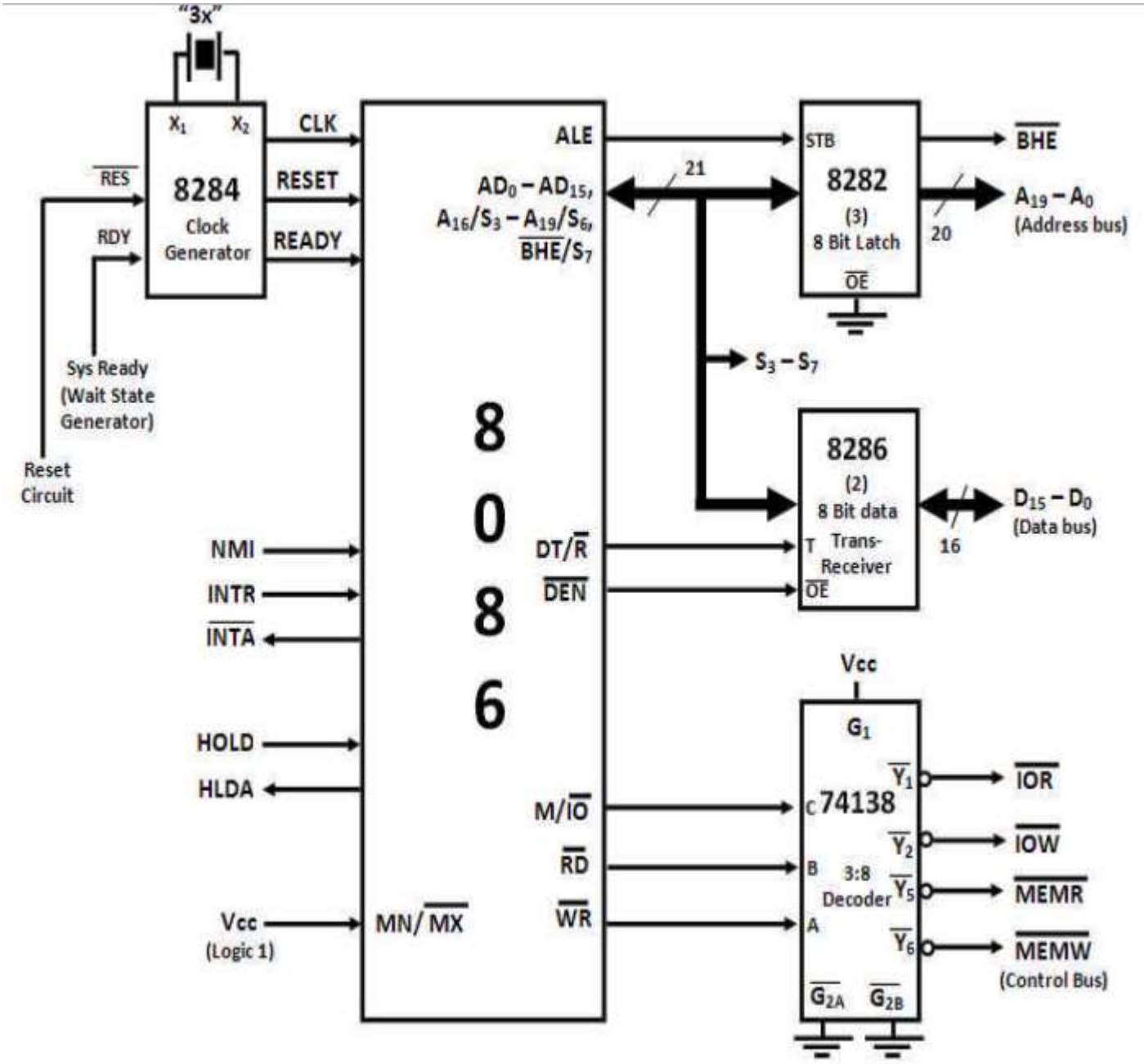
- 8086 works in Minimum Mode, when MN/ ¯MX = 1.
- Minimum Mode, 8086 is the only processor in the system. The Minimum Mode circuit of 8086 is as shown below:
- Clock is provided by the 8284 clock generator, it provides CLK, RESET and READY input to 8086.
- Address from the address bus is latched into 8282 8-bit latch. Three such latches are needed, as address bus is 20-bit. The ALE of 8086 is connected to STB of the latch. The ALE for this latch is given by 8086 itself.
- The data bus is driven through 8286 8-bit trans-receiver. Two such trans-receivers are needed, as the data bus is 16-bit. The trans-receivers are enabled through the DEN signal, while the direction of data is controlled by the DT/ ¯R signal. ¯DEN is connected to ¯OE and DT/ ¯R is connected to T. Both ¯DEN and DT/ ¯R are given by 8086 itself.

$\overline{DEN}$	DT/ $\overline{R}$	Action
1	X	Transreceiver is disabled
0	0	Receive data
0	1	Transmit data

- Control signals for all operations are generated by decoding M/¯IO , ¯RD and ¯WR signals.

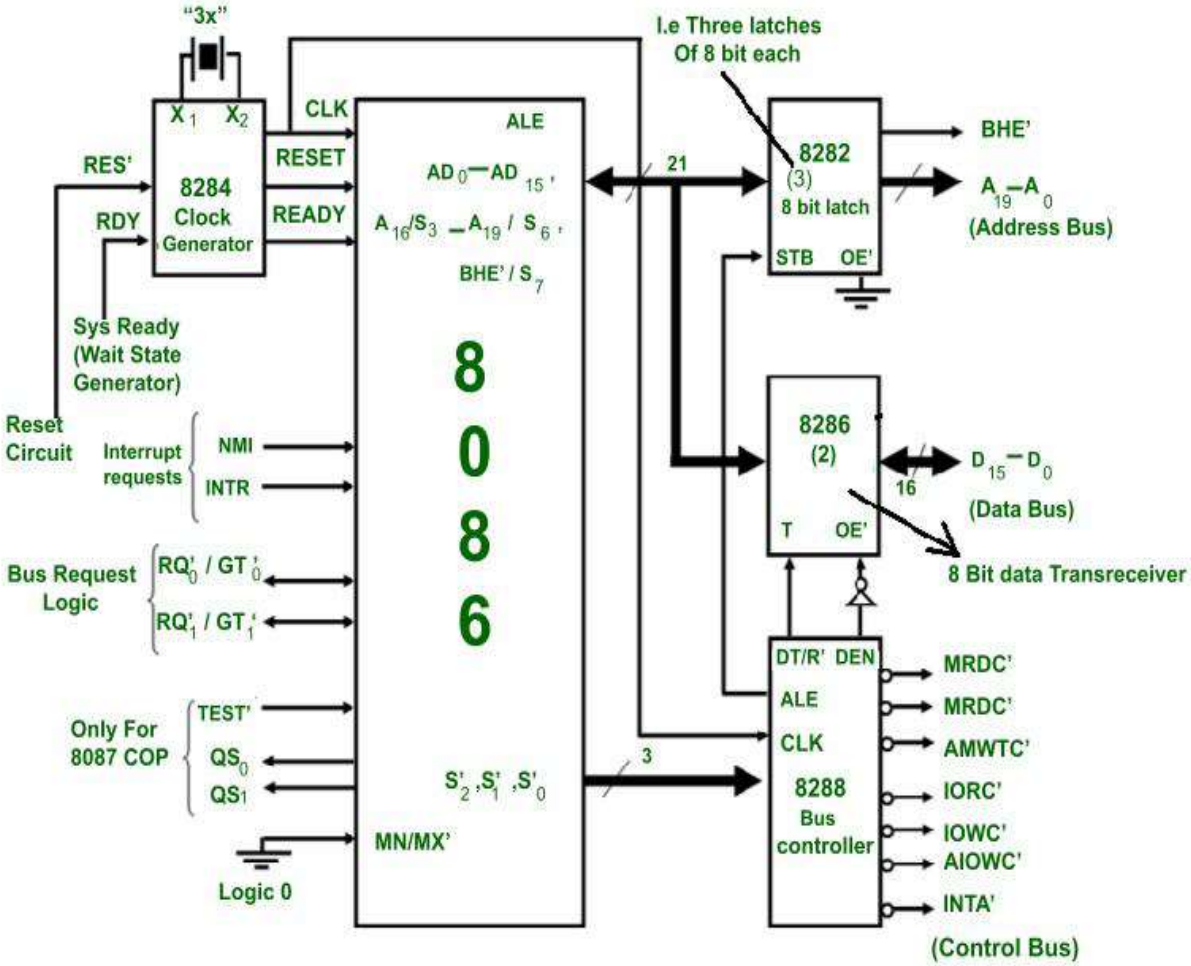
M/ $\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Action
1	0	1	Memory Read
1	1	0	Memory Write
0	0	1	I/O Read
0	1	0	I/O Write

- M/¯IO , ¯RD and ¯WR are decoded by a 3:8 decoder like IC 74138. Bus Request (DMA) is done using the HOLD and HLDA signals.
- ¯INTA is given by 8086, in response to an interrupt on INTR line.


"3x"
X1
X2
8284
Clock
Generator
CLK
RESET
READY
RES
RDY
Sys Ready
(Wait State
Generator)
Reset
Circuit
8
0
8
6
ALE
AD0 – AD15,
A16/S3 – A19/S6,
BHE/S7
21
STB
8282
(3)
8 Bit Latch
OE
BHE
A19 – A0
(Address bus)
20
S3 – S7
8286
(2)
8 Bit data
T Trans-
Receiver
OE
16
D15 – D0
(Data bus)
NMI
INTR
INTA
HOLD
HLDA
Vcc
(Logic 1)
MN/MX
DT/R
DEN
M/IO
RD
WR
Vcc
G1
C 74138
B
A
3:8
Decoder
Y1
Y2
Y5
Y6
G2A
G2B
IOR
IOW
MEMR
MEMW
(Control Bus)

7. Explain maximum mode of 8086 microprocessor.

- In this we can connect more processors to 8086 (8087/8089).
- 8086 max mode is basically for implementation of allocation of global resources and passing bus control to other coprocessor(i.e. second processor in the system), because two processors can not access system bus at same instant.
- All processors execute their own program.
- The resources which are common to all processors are known as global resources.
- The resources which are allocated to a particular processor are known as local or private resources.



Maximum mode circuit

Circuit explanation:

- When MN/ MX' = 0 , 8086 works in max mode.
- Clock is provided by 8284 clock generator.
- **8288 bus controller-** Address form the address bus is latched into 8282 8-bit latch. Three such latches are required because **address bus is 20 bit**. The ALE(Address latch enable) is connected to STB(Strobe) of the latch. **The ALE for latch is given by 8288 bus controller.**
- The data bus is operated through 8286 8-bit transceiver. Two such transceivers are required, because **data bus is 16-bit**. The transceivers are enabled the DEN signal, while the direction of data is controlled by the DT/R signal. DEN is connected to **OE'** and **DT/ R'** is connected to T. Both DEN and DT/ R' are given by 8288 bus controller.

DEN (Of 8288)	DT/ R'	Action
0	X	Transreceiver is disabled
1	0	Receive data
1	1	Transmit data

- Control signals for all operations are generated by decoding S'_2, S'_1 and S'_0 using 8288 bus controller.
- Bus request is done using RQ' / GT' lines interfaced with 8086. RQ_0/GT_0 has more priority than RQ_1/GT_1.
- INTA' is given by 8288, in response to an interrupt on INTR line of 8086.
- In max mode, the advanced write signals get enabled one T-state in advance as compared to normal write signals. This gives slower devices more time to get ready to accept the data, therefore it reduces the number of cycles.

8.

Q. Draw and discuss timing diagram for read operation in minimum mode of 8086. **(MU-Dec. 17, 5 Marks)**

Q. Draw and explain memory read machine cycle timing diagram in minimum mode of 8086. **(MU-Dec. 18, 5 Marks)**

- A machine cycle typically involves transferring a "single" value of data across the system.
- In a "read" machine cycle, data is transferred from memory or i/o to the μP.
- Memory read cycle transfers data from memory to μP, whereas I/O read transfers from I/O to μP.
- One machine cycle (also called bus cycle) involves 4 clock cycles also called T-states.
- 1 T-state = 1 clock cycle = 1 ÷ (clock frequency).
- Assume 8086 operating at 6 MHz, 1 T-state = 1 ÷ (6 MHz) = 0.167 microseconds or 167 nanoseconds.
- These are the activities performed in the 4 T-states

T_1 : ALE goes high as both multiplexed buses carry address.

This allows the latch to capture the address till ALE goes low.

T_2: μP asks for data by making the $\overline{\text{RD}}$ signal low.

As address is no longer present in the multiplexed bus, the transreceiver is enabled by $\overline{\text{DEN}} = 0$.

The data of course does reach the μP immediately.

Data is coming from memory hence it will take time to travel and reach the μP.

This time is called the "Propagation delay".

Please note that all these activities are only happening in nanoseconds.

T_3: Memory starts sending data on the data bus

Towards the end of T_3 μP checks Ready signal.

If Ready = 1 it means device is ready and μP completes the machine cycle in the next T-state (T4)

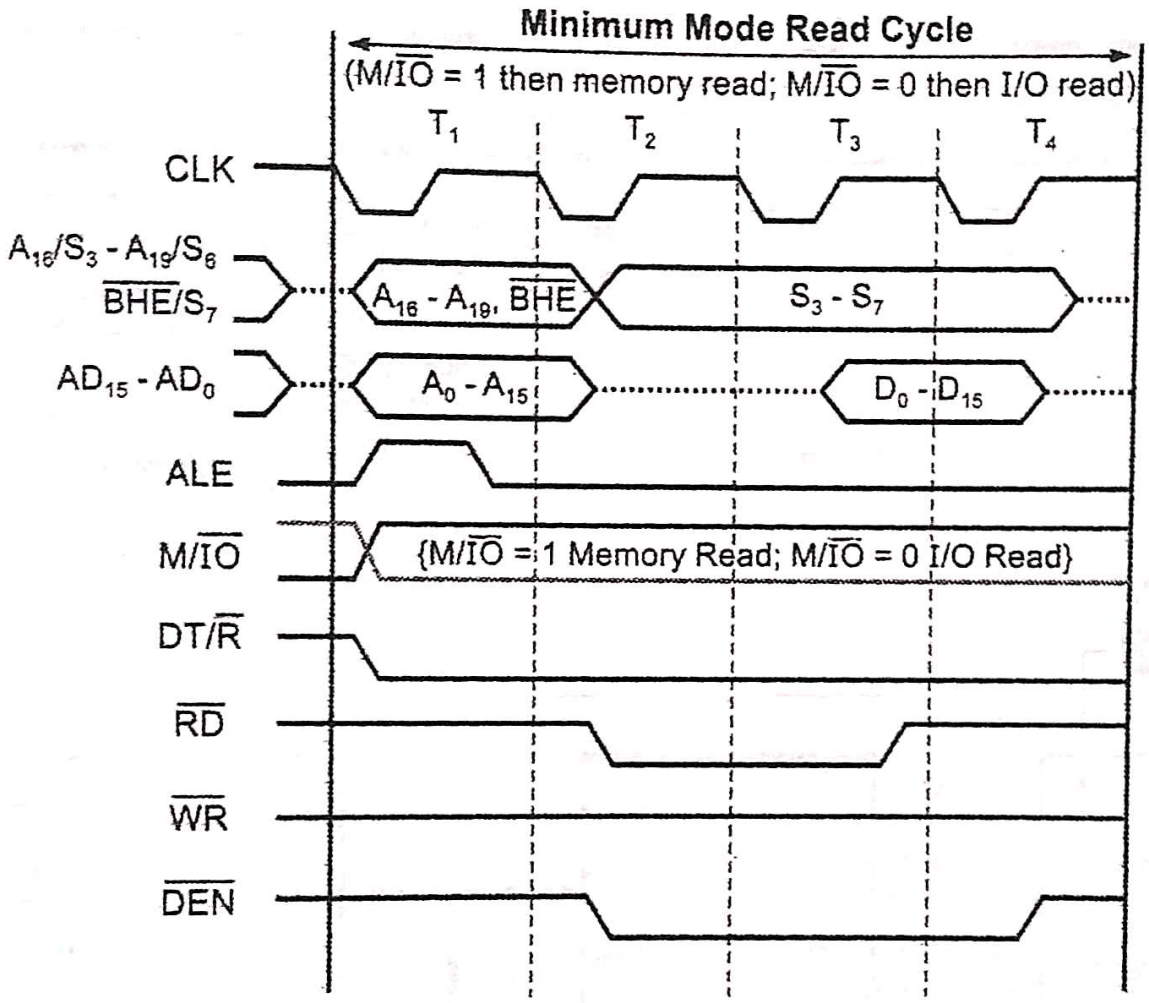
If Ready = 0 it means device is NOT ready

μP will then insert a wait state between T_3 and T_4

T_4: μP captures the data sent by the memory device

The multiplexed buses become tri-stated and will then open up in the next machine cycle

All control signals like $\overline{\text{RD}}$, $\overline{\text{DEN}}$ etc. are de-activated.



(1A28)**Fig. 1.12.2 : Minimum mode read cycle**

1.12.3 Timing Diagram of a "write" Machine Cycle

- A machine cycle typically involves transferring a "single" value of data across the system.
- In a "write" machine cycle, data is transferred from µP to memory or i/o.
- Memory write cycle transfers data from µP to memory, whereas I/O write transfers from µP to I/O.
- One machine cycle (also called bus cycle) involves 4 clock cycles also called T-states.

  1 T-state = 1 clock cycle = 1 ÷ (clock frequency).

  Assume 8086 operating at 6 MHz, 1 T-state = 1 ÷ (6 MHz) = 0.167 microseconds or 167 nanoseconds.

These are the activities performed in the 4 T-states

T_1: ALE goes high as both multiplexed buses carry address.

This allows the latch to capture the address till ALE goes low.

T_2: µP puts out the data on the data bus.

It informs the memory (or I/O device)that it wants to write (send) data.

This is done by making the $\overline{WR}$ signal low.

As address is no longer present in the multiplexed bus, the transreceiver is enabled by $\overline{DEN} = 0$.

T_3: Memory (or I/O) starts becoming ready and accept the data sent by µP

Towards the end of T_3 µP checks Ready signal.

If Ready = 1 it means device is ready and µP completes the machine cycle in the next T-state (T_4)

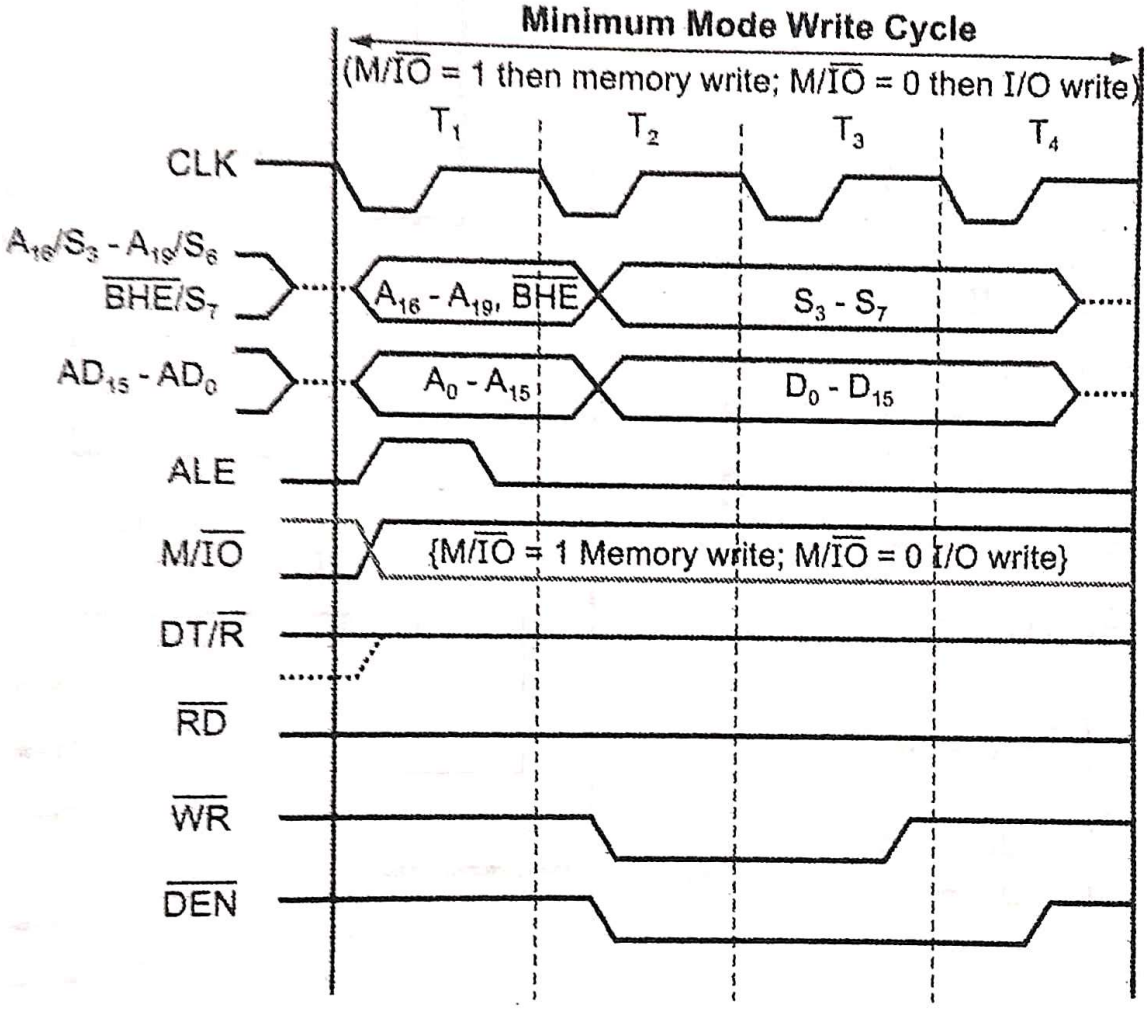
If Ready = 0 it means device is NOT ready

µP will then insert a wait state between T_3 and T_4

T_4: Data is captured by memory (or I/O device) and the operation is complete

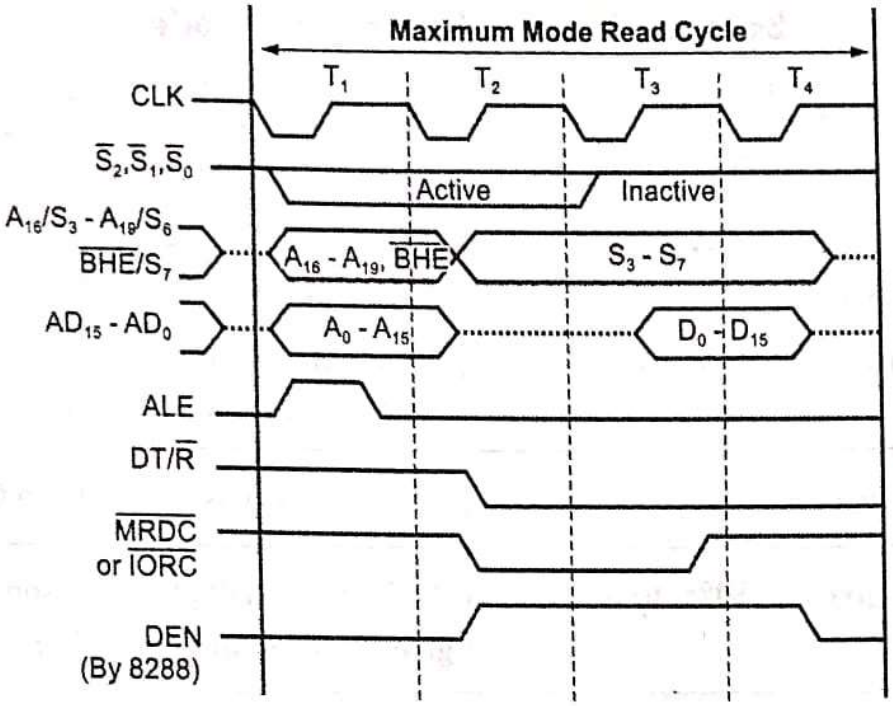
The multiplexed buses become tri-stated and will then open up in the next machine cycle

All control signals like $\overline{WR}$, $\overline{DEN}$ etc. are de-activated.

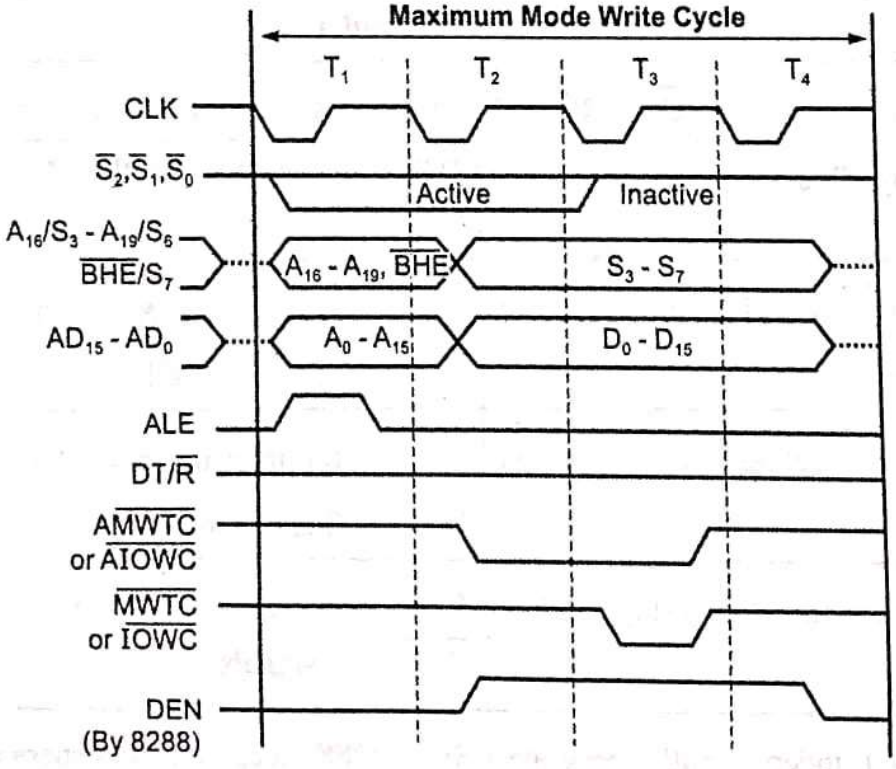


(1A29)**Fig. 1.12.3 : Minimum mode write cycle**

9.



(1A31)**Fig. 1.13.2 : Maximum Mode Read Cycle**



(1A32)**Fig. 1.13.3 : Maximum Mode Write Cycle**

10. Explain IVT Interrupt Vector Table.

1. The interrupt vector (or interrupt pointer) table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code. 8086 supports total 256 types i.e. 00H to FFH.

2. For each type it has to reserve four bytes i.e. double word.

3. The total interrupt vector table is divided into three groups namely,

A. Dedicated interrupts (INT 0…..INT 4)

B. Reserved interrupts (INT 5…..INT 31)

C. Available / User defined interrupts (INT 32…..INT 225)

A. **Dedicated interrupts (INT 0…..INT 4):**

1. **INT 0 (Divide Error)-**
   - This interrupt occurs whenever there is division error i.e. when the result of a division is too large to be stored. This condition normally occurs when the divisor is very small as compared to the dividend or the divisor is zero.
   - Its ISR address is stored at location 0 x 4 = 00000H in the IVT.
2. **INT 1 (Single Step)-**
   - The microprocessor executes this interrupt after every instruction if the TF is set.
   - It puts microprocessor in single stepping mode i.e. the microprocessor pauses after executing every instruction. This is very useful during debugging.
   - Its ISR generally displays contents of all registers. Its ISR address is stored at location 1 x 4 = 00004H in the IVT.
3. **INT 2 (Non mask-able Interrupt)-**
   - The microprocessor executes this ISR in response to an interrupt on the NMI (Non mask-able Interrupt) line.
   - Its ISR address is stored at location 2 x 4 = 00008H in the IVT.
4. **INT 3 (Breakpoint Interrupt)-**
   - This interrupt is used to cause breakpoints in the program. It is caused by writing the instruction INT 03H or simply INT.
   - It is useful in debugging large programs where single stepping is efficient.
   - Its ISR is used to display the contents of all registers on the screen. Its ISR address is stored at location 3 x 4 = 0000CH in the IVT.
5. **INT 4 (Overflow Interrupt)-**
   - This interrupt occurs if the overflow flag is set and the microprocessor executes the INTO (Interrupt on Overflow) instruction.
   - It is used to detect overflow error in signed arithmetic operations.
   - Its ISR address is stored at location 4 x 4 = 00010H in the IVT.

B. **Reserved interrupts (INT 5…..INT 31):**

1. These levels are reserved by Intel to be used in higher processors like 80386, Pentium etc. They are not available to the user.

C. **Available / User Defined interrupts (INT 32…..INT 225):**

1. These are user defined, software interrupts.

2. ISRs for these interrupts are written by the users to service various user defined conditions.
3. These interrupts are invoked by writing the instruction INT n. Its ISR address is obtained by the microprocessor from location n x 4 in the IVT.

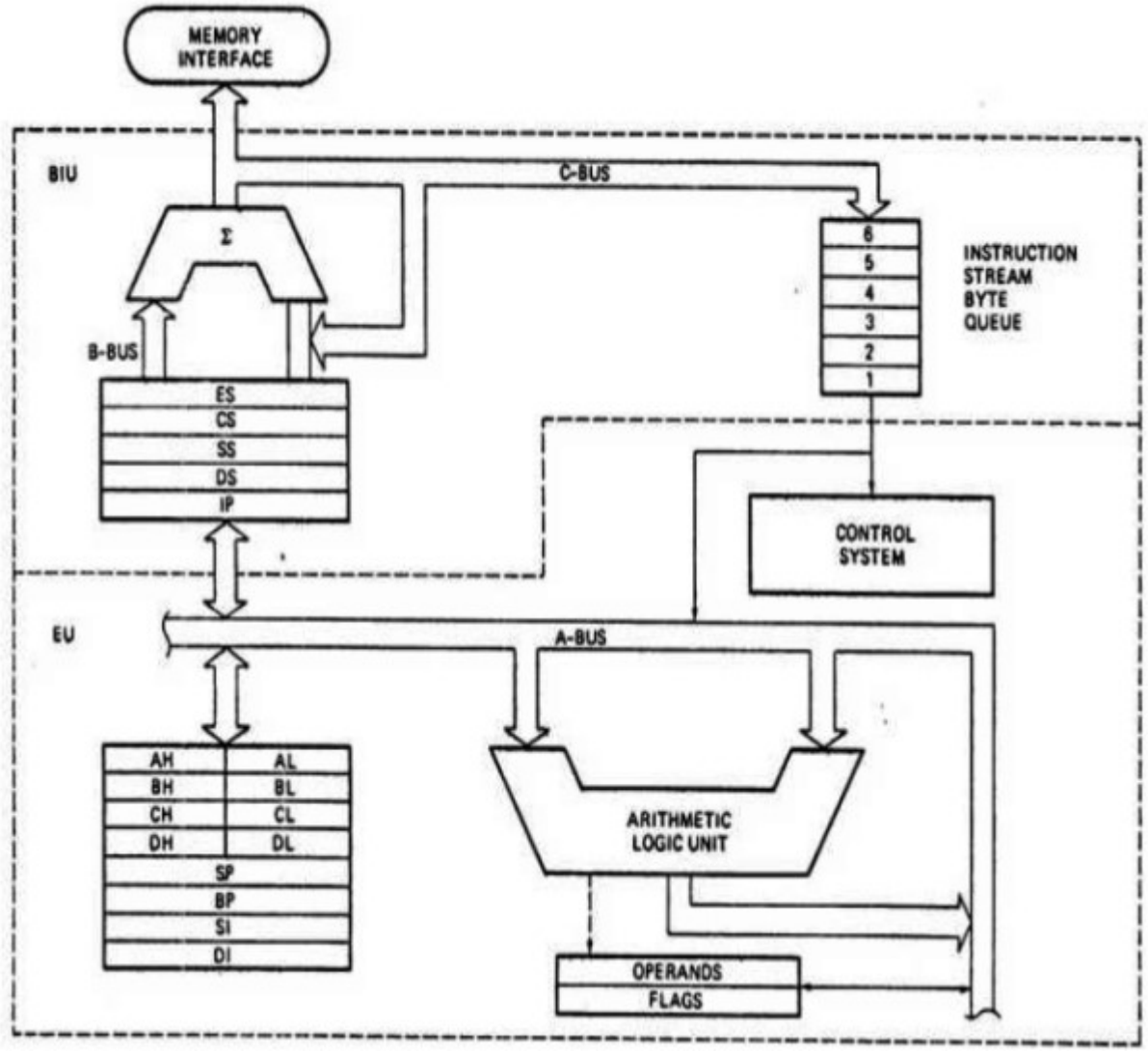
Hardware Interrupts:

1. **NMI (Non mask-able interrupt)-**
    - This is a non-mask-able, edge triggered, high priority interrupt.
    - On receiving an interrupt on NMI line, the microprocessor executes INT
    - Microprocessor obtains the ISR address from location 2 x 4 = 00008H from the IVT.
    - It reads 4 locations starting from this address to get the values for IP and CS to execute the ISR.
2. **INTR-**
    - This is a mask-able, level triggered, low priority interrupt.
    - On receiving an interrupt on INTR line, the microprocessor executes 2 INTA pulses.
    - 1st INTA pulse – The interrupting device calculates (prepares to send) the vector number.
      2nd INTA pulse – The interrupting device sends the vector number 'N' to the microprocessor.
    - Now microprocessor multiplies N x 4 and goes to the corresponding location in the IVT to obtain the ISR address. INTR is a mask-able interrupt.
    - It is masked by making IF = 0 by software through CLI instruction.
    - It is unmasked by making IF = 1 by software through STI instruction.

EXTRA QUE.

11. Explain the Architecture of 8086 with Block diagram.

ANS:



The architecture is divided into two units:

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

BUS INTERFACE UNIT (BIU)

- It provides the interface of 8086 to other devices.
- It operates w.r.t. Bus cycles. This means it performs various machine cycles such as Memory Read, IO Write etc to transfer data with Memory and I/O devices.
- It also generates the 20-bit physical address for memory access, fetches instruction from memory, transfers data to and from the memory and IO.
- It supports Pipelining using the 6-byte instruction queue.

The main components of the BIU are as follows:
SEGMENT REGISTERS:

- **CS Register:** CS holds the base (Segment) address for the Code Segment. All programs are stored in the Code Segment. It is multiplied by 10H (16d), to give the 20-bit physical address of the Code Segment. CS register cannot be modified by executing any instruction except branch instructions.
- **DS Register:** DS holds the base (Segment) address for the Data Segment. It is multiplied by 10H (16d), to give the 20-bit physical address of the Data Segment. Eg: If DS = 4321H then DS × 10H = 43210H è Starting address of Data Segment.
- **SS Register:** SS holds the base (Segment) address for the Stack Segment. It is multiplied by 10H (16d), to give the 20-bit physical address of the Stack Segment.
- **ES Register:** ES holds the base (Segment) address for the Extra Segment. It is multiplied by 10H (16d), to give the 20-bit physical address of the Extra Segment.

Instruction Pointer (IP register): It is a 16-bit register. It holds offset of the next instruction in the Code Segment. Address of the next instruction is calculated as CS x 10H + IP. IP is incremented after every instruction byte is fetched. IP gets a new value whenever a branch occurs.

Address Generation Circuit: The BIU has a Physical Address Generation Circuit. It generates the 20-bit physical address using Segment and Offset addresses using the formula: Physical Address (20 bit) = Segment Address (16 bit) X 10H + Offset Address (16 bit)

Pipelining: It is a 6-byte FIFO RAM used to implement Pipelining. Fetching the next instruction while executing the current instruction is called Pipelining. BIU fetches the next "six instruction-bytes" from the Code Segment and stores it into the queue. Execution Unit (EU) removes instructions from the queue and executes them.

Execution Unit (EU)

- It fetches instructions from the Queue in BIU, decodes and executes them.
- It performs arithmetic, logic and internal data transfer operations. It sends request signals to the BIU to access the external module. It operates in synchronization with T-States (clock cycles).

The main components of the EU are as follows:

a) General Purpose Registers: 8086 has four 16-bit general-purpose registers AX, BX, CX and DX. These are available to the programmer, for storing values during programs. Each of these can be divided into two 8-bit registers such as AH, AL; BH, BL; CL, CH and DL, DH. Beside their general use, these registers also have some specific functions.

- **AX Register (16-Bits):** It holds operands and results during multiplication and division operations. All IO data transfers using IN and OUT instructions use A register (AL/AH or AX). It functions as accumulator during string operations.
- **BX Register (16-Bits):** It holds the memory address (offset address), in Indirect Addressing modes.
- **CX Register (16-Bits):** It holds count for instructions like: Loop, Rotate, Shift and String Operations.
- **DX Register (16-Bits):** It is used with AX to hold 32 bit values during Multiplication and Division. It is used to hold the address of the IO Port in indirect IO addressing mode.

b) Special Purpose Registers

- **Stack Pointer (SP 16-Bits):** It is holds offset address of the top of the Stack. Stack is a set of memory locations operating in LIFO manner. SP is used with the SS Register to calculate physical address for the Stack Segment. It used during instructions like PUSH, POP, CALL, RET etc. During PUSH instruction, SP is decremented by 2 and during POP it is incremented by 2.
- **Base Pointer (BP 16-Bits):** It holds the offset address of any location in the stack segment. It is used to access random locations of the stack.
- **Source Index (SI 16-Bits):** It is normally used to hold the offset address for Data segment but can also be used for other segments using Segment Overriding. It holds offset address of source data in Data Segment, during String Operations.
- **Destination Index (DI 16-Bits):** It is normally used to hold the offset address for Extra segment but can also be used for other segments using Segment Overriding. It holds offset address of destination in Extra Segment, during String Operations.

c) ALU (16-Bits): It has a 16-bit ALU. It performs 8 and 16-bit arithmetic and logic operations.

d) Instruction Register and Instruction Decoder (Present inside the Control Unit): The EU fetches an opcode from the queue into the Instruction Register. The Instruction Decoder decodes it and sends the information to the control circuit for execution.

12. Define MICROPROCESSOR, Features of 8086.

ANS:

A microprocessor is a component that performs the instructions and tasks involved in computer processing. In a computer system, the microprocessor is the central unit that executes and manages the logical instructions passed to it.

SALIENT FEATURES OF 8086 MICROPROCESSOR

1. Single +5V power supply
2. Clock speed range of 5-10MHz
3. capable of executing about 0.33 MIPS (Millions instructions per second)
4. It is 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
5. It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
6. Fetch stage can pre-fetch up to 6 bytes of instructions and stores them in the queue.
7. It has 256 interrupts.

13. Differentiate between minimum and maximum mode of operation of 8086 microprocessor.

Minimum mode	Maximum mode
In minimum mode there can be only one processor i.e. 8086.	In maximum mode there can be multiple processors with 8086, like 8087 and 8089.
$MN/\overline{MX}$ is 1 to indicate minimum mode.	$MN/\overline{MX}$ is 0 to indicate maximum mode.
ALE for the latch is given by 8086 as it is the only processor in the circuit.	ALE for the latch is given by 8288 bus controller as there can be multiple processors in the circuit.
$\overline{DEN}$ and $DT/\overline{R}$ for the trans-receivers are given by 8086 itself.	and $DT/\overline{R}$ for the trans-receivers are given by 8288 bus controller.
Direct control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are given by 8086.	Instead of control signals, each processor generates status signals called $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$.
Control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are decoded by a 3:8 decoder like 74138.	Status signals $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ are decoded by a bus controller like 8288 to produce control signals.
$\overline{INTA}$ is given by 8086 in response to an interrupt on INTR line.	$\overline{INTA}$ is given by 8288 bus controller in response to an interrupt on INTR line.
HOLD and HLDA signals are used for bus request with a DMA controller like 8237.	$\overline{RQ}/\overline{GT}$,lines are used for bus requests by other processors like 8087 or 8089.
The circuit is simpler.	The circuit is more complex.
Multiprocessing cannot be performed hence performance is lower.	As multiprocessing can be performed, it can give very high performance.

MODULE 2. Instruction Set & Programming.

14. Explain different Addressing mode.

ANS:

3. Explain various addressing modes of 8086 microprocessor.

Ans. Addressing modes refer to the different methods of addressing the operands.

Addressing modes of 8086 are as follows:

1. Immediate Addressing mode:- In this mode, the operand is specified in the instruction itself. Instructions are longer but the operands are easily identified. Ex:. MOV CL, 12H
This instruction moves 12 immediately into CL register. CL ← 12H.

2. Register Addressing mode:- In this mode, operands are specified using registers. This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms. Registers may be used as source operands.

destination operands or both. EX. MOV AX, BX. ~~AX ← BX.~~
This instruction copies the contents of BX register into AX register

3. Direct Memory Addressing Mode:- In this mode, address of the operand is directly specified in the instruction. Hereonly the offset address is specified, the segment being indicated by the instruction. Ex. MOV CL, [4321H]
This instruction moves data from location 4321H in the data segment into CL. The physical address is calculated as.
DS * 10H + 4321
Assume DS = 5000H
∴ PA = 50000 + 4321 = 54321H ∴ CL ← [54321H]

4. Register based indirect Addressing Mode:- In this mode, the effective address of the memory may be taken directly from one of the base register or index register specified by instruction. If register is SI, DI and BX then DS is by default segment register. If BP is used, then SS is by default segment register.
EX. MOV CX, [BX]
This instruction moves a word from the address pointed by BX & BX+1 in data segment into CL and CH resp.
CL ← DS : [BX] and CH ← DS : [BX+1]
physical address can be calculated as DS * 10H + BX.

5. Register Relative Addressing Mode:- In this mode, the operand address is calculated using one of the base registers and an 8 bit or a 16 bit displacement. EX. MOV CL, [BX + 04H]
This instruction moves a bytes from the address pointed by BX+4 in data segment to CL CL ← DS: [BX + 04H]
Physical address can be calculated as DS * 10H + BX + 4H.

6. Base Indexed Addressing mode:- Here operand address is calculated as base register plus an index register.
EX. MOV CL, [BX + SI]
This instruction moves a byte from the address pointed by BX + SI in data segment to CL CL ← DS: [BX+SI]

physical address can be calculated as DS * 10H + BX + SI.

Relative based indexed addressing mode :-

In this mode, the address of the operand is calculated as the sum of base register, index register and 8 bit or 16 bit displacement.

EX. MOV CL, [BX + DI + 20H]

This instruction moves a byte from the address pointed by BX + DI + 20H in data segment to CL. CL ← DS : [BX + DI + 20H]

physical address can be calculated as DS * 10H + BX + DI + 20H

15. Explain the string instruction of 8286.

ANS:

4. Explain the string instruction in 8086 microprocessor.

Ans. 1. String is a series of bytes store sequentially in the memory

2. The source string is at location pointed by SI (source index) in data segment.

3. Destination string is at the location pointed by DI in extra segment.

1. MOVS (Move string) :-

1. It is used to transfer the string from data segment to extra segment.

2. String is a series of bytes store sequentially in the memory

3. The source string is at location pointed by SI in the data segment

Destination string is at the location pointed by DI in extra segment.

4. The offset of the source is SI.

5. The offset of the Destination is DI.

6. SI and DI incremented or decremented depending upon direction flag (DF).

EX. MOV SB

MOV SW.

2. LODS (Load String):

1. It is used to load AL or AX register which string from data segment.
2. string is a series of bytes store sequentially in memory
3. The source string is at location pointed by SI in data segment. Destination string is at the location pointed by DI in extra segment.
4. The offset of the source is SI and Destination is DI.

EX. LODSB
LODSW

3. STOS (Store string):

1. It is used to stroe string of AL or AX into extra segment
2. string is a series of bytes store sequentially in memory
3. The source string is at location pointed by SI in data segment. Destination string is at the location pointed by DI in extra segment.
4. The offeset of the source is SI and Destination is DI.

EX. STOSB
STOSW.

4. CMPS (compare string):

1. It is used to compare two string #2,3,4 points as it is from abo

EX: CMPSB
CMPSW

5. SCANS (scan string):

1. It is used to scan the string of AL or AX register in extra segment. #2,3,4 points as it is from 3rd point.

EX: SCANSB
SCANSW

6. REPZ/REPE and 7. REPNZ/REPNE :-

1. It is a conditional repeat instruction.
2. It is used only with string instruction.
3. After each execution. SI & DI are incremented or decremented.

16. Compare MACRO & PROCEDURE.

ANS:

5. Differentiate MACRO and PROCEDURE.

Ans.

MACRO	PROCEDURE
1. The uses of MACRO in the program never Alter the flow of execution.	1. The uses of PROCEDURE Alter the flow of execution.
2. It increases the size of program.	2. It does not increase the size of program.
3. MACRO call is process at translation time.	3. The PROCEDURE call are process at the execution time.
4. MACRO Never returns any value.	4. PROCEDURE may return the value.
5. MACRO needs an overhed of MACRO-PROCESSOR.	5. No such overhead is require in PROCEDURE.
6. It is reccommanded if lines of code is in the range of 5-7.	6. It is recommanded if the line of code is above 7.

7. EX. A1 DATA1 A1 DATA1
A1 DATA2 A1 DATA2
A1 DATA3 A1 DATA3
	7. Ex. int a,b,c;
	int add()
	{
	c = a+b;
	return c;
	}.

17. Explain Instruction Set of 8086.

ANS:

- Instruction Set :-

1. MOV Instruction :-
   - Syntax :- MOV Destination, Source.
   - Explanation :- It moves a byte or word from source to distination
   - Ex :- 1. MOV AX, 0036H
     2. MOV AX, BX

2. XCHG Instruction :-
   - Syntax :- XCHG Destination, Source.
   - Explanation :- It exchange a byte or word b/w source and distination.
   - Ex :- XCHG CX, BX

3. XLAT / XLATB Instruction :-
   - Explanation :- It moves data in to AL, the content of location in data segment in which effective address is formed by sum of BX and AL
   - Syntax :- AL ← DS [BX + AL] XLAT
   - Ex :-
     DS = 1000. BX = 0200 AL 06
     = DS x 10 + BX + AL
     AL ← 10200 H

4. LAHF Instruction :-
   - Explanation : This is a single type instruction it loads AH with lower byte of the flag register
   - Syntax : LAHF

Page No.
Date

5. ADD Instruction :-

- Syntax : ADD destination, source.
- Explaination : It ADD the content of source with the content of destination. The Result will be store in destination
- Ex : ADD AX, BX

It adds the context of BX in AX and story in AX.

```
MOV AX, 0001H
MOV BX, 0002H
ADD AX, BX
```

Content of: AX ; BX AX ← AX + BX
0003H 0002H

6. SUB Instruction :-

- syntax : SUB destination, source.
- Explaination : It SUB the source data to the destination result will be stored in destination

: Ex :

```
SUB AX, BX
MOV AX, 0004H
MOV BX, 0002H
SUB AX, BX
```

Content of AX ; 0002H AX ← AX - BX
BX ; 0002H

7. INC Instruction :-

- Syntax : INC Operand
- Explaination : It will increment the operand
- EX : INC BL

BL ← BL + 1

8 DEC Instruction :-

- syntax : DEC Operand.

Page No.
Date 30 | 1 | 23

- Explaination : It will decrement the operand.
- EX : DEC BL
  BL - 1

9. MUL instruction :-
- syntax : MUL Operand.
- Explaination : This is an multiplication instruction
  The result will be store in accumulator.
- EX : 8 bit
  MUL BL AL ← AL * BL.
  16 bit
  MUL BX DX.AX ← AX * BX

10. DIV instruction :-
- syntax : DIV operand
- Explaination : This is an division instruction
  The result will be store in accumulator.
- EX : DIV BL
  16 bit ÷ 8 bit
  AX ÷ BL
  AL ← Quotient
  AH ← Remainder

11. DAA instruction :- (Decimal Adjust After Addition)
1. It works only on AL register.
2. It is used when we bought to perform addition of 2 decimal no's.
3. We first enter two decimal no's. We add them using a normal add instruction.
4. Then we perform DAA instruction.
   It will adjust decimal after addition

DAA logic :-

1. If the lower four bit of AL is > 9, then Add 06 to AL.

2. If the higher four bit of AL is < 9, then Add 60 to AL.

EX. AL = 14H CL = 28H

Add AL, CL

AL = 3C H

Then DAA

```
   3 C   +   0 6
0011 1100
0000 0110
---------
0100 0010
   4    2   H.
```

2M

** 12. AAA Instruction: (ASCII Adjust After Addition)

1. It makes the result in unpacked BCD form.
2. In ASCII code [0-9] are represented by 30 --- 39
3. When we add we need to mask the higher byte
   EX. (3 of 39)
4. This can be avoided if we use AAA instruction after the addition.
5. AAA updates (A.F) flag (CF) flag

18. Write program to find out largest number in an array.

```
DATA SEGMENT
ARR DB 1, 4, 2, 3, 9, 8, 6, 7, 5, 10
LEN DW $-ARR
LARGE DB?
DATA ENDS
CODE SEGMENT
ASSUME DS: DATA CS: CODE
START:
    MOV AX, DATA
    MOV DS, AX
    LEA SI, ARR
    MOV AL, ARR [SI]
    MOV LARGE, AL
    MOV CX, LEN
REPEAT:
    MOV AL, ARR [SI]
    CMP LARGE, AL
    JG NOCHANGE
    MOV LARGE, AL
NOCHANGE:
    INC SI
    LOOP REPEAT
    MOV AH, 4CH
    INT 21H
CODE ENDS
END START
```

19. Write program to find the factorial of a number.

```
data segment
    num1 dw 0005h
    num2 dw 0001h
    fact dw 0001h

data ends

code segment

    assume ds:data cs:code

start:
      mov ax,data
      mov ds,ax

      mov ax,fact
      mov cx,num2

back:
     mul cx
     cmp cx,num1
     je skip
     inc cx
     jmp back

skip:
     mov fact,ax
     mov ah,4ch
     int 21h

end start
code ends
```

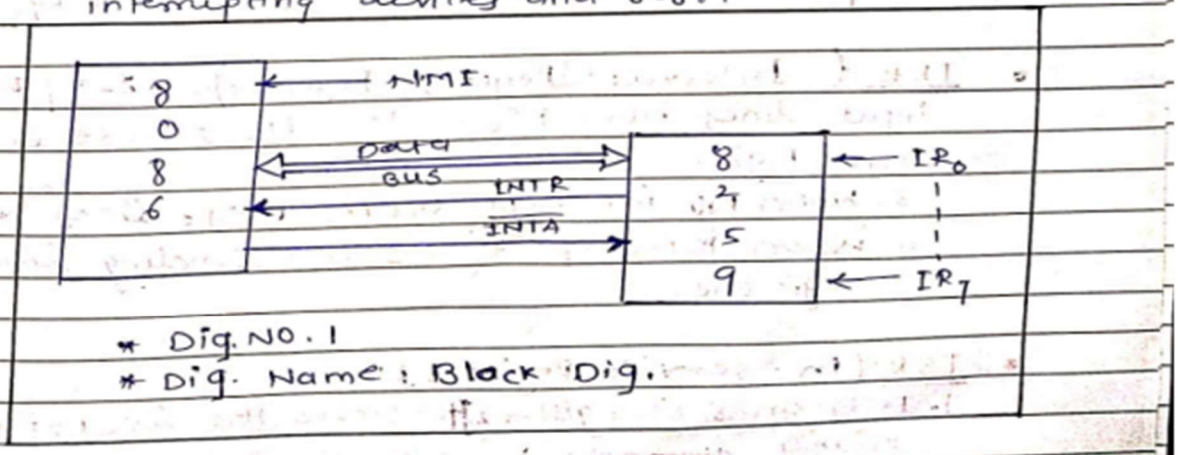
MODULE 3. MEMORY & PERIPHERALS INTERFACING.

20. Explain 8259 PIC (Programmable Interrupt Controller).

ANS:

• **8259 Programmable Interrupt controller ((8259) PIC)) :-**

1. 8259 is a programmable Interrupt controller.
2. It is used to increase the number of hardware interrupts for the processor.
3. single 8259 can provide 8 interrupt.
4. • **Need for 8259 :-**
   1. As 8086 has only two hardware interrupt ① NMI ② INTR
   2. Out of which NMI is vectored.
   3. This means it has a fix vector number in the IVT (Interrupt Vector table) i.e. two.
   4. On the other hand INTR is a non-vectored.
   5. This makes INTR a flexible interrupt line
   6. we can combine several interrupting devices into a common PIN like INTR.
   7. This is because the processor would never really understand which of the devices has sent the interrupt
   8. This is why we need 8259 as an interface beth interrupting devices and 8086.



* Dig. NO. 1
* Dig. Name : Block Dig.

5. • **Features of 8259 :-** 1. It is programmable interrupt controller that can work with 8085, 8086.
   2. It is used to increase NO. of interrupt.
   3. A single 8259 provides 8 interrupt.
   4. Priority of interrupt is flexible

21. Explain 8255 PPI (Programmable Peripheral Interface).

ANS:

5M Q. Write a short not of 8255 PPI (programmable peripheral Interface).
OR
Q. Explain with block diagram working of 8255 (PPI).

- 8255 (PPI) :-
  1. 8255 is programmable peripheral Interface (PPI).
  2. It is used to connect (Interface) microprocessor with I/O devices such as printer, keyboard.
  3. 8255 has three, 8 bit codes, bidirectional I/O codes called as port A ; port B, port C.
  4. They can be used as Input port or as ouput port
  5. These ports can operates in three different modes of data transfer.
  6. 1. Mode zero it called as simple I/O device.
     2. Mode one it used advanced technology for data transfer.
     3. Mode two it performs I/O data transfer as well as

* Data Bus Buffer :-
  1. It is a 8 bit bidirectional bus used to connect internal as well as external data bus.
  ~~2. Read~~

* Read/write control logic :-
  1. It accept Read, write signal

* RESET signal :-
  1. It is used to stop current data transfer. It address lines A_1 and A_0

A_1	A_0	
0	0	Part A
0	1	part B
1	0	part C
1	1	control word

22. Operating Modes of PIC 8259.

The various Operating Modes of 8259 Programmable Interrupt Controller are :

1. Fully Nested Mode of 8259,
2. Special Fully Nested Mode in 8259 (SFNM)
3. Rotating Priority Mode of 8259,
4. Special Mask Mode in 8259, and
5. Polled Mode in 8259.

1. Fully Nested Mode of 8259 (FNM): After initialization, the 8259A operates in fully nested mode so it is called default mode. The 8259 continues to operate in the Fully Nested Mode until the mode is changed through Operation Command Words. In this mode, IR_0 has highest priority and IR_7 has lowest priority. When the interrupt is acknowledged, it sets the corresponding bit in ISR. This bit will inhibit all interrupts of the same or lower level, however it will accept higher priority interrupt requests. The vector address corresponding to this interrupt is then sent. The bit in the ISR will remain set until an EOI command is issued by the microprocessor at the end of interrupt service routine. But if AEOI (Automatic End Of Interrupt) bit is set, the bit in the ISR resets at the trailing edge of the last INTA.

2. Special Fully Nested Mode in 8259 (SFNM): In the FNM, on the acknowledgement of an interrupt, further interrupts from the same level are disabled. Consider a large system which uses cascaded 8259s and where the interrupt levels within each slave have to be considered. An interrupt request input to a slave, in turn causes the slave to place an interrupt request to the master on one of the master's inputs. Further interrupts to the slave will cause the slave to place requests to the master on the same input to the master, but these will not be recognised because further interrupts on the same input level are disabled by the master.

3. Rotating Priority Mode of 8259: The Rotating Priority mode can be set in **(i) Automatic Rotation, and (ii) Specific Rotation.**

(i) Automatic Rotation

In this mode, a device, after being serviced, receives the lowest priority. The device just been serviced, will receive the seventh priority. Here IR_3 has just been serviced.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	5	6	7	0	1	2	3

(ii) Specific Rotation.

In the Automatic Rotation mode, the interrupt request last serviced is assigned the lowest priority, whereas in the Specific Rotation mode, the lowest priority can be assigned to any interrupt input (IR_0 to IR_7) thus fixes all other priorities.

For example if the lowest priority is assigned to IR_2, other priorities are as shown below.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
5	6	7	0	1	2	3	4

4. Special Mask Mode in 8259:

If any interrupt is in service, then the corresponding bit is set in ISR and the lower priority interrupts are inhibited. Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion. In these cases, we have to go for special mask mode. In the special mask mode it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked. Thus any interrupt may be selectively enabled by loading the mask register.

5. Polled Mode in 8259:

In this mode the INT output is not used. The microprocessor checks the status of interrupt requests by issuing poll command. The microprocessor reads contents of 8259A after issuing poll command. During this read operation the 8259A provides polled word and sets ISR bit of highest priority active interrupt request FORMAT.

I	X	X	X	X	W_2	W_1	W_0

I = 1 —> One or more interrupt requests activated.

I = 0 —> No interrupt request activated.

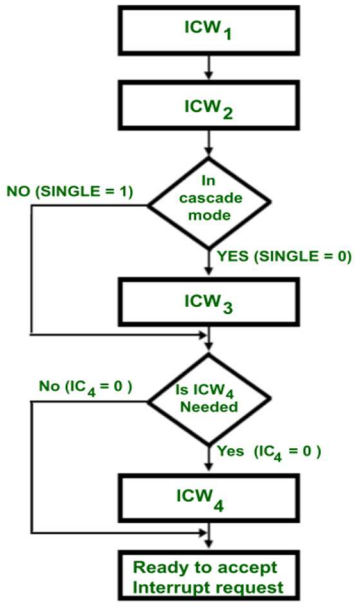
W_2 W_1 W_0 —> Binary code of highest priority active interrupt request.

23. Give formats of initialisation command words of 8259 PIC.

Initialization command words(ICW) :

- ICW is given during the initialization of 8259 i.e. before its start functioning.
- ICW_1 and ICW_2 commands are compulsory for initialization.
- ICW_3 command is given during a cascaded configuration.
- If ICW_4 is needed, then it is specified in ICW_1.
- The sequence order of giving ICW commands is fixed i.e. ICW_1 is given first and then ICW_2 and then ICW_3.
- Any of the ICW commands can not be repeated, but the entire initialization process can be repeated if required.

Initialization sequence of 8259 :



ICW_1 command :

- The control word is recognized as ICW_1 when $A_0 = 0$ and $D_4 = 1$.
- It has the control bits for Edge and level triggering mode, single/cascaded mode, call address interval and whether ICW4 is required or not.
- Address lines A_7 to A_5 are used for interrupt vector addresses.

ICW_2 command :

- The control word is recognized as ICW_2 when $A_0 = 1$.
- It stores the information regarding the interrupt vector address.
- In the 8085 based system, the A15 to A_8 bits of control word is used for interrupt vector addresses.
- In the 8086 based system, T_6 to T_3 bits are inserted instead of A_{15} to A_8 and A_{10} to A_8 are used for selecting interrupt level, i.e. 000 for IR_0 and 111 for IR_7.

ICW_3 :
ICW_3 command word is used when there is more than one 8259 present in the system i.e. when SNGL bit in ICW_1 is 0, then it will load 8-bit slave register.

ICW_4 :

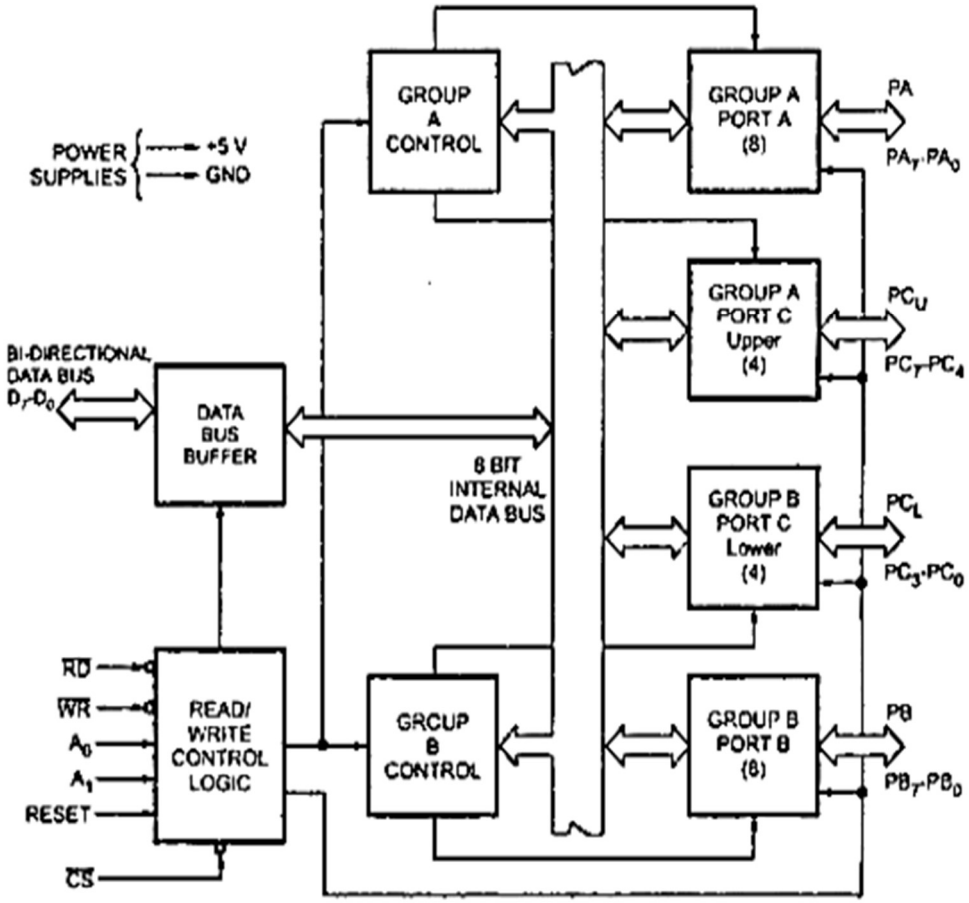
- When AEOI = 1, then Automatic end of interrupt mode is selected.
- When SFMN = 1, then a special fully nested mode is selected.
- when BUF = 0 , then Non buffered mode is used (i.e. M/S is don't care) and when M/S = 1, then 8259 is master, otherwise it is a slave.
- when µPM = 1, then 8086 operations are performed, otherwise 8085 operations are performed.

24. Explain the operation of three 8259 PIC in cascaded mode

1. When more than one 8259s are connected to the microprocessor, it is called as a cascaded configuration.
2. A cascaded configuration increases the number of interrupts handled by the system.
3. As the maximum number of 8259s interfaced can be 9 (1 master and 8 slaves) the maximum number of interrupts handled can be 64.
4. The master 8259 has $\overline{SP}/\overline{EN} = +5V$ and the slave has $\overline{SP}/\overline{EN} = 0V$.
5. Each slave's INT output is connected to the IR input of the master.
6. The INT output of the master is connected to the INTR input of the microprocessor.
7. The master addresses the individual slaves through CAS2, CAS1 and CAS0 lines connected from the master to each of the slaves.
8. First the INTR signal of the microprocessor should be enabled using the STI instruction.
9. Each 8259 (master or slave) has its own address and has to be initialized separately by giving ICWs as per requirement.
10. When an interrupt request occurs on a slave, the events are performed:
11. The slave 8259 resolves the priority of the interrupt and sends the interrupt to the master 8259.
12. The master resolves the priority among the slaves and sends the interrupt to the microprocessor.
13. The microprocessor finishes the current instruction and responds to the interrupt by sending 2 $\overline{INTA}$ pulses.
14. In response to the first $\overline{INTA}$ pulse, the following events occur-
    - The master sends the 3 bit slave identification number on the CAS lines.
    - The master sets the corresponding bit in its InSR.
    - The slave identifies its number on the CAS lines and sets the corresponding bit in its InSR.
15. In response to the second $\overline{INTA}$ pulse, the slave places vector number N on the data bus.
16. During the 2nd $\overline{INTA}$ pulse, the InSR bit of the slave is cleared in AEOI mode otherwise it is cleared by the EOI command at the end of the ISR.
17. The microprocessor pushes the contents of flag register, CS, IP in to the stack; clears IF and TF and transfers program to the address of the ISR.
18. The ISR thus begins.

25. Explain with block diagram working of 8255 PPI.

1. Figure shows the internal block diagram of 8255A. It consists of data bus buffer, control logic and Group A and Group B controls.
2. Data Bus Buffer: This tri-state bi-directional buffer is used to interface the internal data lilts of 8255 to the system data bus. Input or Output instructions executed by the CPU either Read date from or Write data into the buffer. Output data from the CPU to the ports or control register, and input data to the CPU from the ports or status register are all passed through the buffer.



Block Diagram of 8255

1. Control Logic: The control logic block accepts control bus signals as well as inputs from the address bus, and issues commands to the individual group control blocks (Group A control and Group B control). It issues appropriate enabling signals to access the required data/control words or status word. The input pins for the control logic section are described here.
2. Group A and Group B Controls: Each of the Group A and Group B control blocks receives control words from the CPU and issues appropriate commands to the ports associated with it. The Group A control block controls Port A and PC_7-PC_4 while the Group B control block controls Port B and PC_3-PC_0.
3. Port A: This has an 8-bit latched and buffered output and an 8-bit input latch. It can be programmed in three modes: mode 0, mode 1 and mode 2.
4. Port B: This has an 8-bit data I/O latch/ buffer and an 8-bit data input buffer. It can be programmed in mode 0 and mode 1.
5. Port C: This has one 8-bit unlatched input buffer and an 8-bit output latch/buffer. Port C can be spitted into two parts and each can be used as control signals for ports A and B in the handshake mode. It can be programmed for bit set/reset operation.

26. Draw and Explain the block diagram of 8257 DMA controller.

The functional Block Diagram of DMA controller (8257) is shown in Figure and the description are as follows: It consists of five functional blocks:

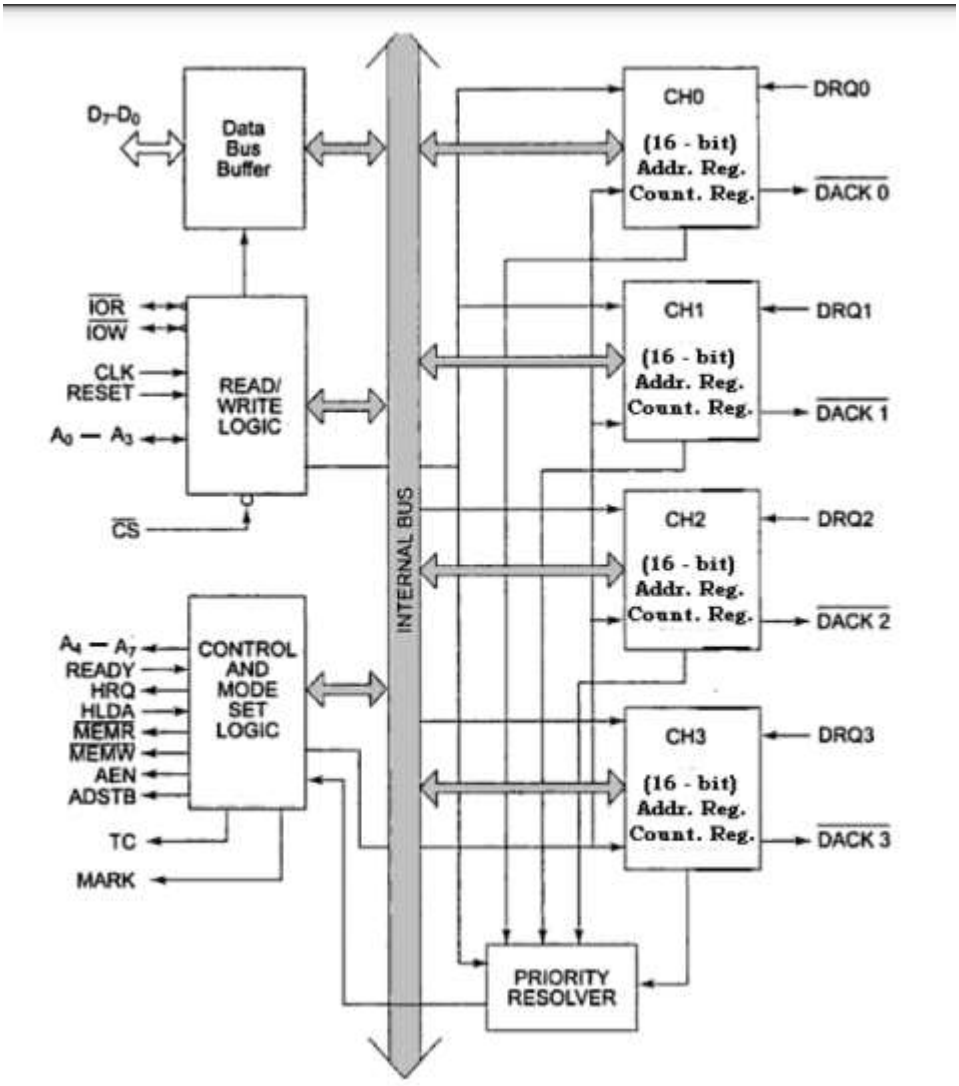
a) Data bus buffer

b) Control logic

c) Read/write logic

d) Priority Resolver

e) DMA channels



Data Bus Buffer:

It is a tri-state, bi-directional, eight bit buffer which interfaces the 8257 to the system data bus. In the slave mode, it is used to transfer data between microprocessor and internal registers of 8257. In master mode, it is used to send higher byte address (A8-A15) on the data bus.

Read/Write logic:

When the CPU is programming or reading one of the internal registers of Pin Diagram of 8257 (i.e, when the 8257 is in the slave mode), the Read/Write logic accepts the I/O Read (IOR) or I/O Write (IOW) signal, decodes the least significant four address bits (A0 – A3) and either writes the contents of the data bus into the addressed register (if IOW is low) or places the contents of the addressed register onto the data bus (if IOR is low). During DMA cycles (i.e. when the 8257 is in the master mode) the Read/Write logic generates the I/O read and memory write (DMA write cycle) or I/O write and memory read (DMA read cycle) signals which control the data transfer between peripheral and memory device.

DMA Channels:

The Pin Diagram of 8257 provides four identical channels, labeled CH0 to CH3. Each channel has two sixteen bit registers:

1. DMA address register, and

2. Terminal count register.

DMA address register: It specifies the address of the first memory location to be accessed. It is necessary to load valid memory address in the DMA address register before channel is enabled.

Terminal Count Register:

The value loaded into the low order 14 bits (C13 — C0) of the terminal count register specifies the number of DMA cycles minus one before the terminal count (TC) output is activated. Therefore, for N number of desired DMA cycles it is necessary to load the value N-1 into the low order 14-bits of the terminal count register. The most significant 2 bits of the terminal count register specifies the type of DMA operation to be performed. It is necessary to load count for DMA cycles and operational code for valid DMA cycle in the terminal count register before channel is enabled.

Control logic:

It controls the sequence of operations during all DMA cycles (DMA read, DMA write, DMA verify) by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed. It consists of mode set register and status register. Mode set register is programmed by the CPU to configure 8257 whereas the status register is read by CPU to check which channels have reached a terminal count condition and status of update flag.

Mode Set Register:

. Least significant four bits of mode set register, when set, enable each of the four DMA channels. Most significant four bits allow four different options for the Pin Diagram of 8257. It is normally programmed by the CPU after initializing the DMA address registers and terminal count registers. It is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up.

Status Register:

As said earlier, it indicates which channels have reached a terminal count condition and includes the update flag described previously. The TC status bit, if one, indicates terminal count has been reached for that channel. TC bit remains set until the status register is read or the 8257 is reset. The update flag, however, is not affected by a status read operation. The update flag bit, if one, indicates CPU that 8257 is executing update cycle. In update cycle 8257 loads parameters in channel 3 to channel 2.

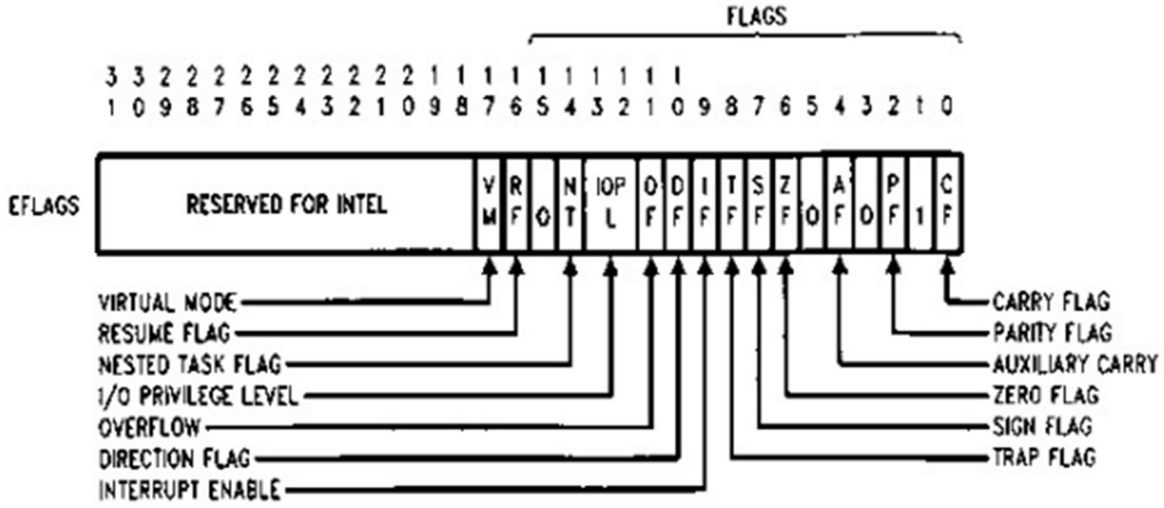
Priority Resolver:

It resolves the peripherals requests. It can be programmed to work in two modes, either in fixed mode or rotating priority mode.

Module 4. Intel 80386DX Processor

27. Explain flag register of 80386 DX.

The flag register in the 80386 DX is a 32-bit register, which is divided into 16 individual bits that represent specific status flags. These status flags provide information about the result of the previous arithmetic or logical operation performed by the processor.



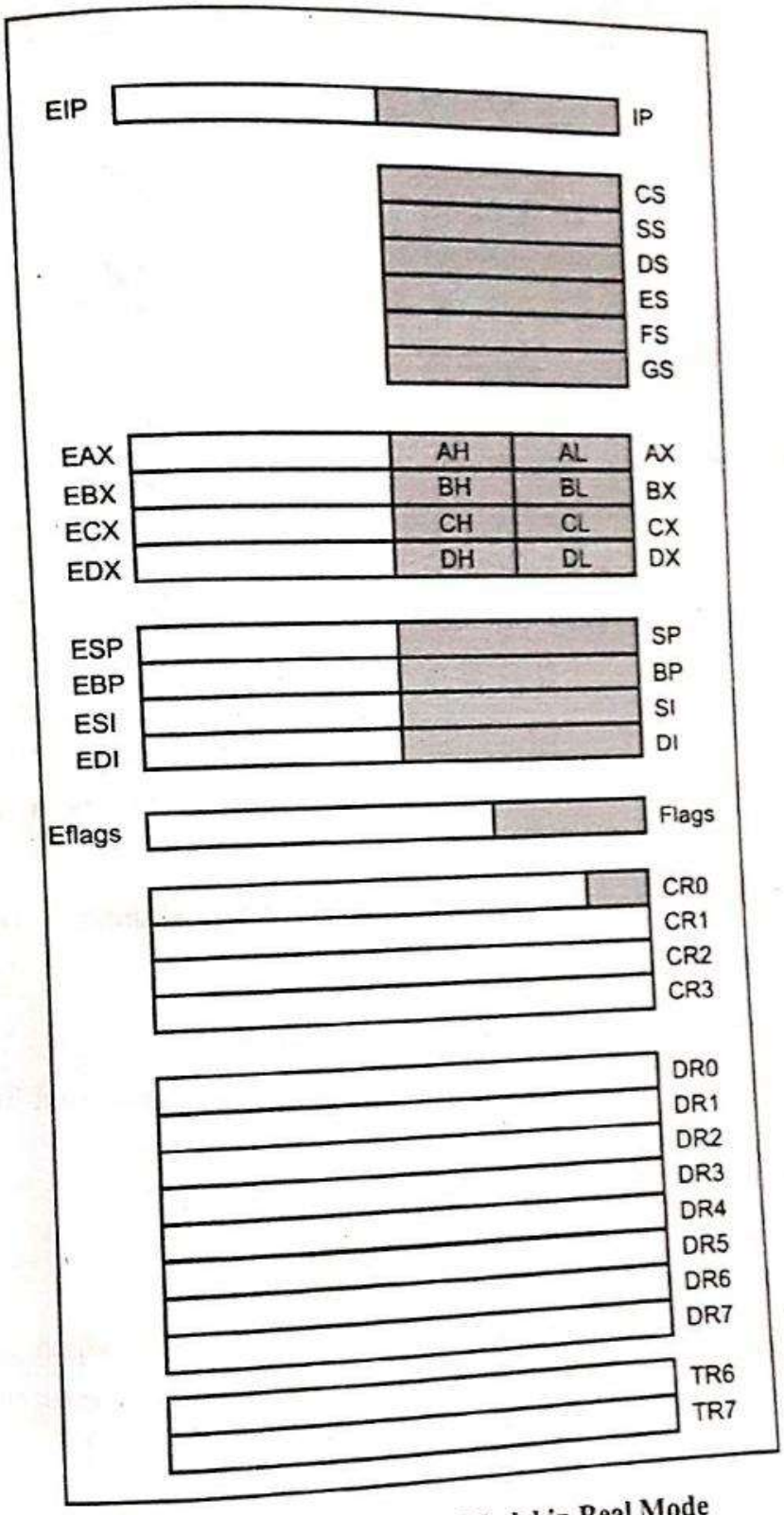
The individual bits of the flag register are as follows:

1. Carry flag (CF): This flag is set if an arithmetic operation generates a carry or borrow out of the most significant bit. It is used for unsigned arithmetic operations.
2. Parity flag (PF): This flag is set if the least significant byte of the result of an arithmetic or logical operation has an even number of 1 bits. It is used for parity checks.
3. Auxiliary Carry flag (AF): This flag is set if an arithmetic operation generates a carry or borrow between bits 3 and 4 of the operands. It is used for binary-coded decimal (BCD) arithmetic.
4. Zero flag (ZF): This flag is set if the result of an arithmetic or logical operation is zero.
5. Sign flag (SF): This flag is set if the result of an arithmetic or logical operation is negative.
6. Trap flag (TF): This flag is used for single-step debugging.
7. Interrupt flag (IF): This flag is used to enable or disable interrupts.
8. Direction flag (DF): This flag is used to control the direction of string operations.
9. Overflow flag (OF): This flag is set if an arithmetic operation generates a result that is too large or too small to be represented in the destination operand.
10. I/O privilege level flag (IOPL): This flag is used to control the level of I/O privilege that is required to execute certain instructions.
11. Nested task flag (NT): This flag is used to control the execution of nested tasks.
12. Resume flag (RF): This flag is used for debugging purposes.
13. Virtual 8086 mode flag (VM): This flag is used to control the execution of virtual 8086 mode.
14. Alignment check flag (AC): This flag is used to detect unaligned memory accesses.
15. Virtual interrupt flag (VIF): This flag is used to control the execution of virtual interrupts.
16. Virtual interrupt pending flag (VIP): This flag is used to indicate the presence of a pending virtual interrupt.

The flag register in the 80386 DX plays a critical role in the operation of the processor and is used by the operating system and applications to determine the state of the processor and the result of arithmetic and logical operations.

28. Explain virtual mode of 80386 DX.

1. In protected mode of operation, 80386DX provides a virtual 8086 operating environment to execute the 8086 programs.
2. The real mode can also use to execute the 8086 programs along with the capabilities of 80386, like protection and a few additional instructions.
3. Once the 80386 enters the protected mode from the real mode, it cannot return back to the real mode without a reset operation.
4. Thus, the virtual 8086 mode of operation of 80386, offers an advantage of executing 8086 programs while in protected mode.
5. V86 Mode is also known as Virtual Mode of 80386.
6. V86 Mode is a Dynamic Mode.
7. It can switch repeatedly & rapidly between V86 Mode & Protected Mode.
8. To execute an 8086 program, the CPU enters in V86 Mode from Protected Mode.
9. CPU Leaves V86 Mode and enters protected mode to continue executing a native 80386 program.
10. The address forming mechanism in virtual 8086 mode is exactly identical with that of 8086 real mode.
11. In virtual mode, 8086 can address 1MB of physical memory that may be anywhere in the 4GB address space of the protected mode of 80386.
12. Like 80386 real mode, the addresses in virtual 8086 mode lie within 1MB of memory.
13. In virtual mode, the paging mechanism and protection capabilities are available at the service of the programmers.
14. The virtual mode allows the multiprogramming of 8086 applications.
15. The virtual 8086 mode executes all the programs at privilege level 3.



(1F5)Fig. 4..6.1 : Software Model in Real Mode

29. Write a short note on control register of 80386DX.

80386 control Register :-

PG	00000000000000000	00000000000	ET	TS	EM	MP	PE	CR_0
NOT USED								CR_1
page fault Linear Address								CR_2
Page Directory Base		000000000000						CR_3

Control Register of 80386.

- CR_0 :- (Control Register zero)

① PG (P G) :- (Paging Enable.)

1. This bit is made one to enable paging mode
2. 80386 Microprocessor implements virtual memory using the techniques of segmentation and paging.
3. In this segmentation is cumpulsory where paging is optional.
4. Paging is enabled using the PG bit of CR_0 the default value of PG is zero.
5. ~~It is extension type~~

② ET (E T) :- (Extension Type)

1. This bit is used to indicate the type of math coprocessor used with 80386.
2. IF ET is 1
   then 80387 MATH coprocessor is used.
3. IF ET is 0
   then 80287 MATH coprocessor is used.

③ TS (T/S) :- (Task switch)

1. This bit made one to indicate if a task switch is performed.
2. 80386 microprocessor performs multitasking that is why it switches beth various tasks.

④ EM (E/M) :- (Emulate coprocessor)

1. This bit is made one in the absence of math coprocessor so that if the coprocessor instruction is encounter then it will be executed on a chip emulator
2. If this bit is zero then the coprocessor instruction will be executed by 80387 and/or 80287.

⑤ MP (M/P) :- (MATH coprocessor)

1. This bit is made one to indicate math coprocessor is available.

⑥ PE (P/E) :- (protection Enable)

1. This bit is made one to enter in protected mode

- CR_1 :- (control Register one.)
  1. Control register 1 not used reserved by intel.
- CR_2 :- (Control Register two.)
  1. This is a 32 bit register giving the linear address that caused the last page fault
  2. A page fault occurs when the desired page is not present in the physical memory.

- CR_3 :- (Control Register three.)
  1. 80386 Microprocessor implements two level page translation Mechanism
  2. The information about various pages is stored in various page tables.
  3. The address of these page tables are stored in the page directory.
  4. CR_3 gives the base address that is the starting addressing of page.

30.

Difference between Real and Protected Mode

Real Mode	Protected Mode (PVAM)
Memory addressing up to 1 MB physical memory	Memory addressing up to 16 MB of physical memory
No virtual memory support	Supports up tp to 64TB of virtual memory
Memory Protection mechanism is not available	Memory Protection Mechanism is avilable
Does not support virtual address space	Gives virtual and physical address space
Does not support LDT and GDT	Supports LDT and GDT
Segment descriptor cache is not available	Segment descriptor cache is available
Supports Segmentation	Supports segmentation and paging.

31. What is GDT and explain the structure of GDT.

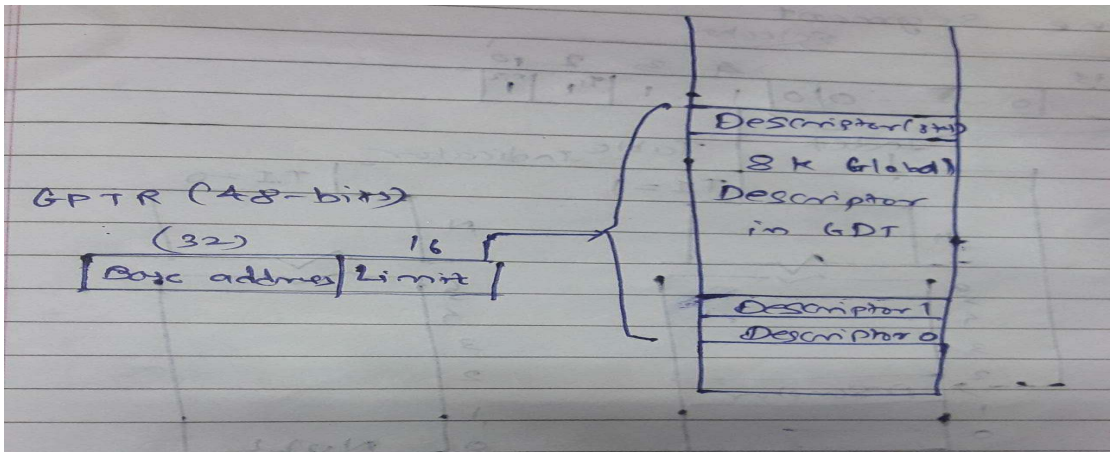
GDT stands for Global Descriptor Table, which is a data structure used by the x86 family of processors to define the memory segments of a computer's physical address space. The GDT is a table that contains a list of descriptors, each of which defines a memory segment.

1. It has 48 bit register.

2. It defines Global Descriptor Table in the physical memory.

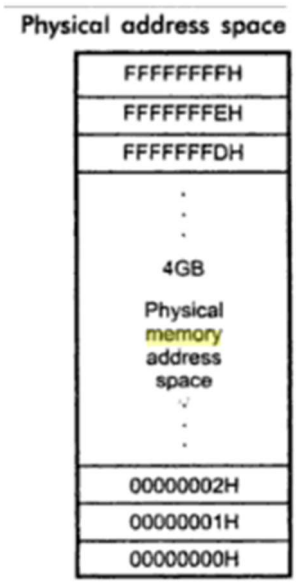
3. The upper 4 bytes (32 bits) gives starting address of the table also called as base address.

4. The lower two byes indicates the length of GDT.



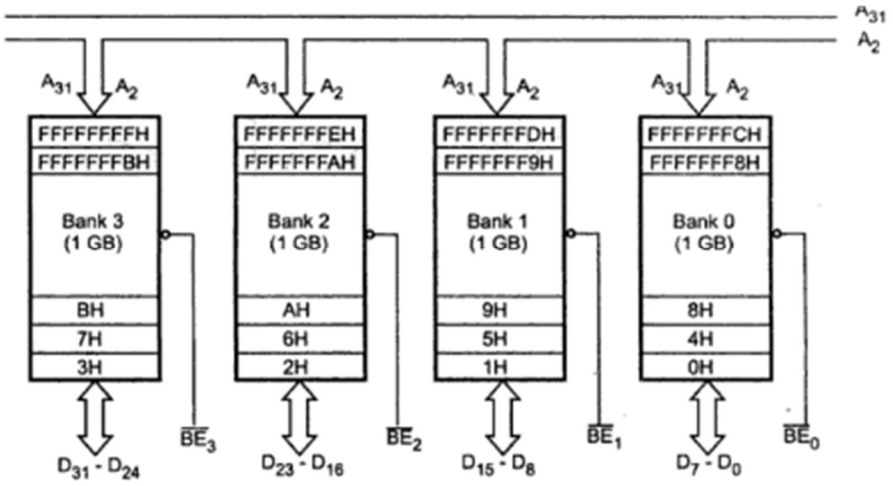
32. Explain Memory Management in details in 80386DX processor.

1. The 80386DX has 32-bit address bus so it can access up to 4 Gbytes (2^32) of memory locations. Figure (a) below shows the physical address space. From the software point of view this memory is organized over the address range from 00000000H through FFFFFFFFH and 80386DX can access data in this memory as byte, word or double words. The words are accessed from two consecutive memory locations whereas double words are accessed from four consecutive memory locations. To implement this entire memory is divided into four independent byte-wide memory banks: Bank0-Bank3. Each bank is 1 Gbyte in size.



(a)Physical address space

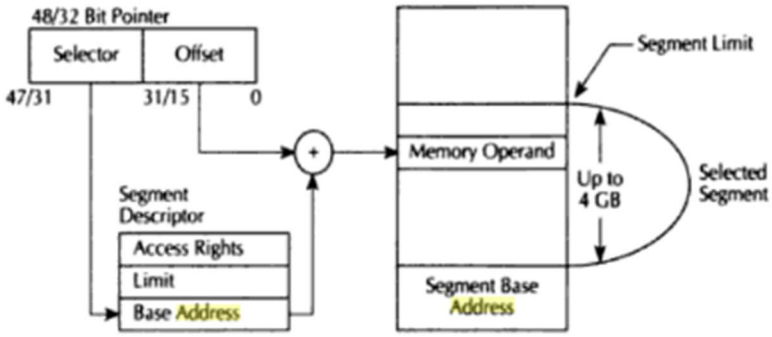
1. Figure (b) shows the organization of four memory banks. As shown in figure bank 0, bank 1, bank 2 and bank 3 are selected using byte enable signals BE0, BE1, BE2 and BE3 signals, respectively.
2. Address lines A31-A2 are connected in parallel to all memory banks which make it possible to access 1G-byte of memory. But the 32-bit data bus is distributed over four memory banks, 8-bit each.
3. The 80386 uses byte enable signals instead of the two least significant address bits, because 80386 has problems involved in addressing more than one byte of memory at a time. When 80386 accesses word from even address, it uses two consecutive memory locations for example,



1. MOV WORD PTR DS: [2000H], 5678H This instruction writes 78 to address 2000H and 56 to address 2001H.
2. Similarly, when 80386 accesses Dword from address divisible by 4, it uses four consecutive memory locations. This works fine when 80386 accesses even byte in case of word access and address divisible by 4 in case of Dword access, since address on the A31-A2 lines is same for word as well as Dword access.
3. The data transfers using such addresses are called Aligned transfers. But to access word from odd address or to access Dwords from address not divisible by 4 unaligned 80386 faces problem in placing the correct address on the address bus. This problem is solved by replacing two address pins AO and Al with four byte enable pins.
4. Without the two least significant address pins, the 80386 produces only addresses that are even multiples of 4. To distinguish between (our addresses (four banks) byte enable signals are used. Table 1.3 shows the use of the four byte enables pins with the address pins.

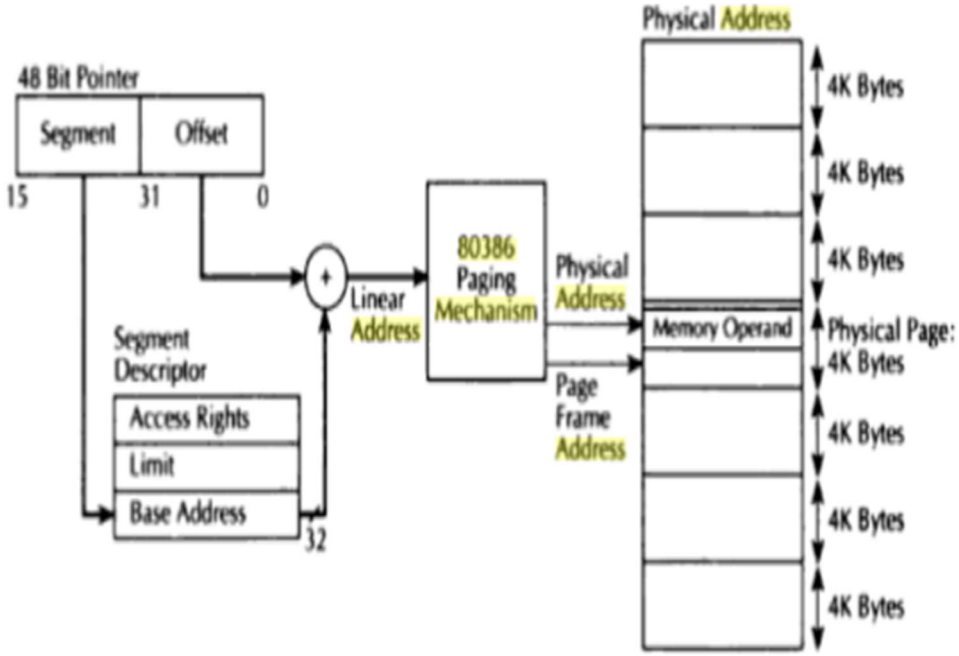
33. Explain address translation mechanism used in protected mode of 80386.

1. Figure below shows the protected mode addressing mechanism.



Protected mode addressing

1. The selector is used to specify an index into a table defined by the operating system. The table includes the 32-bit base address of a given segment. The physical address is obtained by summing the base address obtained from the table with the offset.
2. With the paging mechanism enabled, the 80386 provides an additional memory management mechanism. The paging feature manages large 80386 segments. The paging mechanism translates the protected linear addresses from the segmentation unit into physical addresses. Figure below shows this translation scheme.
3. Segmentation provides both memory management and protection. All information about the segments is stored in an 8-byte data structure called a descriptor. All the descriptors are stored in tables identified by the 80386 hardware. There are three types of tables holding 80386 descriptors: global descriptor table (GDT), local descriptor table (LDT), and interrupt descriptor table (I DT).

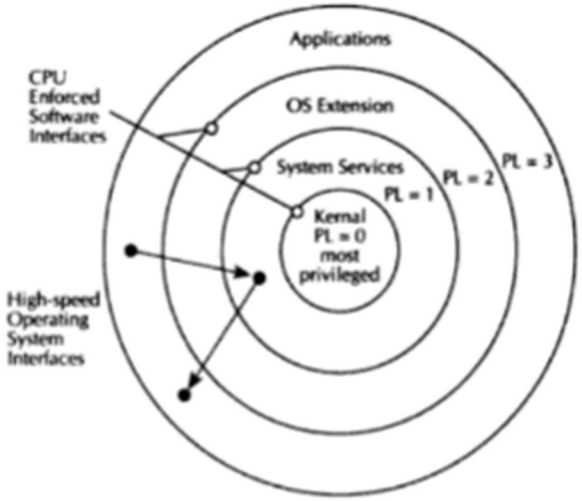


Paging and Segmentation

- The GDT contains descriptors which are available to all the tasks in the system. In general, the GDT contains code and data segments used by the operating system, task state segments, and descriptors for LDTs in a system.
- LDTs store descriptors for a given task. Each task has a separate LDT, while the GDT contains descriptors for segments which are common to all tasks.
- The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. Every interrupt used by a system must have an entry into the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions.

The 80386 provides four protection levels for supporting a multitasking operating system to isolate and protect user programs from each other and the operating system.

The privilege level controls the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The 80386 includes the protection as part of its memory management unit. The four-level hierarchical privilege system is shown in Figure below.



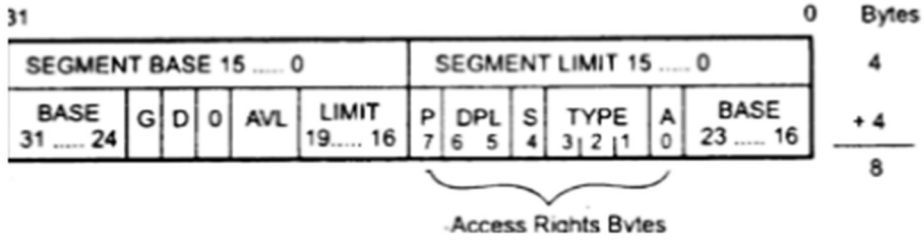
Four level hierarchical protection

1. It is an extension of the user/ supervisor privilege mode used by minicomputers. Note that the user/supervisor mode is supported by the 80386 paging mechanism. The Privilege Levels (PL) is numbered 0 to 3. Level 0 is the most privileged level.
2. procedures between levels of a task:
    - Data stored in a segment with a privilege level x can be accessed only by code executing at a privilege level at least as privileged as x.
    - A code segment/procedure with privilege level x can only be called by a task executing at the same or a higher privilege level than x.
3. The 80386 supports task gates (protected indirect calls) to provide a secure method of privilege transfers within a task. The 80386 also supports a rapid task switch operation via hardware.
4. Paging is another type of memory management for virtual memory multitasking operating systems. The main difference between paging and segmentation is that paging divides programs/data into several equal-sized pages, while segmentation divides programs/data into several variable-sized segments.
5. There are three elements associated with the 80386 paging mechanism. These are page directory, page tables, and the page itself (page frame). The paging mechanism does not have memory fragmentation since all pages have the same size of 4K bytes.
6. The 80386 takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables and the handling of any page faults.
7. The operating system initializes the tables by loading CR3 with the address of the page directory and allocates space for the page directory and the page tables. The operating system also implements a swapping policy and handles all of the page faults.

34. Draw a segment descriptor format and explain different fields.

35. Draw format of selector and explain its fields

1. In protected mode, memory management unit (MMU) uses the segment selector to access a descriptor for the desired segment in a table of descriptors in memory.
2. Segment descriptor is a special structure which describes the segment. Exactly one segment descriptor must be defined for each segment of the memory.
3. Descriptors are eight type quantities which contain attributes about a given region of linear address space (i.e. a segment).
4. These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment.
5. Fig. below shows the general format of a descriptor. As shown in Fig. segment descriptor has following fields.
6. Base: It contains the 32-bit base address for a segment. Thus defines the location of the segment within the 4 gigabyte linear address space. The 80386 concatenates the three fragments of the base address to form a single 32-bit address.
7. Limit: It defines the size of the segment. The 80386 concatenates the two fragments of the limit field to form a 20 bit value. The 80386 interprets this 20-bit value in two ways, depending on the setting of the granularity bit (G) :
    - If G bit 0: In units of one byte, to define a limit of up to 1 M byte (220)
        - If G bit 1: In units of 4 kilobytes, to define a limit of up to 4 gigabytes.
8. Granularity Bit: It specifies the units with which the limit field is interpreted. When bit is 0, the limit is interpreted in units of one byte; otherwise limit is interpreted in units of 4 Kbytes.
9. D (Default size): When this bit is cleared, operands contained within this segment are assured to be 16 bits in size. When it is set, operands are assumed to be 32-bits.



BASE Base Address of the segment
LIMIT The length of the segment
P Present Bill: 1= Present 0= Not present
DPL Descriptor privilege Level 0 – 3
S Segment Descriptor : 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE Type of segment
A Accessed Bit
G Granulaty Bit: 1= Segment length is page granular 0 = Segment length is byte granular

D Default Operation Size (recognized in code segment descriptors only) 1 =32-bit segment 0 = 16 – bit segment
0 Bit must be zero (0) for compatibility with future processors
AVL Available field for user or OS
Note:

In a maximum – size segment (i.e. a segment with G = 1 and segment limit 19……0=FFFFFFH).

The lowest 12 bits of the segment base should be zero. (i.e. segment base 11…..000=000H)

General Segment Descriptor Format

- 0 (Reserved by Intel): It neither can be defined nor can be used by user. This bit must be zero for compatibility with future processors.
- AVL/U (User Bit) : This bit is completely undefined, and 80386 ignores it. This is available field/bit for user or operating system. Access Rights Byte:
- P (Present Bit): The present P bit is 1 if the segment is loaded in the physical memory, if P = 0 then any attempt to access this segment causes a not present exception (exception 11).
- DPL (Descriptor Privilege Level): It is a 2-bit field defines the level of privilege associated with the memory space that the descriptor defines- DPL0 is the most privileged whereas DPL3 is the least privileged.
- S (System Bit): The segment S bit in the segment descriptor determines if a given segment is a system segment or a code or a data segment. If the S bit is I then the segment is either a code or data segment, if it is 0 then the segment is system segment.
- Type: This specifies the specific descriptors among various kinds of descriptors. (Detail explanation is given in the following sections).
- A (Accessed Bit): The 80386 automatically sets this bit when a selector for the descriptor is loaded into a segment register. This means that 80386 sets accessed bit whenever a memory reference is made by accessing the segment.

Module 5. Pentium Processor

36. Draw and explain block diagram of Pentium processor.

Ans:

1) BUS Unit :
- The BUS Unit is responsible for transfering data in & out of the MP.
- It is connected to external memory & I/O devices using system BUS.

2) L1 Code :
- Pentium has a chip with 8Kb L1 code.
- It is two way set associative.
- It contains the most recently used instruction.

3) Prefetch Unit :
- To Prefetch instruction from L1 code.
- It has 2 queue each of 32 bytes.
- one queue act's as a active queue whereas other is used during branch prediction.

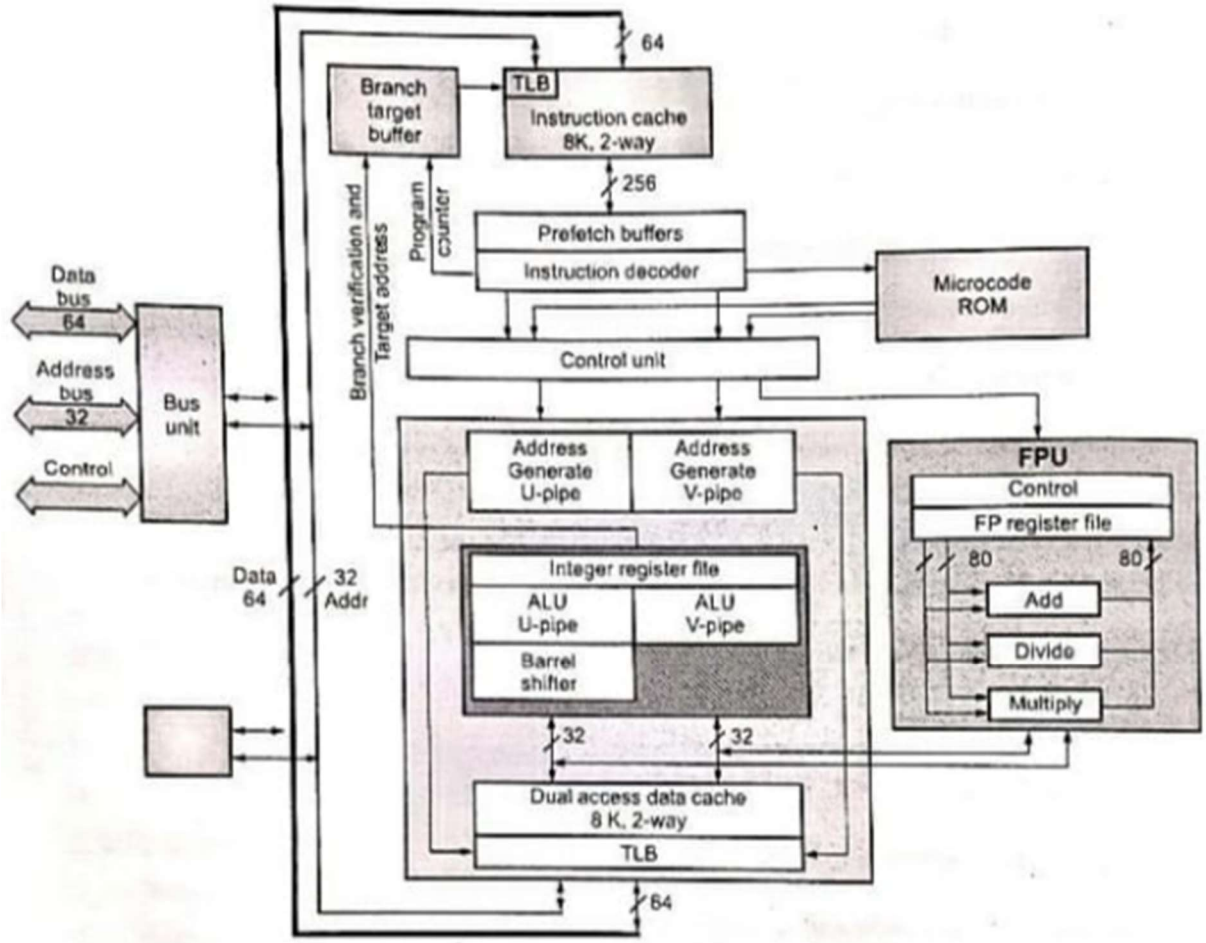
4) Decode Unit :
- It decode the inst. simultaneously for U & V pipeline.
- Simple instruction are decoded by hardwired control unit.
- complex instruction are decoded using softwired control unit.

5) Integer Execution Unit :
- It can handle 2 integer instruction simultaneously.
- The 1st one goes to U pipe & 2nd goes to V pipe.
- Add. generation is performed for each pipe.
- If the inst. uses memory as operand the add. generatn unit generates physical address & fetches

instruction from desired location.

6) Floating Point Unit :
- It performs the floating Pt. operation.
- It uses 80 bit floating Pt. registers
- It has its own floating Pt. control unit.

7) Branch Prediction Unit :
- The Pentium performs Branch prediction to minimize drawbacks of pipeline.



(IGI)Fig. 5.2.1 : Pentium Architecture

37. Explain in brief Pipeline stages of Pentium 1.

Pentium1 uses a 5 stage pipeline with the following stages in the pipeline.

1. Pre-fetch Stage: Here the instruction are fetched from physical memory and stored them into the pre-fetch queue.

Pre-fetch queue is of 32 bytes as it needs at least two full instruction to be present inside the pipeline.

There are two pre-fetch queue but only one of them is active and other is used when bank prediction logic is used

2. Decode stage: The decode stage decodes the instruction of code.

It also decodes and change for instruction pairing and perform branch prediction logic.

Certain rules are provided for the instruction pairing.

If two instruction can be paired then first instruction is given to U pipeline and second instruction is given to V pipeline.

3. Address Generation Stage: It generates the address of the physical memory.

Physical address is required when memory operand is used using segment translation and page translation.

Protection mechanism is also performed at this stage.

4. Execution Stage: The execution stage mainly uses the ALU.

Operands are either provided by register or by cache memory.

Instruction involving such as multiplication division can only be use by U pipeline.

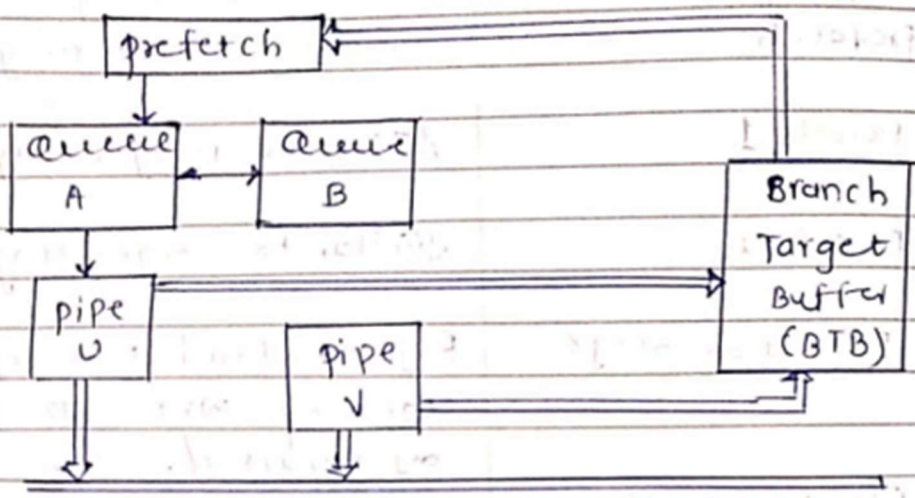
5. Write Back Stage: In this stage as the name suggest the result is written back to memory.

It means it stores the result in the memory.

38. Explain the branch prediction logic used in Pentium processor.

Ans:

• Branch prediction logic:-



The pentium processor includes branch prediction logic that minimize the pipeline problem.

When branch operation is correctly predicted no performance penalty is incurred.

However branch prediction is not correct a three cycle penalty is considered

If branch is executed in 'U' pipeline and fourth cycle penalty is considered if the branch is executed in three pipeline

There are Three bits containing in the following information.

① A valid bit :- That indicates whether or not the entry is in use.

② A Histy bit :- That track a branch has been taken for execution in the pipeline before.

③ Memory Address :- It keeps the address of branch instruction for identification.

If No history bit exists then the prediction is made that the branch will not be taken.

If their is a history bit then the prediction is made as follow:

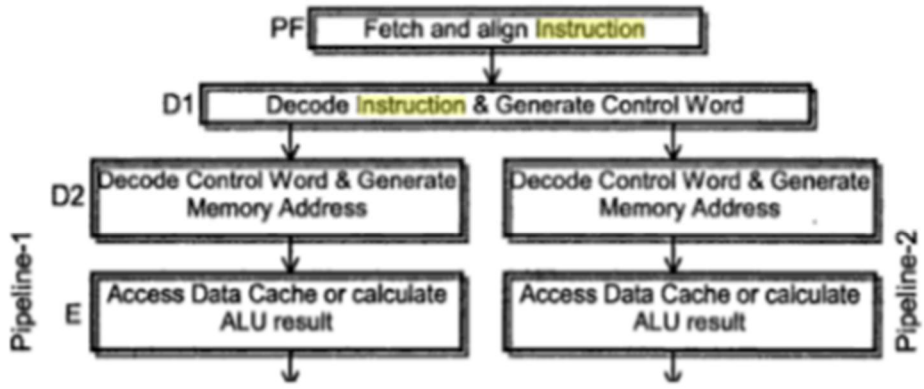
	History bit	discription	prediction made	If actually taken	If actually not taken
1.	00	strongly not taken	Branch will not be taken	Upgrade to weakly not taken	Remains strongly not taken.
2.	01	Weakly not taken	Branch will not be taken	upgrade to weakly taken	Downgrade to strongly not taken.
3.	10	weakly taken	Branch will be taken	Upgrade to strongly taken	Down grade to weakly not taken
4.	11	strongly taken	Branch will be taken	Remains strongly taken	Downgrade to weakly taken

During the execution stage the microprocessor relise whether the prediction was true or false.
The following action are perform.
1. The branch was correctly predicted taken, the entries of history bits are upgrated.
2. If the branch was incorrectly predicted taken, the entry is down graded.

39. Write the instruction issue algorithm used in Pentium.

The Pentium has superscalar organizations. It enables 2-instructions to be executed in parallel. Figure below (a) shows that the resources for address generation and ALU functions have been replicated in independent integer pipeline, called U- and V-.

The ????P in the PF and D1 stages can fetch and decode 2-simple instructions in parallel and issue them to the U- and V-pipelines. Additionally, the ????P for complex instructions in D1 can generate micro coded sequences that control both U- and V-pipelines. Several techniques are used to resolve dependencies between instructions that might be executed in parallel. Most of the logic is contained in the instruction issue algorithm as indicated in Figure (b) of D1.



(a) Superscalar execution

Decode two consecutive instructions: I1 and I2
If the following are all true
I1 is a 'simple' instruction
I2 is a 'simple' instruction
I1 is not a jump instruction
Destination of I1 ≠ Source of I2
Destination of I1 ≠ Destination of I2
Then issue I1 to U-pipe and I2 to V-pipe
Else issue I1 to U-pipe

(b) Instruction issue algorithm

Resource Dependency

When 2-instructions require a single functional unit or data path, a resource dependency occurs. The ????P during the D1 stage issues 2-instructions for parallel execution if both belong to the class of simple instructions, thereby eliminating most resource dependencies. The instructions must be directly executed that does not require micro-coded sequencing. The instruction being issued to the V-pipe can be an ALU operation, memory referencing or a jump. The instruction being issued to the U-pipe can be from the same categories or from an additional set that uses a functional unit available only in the U-pipe, such as the barrel shifter. Although, the set of instructions identified as 'simple' might seem restrictive, more than 90% of the instructions executed in the integer SPEC benchmark suite are simple.

Data dependencies

When one instruction writes a result that is read or written by another instruction, a data dependency occurs. Logic in D1 ensures that the source and the destination registers of the instruction issued to the V-pipe differ from the destination register of instruction issued to the U-pipe. This arrangement eliminates read-after-write (RAW) and write-after-write (WAW) dependencies. Write-after-read (WAR) dependencies need not be checked because reads occur in an earlier stage of the pipelines than writes. The design includes logic that enables instruction with certain special types of data dependency to be executed in parallel. For example, a conditional branch instruction that tests the flag results can be executed in parallel with a compare instruction that sets the flags.

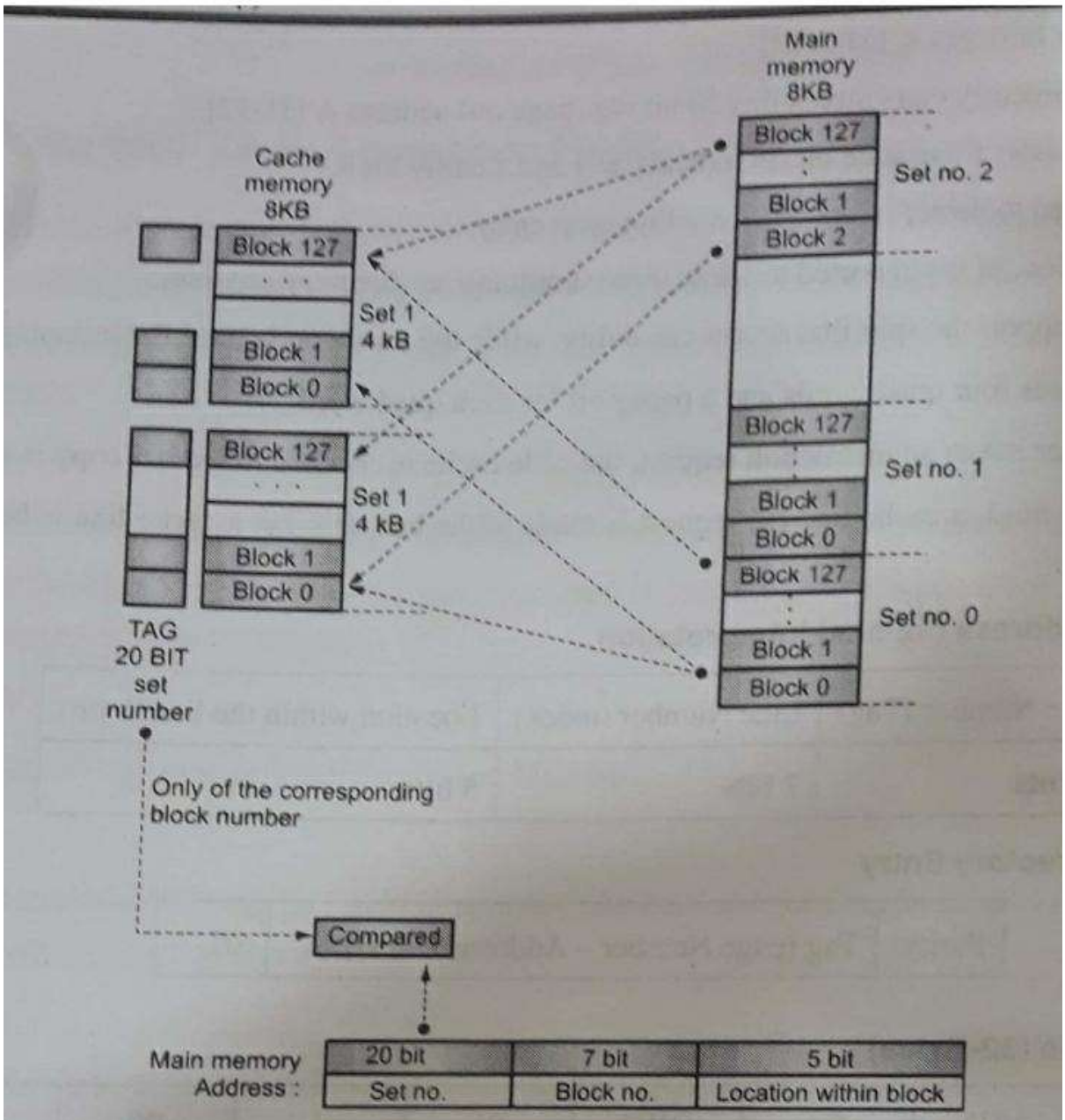
Control dependencies

When the result of one instruction determines whether another instruction will be executed, a control dependency occurs. The ????P in D1 never issues an instruction to the V-pipe when a jump instruction is issued to the U-pipe, thereby eliminating control dependencies.

40. Code cache organization of Pentium.

1. Despite the potential advantages of a unified cache which is used in the 80486 processor, the Pentium microprocessor uses separate code and data caches.
2. The reason is that the superscalar design and branch prediction demand more bandwidth than a unified cache.
3. First, efficient branch prediction requires that the destination of a branch be accessed simultaneously with data references of previous instructions executing in the pipeline.
4. Second, the parallel execution of data memory references requires simultaneous accesses for loads and stores.
5. Third, in the context of the overall Pentium microprocessor design, handling self-modifying code for separate code and data caches is only marginally more complex than for a unified cache.
6. The data and instruction caches on the Pentium processor are each 8 KB, two-way associative designs with 32 byte lines. Each cache has a dedicated translation lookaside Buffer (TLB) to translate linear addresses to physical addresses.
7. The caches can be enabled or disabled by software or hardware. The Pentium microprocessor implements the data cache to supports dual accesses by the U-pipe and V-pipe to provide additional bandwidth and simplify compiler instruction scheduling algorithms.
8. The data cache is write back or write through configured on a line-by-line basis and follows the MESI protocol. The data cache tags arc triple ported to support two data transfers and an inquire cycle in the same clock.
9. The code cache is an inherent write protected cache. The code cache tags of the Pentium processor are also triple ported to support snooping and split-line accesses.
10. The data path, however, is single ported with eight way interleaving of 32-bit-wide banks. When a bank conflict occurs, the U-pipe assumes software or hardware.
11. The Pentium microprocessor implements the data cache to supports dual accesses by the U-pipe and V-pipe to provide additional bandwidth and simplify compiler instruction scheduling algorithms.
12. The data cache is write back or write through configured on a line-by-line basis and follows the MESI protocol.
13. The data cache tags are triple ported to support two data transfers and an inquire cycle in the same clock.
14. The code cache is an inherently write protected cache.
15. The code cache tags of the Pentium processor are also triple ported to support snooping and split-line accesses.
16. The data path, however, is single ported with eight way interleaving of 32-bit-wide banks.

17. When a bank conflict occurs, the U-pipe assumes priority, and the V-pipe stalls for a clock cycle.
18. The bank conflict logic also serves to eliminate data dependencies between parallel memory references to a single location.
19. For memory references to double-precision floating-point data, the processor accesses consecutive banks in parallel, forming a single 64-bit path.
20. Translation lookaside buffers (TLB): Besides general-purpose caches, X86 processors include caches called Translation Lookaside Buffers (TLB) to speed up linear address translation. When a linear address is used for the first time, the corresponding physical address is computed through slow accesses to the page tables in RAM. The physical address is then stored in a TLB entry so that further references to the same linear address are quickly translated. When the CR_3 control register is modified, the hardware automatically invalidates all entries of the TLB.



(1G4)Fig. 5.6.1 : Code and Data Cache Organization

EXTRA QUE WITH

Module. 6 Pentium 4.

41. Explain the Floating point Pipeline stages of Pentium 1.

Ans: Pentium 1 has 8 floating point pipeline stages: The floating point pipeline of Pentium consists of eight stages which are used to speed up the execution of floating point unit.

1. Pre-fetch: Instructions are pre fetched from the on chip instruction cache.

2. Decode 1: Instruction decode to generate control word. A single control word causes direct execution of an instruction and complex instruction require micro-coded control sequence.

3. Decode 2: Address of memory resident operand are calculated.

4. Execution Stage: In this stage, register read, memory read or memory write operation is performed to access an operand as required by the instruction.

5. Floating point Execution 1: In this stage, the floating point data from register or memory is written into floating point register.

6. Floating point Execution 2: In this stage, the floating point operation is performed by floating point unit.

7. Write floating point result: In this stage, floating point operation are rounded and the written to the floating point register.

8. ER reporting: In this stage, if any error is occurred during floating point execution, then it is reported and FPU status word is updated.

42. Explain the features of Pentium4.

Ans: 1.It specifically designed for high end performance or high band width internet.

2. It specifically designed for sophisticated user working with complex internet, image, video, speech.

3. It is based upon intel net burst micro-architecture.

4. It has 400 MHZ system bus.

5. It has rapid execution engine.

6. It has advance transfer cache memory.

7. It supports hyper pipeline technology.

8. It supports hyper threading technology.

9. Pentium 4 has 20 stage pipelining.

10. It supports write back and write through policy.

43.

Comparison of all processors

(Very Important, 5m – features of any one processor)

S No	Attribute	8085	8086	80286	80386	80486	Pentium
1	Processor Size	8 – bit	16 – bit	16 – bit	32 – bit	32 – bit	32 – bit
2	Data Bus	8 – bit	16 – bit	16 – bit	32 – bit	32 – bit	64 – bit
3	Memory Banks	--- NA ---	2 banks	2 banks	4 banks	4 banks	8 banks
4	Address Bus	16 – bit	20 – bit	24 – bit	32 – bit	32 – bit	32 – bit
5	Memory Size	64 KB	1 MB	16 MB	4 GB	4 GB	4 GB
6	Pipeline Stages	--- NA ---	2	3	3	5	5
7	ALU Size	8 – bit	16 – bit	16 – bit	32 – bit	32 – bit	32 – bit
8	No of Transistors	6500	29,000	1,34,000	2,75,000	11,80,235	31,00,000
9	Year of Release	1976	1978	1982	1985	1989	1993
10	Operating Frequency	3 MHz	6 MHz	12 MHz	33 MHz	60 MHz	100 MHz

44. Explain the 20 stage pipelining of Pentium 4.

Ans: The Pentium 4 is a microprocessor that uses a deep pipeline consisting of 20 stages. The purpose of a pipeline is to break up the processing of an instruction into smaller, more manageable stages that can be executed in parallel. This allows the processor to handle multiple instructions at the same time, improving performance.

Here are the 20 stages of the Pentium 4 pipeline, along with a brief description of each:

1. Instruction Fetch: The processor fetches the next instruction from memory.
2. Instruction Decode: The instruction is decoded into its constituent parts.
3. Instruction Queue: The decoded instruction is stored in a queue.
4. Dispatch: The instruction is dispatched to the appropriate execution unit.
5. Register Rename: Registers are renamed to allow for out-of-order execution.
6. Schedule: The instruction is scheduled for execution.
7. Issue: The instruction is issued to the appropriate execution unit.
8. Operand Fetch: The operands required by the instruction are fetched.
9. Operand Queue: The operands are stored in a queue.
10. Execution: The instruction is executed.
11. Memory Address Generation: The address of the memory location to be accessed is generated.
12. Memory Access: The memory location is accessed.
13. Memory Queue: The memory operation is stored in a queue.
14. Retirement: The results of the instruction are written back to the appropriate registers.
15. Instruction Fetch for Next Instruction: The processor fetches the next instruction from memory.
16. Instruction Decode for Next Instruction: The next instruction is decoded.
17. Instruction Queue for Next Instruction: The next instruction is stored in a queue.
18. Dispatch for Next Instruction: The next instruction is dispatched to the appropriate execution unit.
19. Register Rename for Next Instruction: Registers are renamed for the next instruction.
20. Schedule for Next Instruction: The next instruction is scheduled for execution.

45. Draw and explain block diagram of Pentium 4.

Ans: The Pentium 4 is a microprocessor developed and manufactured by Intel. It was released in 2000 and represented a significant departure from the previous Pentium III architecture. The Pentium 4 features a high clock rate and a deep pipeline, which allows for high performance in certain types of applications.

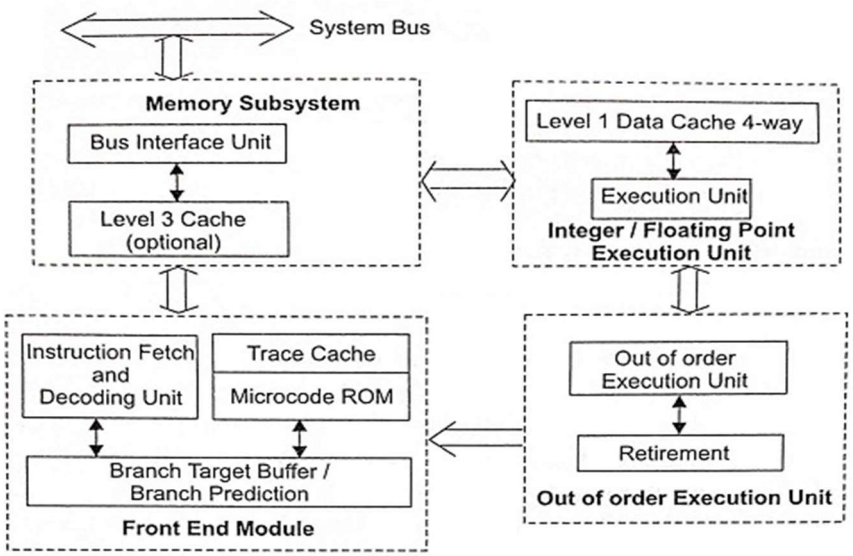


Fig. 12.60 *Simplified block diagram of Pentium 4 processor*

1. Front End: This part of the processor is responsible for fetching and decoding instructions from memory.
2. Instruction Decoder: This unit decodes instructions fetched from memory into a form that can be executed by the processor.
3. Instruction Cache: This is a small, high-speed memory that stores recently used instructions for quick access by the processor.
4. Execution: This part of the processor is responsible for actually executing instructions.
5. Integer Execution Units (EU): These are specialized units that perform operations on integer data.
6. Floating Point Execution Units: These units are responsible for performing floating-point arithmetic operations.
7. Cache Memory: This is a small, high-speed memory that stores frequently accessed data for quick access by the processor.
8. System Bus: This is the pathway that connects the processor to the rest of the computer system, including memory and other peripherals.

Overall, the Pentium 4 was a powerful processor for its time