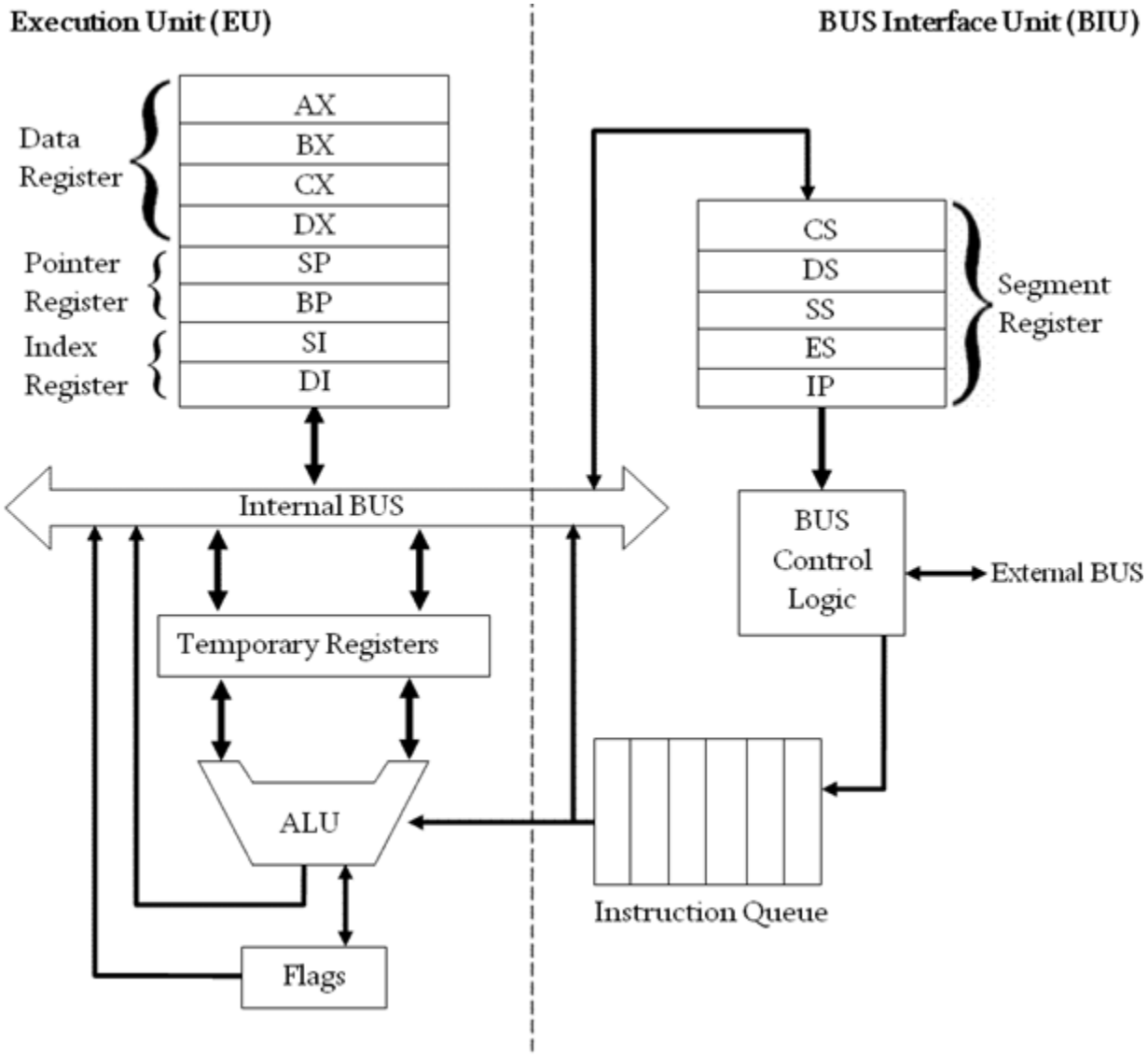
Module 1 - The Intel Microprocessor 8086 Architecture

1. 8086 CPU

Internal Architecture



As 8086 does 2-stage pipelining (overlapping fetching and execution), its architecture is divided into two units:

- Bus Interfacing Unit (BIU)
- Execution Unit (EU)

Bus Interfacing Unit (BIU)

- It provides the interface of 8086 to external memory and I/O devices.
- It operates with respect to bus cycles (machine cycles). This means it performs various machine cycles such as memory read, I/O read etc. to transfer data with memory and I/O devices.
- BIU performs the following functions-
- It generates the 20 bit physical address for memory access.
- It fetches instruction from memory.
- It transfers data to and from the memory and I/O.
- 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 address for the Code Segment. All programs are stored in the Code Segment. CS is multiplied by 10H to give the 20 bit physical address of the Code Segment. E.g. If CS = 4321H then CS x 10H = 43210H→ Starting address of Code Segment.

        - DS register: DS holds the base address for the Data Segment. It is multiplied by 10H to give the 20 bit physical address of the Data Segment. E.g. If DS = 4321H then DS x 10H = 43210H→ Starting address of Data Segment.

        - SS register: SS holds the base address for the Stack Segment. It is multiplied by 10H to give the 20 bit physical address of the Stack Segment. E.g. If SS = 4321H then SS x 10H = 43210H→ Starting address of Stack Segment.

        - ES register: ES holds the base address for the Extra Segment. It is multiplied by 10H to give the 20 bit physical address of the Extra Segment. E.g. If ES = 4321H then ES x 10H = 43210H→ Starting address of Code Segment.

    - **Instruction Pointer (IP)**
        - It is a 16 bit register. It holds offset of the next instructions 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 = Segment Address x 10H + Offset Address

- **6 Byte Pre-fetch Queue**
  - It is a 6 byte first in first out (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. The queue is refilled when at least two bytes are empty as 8086 has a 16 bit data bus. Pipelining increases the efficiency of the microprocessor.
  - Pipelining fails when a branch occurs as the pre-fetched instructions are no longer useful.
  - Hence as soon as 8086 detects a branch operation, it clears/discards the entire queue.
  - Now, the next six bytes from the new location (branch address) are fetched and stored in the queue and pipelining continues.

Execution Unit (EU)

- It fetches instructions from the Queue in BIU, decodes and executes them.
- It performs arithmetic, logic and internal data transfer operations within the microprocessor.
- It sends request signals to the BIU to access the external module.
- It operates with respect to T-stats (clock cycles) and does not depend upon which machine cycle is being performed by the BIU.
- The main components of the EU are as follows:

- **General purpose registers**
  - 8086 microprocessor 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; etc. 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 I/O 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 I/O port in indirect I/O addressing mode.
- **Special purpose registers-**
- Stack Pointer (SP 16 bits): It holds offset address of the top of the Stack. Stack is a set of memory locations operating in LIFO manner. Stack is present in the memory in Stack Segment. It is used during instructions like PUSH, POP, CALL, RET etc.

- Base Pointer (BP 16 bits): BP can hold 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.

ALU (Arithmetic Logic Unit)
It has a 16 bit ALU. It performs 8 and 16 bit arithmetic and logic operations.

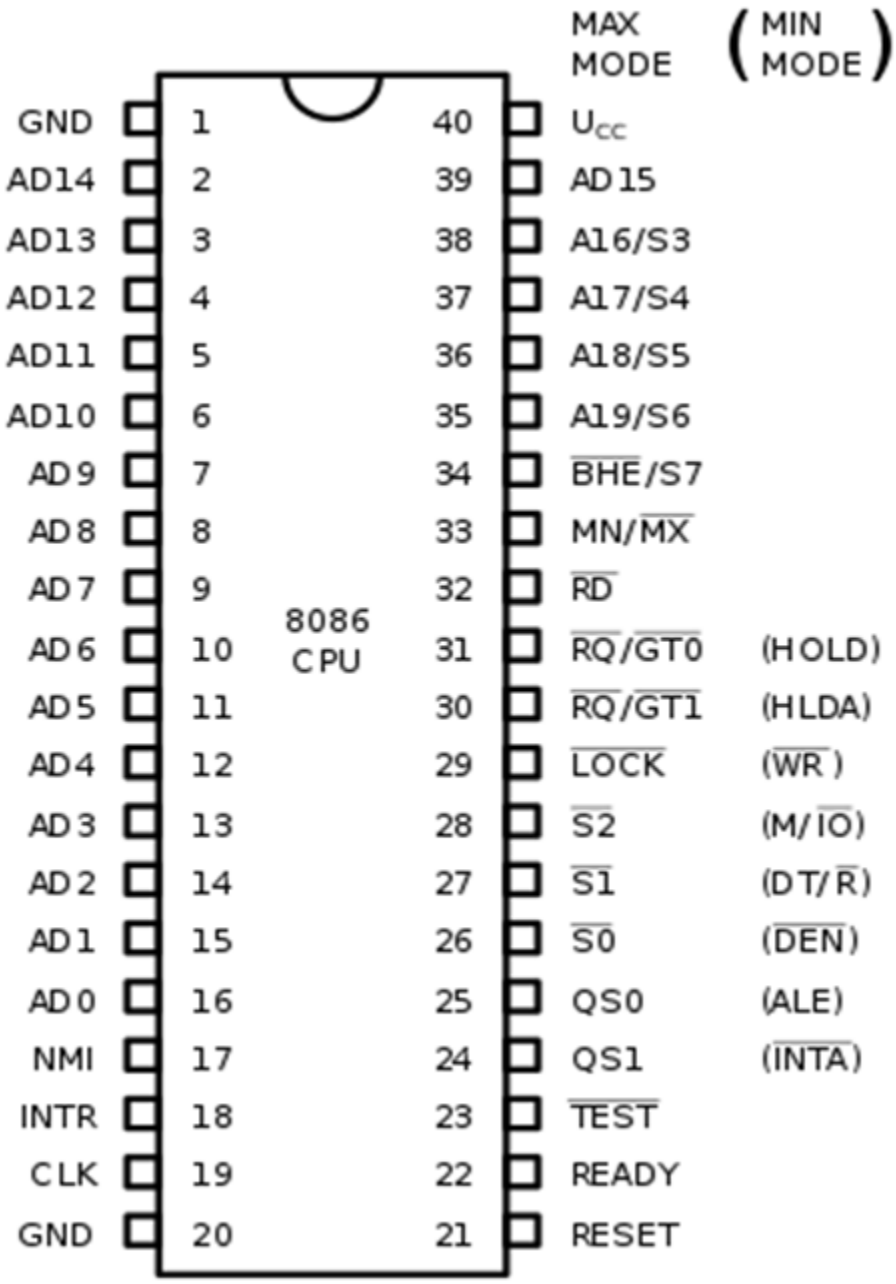
Operand register
It is a 16 bit register used by the control register to hold the operands temporarily. It is not available to the programmer.

Instruction Register and Instruction Decoder
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.

Flag register (16 bits)
- It has 9 flags.
- These flags are of two types: 6 Status flags namely carry flag, parity flag, auxiliary carry flag, zero flag, sign flag and 3 Control flags namely trap flag, interrupt flag and direction flag.
- Status flags are affected by the ALU after every arithmetic or logic operation. They give the status of the current result.
- The Control flags are used to control certain operations. They are changed by the programmer.

Pin Diagram

1. 8086 microprocessors works on voltage level 5 V DC power supply.
2. It comes in a 40-Lead Cerdip and Plastic Package, with 20 on each side.



- **PIN 1 & 20** are GND Pins which should be connected to low potential of the power supply or to the ground of the whole system.
- **PIN 2-16 & 39 : AD15-AD0**,
    - These are ADDRESS DATA BUS. These lines are used for I/O function with time multiplexed memory. These are active high and 3-state during interrupt acknowledge.
- **PIN 17 : NMI, NON-MASKABLE INTERRUPT.**

    - This pin is for creating a type 2 interrupts, which occurs due to a triggered input. This cannot be done internally with software as you have to make it happen physically from LOW to HIGH, but the interrupt is internally synchronized.
- **PIN 18 : INTR, INTERRUPT REQUEST.**
    - This is a level triggered input (active HIGH) which usually occurs at the last clock cycle of each instruction to check whether or not the processor into interrupt acknowledge operation mode.
    - This can be set internally with the help of software by resetting the interrupt enable bit, also the interrupt is synchronized internally.
- **PIN 19 : CLK, CLOCK.**
    - This pin provides the clock signal for the processor and bus control.
    - It is asymmetric with 33% duty cycle to provide optimized internal timing.
- **PIN 21 : RESET.**
    - This pin is used to reset the processor from whatever state it is in. The signal should be active high for at least 4 clock cycles.
    - When the pin is pulled low, it restarts execution from the beginning as in the code or program, all this is internally synchronized.
- **PIN 22 : READY.**
    - This pin reads the ready signal that indicated that the addressed memory or I/O has completed the data transfer. the signal is synchronized by the 8284A Clock Generator along with the address bus to from READY.
    - The signal is active HIGH, but it is not synchronized also it may malfunction if signal timings are not correctly matched.
- **PIN 23 : TEST.**
    - This input is detected by a “WAIT” instruction in 8086 microprocessor. If the input is LOW execution of the program continues, else the processor will wait or delay the task until the signal is LOW.
    - The input is synchronized internally during each clock cycle at the leading edge (start or LOW to HIGH) of CLK signal.
- **PIN 24, 25 : QS1, QS0; QUEUE STATUS.**
    - These two pins are used to determine the queue status.
    - This signal is valid during the CLK cycle after which the queue operation is performed.
    - These are usually used for external visualization of tasks performed by the processor. These pin functions only in maximum mode.
- **PIN 26-28 : S2—S0.**
    - These pins give output which floats to 3-state OFF in hold acknowledge.
    - These have a truth table for all type of combinations, which you van refer to datasheet attached below on pg4.
    - These pin also functions only in maximum mode.
- **PIN 29 : LOCK.**
    - This pin output indicated the supremacy of bus control over other system bus masters.
    - The signal is active low and remains low until completion of the next instruction.
- **PIN 30,31 : RQ/GT, REQUEST/GRANT.**

    - These pins are bidirectional and used by other bus master to request the control over bus at the end of the current bus cycle also they have internal pull-up resistor.
- **PIN 32 : RD, READ.**
    - This output pin indicated that the processor is performing a memory size of 8086 microprocessor or I/O cycle, which totally depends on the state of S2 pin.
    - This pin is used to read devices, which resides on the 8086 local bus.
- **PIN 33 : MN/MX, MINIMUM/MAXIMUM.**
    - This is used to set the processor in either MINIMUM or MAXIMUM mode. And pins function accordingly.
- **PIN 34 : BHE/S7, BUS HIGH ENABLE/STATUS.**
    - This pin provides a signal to enable the data on the most significant half of the data bus (D15-D8).
- **PIN 35-38 : A19/S6-A16/S3, ADDRESS/STATUS.**
    - These pins are the most significant line for memory operations during t1. During I/O operation, these pins are LOW.
- **PIN 40 : VCC.**
    - This is the power pin which should be connected to high potential of the system. The working voltage level of the IC s 5V, and it should not exceed that.

2. 8086 Minimum & Maximum Modes

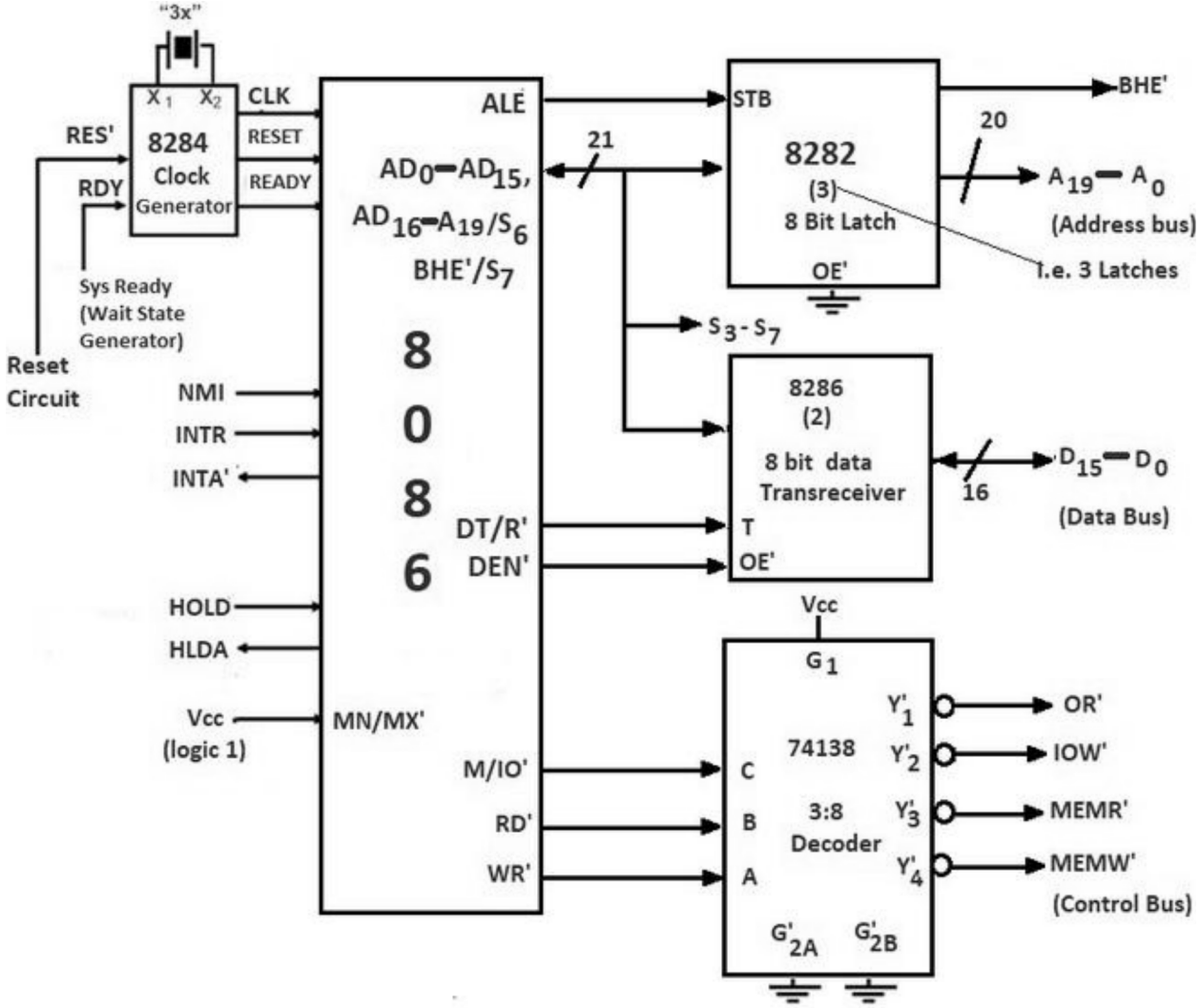
Minimum Mode of 8086

1. The 8086 microprocessor operates in minimum mode when MN/MX' = 1.
2. In minimum mode,8086 is the only processor in the system which provides all the control signals which are needed for memory operations and I/O interfacing.
3. Here the circuit is simple but it does not support multiprocessing.
4. The other components which are transceivers, latches, 8284 clock generator, 74138 decoder, memory and i/o devices are also present in the system.
5. The address bus of 8086 is 20 bits long. By this we can access 220 byte memory i.e. 1MB .
6. Out of 20 bits, 16 bits A0 to A15(or 16 lines) are multiplexed with a data bus.
7. By multiplexing, it means they will act as address lines during the first T state of the machine cycle and in the rest, they act as data lines. A16 to A19 are multiplexed S3 to S6 and BHE' is multiplexed with S7.

Control signals provided by 8086 for memory operations and i/o interfacing :

- They are used to identifying whether the bus is carrying a valid address or not , in which direction data is needed to be transferred over the bus, when there is valid write data on the data bus and when to put read data on the system bus.

- Therefore, their sequence pattern makes all the operations successful in a particular machine cycle.



<u>8282 (8 bits) latch :</u>

- The latches are buffered D FF. They are used to separate the valid address from the multiplexed Address/data bus by using the control signal ALE, which is connected to strobe(STB) of 8282.
- The ALE is active high signal. Here three such latches are required because the address is 20 bits.

<u>8286 (8 bits) transceivers :</u>

- They are bidirectional buffers and also known as data amplifiers. They are used to separate the valid data from multiplexed add/data bus.
- Two such transceivers are needed because the data bus is 16 bits long. 8286 is connected to DT/R' and DEN' signals.
- They are enabled through the DEN signal .The direction of data on the data bus is controlled by the DT/R' signal. DT/R' is connected to T and DEN' is connected to OE'.

DEN'	DT/R'	Action
1	X (don't care)	Transreceiver is disabled
0	0	Receiver data
0	1	Transmit data

- 8284 clock generator is used to provide the clock.
- M/IO'= 1,then I/O transfer is performed over the bus. and when M/IO' = 0, then I/O operation is performed.
- The signals RD' and write WR' are used to identify whether a read bus cycle or a write bus cycle is performing. When WR' = 0 ,then it indicates that valid output data on the data bus.
- RD' indicates that the 8086 is performing a read data or instruction fetch process is occurring .During read operations, one other control signal is also used, which is DEN (data enable) and it indicates the external devices when they should put data on the bus.
- Control signals for all operations are generated by decoding M/IO', RD', WR'. They are decoded by 74138 3:8 decoder.

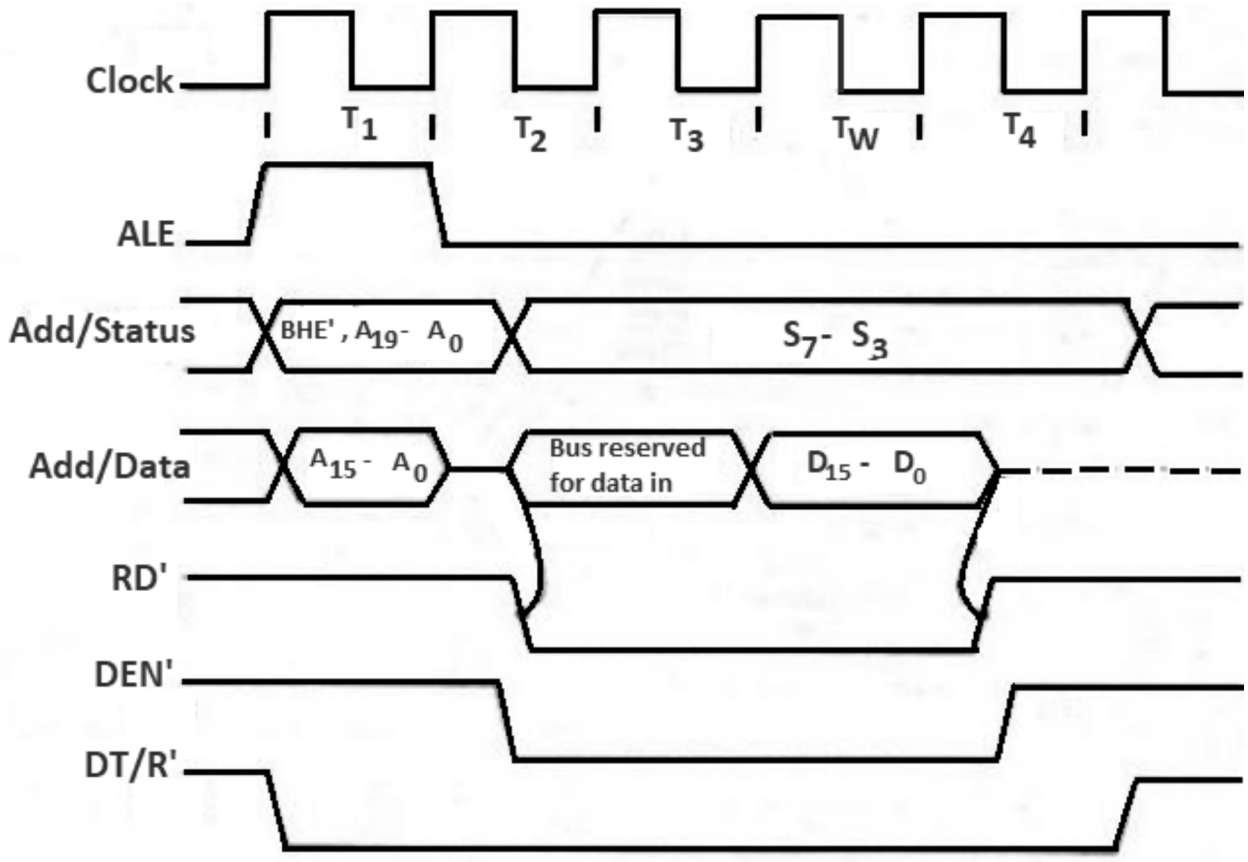
M/IO'	RD'	WR'	Action
1	0	1	Memory Read
1	1	0	Memory Write
0	0	1	I/O Read
0	1	0	I/O Write

INTR and INTA :

- When INTR = 1,then there is an interrupt to 8086 by other devices for their service. When INTA'= 0,then it indicates that the processor is ready to service them.
- The bus request is made by other devices using the HOLD signal and the processor acknowledges them using the HLDA output signal.

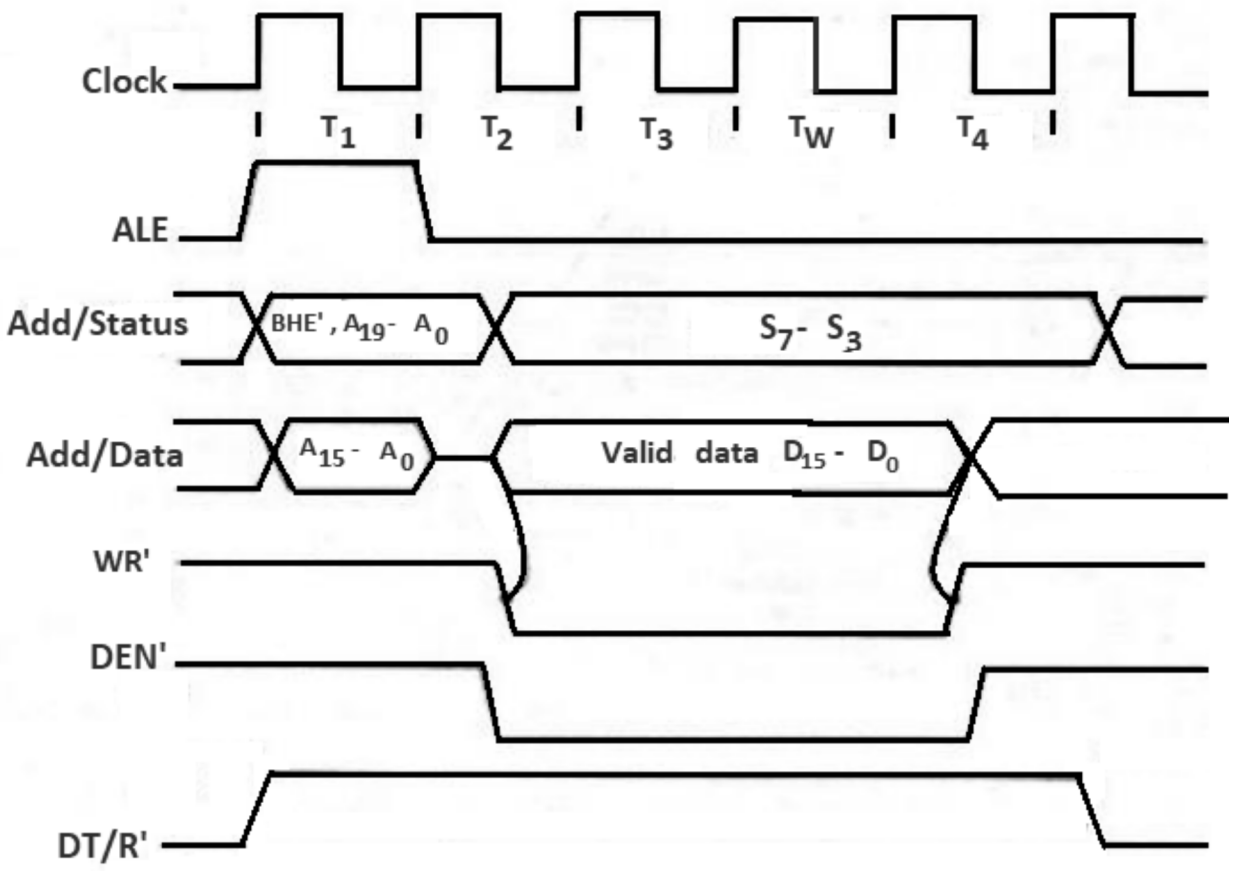
Timing Diagram (Separate Question):

- The working of min mode can be easily understood by timing diagrams.
- All processors bus cycle is of at least 4 T-states(T1,T2,T3,T4) .The address is given by processor in the T1 state. It is available on the bus for one T-state.
- In T2, the bus is tristated for changing the direction of the bus(in the case of a data read cycle.)
- The data transfer takes place between T3 and T4.
- If the addressed device is slower, then the wait state is inserted between T3 and T4.



Opcode fetch or read timing diagram

- At T1 state ALE =1 ,this indicates that a valid address is latched on the address bus and also M / IO'= 1, which indicates the memory operation is in progress.
- In T2, the address is removed from the local bus and is sent to the addressed device. Then the bus is tristated.
- When RD' = 0 , the valid data is present on the data bus.
- During T2 DEN' =0, which enables transceivers and DT/R' = 0 ,which indicates that the data is received.
- During T3, data is put on the data bus and the processor reads it.
- The output device makes the READY line high. This means the output device has performed the data transfer process. When the processor makes the read signal to 1, then the output device will again tristate its bus drivers.

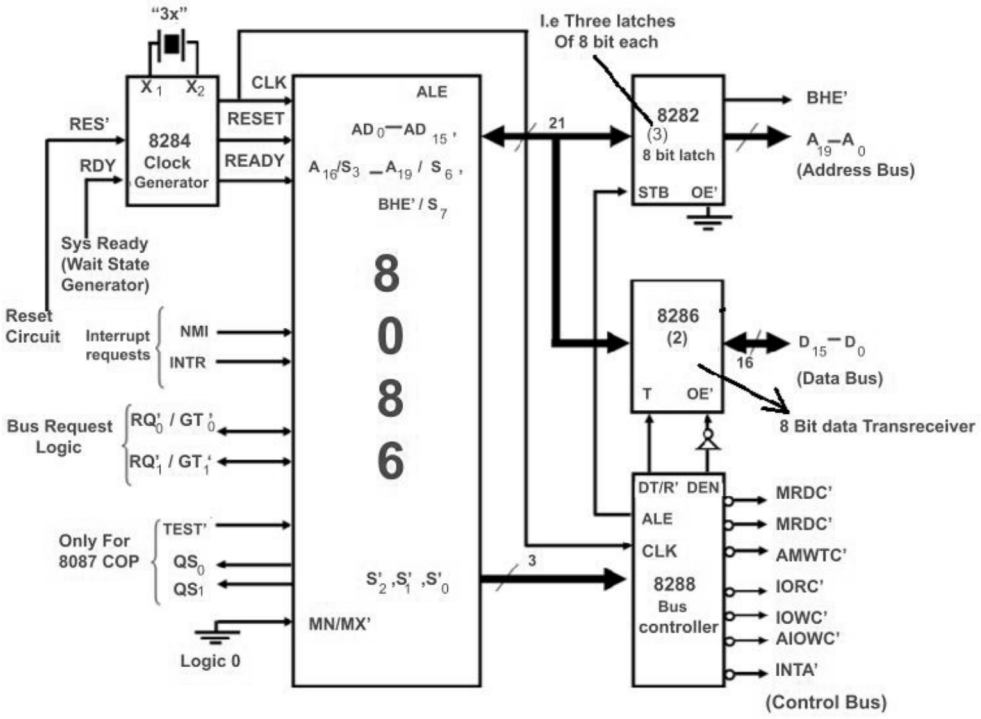


Write memory cycle

- At T1 state ALE =1 ,this indicates that a valid address is latched on the address bus and also M / IO'= 1, which indicates the memory operation is in progress.
- In T2, the processor sends the data to be written to the addressed location.
- The data is buffered on the bus until the middle of T4 state.
- The WR'=0 becomes at the beginning of T2.
- The BHE' and A0 signals are used to select the byte or bytes of memory or I/O word.
- During T2 DEN' =0, which enables, transceivers and DT/R' = 1 ,which indicates that the data is transferred by the processor to the addressed device.
- All kinds of memory and i/o operations are performed using the decoding of M/IO'and RD' WR' as shown in the table above.

Maximum Mode of 8086

- 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.

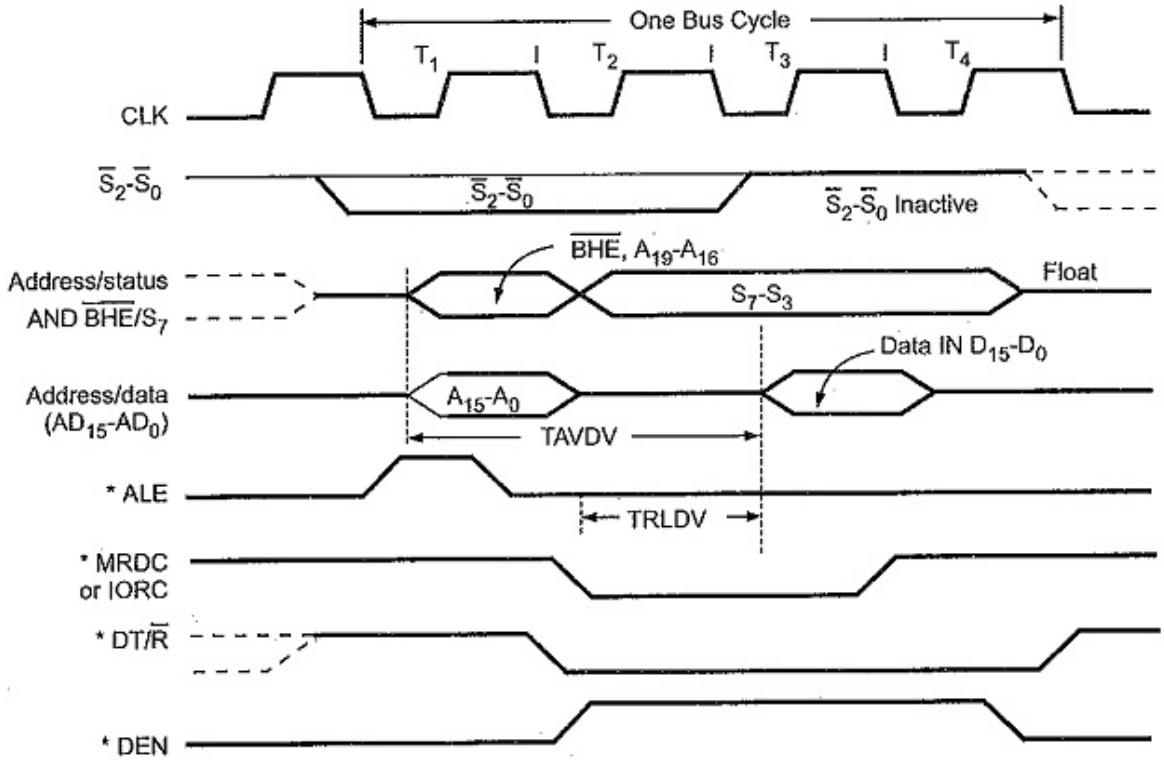
DEN (Of 8288)	DT/ R'	Action
0	X	Transreceiver is disabled
1	0	Receive data
1	1	Transmit data

- Control signals for all operations are generated by decoding S'2, S'1 and S'0 using 8288 bus controller.

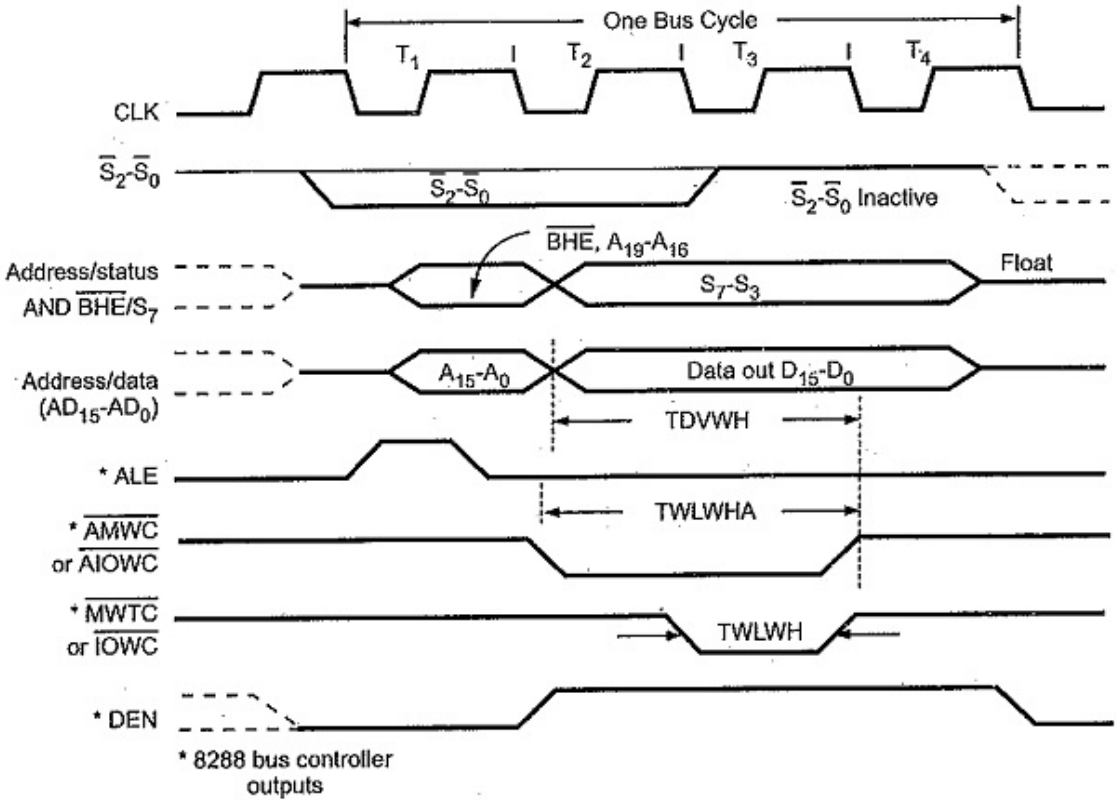
S'_2	S'_1	S'_0	Processor State (What the µP wants to do)	8288 Active Output (What Control signal should 8288 generate)
0	0	0	Interrupt Acknowledge	**INTA'**
0	0	1	Read I/O Port	**IORC'**
0	1	0	Write I/O Port	**IOWC' and AIOWC'**
0	1	1	Halt	None
1	0	0	Instruction Fetch	**MRDC'**
1	0	1	Memory Read	**MRDC'**
1	1	0	Memory Write	**MWTC' and AMWTC'**
1	1	1	Inactive	None

- Bus request is done using RQ' / GT' lines interfaced with 8086. RQ0/GT0 has more priority than RQ1/GT1.
- 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.

Timing Diagram (Separate Question):



Read Operation



Write Operation

These are explained in steps.

1. S0,S1,S2 are set at the beginning of bus cycle. On detecting the change on
2. passive state S0 = S1 = S2 = 1, the 8288 bus controller will output a pulse on its ALE and apply a required signal to its DT/R pin during T1.
3. In T2, 8288 will set DEN = 1 thus enabling transceiver. For an input, 8288 it will activates MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
4. The status bits S0 to S2 remain active until T3, and become passive during T3 and T4.
5. If ready input is not activated before T3, wait state will be inserted between T3 and T4.

Design 8086 microprocessor based system using minimum mode with following specifications:

i) 8086 microprocessor working at 10 MHz.

ii) 128 KB EPROM using 32 KB Chip

iii) 64 KB SRAM using 16 KB Chip

Memory Calculations:
EPROM:

- Required Memory = 128 KB,
- Available Memory = 32 KB
- Number of chip required = 4
- Number of Address Lines = 15 lines (A15 – A1)

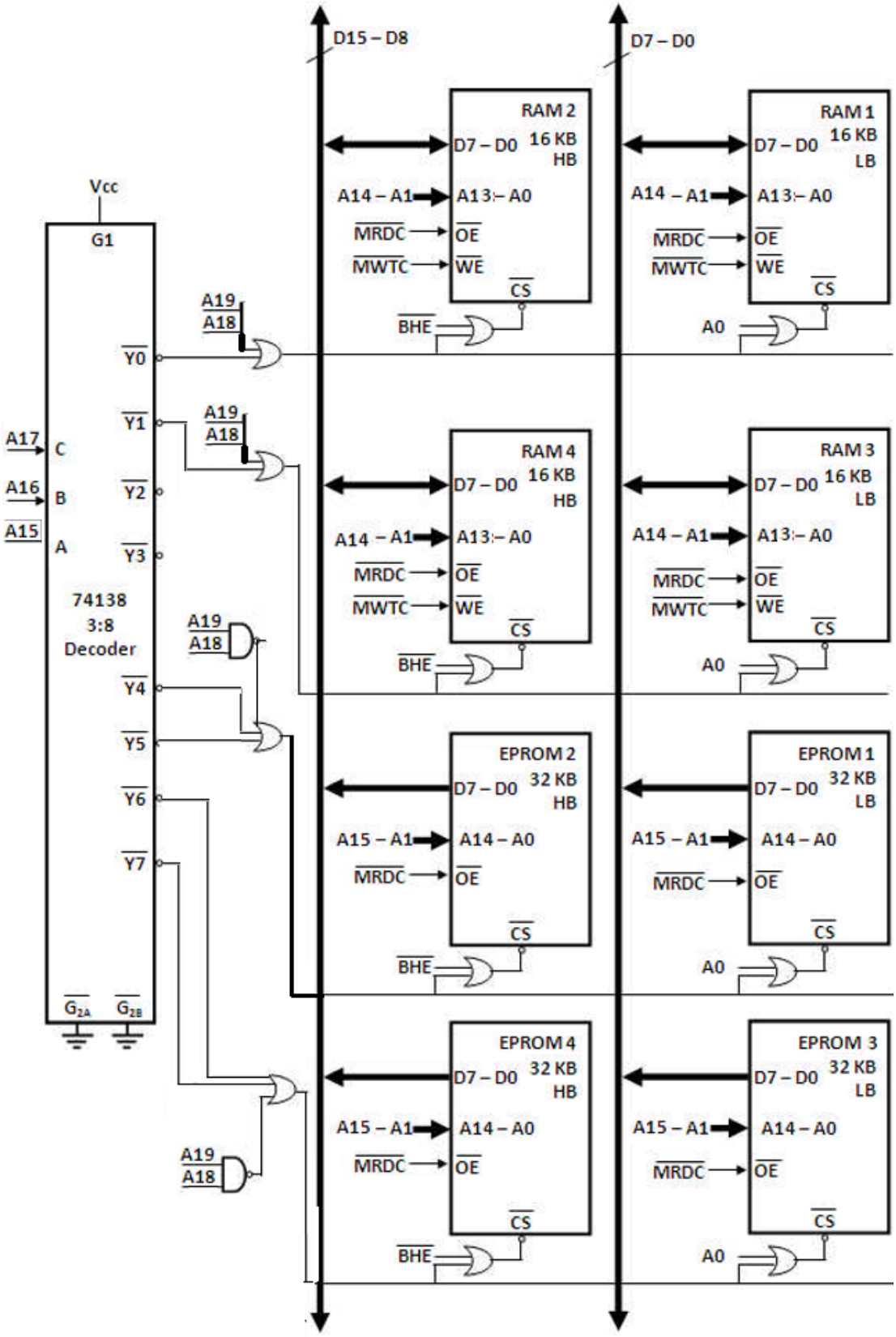
RAM:

- Required Memory = 64 KB,
- Available Memory = 16 KB
- Number of chip required = 4
- Number of Address Lines = 14 lines (A14 – A1)

Memory Map:

	Chip Select					Address Lines														
Memory Chip	**A19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**A0**	**Memory Address**
RAM 1 (LB)	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**00000H**
	0	**0**	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**07FFEH**
RAM 2 (HB)	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**00001H**
	0	**0**	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**07FFFH**
RAM 3 (LB)	**0**	**0**	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**08000H**
	0	**0**	**0**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0FFFEH**
RAM 4 (HB)	**0**	**0**	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**08001H**
	0	**0**	**0**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0FFFFH**
	Chip Select					**Address Lines**														
ROM 1 (LB)	**1**	**1**	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**E0000H**
	1	**1**	**1**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**EFFFEH**
ROM 2 (HB)	**1**	**1**	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**E0001H**
	1	**1**	**1**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**EFFFFH**
ROM 3 (LB)	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**F0000H**
	1	**1**	**1**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**FFFFEH**
ROM 4 (HB)	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**F0001H**
	1	**1**	**1**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**FFFFFH**

Final Configuration:

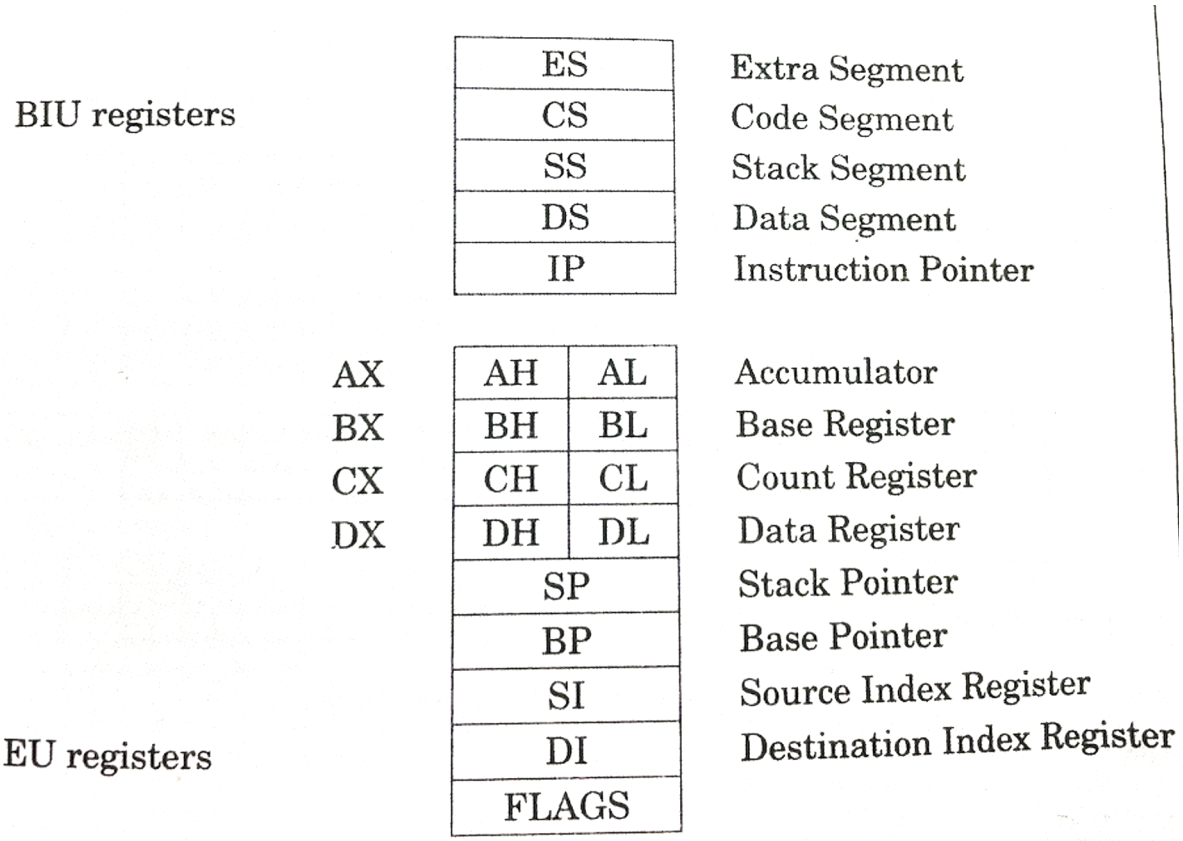


3. Programmer's Model

The 8086 programmer's model is a picture of the processor as available to the grammer. These are the registers used to hold numbers and addresses, as well as indicate status and act as controls.

Data Registers

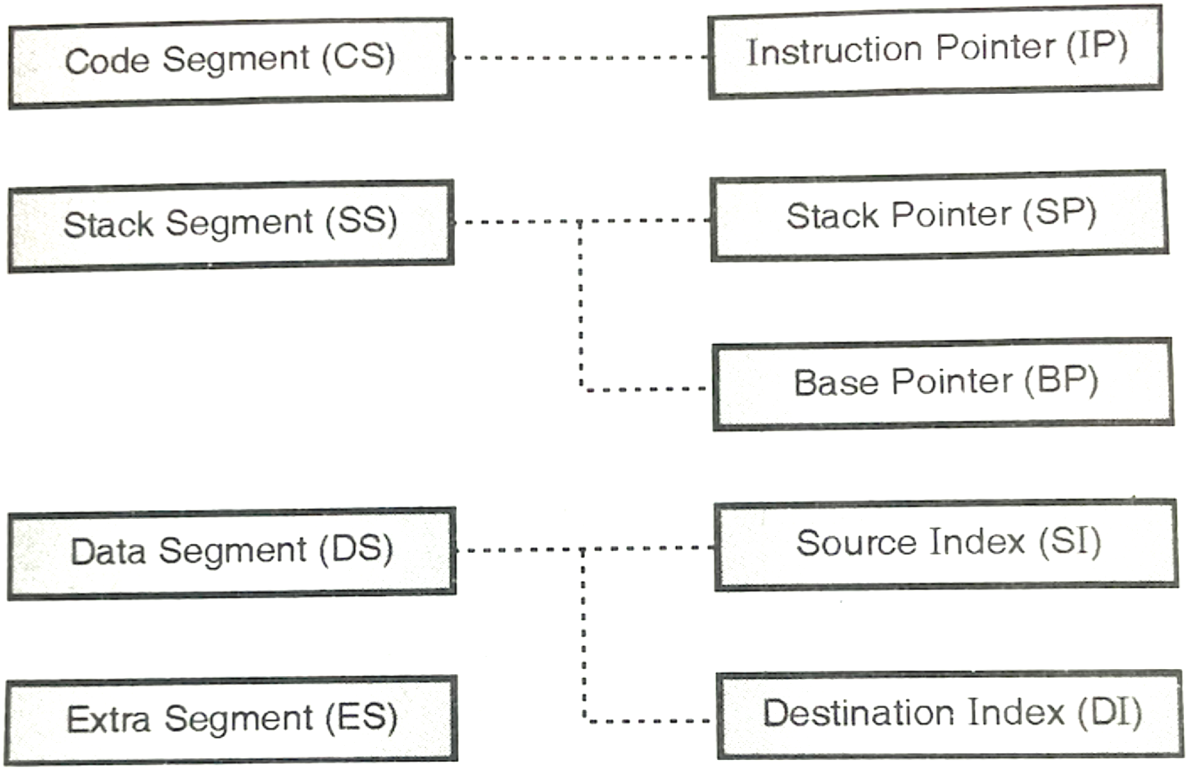
1. For 16-bit data registers can be accessed by names AX, BX, CX and DX,
2. Individual bytes of these registers can be accessed by name, AH, AL, BH, BL, CH, CL, DH and DL



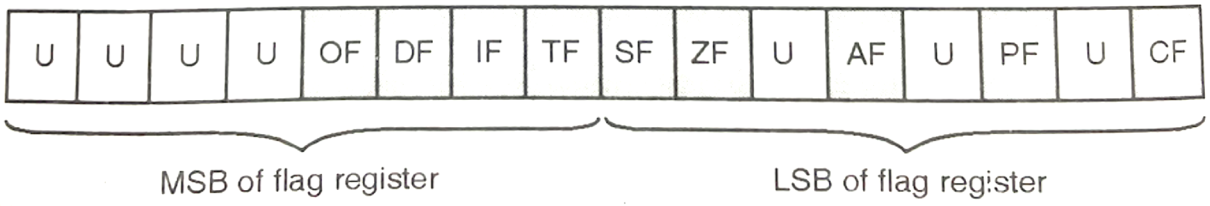
Address Registers:

1. The segment register are CS, DS, ES and SS and offset is stored in instruction pointer IP or stack pointer SP or base registers BX or BP, or one of the index registers SI or DI
2. The CS:IP pair gives the address of the next instruction to be executed in the program sequence.
3. The SS.SP pair gives the address of the top of the stack, a temporary storage area often automatically used by the computer. SP is used for sequential access of the stack.
4. The SS:BP pair is used as a pointer into the stack. BP is used for random access of the stack
5. The DS:SI pair is used as a source pointer and ES: DI pair is used as destination pointer for string instructions. For all other instructions DI register is used with DS segment register.

6. The DS:SI and ES:DI registers are used as general purpose pointers for copying and data processing.
7. The dotted lines below show common default couplings. These defaults can be overridden by a notation such as DS:[BP], where DS will be used in place of the default SS.
8. The BX data register not shown is also used as a pointer in the data segment (by default).



Flag Registers:

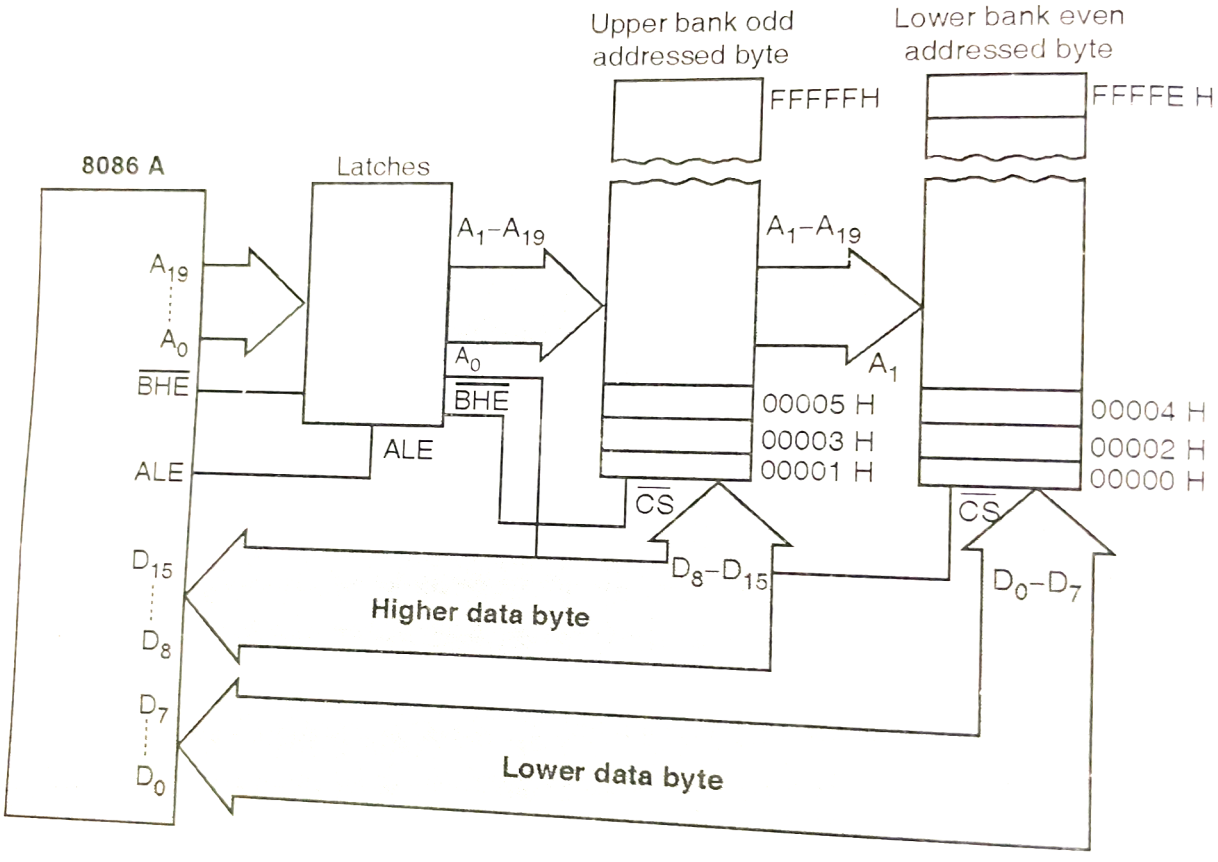


1. CF = Carry Out
2. PF = Parity Of Last Operation
3. ZF = Zero result
4. TF = Trap Flag
5. DF = Direction Flag
6. U = It can be 0 or 1. It is undefined
7. AF = Auxiliary Carry (used in BCD arithmetic)
   SF = Sign bit from last operation

8. IF = Interrupt enable flag
9. OF = Overflow (error for signed numbers)

4. Memory Banking in 8086

1. 8086 has a 16-bit data bus hence it should be able to access 16-bit data in one operation.
2. But the memory chips available are normally such that each location has 8-bits i.e. a byte.
3. Hence, to read 16-bits it needs to access 2 memory locations. However, if both these memory locations are in same memory chip then the address bus will have to provide two addresses sequentially and will require double the time also 16-bits would not be accessed simultaneously.
4. To solve this problem, the memory of 8086 is divided into two banks each bank provides 8-bits.
5. One bank contains all even addresses called the "Even bank", while the other is called "Odd bank" containing all odd addresses.



Memory location	Address	Data Type	$\overline{BHE}$	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	$D_8 - D_{15}$

Case I: Byte access from an even location

1. Since the address in even A_0 is '0'
2. $\overline{BHE}$ signal is kept disabled as only even bank access is required
3. The access is completed in one cycle
4. The data is accessed on data lines D_0 to D_7
5. For e.g. when the byte required is from an even address like (00000)H, it implies on byte from 00000H i.e. first location of even bank as shown in the figure.

Case II: Byte access from an odd location

1. Since the address in odd A_0 is '1'
2. $\overline{BHE}$ signal is kept enabled as odd bank access is required
3. The access is completed in one cycle
4. The data is accessed on data lines D_8 to D_{15}
5. For e.g. when the byte required is from an odd address like (00001 H, it implies one byte from 00001H i.e. first location of odd bank as shown in the figure.

Case III: Word access from an even location

1. Since the address in even A_0 is '0'
2. $\overline{BHE}$ signal is kept enabled as odd bank access is also required
3. The access is completed in one cycle
4. The data is accessed on data lines D_0 to D_7 (from even bank) and D_8 to D_{15} (from odd bank)
5. For eg. when the word required is from an even address like (00000)H, it implies
    a. One byte from 00000H i.e. first location of even bank as shown in the figure.
    b. One byte from 00001H i.e. first location of odd bank as shown in the figure
6. Hence, this is an aligned access and can be done in one cycle.

Case IV: Word access from an odd location

1. Since the address in odd A_0 is '1'
2. $\overline{BHE}$ signal is kept enabled as odd bank access is also required
3. The access is completed in two cycles
4. The data is accessed on data lines D_8 to D_{15} in the first access from odd bank
5. The data is accessed on data lines D_0 to D_7 in the second access from the even bank
6. This access is not possible in the same cycle as it is a unaligned access

7. Unaligned access is one wherein the data to be accessed are in different rows of the two memory chips
8. For e.g. when the word required is from an odd address like (00001)H, it implies
    a. One byte from 00001H L.e. first location of odd bank as shown in the figure.
    b. And another byte from 00002H i.e. second location of even bank as shown in figure.
9. Since the locations in the two banks are not same, two accesses are required

5. Interrupts in 8086 & IVT (Interrupt Vector Table)

Interrupts

1. An interrupt is a condition that halts the microprocessor temporarily to work on a different task and then returns to its previous task.
2. An interrupt is an event or signal that requests the CPU's attention. This halt allows peripheral devices to access the microprocessor.
3. Whenever an interrupt occurs, the processor completes the current instruction and starts the implementation of an Interrupt Service Routine (ISR) or Interrupt Handler.
4. ISR is a program that tells the processor what to do when the interrupt occurs. After the ISR execution, control returns to the main routine where it was interrupted.
5. In the 8086 microprocessor following tasks are performed when the microprocessor encounters an interrupt:
6. The value of the flag register is pushed into the stack. It means that first, the value of SP (Stack Pointer) is decremented by two then the value of the flag register is pushed to the memory address of the stack segment.
7. The value of starting memory address of CS (Code Segment) is pushed into the stack.
8. The value of IP (Instruction Pointer) is pushed into the stack.
9. IP is loaded from word location (Interrupt type) * 04.
10. CS is loaded from the following word location.
11. Interrupt, and Trap flags are reset to 0.
12. The different types of interrupts present in the 8086 microprocessor are given by:

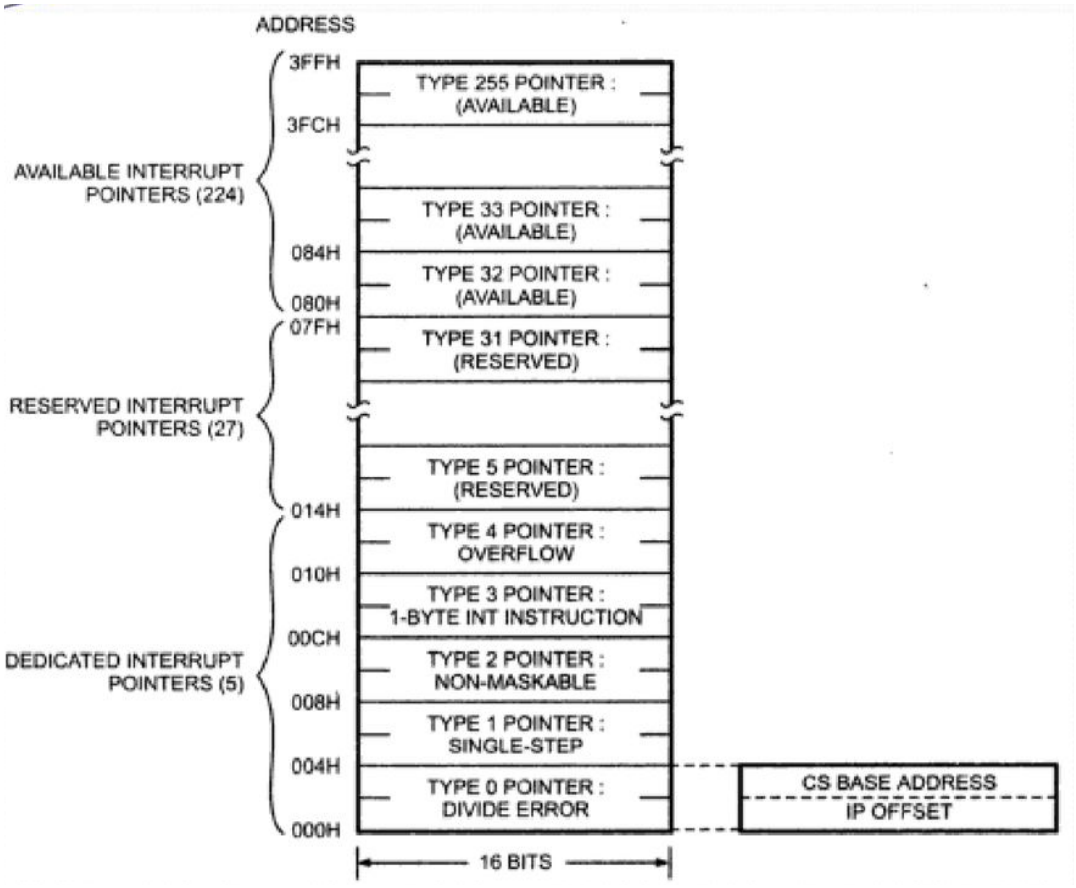
Hardware Interrupts

1. Hardware interrupts are those interrupts that are caused by any peripheral device by sending a signal through a specified pin to the microprocessor. There are two hardware interrupts in the 8086 microprocessor. They are:
    a. NMI (Non-Maskable Interrupt): It is a single pin non-maskable hardware interrupt that cannot be disabled. It is the highest priority interrupt in the 8086 microprocessor. After its execution, this interrupt generates a TYPE 2 interrupt. IP is loaded from word location 00008 H, and CS is loaded from the word location 0000A H.
    b. INTR (Interrupt Request): It provides a single interrupt request and is activated by the I/O port. This interrupt can be masked or delayed. It is a level-triggered interrupt. It can receive any interrupt type, so the value of IP and CS will change on the interrupt type received.

Software Interrupts

1. These are instructions inserted within the program to generate interrupts. There are 256 software interrupts in the 8086 microprocessor.
2. The instructions are of the format INT type, where the type ranges from 00 to FF. The starting address ranges from 00000 H to 003FF H.
3. These are 2-byte instructions. IP is loaded from type * 04 H, and CS is loaded from the following address given by (type * 04) + 02 H. Some important software interrupts are:
    a. TYPE 0 corresponds to division by zero(0).
    b. TYPE 1 is used for single-step execution for debugging the program.
    c. TYPE 2 represents NMI and is used in power failure conditions.
    d. TYPE 3 represents a break-point interrupt.
    e. TYPE 4 is the overflow interrupt.

Interrupt Vector Table (IVT):



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.
2. 8086 supports total 256 types i.e. 00H to FFH.
3. For each type it has to reserve four bytes i.e. double word. This double word pointer contains the address of the procedure that is to service interrupts of that type.

4. The higher addressed word of the pointer contains the base address of the segment containing the procedure. This base address of the segment is normally referred as NEW CS.
5. The lower addressed word contains the procedure's offset from the beginning of the segment. This offset is normally referred as NEW IP.
6. Thus NEW CS: NEW IP provides NEW physical address from where user ISR routine will start.
7. As for each type, four bytes (2 for NEW CS and 2 for NEW IP) are required; therefore interrupt pointer table occupies up to the first 1k bytes (i.e. 256 x 4 = 1024 bytes) of low memory.
8. 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 interrupts (INT 32…..INT 225)

Dedicated interrupts (INT 0…..INT 4):

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.

1. This condition normally occurs when the divisor is very small as compared to the dividend or the divisor is zero.
2. Its ISR address is stored at location 0 x 4 = 00000H in the IVT.

INT 1 (Single Step)

1. The microprocessor executes this interrupt after every instruction if the TF is set.
2. It puts microprocessor in single stepping mode i.e. the microprocessor pauses after executing every instruction. This is very useful during debugging.
3. Its ISR generally displays contents of all registers. Its ISR address is stored at location 1 x 4 = 00004H in the IVT.

INT 2 (Non mask-able Interrupt)

1. The microprocessor executes this ISR in response to an interrupt on the NMI (Non mask-able Interrupt) line.
2. Its ISR address is stored at location 2 x 4 = 00008H in the IVT.

INT 3 (Breakpoint Interrupt)

1. This interrupt is used to cause breakpoints in the program. It is caused by writing the instruction INT 03H or simply INT.
2. It is useful in debugging large programs where single stepping is efficient.
3. 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.

INT 4 (Overflow Interrupt)

1. This interrupt occurs if the overflow flag is set and the microprocessor executes the INTO (Interrupt on Overflow) instruction.

2. It is used to detect overflow error in signed arithmetic operations.
3. Its ISR address is stored at location 4 x 4 = 00010H in the IVT.

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.

Available interrupts (INT 32…..INT 225):

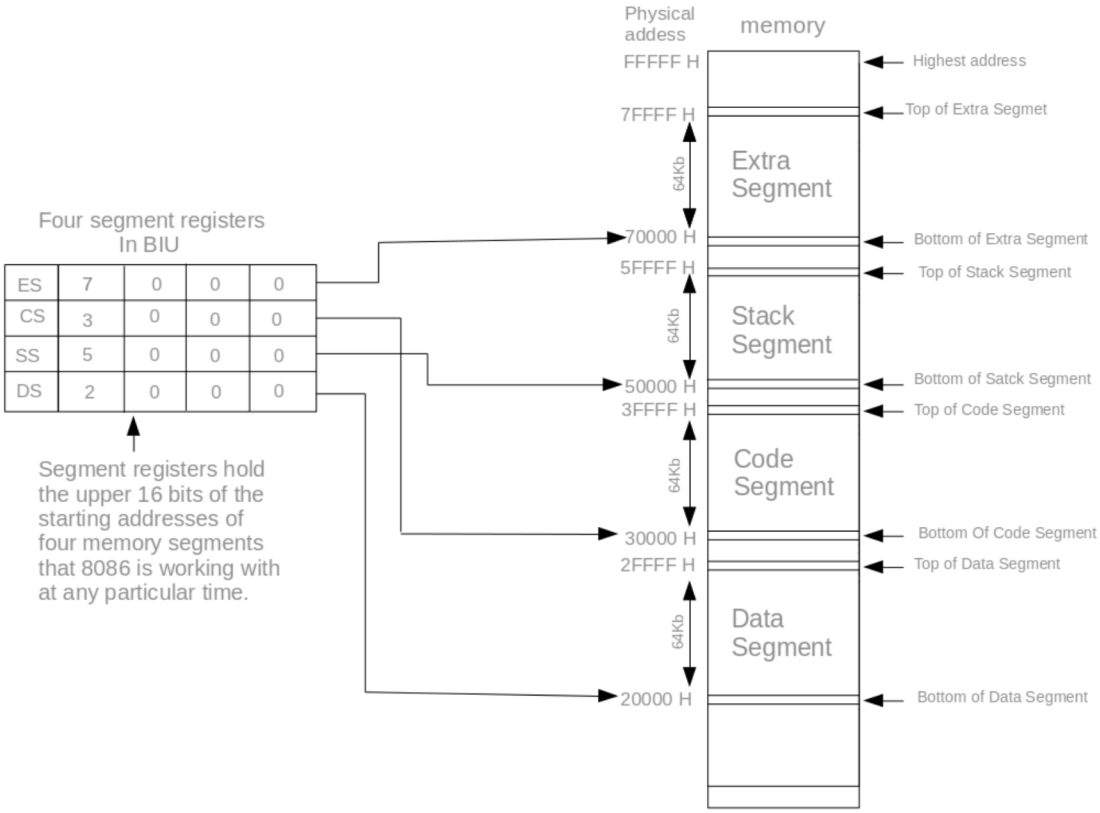
1. These are user defined, software interrupts.
2. ISRs for these interrupts are written by the users to service various user defined conditions.
3. These interrupts are invoked by writing the instruction INT n.
4. Its ISR address is obtained by the microprocessor from location n x 4 in the IVT.

6. Segmentation in 8086

- 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.

- **Need for Segmentation**
  The Bus Interface Unit (BIU) contains four 16 bit special purpose registers (mentioned below) called as Segment Registers.
    - Code segment register (CS): is used for addressing memory location in the code segment of the memory, where the executable program is stored.
    - Data segment register (DS): points to the data segment of the memory where the data is stored.
    - Extra Segment Register (ES): also refers to a segment in the memory which is another data segment in the memory.
    - Stack Segment Register (SS): is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.
- The number of address lines in 8086 is 20, 8086 BIU will send 20bit address, so as to access one of the 1MB memory locations.
- The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time.
- A segment is a logical unit of memory that may be up to 64 kilobytes long. Each segment is made up of contiguous memory locations.
- It is an independent, separately addressable unit. Starting address will always be changing. It will not be fixed.
- Note that the 8086 does not work the whole 1MB memory at any given time. However, it works only with four 64KB segments within the whole 1MB memory.

- Below is the one way of positioning four 64 kilobyte segments within the 1M byte memory space of an 8086.



Types Of Segmentation

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.

Rules of Segmentation:

The starting address of a segment should be such that it can be evenly divided by 16.
Minimum size of a segment can be 16 bytes and the maximum can be 64 kB.

Segment	**Offset registers**	**Function**
CS	IP	Address of the next function
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (string operations)

Advantages

1. It provides a powerful memory management mechanism.
2. Data related or stack related operations can be performed in different segments.
3. Code related operation can be done in separate code segments.
4. It allows to processes to easily share data.
5. 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.
6. 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.

Module 2 - Instruction Set & Programming

1. Addressing Modes

The different ways in which a source operand is denoted in an instruction is known as addressing modes.
There are 8 different addressing modes in 8086 programming −

1. Immediate addressing mode

1. The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

Example

```
MOV CX, 4929 H, ADD AX, 2387 H,  MOV AL, FFH
```

2. Register addressing mode

1. It means that the register is the source of an operand for an instruction.

Example

```
MOV CX, AX    ; copies the contents of the 16-bit AX register into
              ; the 16-bit CX register),
ADD BX, AX
```

3. Direct addressing mode

1. The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example

```
MOV AX, [1592H], MOV AL, [0300H]
```

4. Register indirect addressing mode

1. This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

Example

```
MOV AX, [BX]  ; Suppose the register BX contains 4895H, then the contents
              ; 4895H are moved to AX
ADD CX, {BX}
```

5. Based addressing mode

1. In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

Example

```
MOV DX, [BX+04], ADD CL, [BX+08]
```

6. Indexed addressing mode

1. In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

Example

```
MOV BX, [SI+16], ADD AL, [DI+16]
```

7. Based-index addressing mode

1. In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

Example

```
ADD CX, [AX+SI], MOV AX, [AX+DI]
```

8. Based indexed with displacement mode

1. In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

Example

```
MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]
```

2. Instruction Set

Note: If this is asked as a theory question "Explain instruction set of 8086 read 5 instructions from every set, but in general you should be aware of the description because we can expect a question like "Explain MOV, LDS, LEA instructions of 8086"

Instructions are classified on the basis of functions they perform. They are categorized into the following main types:

Data Transfer instruction

All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

MOV	Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc.
LDS	Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register.
LES	Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register.
LEA	Loads offset address into the specified register.
LAHF	Loads low order 8-bits of the flag register into AH register.
SAHF	Stores the content of AH register into low order bits of the flags register.
XLAT/XLATB	Reads a byte from the lookup table.
XCHG	Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations.
PUSH	Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack.
POP	Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s).

POPF	Pops (reads) two bytes from the top of the stack and keeps them in the flag register.
IN	Transfers data from a port to the accumulator or AX, DX or AL register.
OUT	Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction.

Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	Decimal Adjust after BCD Addition: When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.

DAS	Decimal Adjust after BCD Subtraction: When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD.
AAA	ASCII Adjust for Addition: When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD.
AAD	Adjust AX Register for Division: It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte.
AAM	Adjust result of BCD Multiplication: This instruction is used after the multiplication of two unpacked BCD.
AAS	ASCII Adjust for Subtraction: This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number.
CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory location(s).

Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations.

AND	Performs bit by bit logical AND operation of two operands and places the result in the specified destination.
OR	Performs bit by bit logical OR operation of two operands and places the result in the specified destination.

XOR	Performs bit by bit logical XOR operation of two operands and places the result in the specified destination.
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Rotate Instructions

RCL	Rotate all bits of the operand left by specified number of bits through carry flag.
RCR	Rotate all bits of the operand right by specified number of bits through carry flag.
ROL	Rotate all bits of the operand left by specified number of bits.
ROR	Rotate all bits of the operand right by specified number of bits.

Shift Instructions:

SAL or SHL	Shifts each bit of operand left by specified number of bits and put zero in LSB position.
SAR	Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB.
SHR	Shift each bit of operand right by specified number of bits and put zero in MSB position.

Branch Instructions:

It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target.

JA or JNBE	Jump if above, not below, or equal i.e. when CF and ZF = 0
JAE/JNB/JNC	Jump if above, not below, equal or no carry i.e. when CF = 0
JB/JNAE/JC	Jump if below, not above, equal or carry i.e. when CF = 0

JBE/JNA	Jump if below, not above, or equal i.e. when CF and ZF = 1
JCXZ	Jump if CX register = 0
JE/JZ	Jump if zero or equal i.e. when ZF = 1
JG/JNLE	Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF
JGE/JNL	Jump if greater, not less or equal i.e. when SF = OF
JL/JNGE	Jump if less, not greater than or equal i.e. when SF ≠ OF
JLE/JNG	Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF
JMP	Causes the program execution to jump unconditionally to the memory address or label given in the instruction.
CALL	Calls a procedure whose address is given in the instruction and saves their return address to the stack.
RET	Returns program execution from a procedure (subroutine) to the next instruction or main program.
IRET	Returns program execution from an interrupt service procedure (subroutine) to the main program.
INT	Used to generate software interrupt at the desired point in a program.
INTO	Software interrupts to indicate overflow after arithmetic operation.
LOOP	Jump to defined label until CX = 0.
LOOPZ/LOOPE	Decrement CX register and jump if CX ≠ 0 and ZF = 1.
LOOPNZ/LOOPNE	Decrement CX register and jump if CX ≠ 0 and ZF = 0.

Here, CF = Carry Flag, ZF = Zero Flag, OF = Overflow Flag, SF = Sign Flag, CX = Register

String Instructions:

String is series of bytes or series of words stored in sequential memory locations.
The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

MOVS/MOVSB/MOVSW	Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register.
CMPS/CMPSB/CMPSW	Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register.
SCAS/SCASB/SCASW	Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES.
LODS/LODSB/LODSW	Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register.
STOS/STOSB/STOSW	Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register.
REP	Repeats the given instruction until CX ≠ 0
REPE/ REPZ	Repeats the given instruction till CX ≠ 0 and ZF = 1
REPNE/REPNZ	Repeats the given instruction till CX ≠ 0 and ZF = 0

3. Assembly Language Program

Assembly language program to display the contents of 16 bit flag register.

The flag register is one of the special purpose register. The flag bits are changed to 0 or 1 depending upon the value of result after arithmetic or logical operations.
8086 has 16-bit flag register, and there are 9 valid flag bits. The format of flag register is like below.

Bits	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Flags					O	D	I	T	S	Z		AC		P		CY

We can divide the flag bits into two sections. The Status Flags, and the Control Flags.

Status Flags

In 8086 there are 6 different flags which are set or reset after 8-bit or 16-bit operations. These flags and their functions are listed below.

Flag Bit	Function
S	After any operation if the MSB is 1, then it indicates that the number is negative. And this flag is set to 1
Z	If the total register is zero, then only the Z flag is set
AC	When some arithmetic operations generates carry after the lower half and sends it to upper half, the AC will be 1
P	This is even parity flag. When result has even number of 1, it will be set to 1, otherwise 0 for odd number of 1s
CY	This is carry bit. If some operations are generating carry after the operation this flag is set to 1
O	The overflow flag is set to 1 when the result of a signed operation is too large to fit.

Control Flags

In 8086 there are 3 different flags which are used to enable or disable some basic operations of the microprocessor. These flags and their functions are listed below.

Flag Bit	Function
D	This is directional flag. This is used in string related operations. D = 1, then the string will be accessed from higher memory address to lower memory address, and if D = 0, it will do the reverse.
I	This is interrupt flag. If I = 1, then MPU will recognize the interrupts from peripherals. For I = 0, the interrupts will be ignored
T	This trap flag is used for on-chip debugging. When T = 1, it will work in a single step mode. After each instruction, one internal interrupt is generated. It helps to execute some program instruction by instruction.

Algorithm to display the contents of flag register

1. Initialize the data segment
2. Initialize the array FDATA with the flag register names
3. Also initialize array NEW1 with blank contents and 16bit array FLAG
4. In code segment, first display the arrays FDATA and NEW1 using int21 and function 09h
5. Write flag manipulation instructions to clear interrupt flag, set Carry and Direction flags
6. To access the content of flag, push the flag register to stack
7. Pop the flag content from stack to bx register
8. Flag content from bx is moved to the variable flag defined in data segment
9. Move 16 to CX to set count
10. Move 8000h to BX
11. Move Flag to AX and do AND operation with BX
12. Display 1, if result of AND is 1
13. Else if result of AND is zero,then jump to label ifzero and display 0
14. Add space after displaying either 0 or 1
15. Rotate content of bx to right by 1
16. Repeat from Step 11, until CX becomes 0.
17. Terminate the program

```
DATA SEGMENT
FDATA DB 0DH, 0AH, "-- -- -- -- 0F DF IF TF SF ZF -- AF -- PF -- CF $"
NEW1 DB 0DH, 0AH, "$"
FLAG DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:
MOV AX, DATA
MOV DS, AX


MOV DX, OFFSET FDATA
MOV AH, 09H     ; to print a string, ah should have function value 09h which works along with interrupt 21h
int 21h
MOV DX, OFFSET NEW1
MOV AH, 09H
INT 21H

CLI
STC
STD        ; some string manipulation operation

PUSHF  ; to access the content of flag, push the flag register to stack
POP BX   ; pop the flag content from stack to bx register
```

```
MOV FLAG, BX  ;flag content from bx is moved to the variable flag defined in data segment
MOV CX, 16     ; to set count as 16
MOV BX, 8000H  ;1000 0000 0000 0000 - to do AND  on flag content (to check the flag value one bit at a time)
AGAIN: MOV AX, FLAG
       AND AX, BX
       JZ IFZERO   ; if result of AND is zero,then jump to label ifzero

       MOV DL, 31H   ;to display 1, if result of AND is 1.  ASCII value of 1 is 31h
       MOV AH, 02h
      INT 21h
       JMP SPACE

IFZERO: MOV DL, 30h   ; ascii value of 0 is 30h
       MOV AH, 02H
       INT 21H

SPACE: MOV DL, ' '
       MOV AH, 02H
       INT 21H

       MOV AH, 02H     ;to print 2 space
       INT 21H

       ROR BX, 1     ; rotate content of bx to right by 1.
                ; so it will be 0100 0000 0000 0000 in the second loop

       LOOP AGAIN    ;loop instruction will automatically decrement cx value after each iteration

       MOV AH, 4CH   ;to terminate the program
       INT 21H
CODE ENDS
       END START
```

Assembly language program using 8086 microprocessor for addition of 16/32 bit numbers

PROGRAM CODE:

LABEL	PROGRAM CODE	COMMENTS
	DATA SEGMENT	
	NUM1 DW 6000H	create a 16-bit variable initialized to some value
	NUM2 DW 3000H	Create a 16-bit variable initialized to some value
	SUM DW ?	create a 16-bit variable without initializing
	CARRY DB 00H	This variable will store whether there was a carry bit during add
	DATA ENDS	
	CODE SEGMENT	
START:	ASSUME CS:CODE, DS:DATA	Initialize CS and DS registers
	MOV AX,DATA	
	MOV DS,AX	
	MOV AX,NUM1	move value of NUM1 to AX
	MOV BX,NUM2	move value of NUM2 to BX
	ADD AX,BX	Add registers AX and BX
	MOV SUM,AX	moves AX into SUM
	JNC NEXT	Jump if no carry to NEXT
	MOV CARRY, 01H	Store 01H in CARRY
NEXT:	INT 03H	breakpoint interrupt
	CODE ENDS	
	END START	

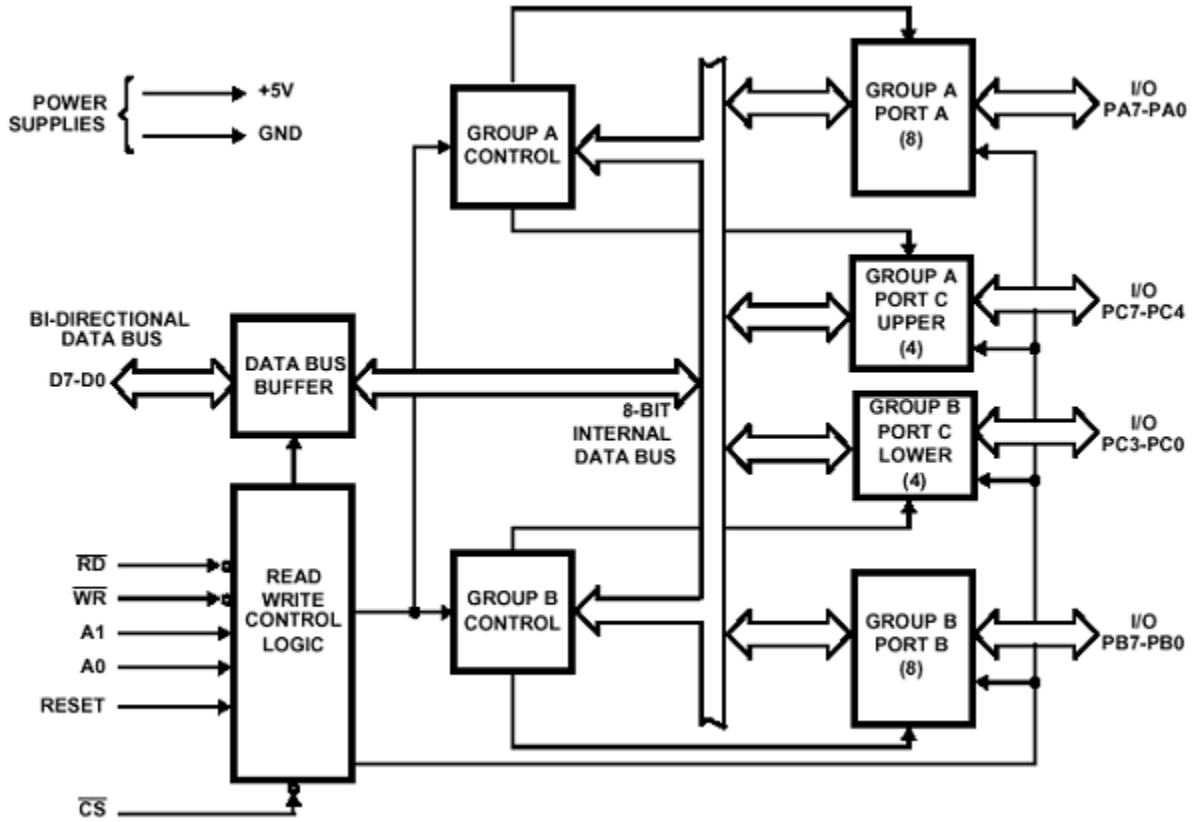
4. Macros v/s Procedures

PROCEDURE	MACROS
Procedures are used for large group of instructions to be repeated.	Macros are used for small group of instructions to be repeated.
Object code is generated only once in memory.	Object code is generated everytime the macro is called.
CALL & RET instructions are used to call procedure and return from procedure	Macro can be called just by writing its name.
Length of the object file is less.	Object file becomes lengthy.
Directives PROC & ENDP are used for defining procedure..	Directives MACRO and ENDM are used for defining MACRO.
More time is required for it's execution.	Less time is required for it's execution.
Procedure can be defined as Procedure_name PROC ------ ------ Procedure_name ENDP	Macro can be defined as MACRO-name MACRO [ARGUMENT ,…….. ARGUMENT N] ------ ------- ENDM
For Example Addition PROC near ------ Addition ENDP	For Example Display MACRO msg ---------

Module 3 - Memory & Peripherals Interfacing

1. 8255 - Programmable Peripheral Interface

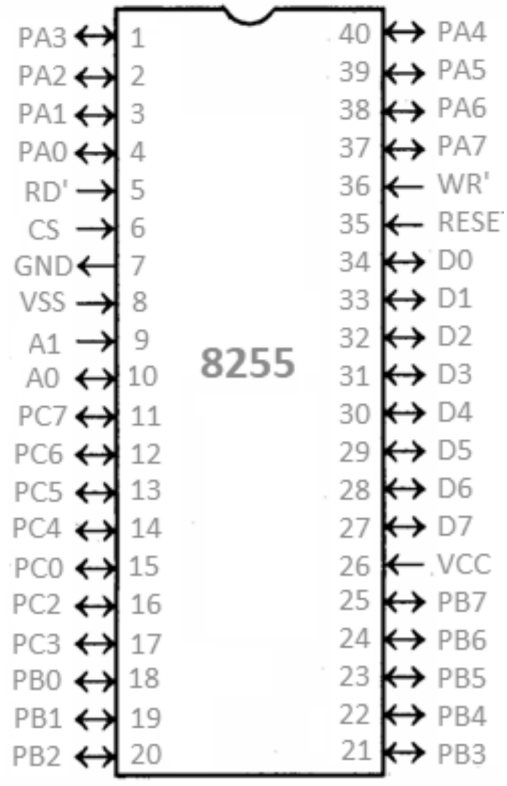
1. PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.
2. We can program it according to the given condition. It can be used with almost any microprocessor.
3. It consists of three 8-bit bidirectional I/O ports i.e. PORT A, PORT B and PORT C. We can assign different ports as input or output functions.



4. It consists of 40 pins and operates in +5V regulated power supply. Port C is further divided into two 4-bit ports i.e. port C lower and port C upper and port C can work in either BSR (bit set rest) mode or in mode 0 of input-output mode of 8255.
5. Port B can work in either mode 0 or in mode 1 of input-output mode. Port A can work either in mode 0, mode 1 or mode 2 of input-output mode. It has two control groups, control group A and control group B.
6. Control group A consist of port A and port C upper. Control group B consists of port C lower and port B. Depending upon the value if CS', A1 and A0 we can select different ports in different modes as input-output function or BSR. This is done by writing a suitable word in control register (control word D0-D7).

CS’	A1	A0	Selection	Address
0	0	0	PORT A	80 H
0	0	1	PORT B	81 H
0	1	0	PORT C	82 H
0	1	1	Control Register	83 H
1	X	X	No Selection	X

Pin Diagram:



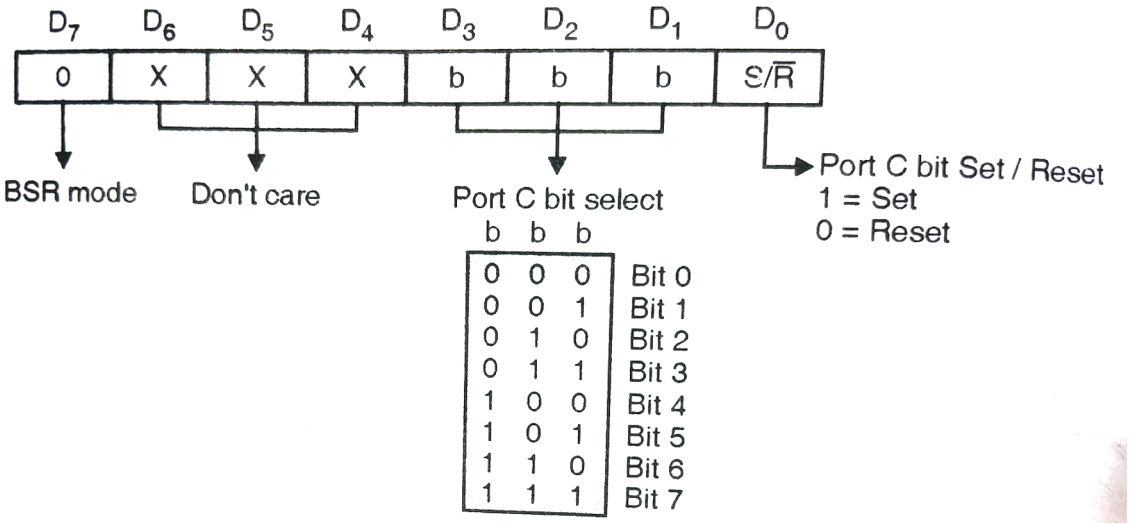
- PA0 – PA7 – Pins of port A
- PB0 – PB7 – Pins of port B
- PC0 – PC7 – Pins of port C
- D0 – D7 – Data pins for the transfer of data
- RESET – Reset input
- RD’ – Read input

- WR' – Write input
- CS' – Chip select
- A1 and A0 – Address pins

Operating modes (Important)

Bit Reset Mode(BSR):

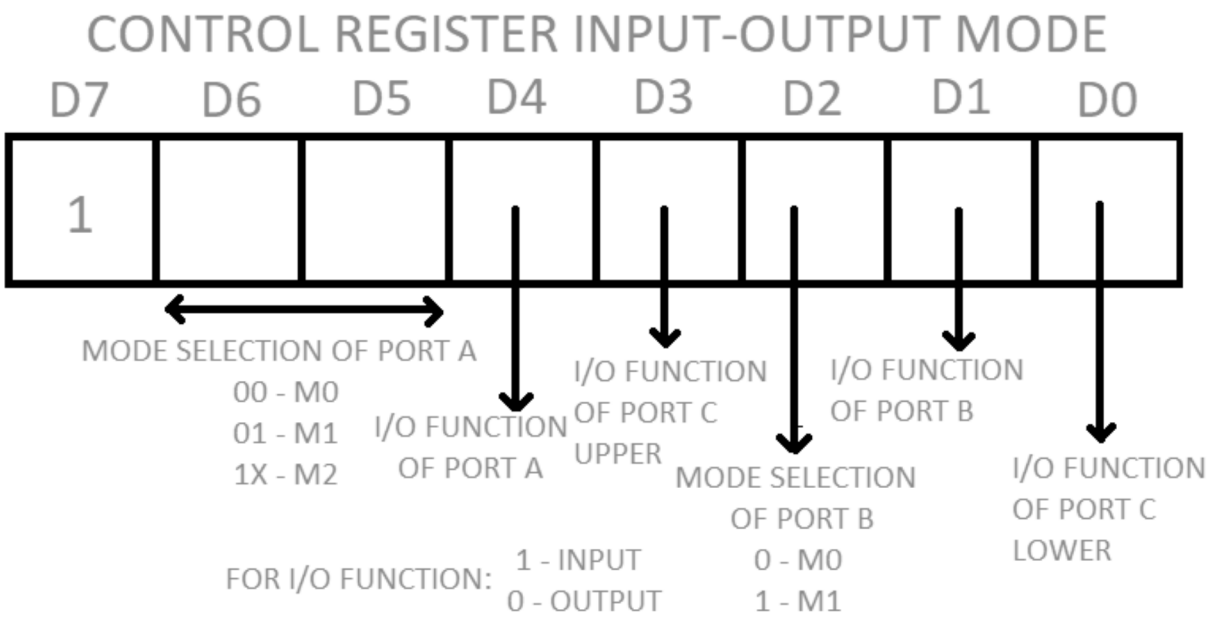
1. The BSR mode is a port C bit set/reset mode.
2. The individual bit of port C can be set or reset by writing control word in the control register.
3. The control word format of BSR mode is as shown in Figure
4. The pin of port C is selected using bit select bits [b b b] and set or reset is decided by bit S/R.
5. The BSR mode affects only one bit of port C at a time.
6. The bit set using BSR mode remains set unless and until you change the bit. So to set any bit of port C, bit pattern is loaded in control register.
7. If a BSR mode is selected it will not affect I/O mode



Control Word Register of 8255

Input-Output mode

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



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

Advantages:

1. Versatility: The PPI 8255 can be programmed to operate in a variety of modes, which makes it a versatile component in many different systems. It provides three 8-bit ports that can be configured as input or output ports, and supports multiple modes of operation for each port.
2. Ease of use: The PPI 8255 is relatively easy to use and program, even for novice programmers. The control register of the PPI can be programmed using simple commands, which makes it easy to interface with other devices.

3. Compatibility: The PPI 8255 is widely used and has been around for many years, which means that it is compatible with a wide range of devices and software.
4. Low cost: The PPI 8255 is a relatively low-cost component, which makes it an affordable option for many different applications.

Disadvantages:

1. Limited functionality: While the PPI 8255 is versatile, it has limited functionality compared to newer I/O interface components. It is not capable of high-speed data transfer and has limited memory capacity.
2. Limited number of ports: The PPI 8255 provides only three 8-bit ports, which may not be sufficient for some applications that require more I/O ports.
3. Limited resolution: The PPI 8255 provides only 8 bits of resolution for each port, which may not be sufficient for some applications that require higher resolution.
4. Obsolete technology: While the PPI 8255 is still used in some applications, it is considered an older technology and is being replaced by newer, more advanced I/O interface components.

2. 8257 - Direct Memory Access Controller

1. DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU.
2. Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory.
3. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

How DMA Operations are Performed?

Following is the sequence of operations performed by a DMA −

1. Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
2. The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
3. Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
4. Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

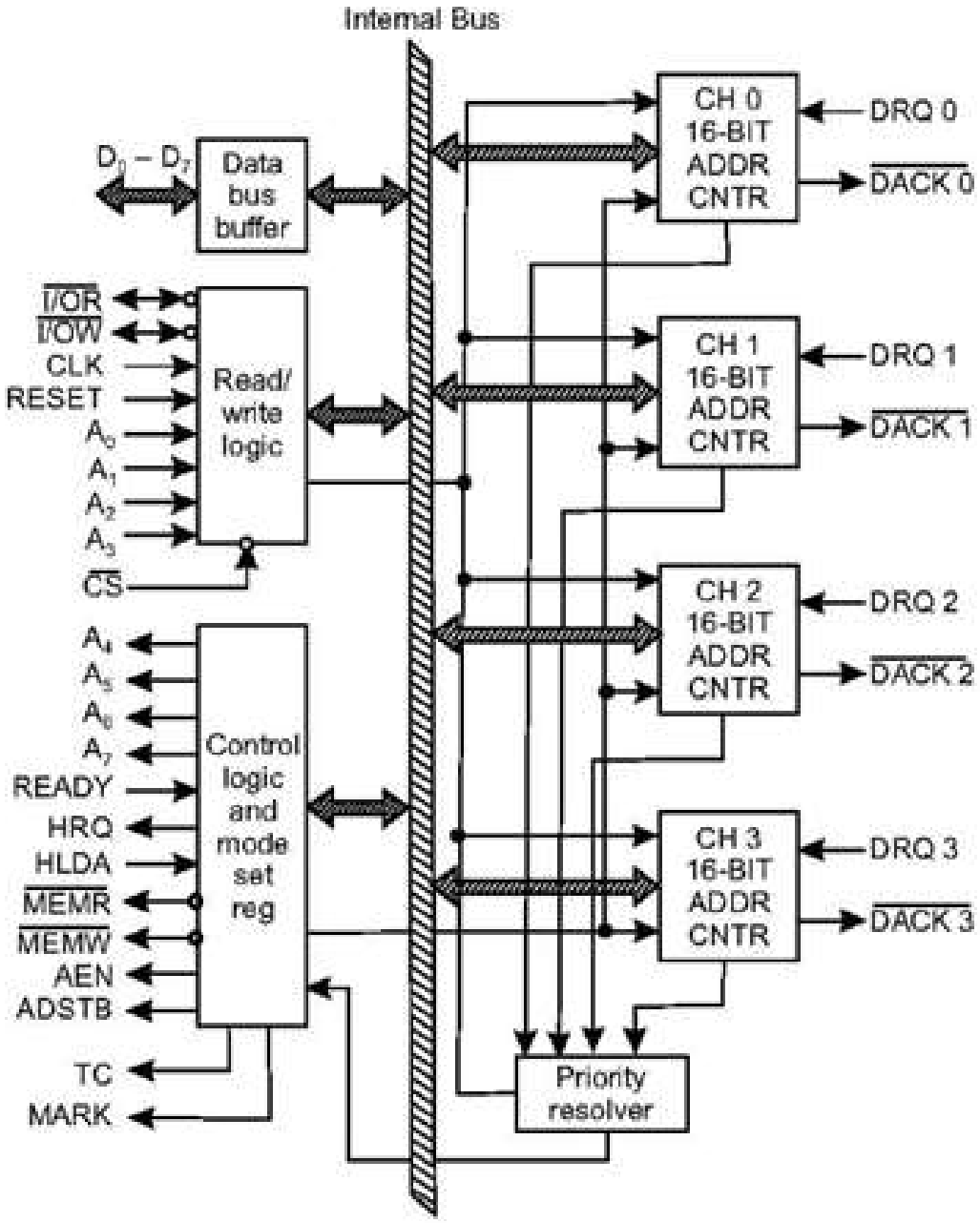
Features of 8257

1. Here is a list of some of the prominent features of 8257 −
2. It has four channels which can be used over four I/O devices.
3. Each channel has 16-bit address and 14-bit counter.
4. Each channel can transfer data up to 64kb.
5. Each channel can be programmed independently.
6. Each channel can perform read transfer, write transfer and verify transfer operations.

7. It generates MARK signal to the peripheral device that 128 bytes have been transferred.
8. It requires a single phase clock.
9. Its frequency ranges from 250Hz to 3MHz.
10. It operates in 2 modes, i.e., Master mode and Slave mode.

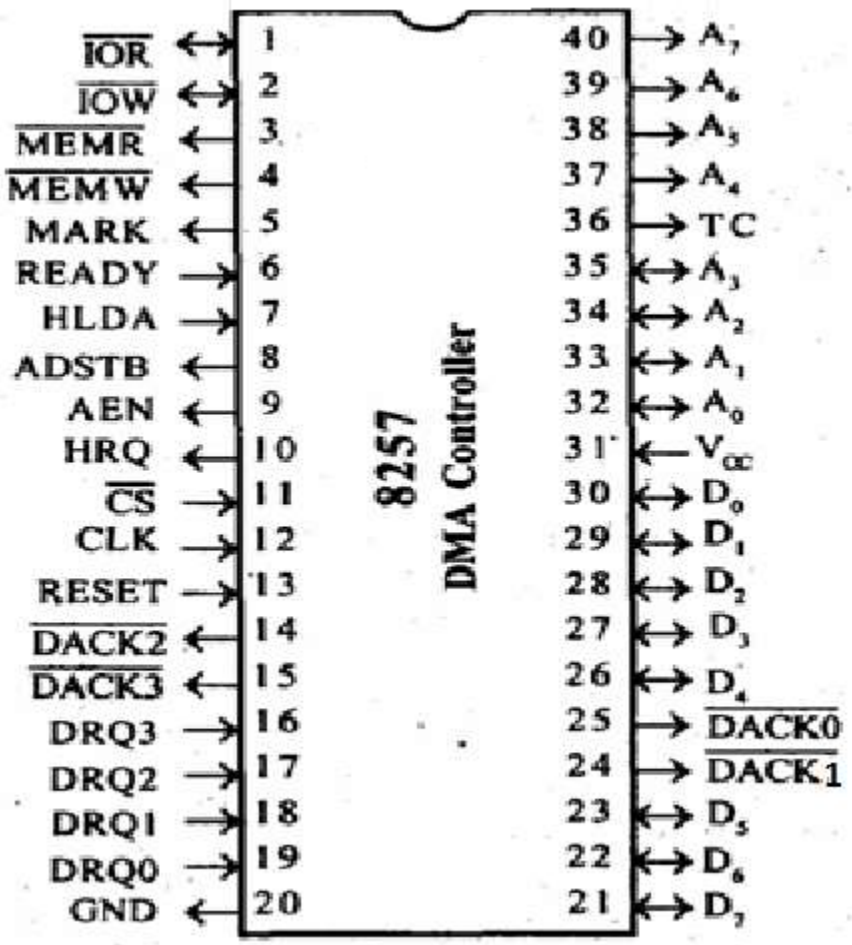
8257 Architecture

The following image shows the architecture of 8257 −



8257 Pin Description

The following image shows the pin diagram of a 8257 DMA controller −



DRQ0−DRQ3

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ0 has the highest priority and DRQ3 has the lowest priority among them.

DACKo − DACK3

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

Do − D7

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is used to load the data to the peripheral devices during DMA memory read cycle.

CLK

It is a clock frequency signal which is required for the internal operation of 8257.

RESET

This signal is used to RESET the DMA controller by disabling all the DMA channels.

Ao - A3

These are the four least significant address lines. In the slave mode, they act as an input, which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A4 - A7

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN

This signal is used to disable the address bus/data bus.

TC

It stands for ‘Terminal Count’, which indicates the present DMA cycle to the present peripheral devices.

MARK

The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

Vcc

It is the power signal which is required for the operation of the circuit.

Advantages

1. DMA accelerates memory operations by avoiding the participation of the CPU.
2. The workload on the CPU is reduced.
3. Only a few clock cycles are required for each transmission.

Disadvantages

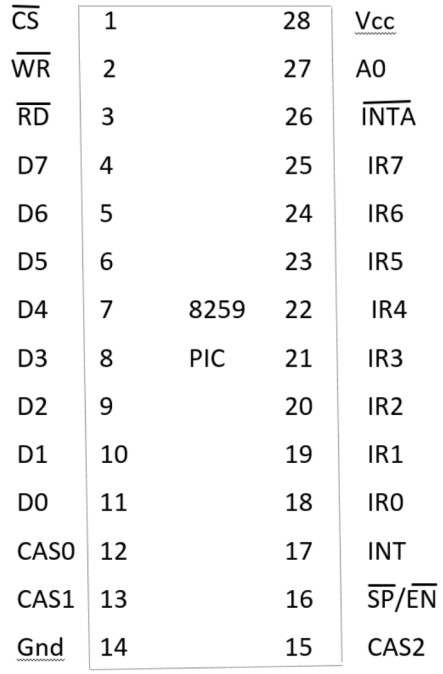
1. When DMA is utilized for data transport, a cache coherence problem might occur.
2. Increases the system's cost.

3. 8259 - Programmable Interrupt Controller

1. Intel 8259 is a Programmable Interrupt Controller (PIC). There are 5 hardware interrupts and 2 hardware interrupts in Intel 8085 and Intel 8086 microprocessors respectively.
2. But by connecting Intel 8259 with these microprocessors, we can increase their interrupt handling capability. Intel 8259 combines the multi-interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.
3. For example, Interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.

Features of Intel 8259 PIC are as follows:

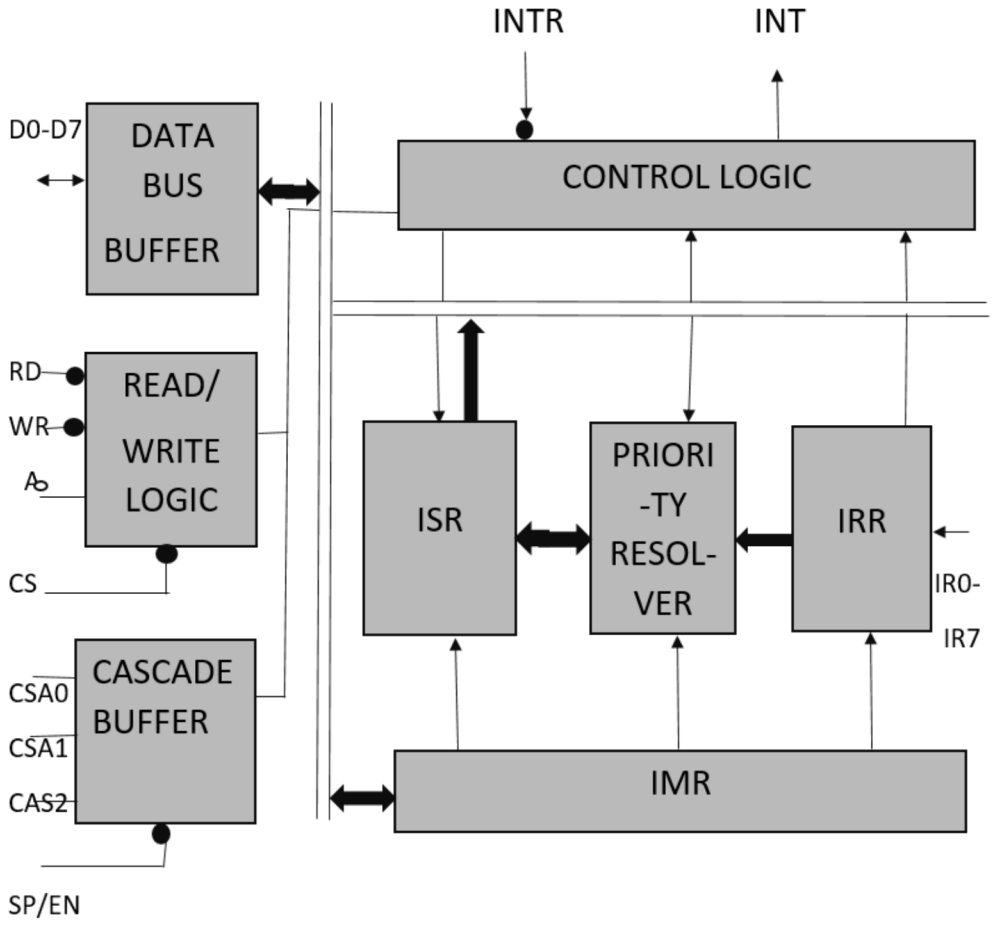
1. Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
2. It can be programmed either in level triggered or in edge triggered interrupt level.
3. We can mask individual bits of interrupt request register.
4. We can increase interrupt handling capability upto 64 interrupt level by cascading further 8259 PICs.
5. Clock cycle is not required.



Pin Diagram of 8259

We can see through above diagram that there are total 28 pins in Intel 8259 PIC where Vcc : 5V Power supply and Gnd : ground. Other pins use are explained below.

Block Diagram of 8259 PIC microprocessor



1. The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.
2. Data bus buffer – This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. After selection of Interrupt by 8259 microprocessor (based on priority of the interrupt), it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.
3. Read/Write logic – This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

4. Control logic – It is the center of the PIC and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.
5. Interrupt request register (IRR) – It stores all the interrupt level which are requesting for Interrupt services.
6. Interrupt service register (ISR) – It stores the interrupt level which are currently being executed.
7. Interrupt mask register (IMR) – It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.
8. Priority resolver – It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.
9. Cascade buffer – To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.
10. SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus.

Advantages:

1. Interrupt Management: The 8259 PIC is designed to handle interrupts efficiently and effectively, allowing for faster and more reliable processing of interrupts in a system.
2. Flexibility: The 8259 PIC is programmable, meaning that it can be customized to suit the specific needs of a given system, including the number and type of interrupts that need to be managed.
3. Compatibility: The 8259 PIC is compatible with a wide range of microprocessors, making it a popular choice for managing interrupts in many different systems.
4. Multiple Interrupt Inputs: The 8259 PIC can manage up to 8 interrupt inputs, allowing for the management of complex systems with multiple devices.
5. Ease of Use: The 8259 PIC includes simple interface pins and registers, making it relatively easy to use and program.

Disadvantages:

1. Cost: While the 8259 PIC is relatively affordable, it does add cost to a system, particularly if multiple PICs are required.
2. Limited Number of Interrupts: The 8259 PIC can manage up to 8 interrupt inputs, which may be insufficient for some applications.
3. Complex Programming: Although the interface pins and registers of the 8259 PIC are relatively simple, programming the 8259 can be complex, requiring careful attention to interrupt prioritization and other parameters.
4. Limited Functionality: While the 8259 PIC is a useful peripheral for interrupt management, it does not include more advanced features, such as DMA (direct memory access) or advanced error correction.

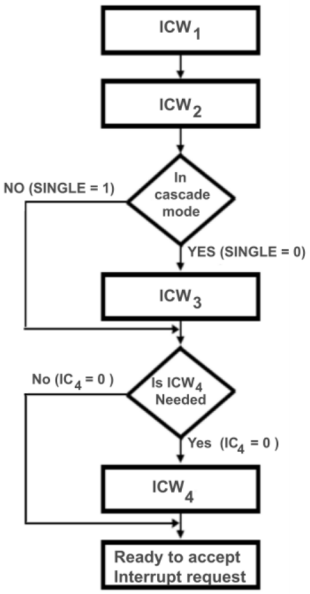
Command word of 8259 is divided into two parts :

1. Initialization command words(ICW)
2. Operating command words(OCW)

Initialization command words(ICW) :

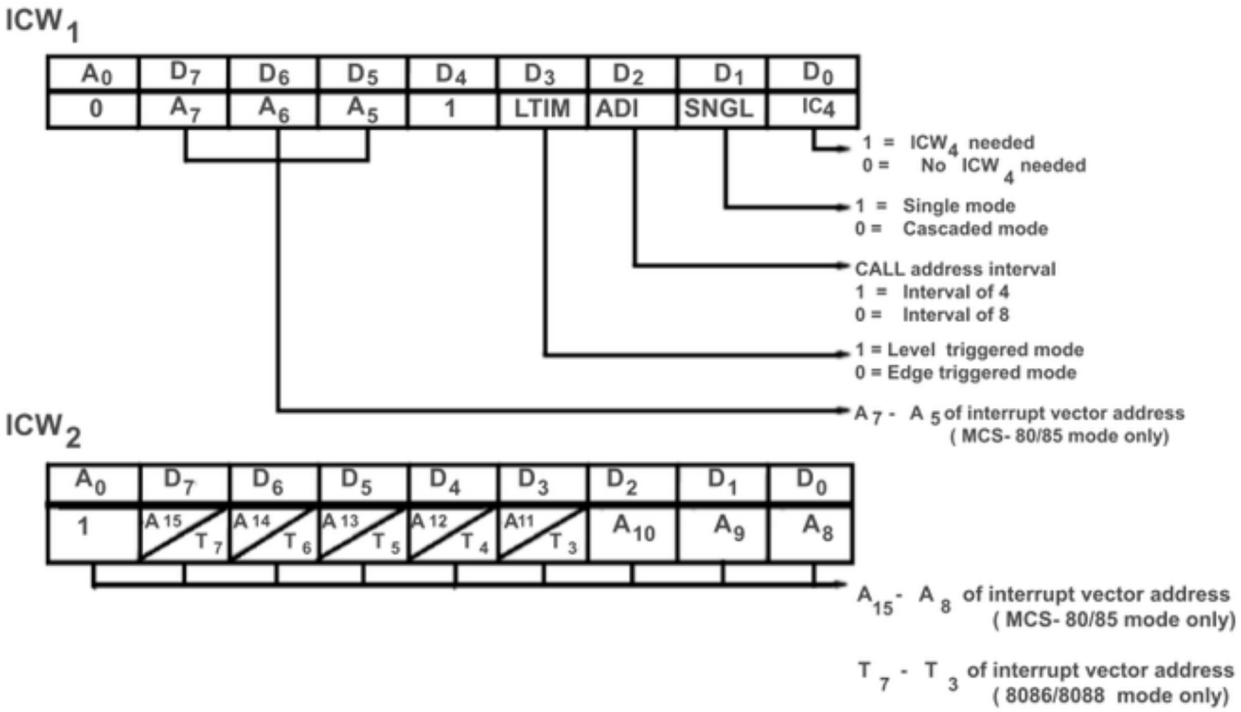
1. ICW is given during the initialization of 8259 i.e. before its start functioning.
2. ICW1 and ICW2 commands are compulsory for initialization.
3. ICW3 command is given during a cascaded configuration.
4. If ICW4 is needed, then it is specified in ICW1.
5. The sequence order of giving ICW commands is fixed i.e. ICW1 is given first and then ICW2 and then ICW3.
6. Any of the ICW commands can not be repeated, but the entire initialization process can be repeated if required.

Initialization sequence of 8259 :



ICW1 command :

1. The control word is recognized as ICW1 when A0 = 0 and D4 = 1.
2. It has the control bits for Edge and level triggering mode, single/cascaded mode, call address interval and whether ICW4 is required or not.
3. Address lines A7 to A5 are used for interrupt vector addresses.
4. When the ICW1 is loaded, then the initializations performed are:
    a. The edge sense circuit is reset because, by default, 8259 interrupt is edge triggered.
    b. The interrupt mask register is cleared.
    c. IR7 is assigned to priority 7.
    d. Slave mode address is assigned as 7.
    e. When D0 = 0, this means IC4 command is not required. Therefore, functions used in IC4 are reset.
    f. Special mask mode is reset and status read is assigned to IRR.

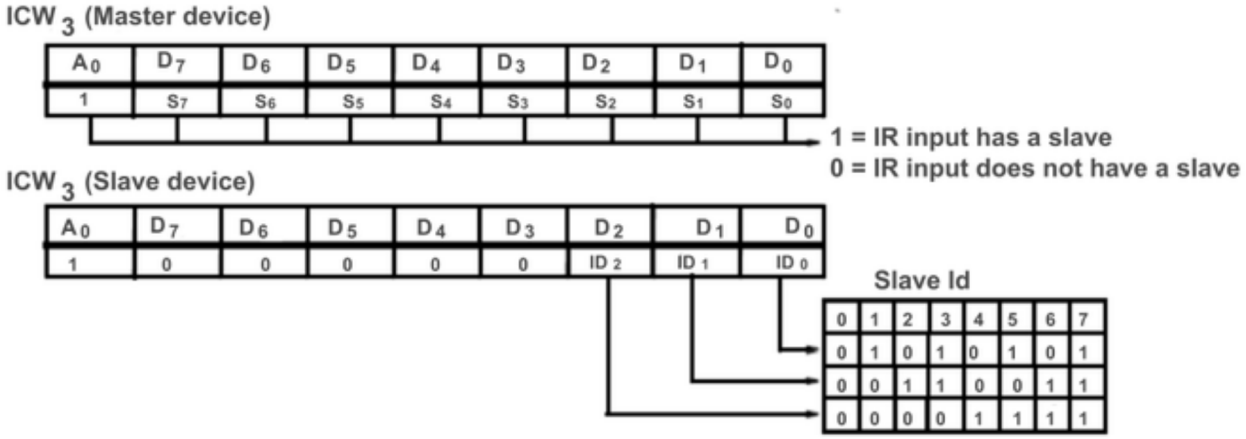


ICW2 command :

1. The control word is recognized as ICW2 when A0= 1.
2. It stores the information regarding the interrupt vector address.
3. In the 8085 based system, the A15 to A8 bits of control word is used for interrupt vector addresses.
4. In the 8086 based system, T6 to T3 bits are inserted instead of A15 to A8 and A10 to A8 are used for selecting interrupt level, i.e. 000 for IR0 and 111 for IR7.
5. Initialization of 8259 by ICW1 and ICW2 command words

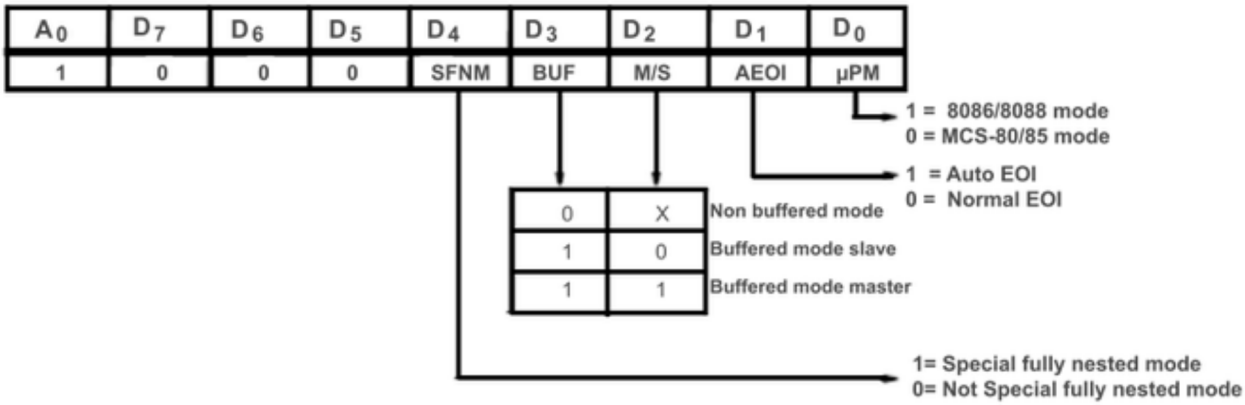
ICW3 :

1. ICW3 command word is used when there is more than one 8259 present in the system i.e. when SNGL bit in ICW1 is 0, then it will load 8-bit slave register.



ICW4 :

1. When AEOI = 1, then Automatic end of interrupt mode is selected.
2. When SFMN = 1, then a special fully nested mode is selected.
3. 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.
4. when µPM = 1, then 8086 operations are performed, otherwise 8085 operations are performed.



Operating command words(OCW) :

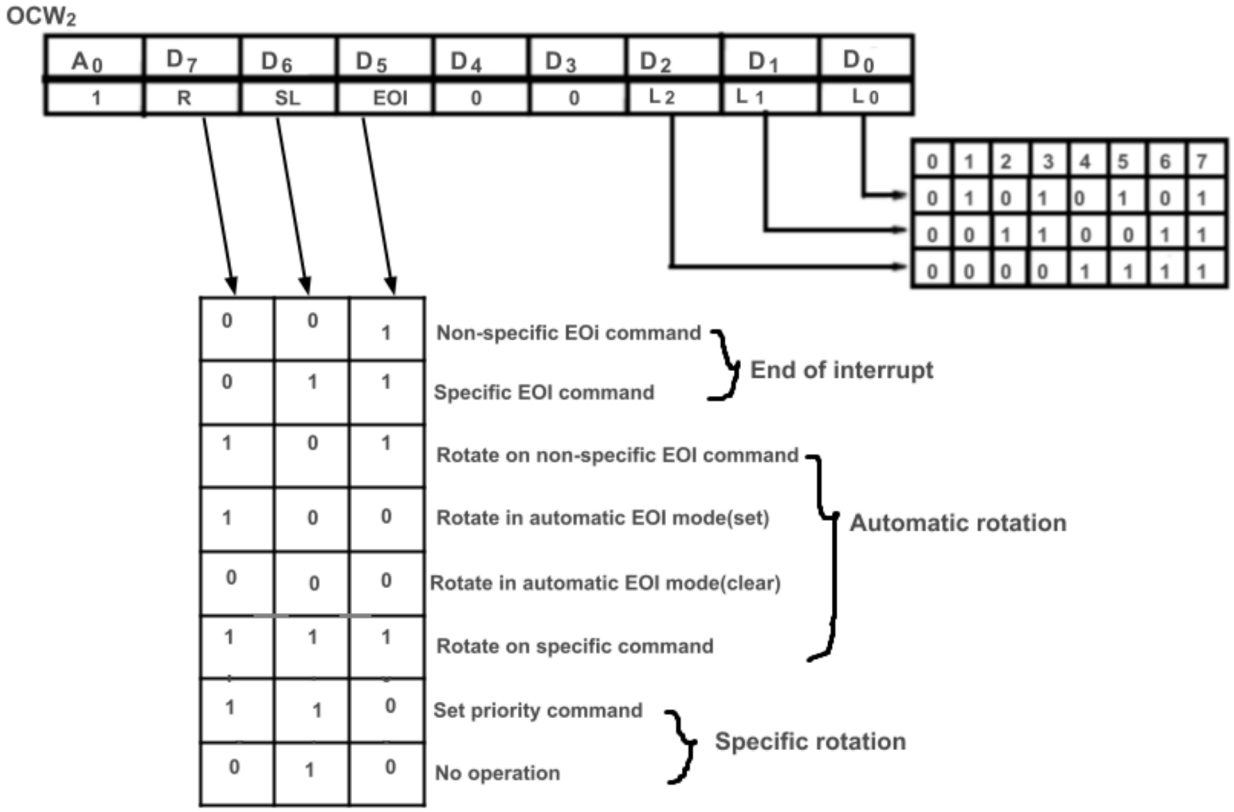
1. OCW is given during the operation of 8259 i.e. microprocessor starts using 8259.
2. OCW commands are not compulsory for 8259.
3. The sequence order of giving OCW commands is not fixed.
4. The OCW commands can be repeated.

OCW1 –

1. It is used to set and reset the mask bits in IMR(interrupt mask register). M7 – M0 describes 8 mask bits

OCW2 –

1. It is used for selecting the mode of operation of 8259. Here L2 to L0 are used to describe interrupt level on which action need to be performed.
2. Detailed operations are described in the diagram below.



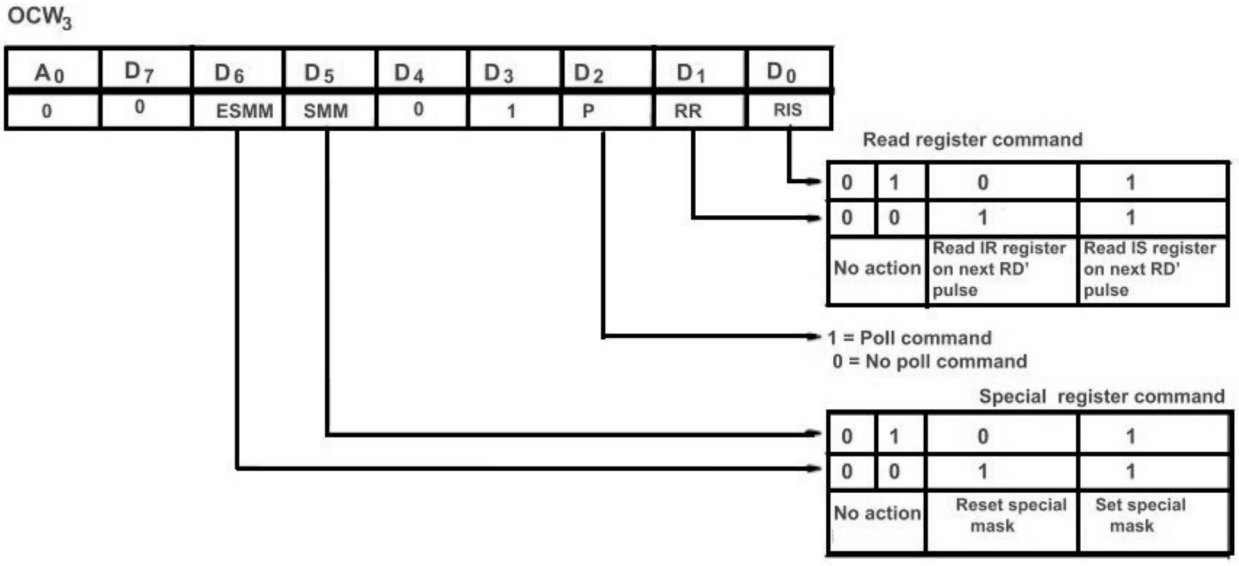
OCW3 –

When the ESMM (Enable special mask mode) bit is set, then the SMM bit is don't care. If SMM = 1 and ESMM = 1, then 8259 will enter in Special mask mode.

If ESMM = 1 and SMM = 0, then 8259 will return into normal mask mode.

RR and RIS are used to give the read register command.

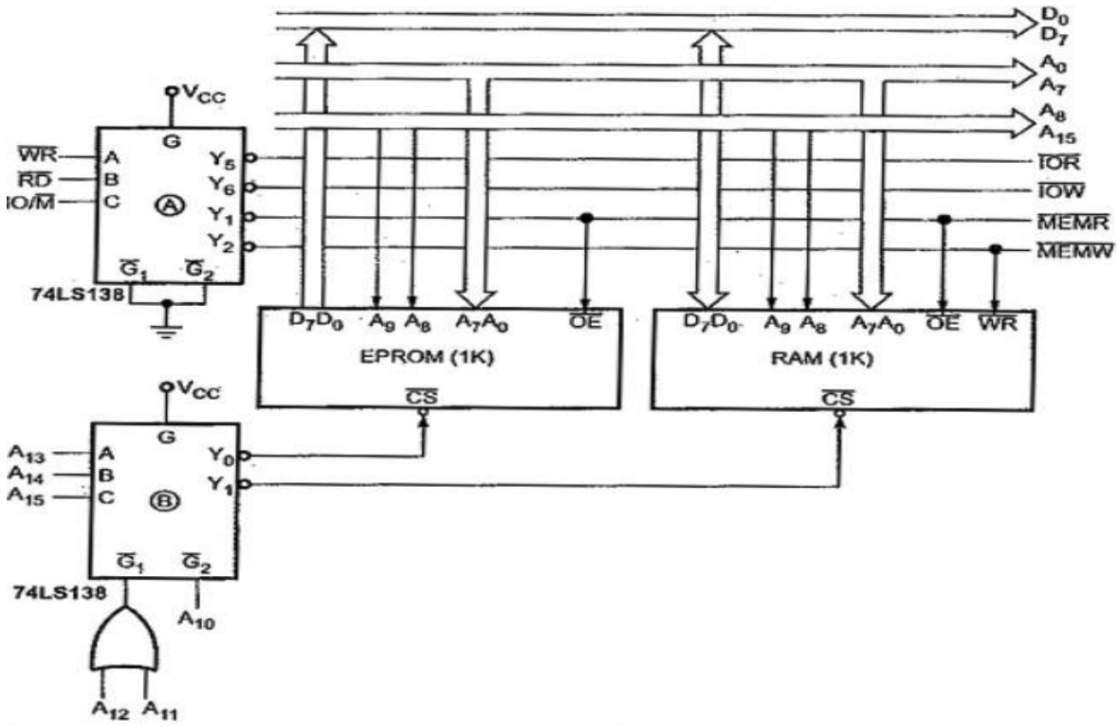
P = 1 is used for poll command.



4. Partial & Absolute decoding

Absolute decoding:

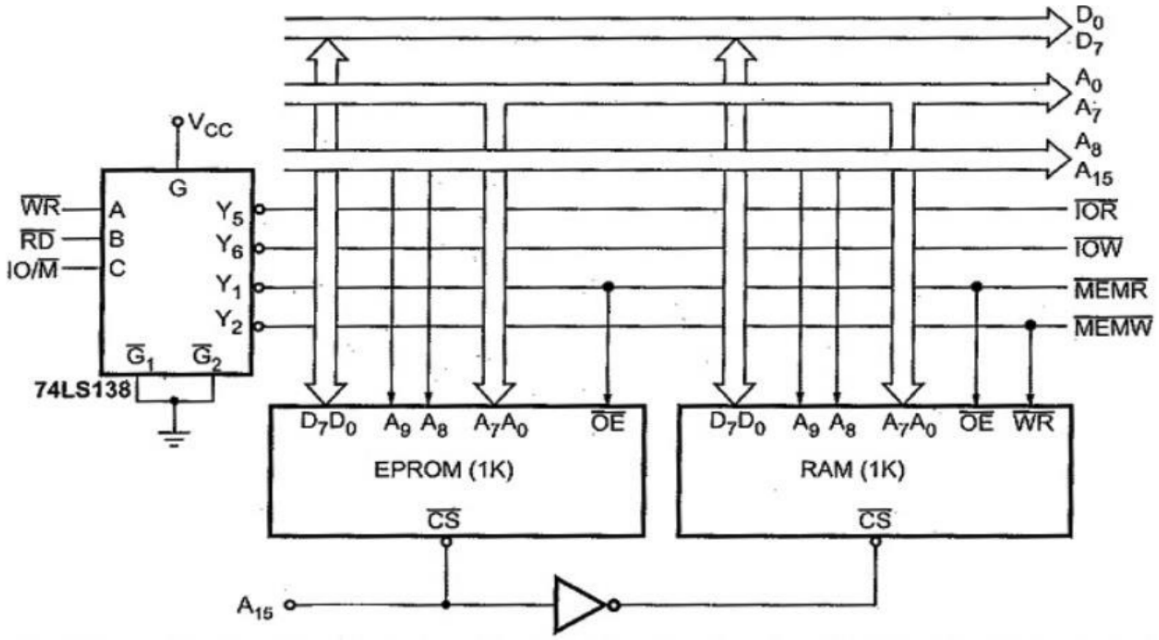
- In absolute decoding technique, all the higher address lines are decoded to select the memory chip, and the memory chip is selected only for the specified logic levels on these high-order address lines; no other logic levels can select the chip.



- Figure shows the Memory Interfacing in 8085 with absolute decoding.
- This addressing technique is normally used in large memory systems.

Partial decoding:

- In small systems, hardware for the decoding logic can be eliminated by using individual high-order address lines to select memory chips.
- This is referred to as linear decoding. Figure shows the addressing of RAM with linear decoding technique. This technique is also called partial decoding.
- It reduces the cost of decoding circuit, but it has a drawback of multiple addresses (shadow addresses).
- Figure shows the addressing of RAM with linear decoding technique.
- A15 address line, is directly connected to the chip select signal of EPROM and after inversion it is connected to the chip select signal of the RAM.
- Therefore, when the status of A15 line is ‘zero’, EPROM gets selected and when the status of A15 line is ‘one’ RAM gets selected.
- The status of the other address lines is not considered, since those address lines are not used for generation of chip select signals.

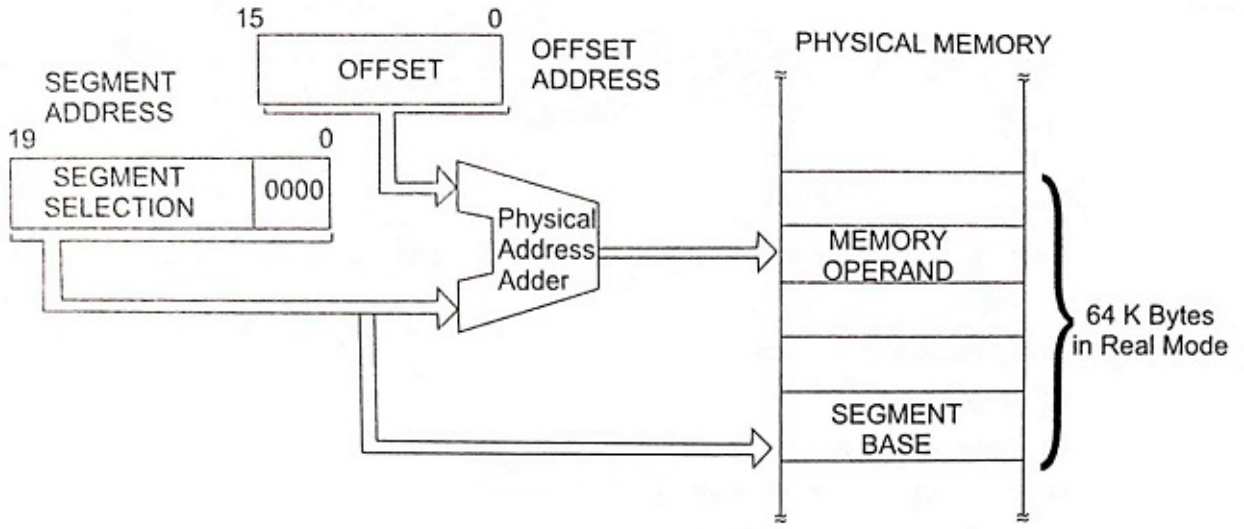

D0
D7
A0
A7
A8
A15
IOR
IOW
MEMR
MEMW
VCC
G
WR
RD
IO/M
A
B
C
Y5
Y6
Y1
Y2
G1
G2
74LS138
D7D0
A9 A8
A7A0
OE
EPROM (1K)
CS
D7D0
A9 A8
A7A0
OE
WR
RAM (1K)
CS
A15

Module 4 - Intel 80386DX Processor

1. Real + Protected + Virtual Mode (8086) in 80386

Real Mode

- The 80386 always starts from the memory location FFFFFFF0H in the real address mode whenever the processor is reset.
- In this Operating Modes of 80386, 80386 works as 8086 processor with 32-bit registers and data types. The addressing modes, memory size, interrupt handling of 80386 are same as the real address mode of 80286.
- Initially, the 80386 starts with real mode and then prepares for protected mode operation. All the instructions of 80386 are available in this mode except protected address mode instructions. In this Operating Modes of 80386, the operand size is 16 bits by default.
- The 32-bit operands and addressing modes can he used with the help of override prefixes. In this mode, the segment size is 64 k.



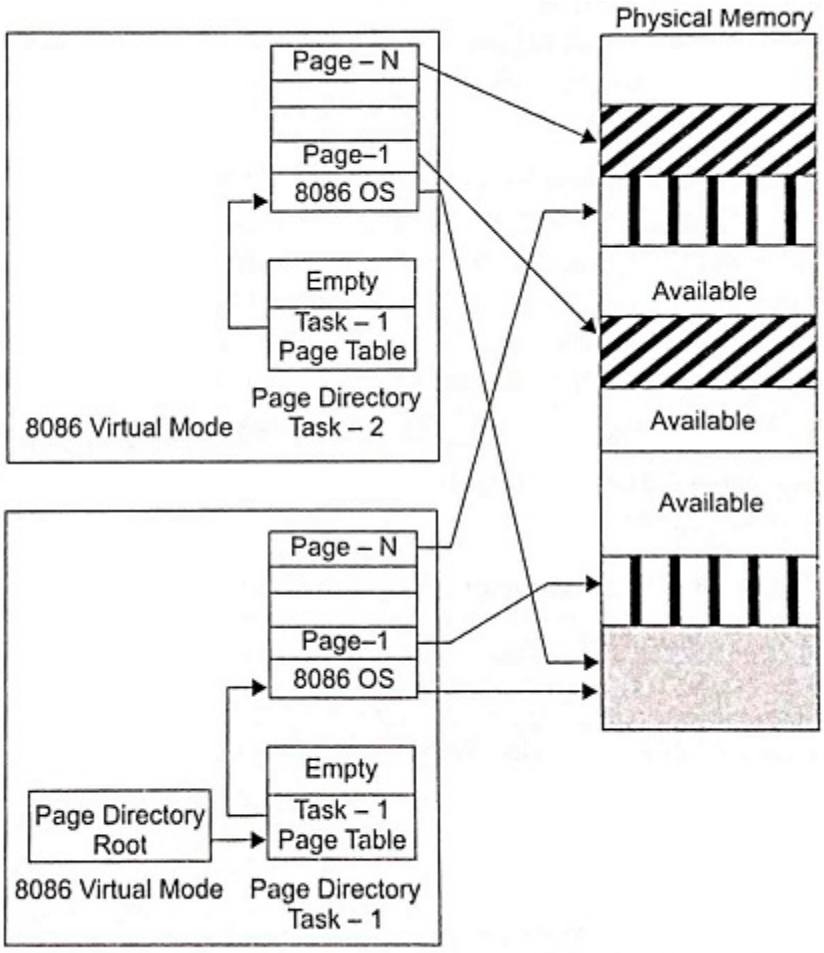
- During real addressing mode, the 80386 can address up to 1 MB of physical memory using address lines A19-A0.
- In this address mode, the paging unit is disabled so that the real addresses are the same as the physical addresses.
- To compute a physical memory address, the contents of the segment register are shifted left by four bit positions and then added to the 16-bit offset address formed using one of the addressing modes just like the 8086 real address mode.
- Figure shows the physical address computation in real mode of 80386. In real-mode operation of 80386, the segments can be read, written or executed.
- The segments in 80386 real modes can be overlapped or non-overlapped.

Protected Mode

1. A 32-bit address space is available in protected mode, a sophisticated operating mode that gives users access to up to 4GB of memory.
2. Additionally, it offers sophisticated memory management and security features including segmentation and paging.
3. Pages are fixed-size units of memory that can be moved in and out of physical memory as needed. Paging enables this.
4. Memory can be separated into logical units called segments through segmentation, which can be used to restrict access to particular memory locations.
5. The 80386 also has access to a number of privileged instructions and registers in a protected mode that are not present in regular mode.
6. The protected mode also supports virtual memory which allows the system to use more memory than the physical memory available by swapping memory pages to and from the disk.

Virtual 8086

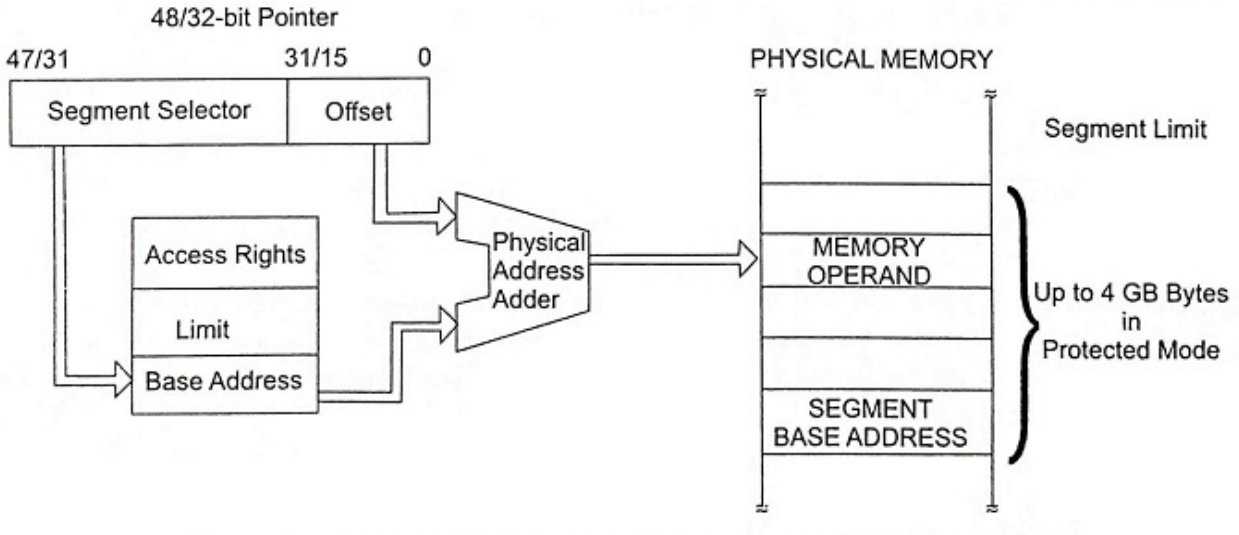
- In real mode, 80386 is able to execute the 8086 programs along with all capabilities of 80386.
- But once the 80386 processor enters into the protected mode from real mode, it cannot revert back to the real mode without a reset operation.
- During the protected mode of operation, 80386 processors confer a virtual 8086 operating environment to execute the application programs of 8086.
- Therefore, the virtual mode operation of 80386 provides an advantage of executing 8086 programs although the 80386 processor is in protected mode.
- The address computation mechanism in virtual 80386 modes is same as 8086 real mode. In this Operating Modes of 80386, 80386 can address 1 Mbytes of physical memory, which will be within the 4 Gbytes memory address of the protected mode of 80386.
- The paging mechanism and protection capabilities are also available in this mode of operation. In the virtual mode, the paging unit provides 256 pages, each of 4 Kbytes size. Each of the pages will be anywhere within the maximum 4 Gbytes physical memory.
- The 80386 can support multiprogramming; hence the multiple 8086 real-mode software applications can be executed at a time.
- Figure shows the memory management in virtual 386 modes in a multitasking virtual 8086 environment.



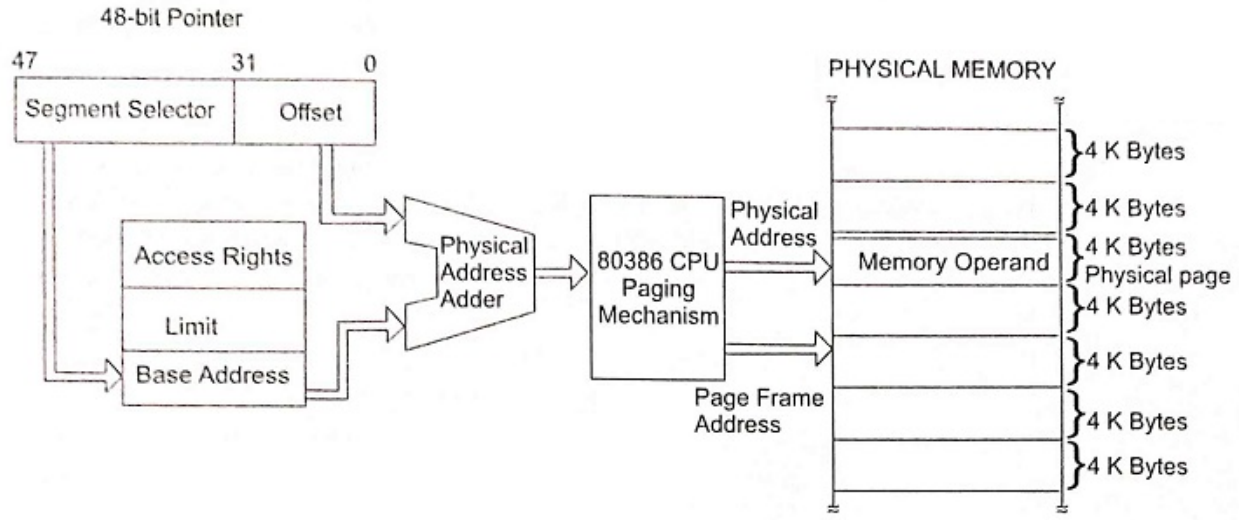
2. Explain memory management in protected mode of 80386

- In protected mode, the 80386 can able to address 4 gigabytes of physical memory and 64 terrabytes of virtual memory.
- In this Operating Modes of 80386, the 80386 has capability to support all programs written for 80286 and 8086 and to be executed.
- The controls of memory management and protection abilities of 80386 are possible in this Operating Modes of 80386.
- All additional instructions and addressing modes of 80386 feasible in protected mode.
- In protected mode addressing, the contents of segment registers are used as selectors which can address the segment descriptors.
- The segment descriptors consist of the segment limit, base address and access rights byte of the segment.
- The effective or offset address is added with segment base address to determine linear address.
- When the paging unit is disabled, the linear address is used as physical address. If the paging unit becomes enabled, the paging unit converts the linear address into physical address.

- Figures show the protected mode addressing without paging and with paging unit respectively. In general, the paging unit is a memory management unit which is enabled only in the protected mode.
- The paging mechanism is able to handle memory segments in terms of pages of 4 KB size. Usually, a paging unit operates under the control of segmentation unit.



Protected mode addressing of 80386 without paging unit



Protected mode addressing of 80386 with paging unit

- The 80386 starts with real mode and then changes the operation from real mode to the protected mode operation. To change the operation from real mode to the protected mode, the following steps must be followed:
    - **Step 1** Initialize the IDT so that it contains valid interrupt gates for at least the first 32 interrupt type numbers. Usually, IDT contains up to 256 8-byte interrupt gates to define all 256 interrupt type.

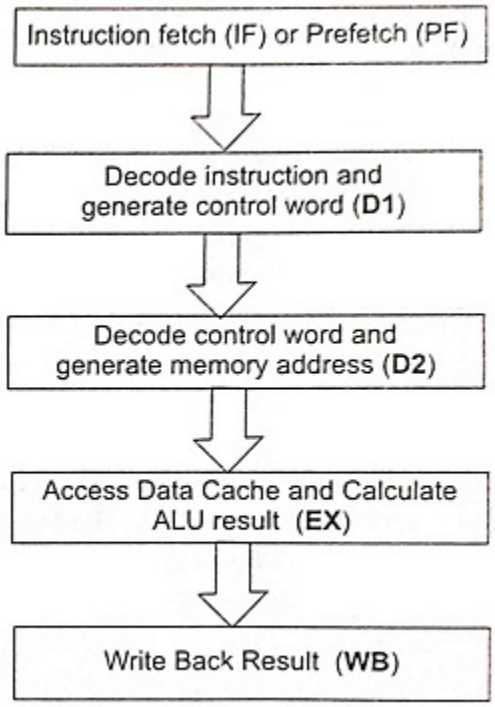
- **Step 2** Initialize the GDT so that it contains a null descriptor at Descriptor 0. The valid descriptors are used for at least one code, one stack and one data segment.
- **Step 3** Switch to protected mode after setting the PE bit in CR0.
- **Step 4** Perform an intrasegment near JMP operation to flush the internal instruction queue and load the TR with the base TSS descriptor.
- **Step 5** After that, load all the segment registers with their initial selector values.
- **Step 6** 80386 operates in the protected mode using segment descriptors that are defined in GDT and IDT.

Module 5 - Pentium Processor

1. Integer & Floating Point Pipeline Stages

Integer Pipeline Stages

- The Pentium is a superscalar processor and it has two integer pipelines, called U and V. The process of issuing two instructions in parallel is known as pairing.
- The U-pipeline is able to handle the full instruction set of the Pentium but the V-pipeline has limited handling capability.
- The V-pipeline is able to handle only simple instructions without any microcode support. The V-pipeline is used to execute ‘simple integer instructions’ such as load/store type instructions and the FPU instruction FXCH, but the U-pipeline executes any legitimate Pentium instructions.
- Actually, Architecture of Pentium Processor use a set of pairing rules to select a simple instruction which can go through the V pipeline.
- When instructions are paired, initially the instruction is issued to the U-pipe and then the next sequential instruction is issued to the V-pipe.



- Each integer unit has the basic five-stage pipeline as given below:
    - Prefetch (PF)
    - Decode-1 (D1)
    - Decode-2 (D2)
    - Execute (E)
    - Write Back (WB)

Prefetch (PF)
In the prefetch stage of integer pipeline of the Pentium processor, instructions are fetched from the instruction cache as instructions are stored initially in the instruction cache.

Decode-1 (D1)
In the decode-1 (D1) pipeline stage, the CPU decodes the instruction and generates a control word. The D1 pipeline stage has two parallel instruction decoders. These implement the pairing rules.

Decode-2 (D2)
The decode-2(D2) pipeline stage is required whenever the control word from D1 stage is decoded to complete the instruction decoding. In this stage, the CPU generates addresses for data memory.

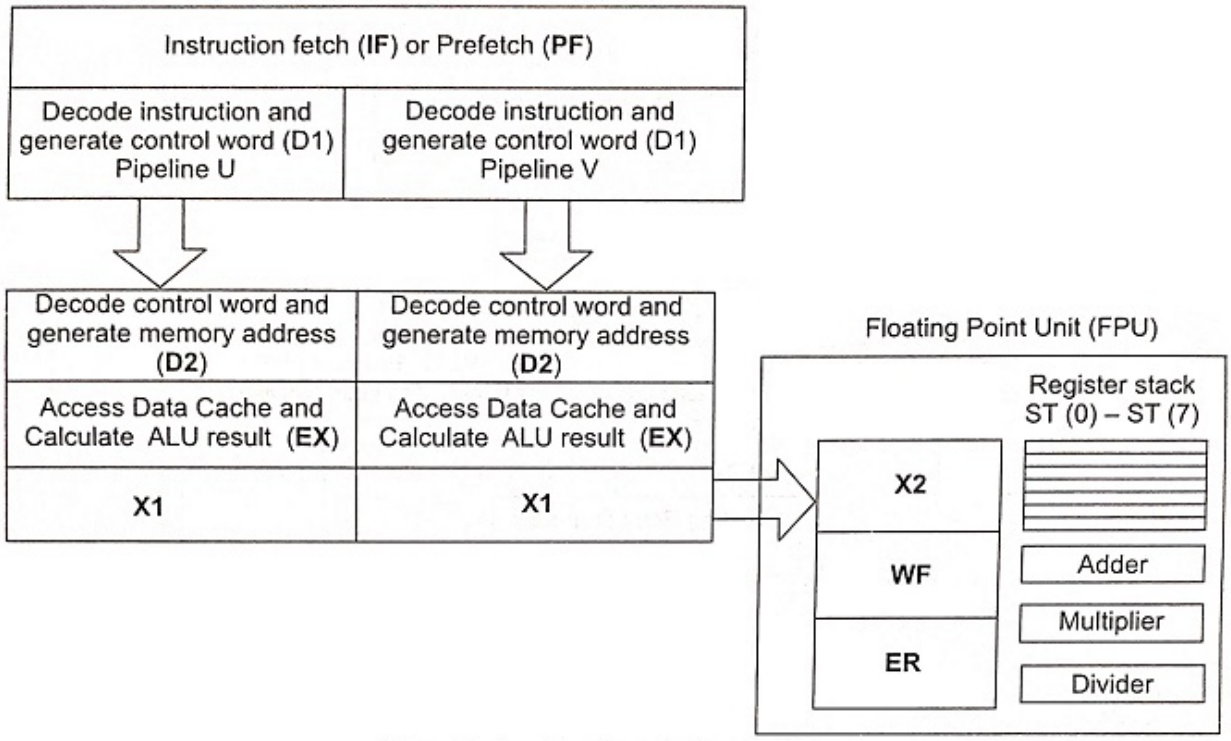
Execute (E)
The execution stage is used for both ALU operations and data cache access. The data cache is used for data operands and ALU performs arithmetic logic computations or floating-point operations.

Write Back (WB)
The final stage of the five-stage pipeline is Write Back (WB). In the WB stage, the CPU updates the contents of registers and status of the flag register after completion of execution.

- The Pentium pipeline structure is similar to 80486 pipeline structure.
- Usually, the 80486 takes two clock cycles to decode instructions, but the Pentium processor takes only one clock cycle as Pentium processor has an additional integrating hardware in each pipeline stages to speed up the process.

Floating Point Pipeline Stages

- The Architecture of Pentium Processor has been designed for incorporating on the chip numeric data processor.
- The Floating-Point Unit (FPU) of Pentium has an eight-stage pipeline as shown in figure.
- The eight pipeline stages are
    - Prefetch (PF)
    - Decode-1 (D1)
    - Decode-2 (D2)
    - Execute (dispatch)
    - Floating Point Execute-1 (X1)
    - Floating Point Execute-2 (X2)
    - Write Float (WF)
    - Error Reporting (ER)



Prefetch (PF)
The prefetch stage is same as the integer pipeline of Pentium processor.

Decode-1 (D1)
The decode-1 (D1) pipeline stage is also same as the integer pipeline of Pentium processor.

Decode-2 (D2)
The decode-2 (D2) pipeline stage is worked as required whenever the control word from D1 stage is decoded to complete the instruction decoding. In this stage, it is the integer pipeline of Pentium processor.

Operand Fetch
During the execution stage (E), the floating-point unit accesses the data cache and the floating-point register to fetch operands. Before writing the floating-point data to the data cache, the floating-point unit converts internal data format into appropriate memory representation format.

Floating Point Execute-1 (X1)
In the Floating Point Execute-1 (X1) stage, the floating-point unit executes the first steps of the floating-point calculations. While reading the floating-point data from the data cache, the floating-point unit writes the data into the floating-point register.

Floating Point Execute-2 (X2)
During the Floating Point Execute-2 (X2) stage, the Floating Point unit execute the remaining steps of the floating-point computations.

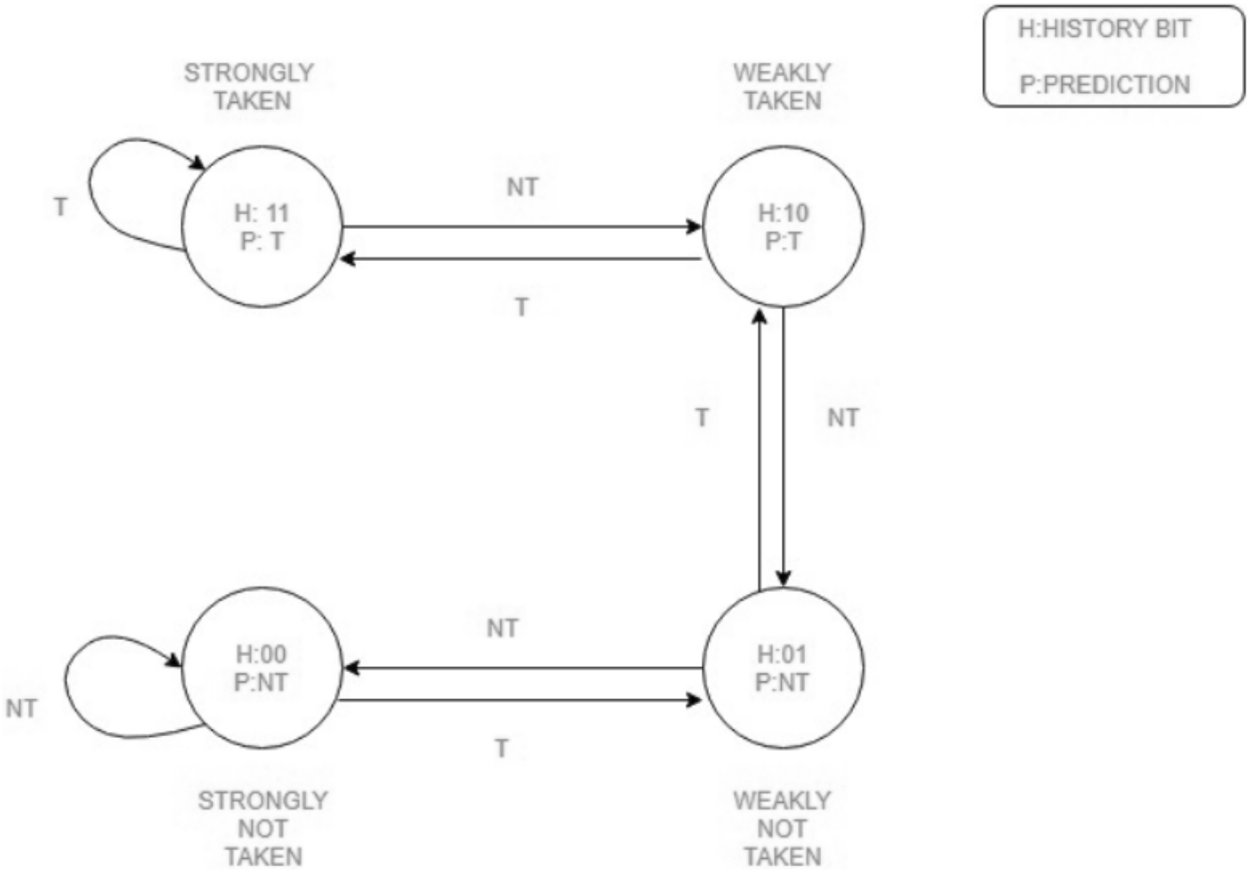
Write Float (WF)
In the Write Float (WF) stage, the floating-point unit completes the execution of the floating-point calculations and then writes the computed result into the floating-point register file.

Error Reporting (ER)
In the error reporting(ER) stage, the floating-point unit generates a report about the internal special situations and updates the floating point status.

2. Branch Prediction Logic

- The gain produced by Pipelining can be reduced by the presence of program transfer instructions eg JMP, CALL, RET etc
- They change the sequence causing all the instructions that entered the pipeline after program transfer instructions invalid
- Thus no work is done as the pipeline stages are reloaded.
- To avoid this problem, Pentium uses a scheme called Dynamic Branch Prediction. In this scheme, a prediction is made for the branch instruction currently in the pipeline.
- The prediction will either be taken or not taken. If the prediction is true then the pipeline will not be flushed and no clock cycles will be lost.
- If the prediction is false then the pipeline is flushed and starts over with the current instruction. It is implemented using 4 way set associated cache with 256 entries.
- This is called Branch Target Buffer (BTB).
- The directory entry for each line consists of:
    - Valid bit: Indicates whether the entry is valid or not.
    - History bit: Track how often bit has been taken.
- Source memory address is from where the branch instruction was fetched. If the directory entry is valid then the target address of the branch is stored in corresponding data entry in BTB.

Working:
- BTB is a lookaside cache that sits to the side of Decode Instruction(DI) stage of 2 pipelines and monitors for branch instructions.
- The first time that a branch instruction enters the pipeline, the BTB uses its source memory to perform a lookup in the cache.
- Since the instruction was never seen before, it is BTB miss. It predicts that the branch will not be taken even though it is unconditional jump instruction.
- When the instruction reaches the EU(execution unit), the branch will either be taken or not taken. If taken, the next instruction to be executed will be fetched from the branch target address. If not taken, there will be a sequential fetch of instructions.
- When a branch is taken for the first time, the execution unit provides feedback to the branch prediction. The branch target address is sent back which is recorded in BTB.
- A directory entry is made containing the source memory address and history bit is set as strongly taken.



- The diagram is explained by the following table:

History Bits	**Resulting Description**	**Prediction made**	**If branch taken**	**If branch not taken**
11	Strongly Taken	Branch Taken	Remains in same state	Downgraded to weakly taken
10	Weakly Taken	Branch Taken	Upgraded to strongly taken	Downgraded to weakly not taken
01	Weakly Not Taken	Branch Not Taken	Upgraded to weakly taken	Downgraded to strongly not taken
00	Strongly Not Taken	Branch Not Taken	Upgraded to weakly not taken	Remains in same state

Advantages:

1. Improved performance
2. Increased instruction throughput
3. Reduced branch misprediction penalty
4. More efficient use of processor resources
5. Better handling of large code bases
6. More accurate predictions over time
7. Improved pipelining

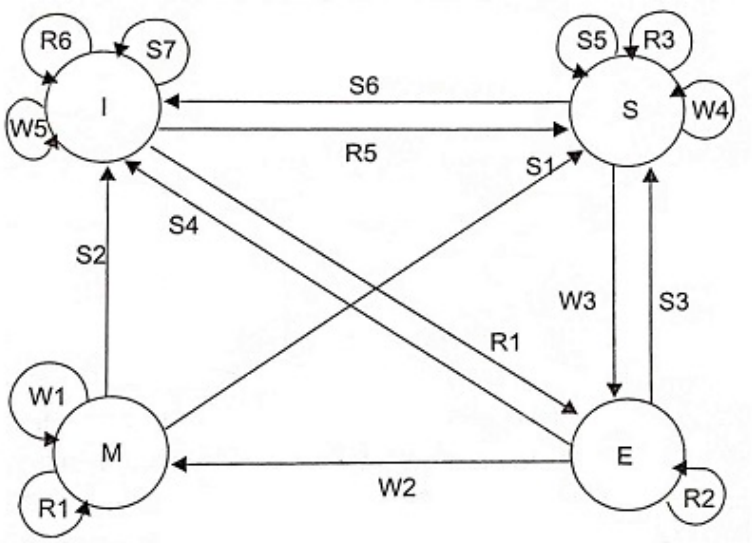
Disadvantages:

1. Increased complexity
2. Increased power consumption
3. Limited accuracy
4. Increased memory usage
5. Difficulty predicting indirect branches
6. Increased vulnerability to side-channel attacks
7. Increased software complexity

3. MESI Protocol

MESI (Modified, Exclusive, Shared, Invalid) Protocol

1. The abbreviation of MESI is Modified, Exclusive, Shared, and Invalid, which are the four possible states of a cache line.
2. The MESI protocol is a general mechanism to control cache consistency, using snooping techniques.
3. The Pentium processors can change the state of a cache line through read or write cycles or internal snooping and other devices such as L2-cache controller can change the state through external snooping.
4. The MESI protocol provides each cache line that can be one of the four states, and the MESI protocol is managed by the two MESI bits. Figure shows the state transition diagram of MESI protocol.



Modified (M)
When the data of a cache line is marked as modified (M), it is available in a single cache of the complete system only. This cache line can be read or written to without an external cycle.

Exclusive (E)
The exclusive (E) cache line is always stored in only one of the caches in a computer system, though it has not been modified. Hence its values are the same as in the rest of the system. The cache line can be read or written to without an external cycle. Once it is written, to the cache line should be set to modify.

Shared (S)
The shared (S) line can be stored in other caches of the system. The shared line always has the current value so that read accesses can be obtained from the cache. Write accesses to a shared cache line are switched through the external data bus, whenever any cache write strategy is used. Therefore, the shared cache lines in the other caches are invalidated.

Invalid (I)
The cache line which is marked as invalid is not available in the cache. The catch lines marked as invalid (I) lines might be empty or could have invalid data in the cache. Each access to an invalid cache line generates a cache miss. During read access, the cache controller starts a catch line fill and the cache controller switches the write through to the external bus, rather than a write-allocate.

Six Read Access States R1 to R6

- R1 Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor (M←M).
- R2 Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor (E←E).
- R3 Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor (S←S).
- R4 Read access leads to a miss. Data is not available in the cache. The cache controller performs an external read and a line fill (E←I).
- R5 Read access leads to a miss. Data is not available in the cache. The cache controller performs an external read and a line fill (S←I).
- R6 Read access leads to a miss. Data is not available in the cache. The cache controller is not able to perform a line fill and the cache line remains invalid (I←I).

Five Write Access States, W1 to W5

- W1 Write access leads to a hit. Data is available in the cache. As MESI protocol operates with write-back cache strategy, no write cycle is sent to the external bus (M←M).
- W2 Write access leads to a hit. Data is available in the cache and it was not previously overwritten. As MESI protocol operates with write-back cache strategy, no write cycle is sent to the external bus (M←E).

- W3 Write access leads to a hit. Data is available in the cache but it was shared. The cache controller sends a write cycle to the external bus. Then cache line is used in one cache. The main memory is updated, and the cache becomes exclusive (E←S).
- W4 Write access leads to a hit. This state will operate for a write-through cache. At this time, all writes switched to external bus and the cache line stay as shared (S←S).
- W5 Write access leads to a miss. Data will be written to the main memory, but not in the cache. The cache line remains invalid (I←I).

Seven Inquiry or Snooping States, S1 to S7

- S1 The snooping (inquiry) cycle hits a modified cache line. The reference cache line will be written back to main memory (S←M).
- S2 The snooping (inquiry) cycle hits a modified cache line and this state becomes invalidated. The reference cache line is written back to main memory anyway (I←M).
- S3 The snooping (inquiry) cycle hits an exclusive cache line. This state has not been modified and it does not require writing back to main memory. The content of main memory is written to another cache line and the previously exclusive line is now shared (S←E).
- S4 This snooping (inquiry) cycle hits an exclusive cache line. This state has not been modified and does not need to write back to main memory. The content of the main memory is written to another cache line. Due to some reason, this line becomes invalidated (I←E).
- S5 The snooping (inquiry) cycle hits a shared cache line. This snooping cycle informs the system that this cache line is available in the cache (S←S).
- S6 The snooping (inquiry) cycle hits a shared cache line. For some reason, this line will be invalidated (I←S).
- S7 The snooping (inquiry) cycle hit an invalid cache line (I←S).

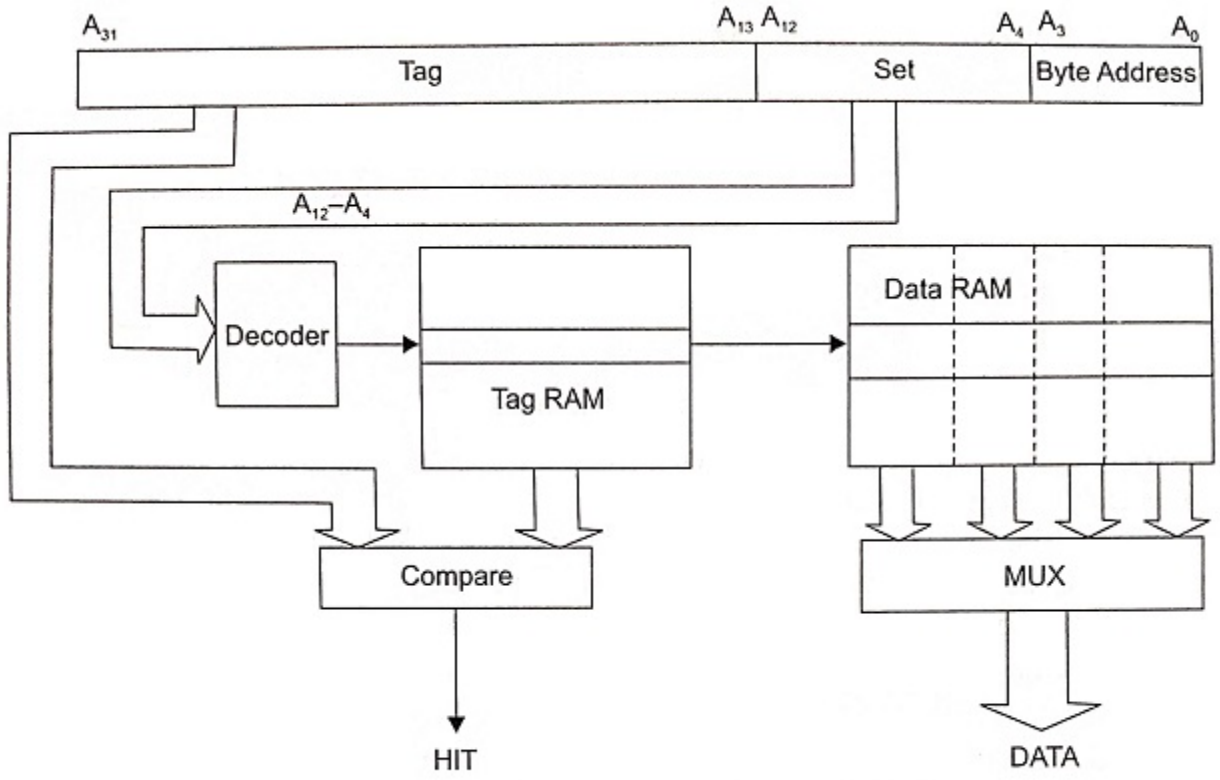
4. Cache Organization

A Cache Memory in Pentium Processor is used to store both the data and the address where the data is stored in the main memory.
There are methods of cache organization such as direct mapped cache and two-way set-associative cache.

Direct Mapped Cache

- The simplest cache organization is the direct mapped cache as shown in figure.
- The features of a direct-mapped cache are given below:
    - A specified data or a memory item is stored in a unique location in the cache memory and two items with the same cache address will compete for use of that location. This is known as contention.
    - Those bits are not used to select within the line or to address the cache RAM, these bits will be stored in the tag field.
    - The tag and data access can be performed simultaneously and also provide the fastest cache access time of any organization.

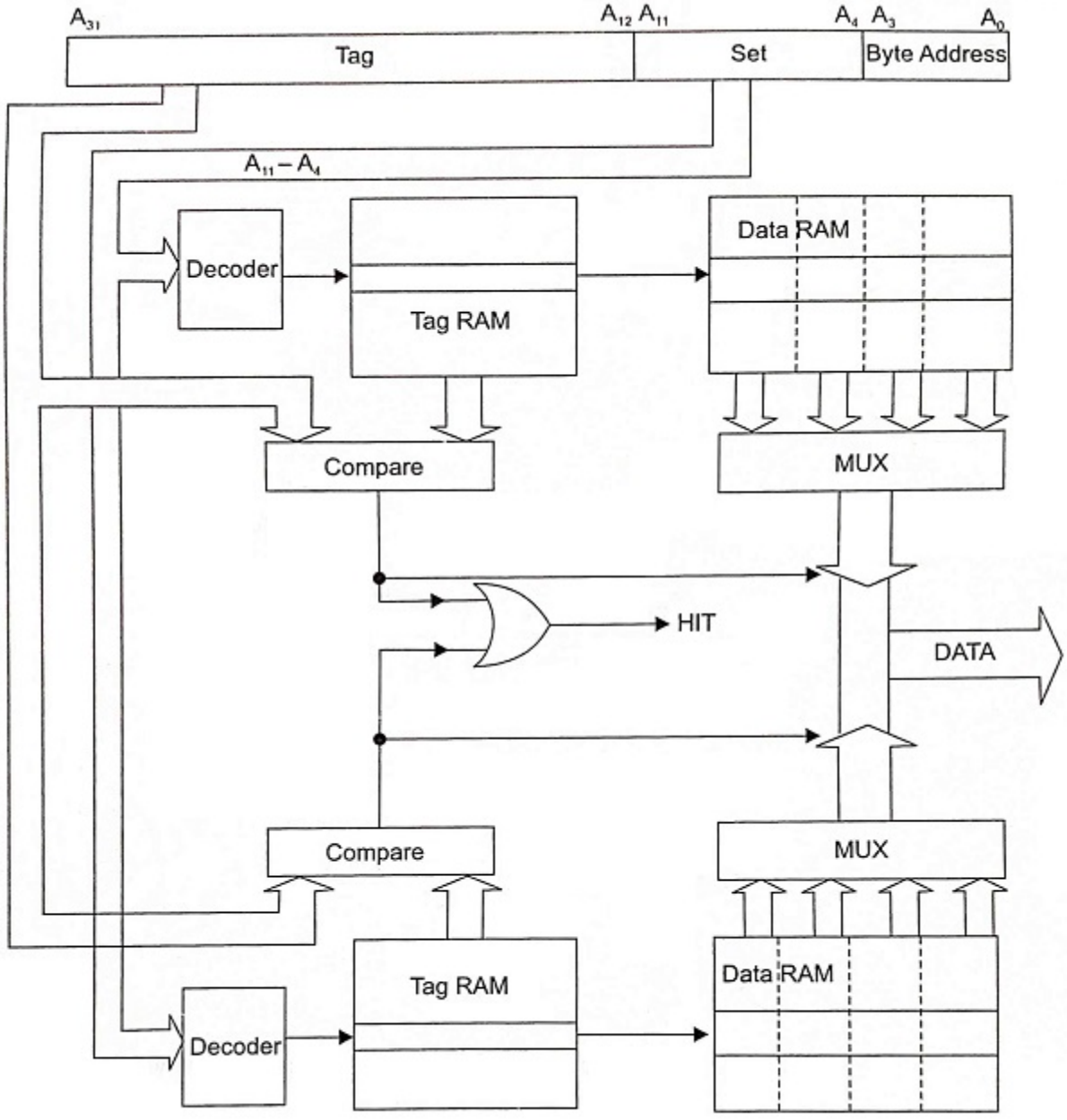
- The tag RAM is always smaller than the data RAM. Hence, the access time of the tag RAM is shorter than data RAM and the tag comparison can be completed within the data access time.
- The direct mapped cache organization can store 8 Kbytes in 16-byte cache lines. The direct mapped cache has 9 address bits A12-A4 or 512 lines. The first 4bits (A3-A0) of the 32-bit address bus A31-A0 are used to address the bytes within the line and there are 19 tag bits (A31-A13). Therefore, 512 x 19 (9728) bits are required for storing the tag.
- While data is loaded into the Cache Memory in Pentium Processor, a block of data will be fetched from memory using burst cycles.



Two-Way Set-Associative Cache

- To reduce the problems due to contention, the set-associative cache is used by allowing a particular memory item to store in more than one cache location.
- In this cache organization, two direct-mapped caches work in parallel. The address applied to the cache may find its data and each memory address can be stored in one of the two places.
- Each of the two items, which were in contention for a single location in the direct-mapped cache organization, may be stored at one of the two places after allowing the cache to hit on both.
- The two-way set-associative caches have 8 Kbyte caches with 16-byte cache lines and eight address bits A11-A4 and 256 lines in either half of the cache.
- Therefore, four bits of the 32-bit address, A3-A0, are used to select a byte from the cache line and eight bits A11-A4 are used to select one line from each half of the cache.
- Consequently, there are 20 address bits A31-A12 for the address tag.

- The access time of a two-way set-associative cache is slightly larger than the direct-mapped cache as the extra time is required to perform the multiplexing operation.



Module 6 - Pentium 4

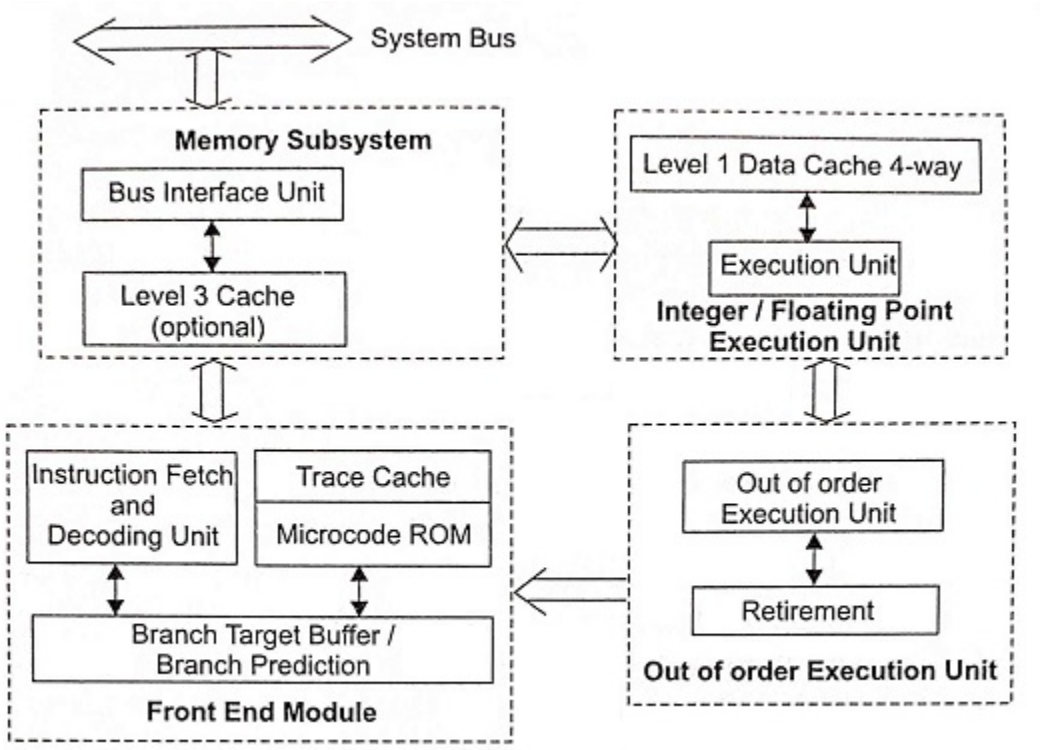
1. Difference / comparison between 8086, 80386, Pentium 1, Pentium 2, Pentium 3

Point	8086	80386	Pentium 1	Pentium 2	Pentium 3
Architecture	16-bit architecture with a segmented memory model.	32-bit architecture with support for protected mode.	32-bit architecture with superscalar execution and on-chip FPU.	32-bit architecture with improved superscalar execution and larger cache.	32-bit architecture with enhanced SSE (Streaming SIMD Extensions) and improved cache.
Clock Speed	Operated at a clock speed of 5 MHz.	Operated at clock speeds ranging from 12 MHz to 40 MHz.	Operated at clock speeds ranging from 60 MHz to 200 MHz.	Operated at clock speeds ranging from 233 MHz to 450 MHz.	Operated at clock speeds ranging from 450 MHz to 1.4 GHz.
Data bus	16-bit data bus.	32-bit data bus.	64-bit data bus.	64-bit data bus.	64-bit data bus.
Addressable Memory	Maximum addressable memory of 1 MB.	Maximum addressable memory of 4 GB.	Maximum addressable memory of 4 GB.	Maximum addressable memory of 64 GB.	Maximum addressable memory of 64 GB.
Instruction Set Extensions	No specific instruction set extensions.	Introduced protected mode and virtual memory support.	Introduced MMX (MultiMedia eXtensions) for enhanced multimedia processing.	Introduced SSE (Streaming SIMD Extensions) for improved multimedia and 3D performance.	Enhanced SSE with additional instructions for improved multimedia and 3D performance.

Cache	No on-chip cache.	Introduced a built-in cache.	Included a 8 KB or 16 KB Level 1 cache.	Featured a larger Level 1 cache (16 KB or 32 KB) and separate Level 2 cache.	Improved cache with larger Level 1 cache (32 KB) and larger Level 2 cache.
Multimedia Performance	8086 processor did not have specific multimedia instructions or extensions.	80386 processor introduced improved multimedia capabilities compared to the 8086.	Pentium 1 processor introduced MMX (MultiMedia eXtensions) technology, which significantly improved multimedia performance.	Pentium 2 processor further enhanced multimedia performance over the Pentium 1.	Pentium 3 processor continued to enhance multimedia performance over the Pentium 2.
Arithmetic Operations	The 8086 processor did not have native support for floating-point arithmetic. Floating-point operations had to be emulated using software routines.	The 80386 processor featured a built-in floating-point unit (FPU) for improved and faster floating-point arithmetic operations.	The Pentium 1 processor introduced the MMX (MultiMedia eXtensions) instruction set, which improved multimedia and vector arithmetic operations.	Improved the MMX technology of the Pentium 1 and introduced the Katmai New Instructions (KNI) set, which enhanced multimedia and 3D graphics operations.	The Pentium 3 processor continued to enhance multimedia performance with the introduction of the SSE (Streaming SIMD Extensions) instruction set.
Segmentation	Divided memory into multiple segments	Introduced a flat memory model in protected mode	Operate in a flat memory model in protected mode.	Operate in a flat memory model in protected mode.	Offered improved memory management capabilities in protected mode.

Transistors	Approx 29,000 transistors.	Approx 275,000 transistors.	Approx 3.1 million transistors.	Approx 7.5 million transistors.	Approx 9.5 million transistors.
Manufacturing process	Fabricated using a 3-micron process.	Fabricated using a 1.5-micron process.	Fabricated using a 0.8-micron process.	Fabricated using a 0.35-micron process.	Fabricated using a 0.25-micron process.

2. Explain Net Burst Micro Architecture in Pentium 4



Bus interface Unit (BIU)

The Bus Interface Unit (BM) is used to communicate with the system bus, cache bus, L2 cache, L1 data cache and L1 code cache.

Instruction Decoder

The instruction decoder is used to decode all instructions of the Pentium 4 processor concurrently and translate them into micro-operations (μ-ops). One instruction decoder decodes one instruction per clock cycle.

Trace Cache (TC)

After translation of instructions into micro-operations (μ-ops) by using an instruction decoder, the streams of decoded instructions are fed to an L1 instruction cache, which is known as trace cache.

Microcode ROM
As complex instructions perform string and interrupt operations, etc., the trace cache transfers the control operation of complex instructions to a micro-code ROM.

Branch Prediction Logic Unit
It predicts the memory locations from where the next instruction will be fetched. Usually, the predictions are performed using the past information of the program execution.

Instruction Translation Look aside Buffer (ITLB)
When a trace cache miss occurs, instruction bytes are required to be fetched from the L2 cache.

Execution Unit
A superscalar processor has multiple parallel execution units, which can process the instructions simultaneously. Actually, the executions of instructions are sequentially dependent on each other.

Allocator
The allocator accepts micro-operations (μ-ops) from the μ-ops queue and allocates the key machine buffers to execute micro-operations.

Register Rename
The register rename logic is used to rename the registers of Intel Architecture 32-bit Pentium processors onto the machine's physical registers.

Instruction Schedulers
The instruction scheduler is used to schedule micro-operations (μ-ops) to an appropriate execution unit. There are five instruction schedulers to schedule micro-operations in different execution units.

Rapid Execution Module
There are two ALUs (Arithmetic Logic Unit) and two AGUs (Address Generation Unit) in a Pentium 4 processor.

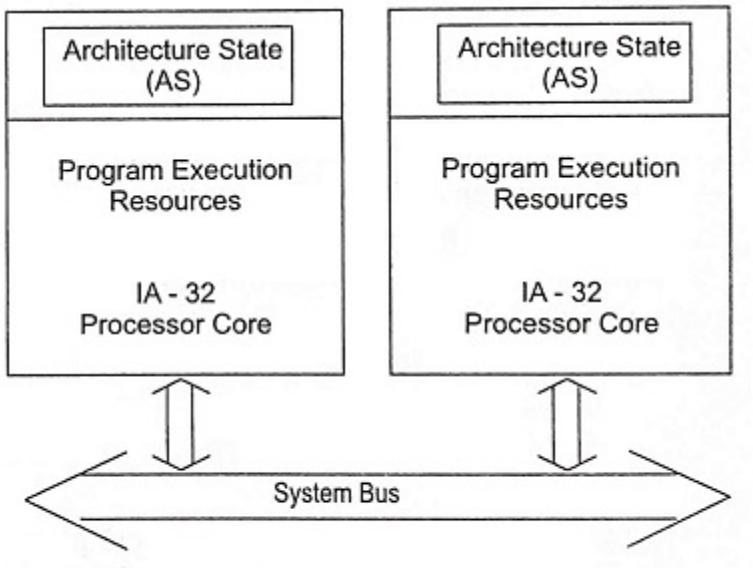
Memory Subsystem
The virtual memory and paging technique are used in memory-subsystem representation. The linear address space can be mapped into the processor's physical address space, either directly or using a paging technique. In direct mapping, paging is disabled and each linear address represents a physical address

3. Explain hyper threading technology in Pentium 4

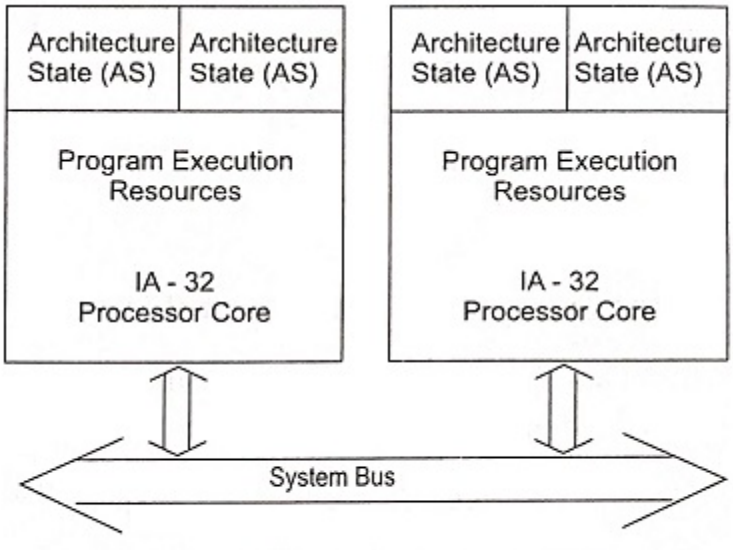
- This technology was first implemented in Pentium® 4 Xeon processor in 2002. The features of hyper-threading technology are given below:
    - HT makes a single physical processor appear as multiple logical processors.

    - Each logical processor has its own architecture state where a set of single execution units are shared between logical processors.
    - HT allows a single processor to fetch and execute two separate code streams simultaneously.
    - In most of the applications, the physical unit is shared by two logical units.
- It is well known to us that each process has a context in which all the information related with the current state of execution of the process are described.
- In any process, the contents of the CPU registers, the program counter, the flag register are used as context. Each process should have at least one thread and sometimes more than one thread is present in a process.
- Each threads has its own local context. Sometimes the context of a process is shared by the other threads in that process.
- The common features of threads are as follows:
    - The threads can be independent in a process.
    - The threads can be bunched together into a process.
    - The threads may be simple in structure and can be used to increase the speed of operation of the process.

- A single processor with multi-processing/multi-threading or CMP can be supported in different ways such as
    - Time-Sliced Multi-threading (TSM) In time-sliced multi-threading, the processor switches from one task to another after a fixed amount of time has passed. This technique is also called real multitasking.
    - Switch-on Event Multi-threading (SEM) The processor could be designed to switch to another task whenever a cache miss occurs.
    - Simultaneous Multi-threading (SM) In simultaneous multi-threading or hyper-threading, multiple threads may be executed on a single processor without switching.



Processor without hyper-threading



Processor without hyper-threading

Architecture State (AS)

The architecture state has the following registers:

1. Registers including the general-purpose registers
2. Control register
3. Advanced Programmable Interrupt Controller (APIC) registers
4. Machine state registers

Replicated Resources

Each processor has general-purpose registers, control registers, flags, time stamp counters, and APIC registers. The content of these registers are used as replicated resources.

Shared Resources

Memory and range registers can be independently read/write. Therefore, memory, range registers and data buses are used as shared resources.

Shared/Replicated Resources

The caches and queues in the hyper-threading pipeline can he shared or not shared according to the situation.