Explain Interrupt structure of 8086 processor

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

1. There are three sources of interrupts for 8086:

1. **Hardware interrupt-**

   These interrupts occur as signals on the external pins of the microprocessor. 8086 has two pins to accept hardware interrupts, NMI and INTR.

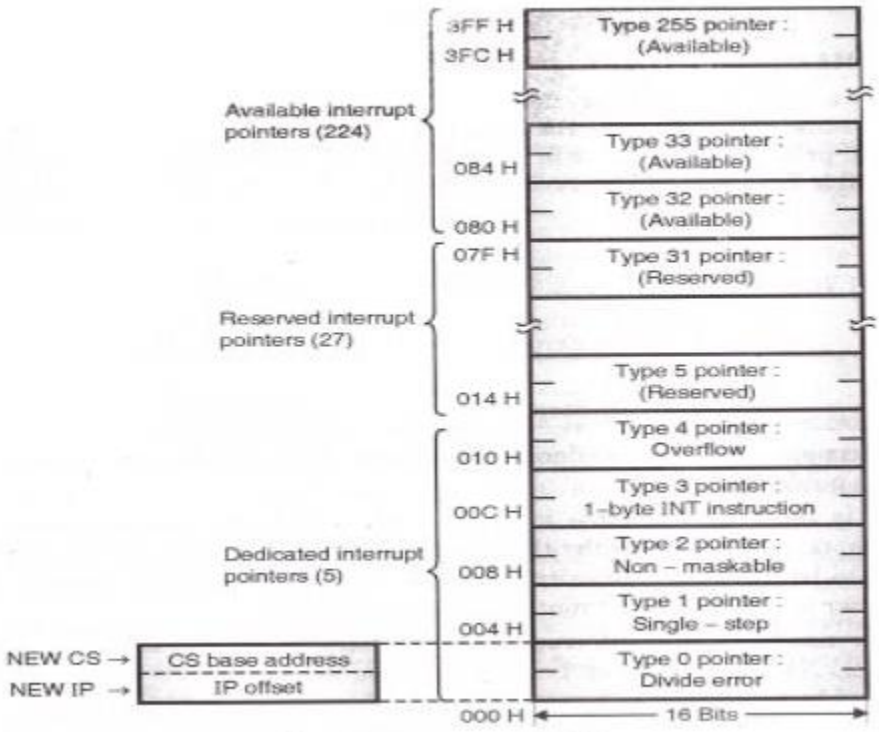
2. **Software interrupt-**

   These interrupts are caused by writing the software interrupt instruction INT n where 'n' can be any value from 0 to 255 (00H to FFH). Hence all 256 interrupts can be invoked by software.

3. **Error conditions (Exception or types)-**

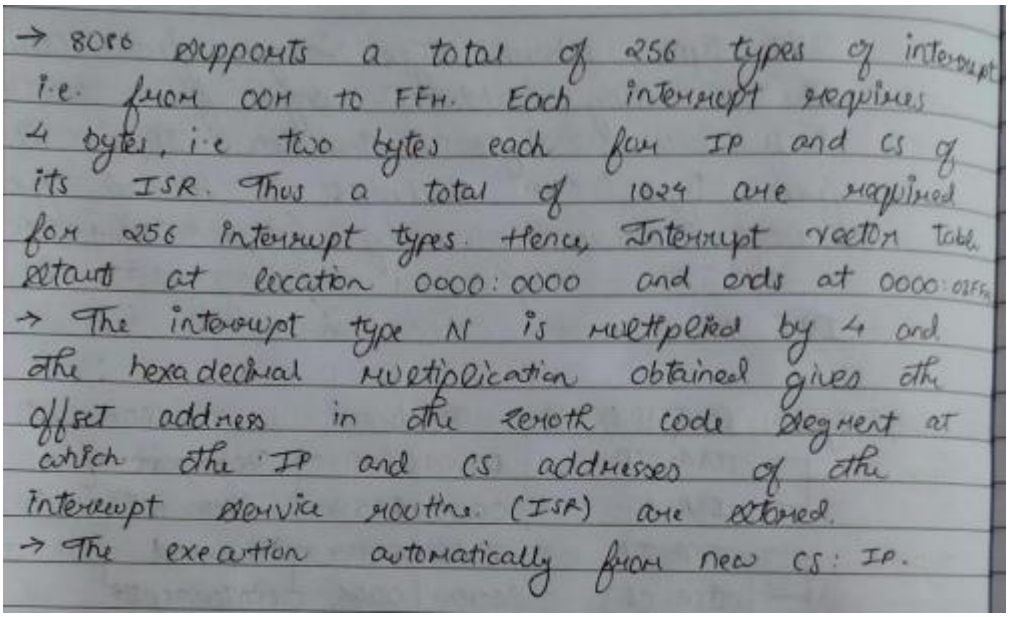
   8086 is interrupted when some special conditions occur while executing certain instructions in the program. Example: An error in division automatically causes the INT 0 interrupt.

Interrupt Vector Table (IVT):



Interrupt vector table

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.



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

INT 1 (Single Step)-

- The microprocessor executes this interrupt after every instruction if the TF is set.
- It puts microprocessor in single stepping mode

INT 2 (Non mask-able Interrupt)-

The microprocessor executes this ISR in response to an interrupt on the NMI (Non mask-able Interrupt) line

INT 3 (Breakpoint Interrupt)-

- This interrupt is used to cause breakpoints in the program.

- It is useful in debugging large programs where single stepping is efficient.

INT 4 (Overflow Interrupt)-

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

. **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.

Hardware Interrupts:

1. **NMI (Non mask-able interrupt)-**
    - This is a non-mask-able, edge triggered, high priority interrupt.
    - On receiving an interrupt on NMI line, the microprocessor executes INT

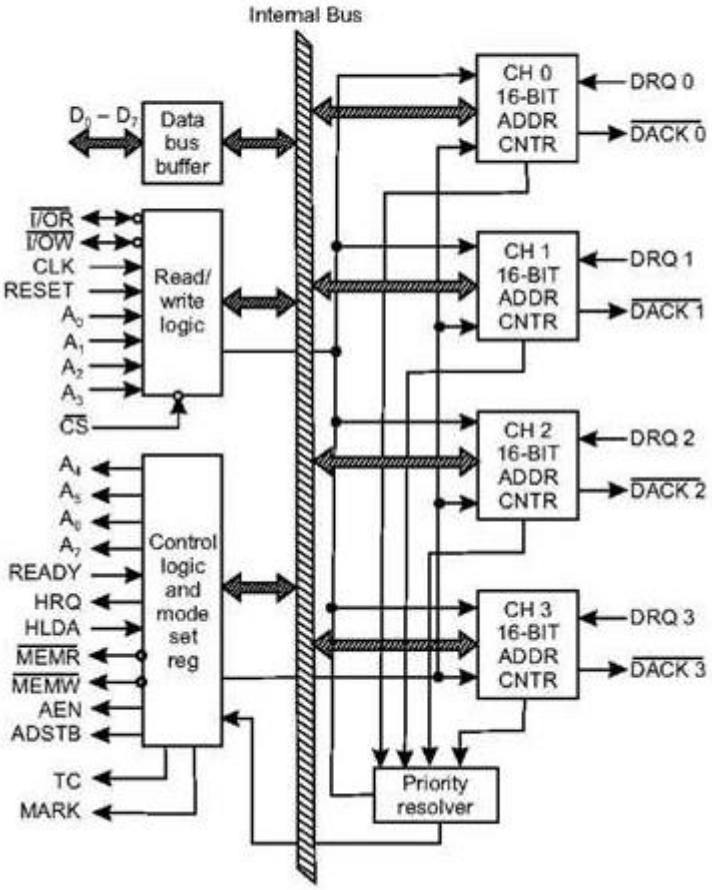
INTR-

- This is a mask-able, level triggered, low priority interrupt.
- On receiving an interrupt on INTR line, the microprocessor executes 2 $\overline{\text{INTA}}$ $\overline{\text{INTA}}$ pulses.
- 1st $\overline{\text{INTA}}$ $\overline{\text{INTA}}$ pulse – The interrupting device calculates (prepares to send) the vector number.
  2nd $\overline{\text{INTA}}$ $\overline{\text{INTA}}$ pulse – The interrupting device sends the vector number ‘N’ to the microprocessor.

Draw and Explain block diagram of 8257? How DMA operations are performed?

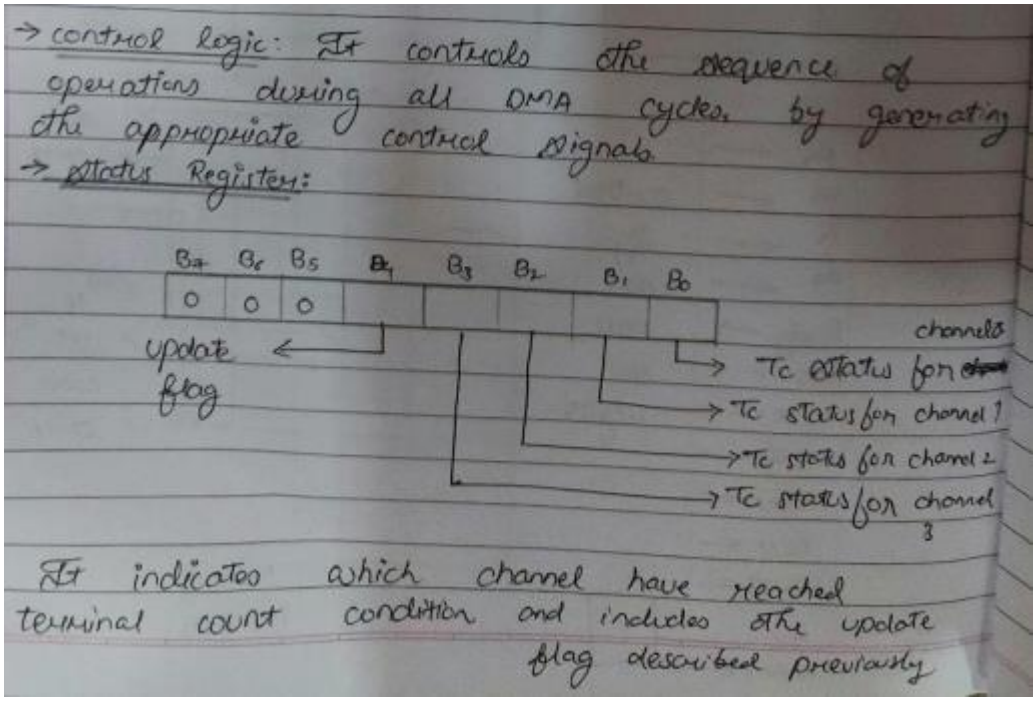
8257 Architecture

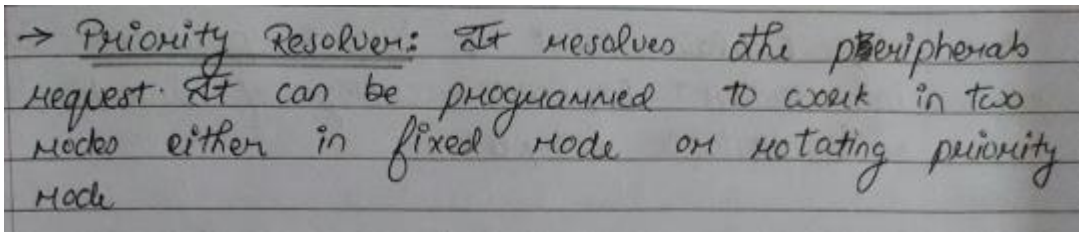
The following image shows the architecture of 8257 −



→ Data bus Buffer: It is a tri-state, bi-directional, eight bit buffer which interfaces the 8257 to the system data bus. It is used to transfer data between up and internal register of 8257.

→ Read/write logic: when the CPU is programming or reading one of the internal registers of 8257 pin diagram, the Read/write accepts the I/O Read or I/O write signal, decodes the least significant four addles bit and either writes the contents of the data bus into the addressed register or places the contents of the addressed register onto the data bus.





DRQ_0–DRQ**3**

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 DRQ_0 has the highest priority and DRQ_3 has the lowest priority among them.

$DACK_o$ – $DACK_3$

the requesting peripheral about the status of their request by the CPU.

D_o – D_7

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257

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

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.

A_o - A_3

These are the four least significant address lines.

CS

It is an active-low chip select line.

A_4 - A_7

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.

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

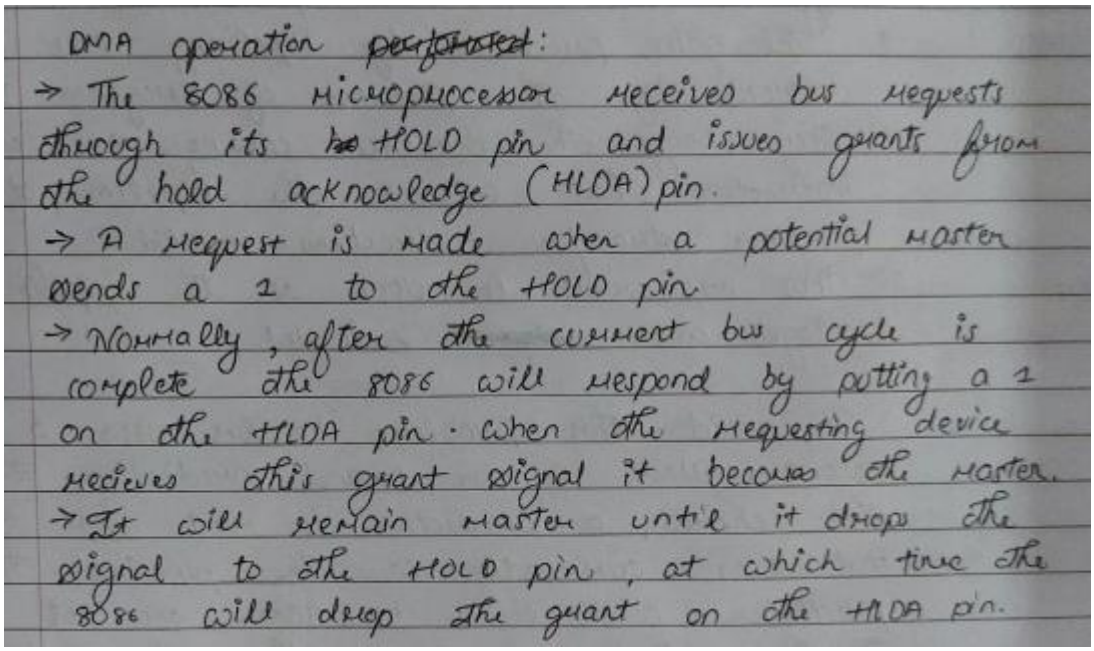
This signal is used to disable the address bus/data bus.

TC

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

MARK

It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.



DMA operation ~~performed~~:
→ The 8086 microprocessor received bus requests through its ~~he~~ HOLD pin and issues grants from the hold acknowledge (HLDA) pin.
→ A request is made when a potential master sends a 1 to the HOLD pin.
→ Normally, after the current bus cycle is complete the 8086 will respond by putting a 1 on the HLDA pin. When the requesting device receives this grant signal it becomes the master.
→ It will remain master until it drops the signal to the HOLD pin, at which time the 8086 will drop the grant on the HLDA pin.

(3) Explain what is branch prediction logic in Pentium? Explain working of Branch prediction with suitable example?

Ans:

→ Why do we need branch prediction?

1. The gain produced by Pipelining can be significantly reduced by the presence of program transfer instructions.
2. They change the sequence causing all the instruction that entered the pipeline after program transfer instructions invalid.
3. This 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 branch instructions 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 will be flushed and restarts over with the current instruction.

Working of Branch prediction:

→ BTB is a lookaside cache that sits to the side of Decode instruction (DI) stage of two pipelines and monitors for branch instruction.

→ The first time the 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, the branch will be either 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 ~~delayed~~ sequential fetch of instruction.
→ When the branch is taken for the first time, the execution unit provides feedback to the branch prediction. The branch target address ~~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.

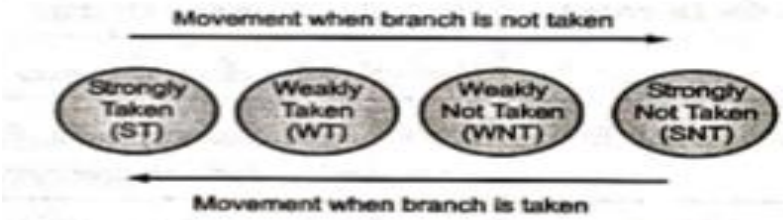


Fig. 11.5.2 : History bits

- The history bits can indicate one of four possible states.

History Bits	Resulting Description	Prediction made	If actually taken	If actually not taken
00	Strongly Not Taken	Branch Will Not Be Taken	Upgrades to Weakly Not Taken	Remains Strongly Not Taken
01	Weakly Not Taken	Branch Will Not Be Taken	Upgrades to Weakly Taken	Downgrades to Strongly Not Taken
10	Weakly Taken	Branch Will Be Taken	Upgrades to Strongly Taken	Downgrades to Weakly Not Taken
11	Strongly Taken	Branch Will Be Taken	Remains Strongly Taken	Downgrades to Weakly Taken

---

Q.4 Compare the 8086, 80386, Pentium Processor.

Product	8086	80386	Pentium
Year introduced	1978	1985	1992
Technology	NMOS	CMOS	BICMOS
Clock rate (MHz)	3 - 10	16 - 33	60, 66
Number of pins	40	132	273
Number of transistors	29,000	275,000	3.1 million
Physical memory	1M	4G	4G
Virtual memory	None	64T	64T
Internal data bus	16	32	32
External data bus	16	32	64
Address bus	20	32	32
Data type (bits)	8, 16	8, 16, 32	8, 16, 32

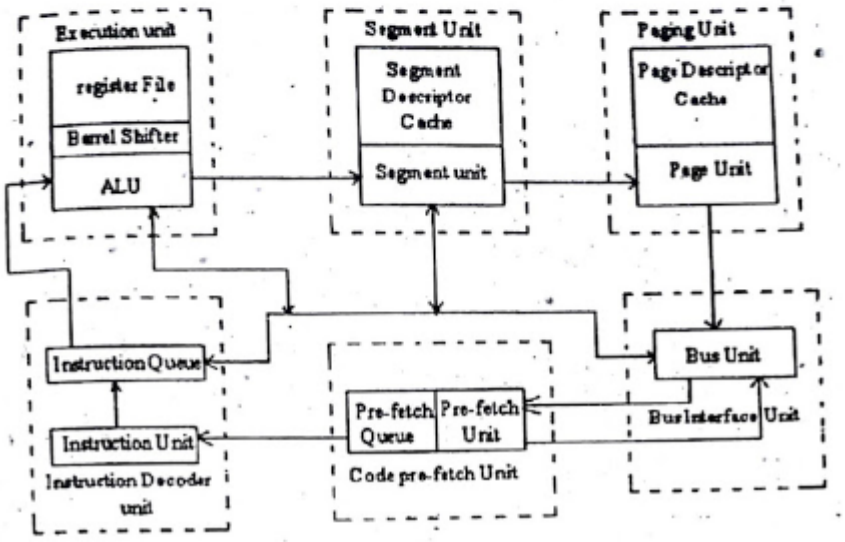
Draw and explain the internal architecture of 80386 microprocessor.

MMRCSE 12/07/2018 04:38:00 PM 2

The internal architecture of the 80386 inch1des six functional units that operate in parallel. The parallel operation is called as pipeline processing. Moreover, Fetching, decoding execution, memory management, and bus access, for several instructions are performed simultaneously.

Also, the six functional units of the 80386 Architecture are

- Bus Interface Unit
- Code Pre-fetch Unit
- Instruction Decoder Unit
- Execution Unit
- Segmentation Unit
- Paging Unit



The Bus Interface Unit connects the 80386 with memory and I/O. Based on internal requests for fetching instructions and transferring data from the code pre-fetch unit, the 80386 Architecture generates the address, data and control signals for the current bus cycles.

Also, The code pre-fetch unit pre-fetches instructions when-the bus interface unit is not executing the bus cycles. It then stores them in a 16-byte instruction queue for decoding by the instruction decode unit.

Moreover, The instruction decode unit translates instructions from the pre-fetch queue into microcode. The decoded instructions then stored in an instruction queue (FIFO) for processing by the execution unit.

The execution unit processes the instructions from die instruction queue. It contains a control unit a data unit and a protection test unit.

The control unit contains microcode and parallel hardware for fast multiply, divide, and effective address calculation. The unit includes a 32-bit ALU, 8 general purpose registers and a 64-bit barrel shifter for performing multiple bit shifts in one clock. The data unit carries out data operations requested by the control unit.

Moreover, The protection test unit checks for segmentation violations under the control of microcode.

Also, The segmentation unit calculates and translates the logical address into linear addresses at the request of the execution unit.

The translated linear address sent to the paging unit. Upon enabling the paging mechanism, the 80386 translates these linear addresses into Physical addresses.

Also if paging not enabled, the physical address is identical to the linear address and no translation is necessary.

Q.5) Explain the operating modes of 80386?

=> The operating mode of the 80386 has three processing modes

1) Real-Address mode

→ i) Power on default mode

ii) In real address mode, 80386 acts as a version of 8086 which is faster than original 8086 as 80386 operates on higher clock frequency.

iii) It offers memory addressability of 1MB of physical memory

iv) Segmented addressing: Total 6 segments

2) Protected Mode

i) It is the natural 32-bit environment of the 80386 processor

ii) In this mode all instructions and features are available.

iii) Segment size = 1 to 4GB

iv) If 80386 is working in protected mode it can not switch to real mode unless it is reset.

3) Virtual 8086 Mode

i) ~~V86 mode of~~ Virtual 8086 mode also called as V86 mode. It is a dynamic in the sense that the processor can switch repeatedly and rapidly b/w V86 mode and protected mode

ii) The CPU enters V86 mode from protected mode to execute an 8086 program, then leaves V86 mode and enters protected mode to continue executing a native 80386 program.

iii) The features that are available to applications programs in protected mode and to all programs in V86 mode are the same.

(7) Explain internal architecture of 8086 µp ? Differentiate the funtioning of minimum mode and maximum mode?

Ans:- → The architecture of 8086 provides a number of improvements over 8085 µp architecture.

→ It supports 16-bit ALU, a set of 16-bit registers, a rich instruction set, powerful interrupt structure, fetched instruction queue.

→ The 8086 architecture is divided into two parts (a) Bus Interface unit (BIU) and (b) Execution unit.

(a) Bus Interface unit (BIU):

It provides the interface of 8086 to external memory and I/O devices via the system Bus. It performs various machine cycles such as memory read, I/O read, etc. to transfer data between memory and I/O devices.

BIU mainly contains 4 segment registers, instruction pointer, prefetch queue and an address generation circuit.

The four segment registers are,

code segment (CS)

Data segment (DS)

stack segment (SS)

Extra segment (ES)

(b) Execution unit (EU): The main components of EU are General purpose register, the ALU, special purpose register, Instruction register and decoder, and flag registers.

There are 16 general purpose registers:

→ AX register

It holds operands and result during multiplication and division operation. Also an accumulator during string operation.

→ BX register.

Holds memory address in indirect addressing mode.

→ CX register

It holds count for instruction like loop, rotate, shift and string operation.

→ DX register

It is used with AX to hold 32 bit value during multiplication and division.

Special purpose registers

→ stack pointer

→ Base pointer

→ source index

→ Destination index.

Flag register consist of 9 flags.

6 status flags:

carry flag (CF)
Parity flag (PF)
Auxiliary carry flag (AF)
Zero flag (ZF)
Sign flag (S)
Overflow flag (O)

3 control flags

Trap flag (TF)
interrupt flag (IF)
Direction flag (DF)

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

(8) Write an assembly language program to find the largest number from an ~~array~~ unordered array of 8-bit numbers.

Sol:

example:

	no of element in array	element of array			
Input data →	04	20	1A	55	2B
Memory address →	2050	2051	2052	2053	2054

output data → 55 ← largest element
memory address → 3050

Program:

Memory address	Mnemonics	comment
2000	LXI H, 2050	H ← 20, L ← 50
2003	MOV C, M	C ← M
2004	DCR C	C ← C-01
2005	INX H	HL ← HL+0001
2006	MOV A, M	A ← M
2007	INX H	HL ← HL+0001
2008	CMP M	A - M
2009	JNC 2000	If carry flag=0, go to 2000
200C	MOV A, M	A ← M
200D	DCR C	C ← C-1
200E	JNZ 2007	If zero flag = 0, go to 2007
2011	STA 3050	A → 3050
2014	HLT	

Q. 2: Interface 32 K word of memory to the 8086 microprocessor system . Available memory chips are 16 K x 8 RAM. Use suitable decoder for generating chip select logic.

Step_1: Total memory = 32 K word = 32*2 K = 64 K

IC available = 16 K

hence,

number of RAM IC required = 64 K x 8/ 16 Kx8 = 4 ICs

so,

EVEV Bank = 2 ICs of 16 Kx8 RAM

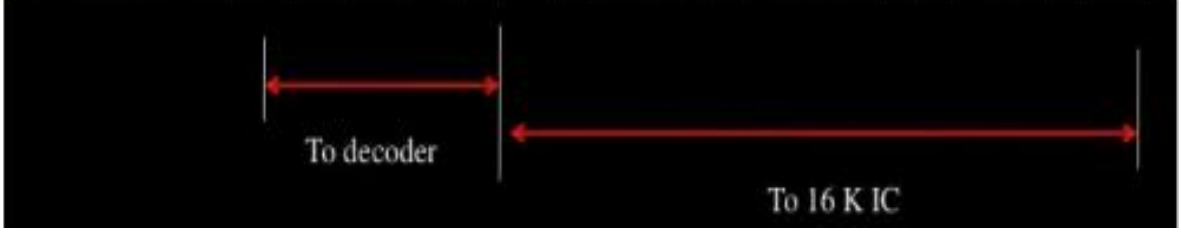
ODD Bank = 2 ICs of 16 Kx8 RAM

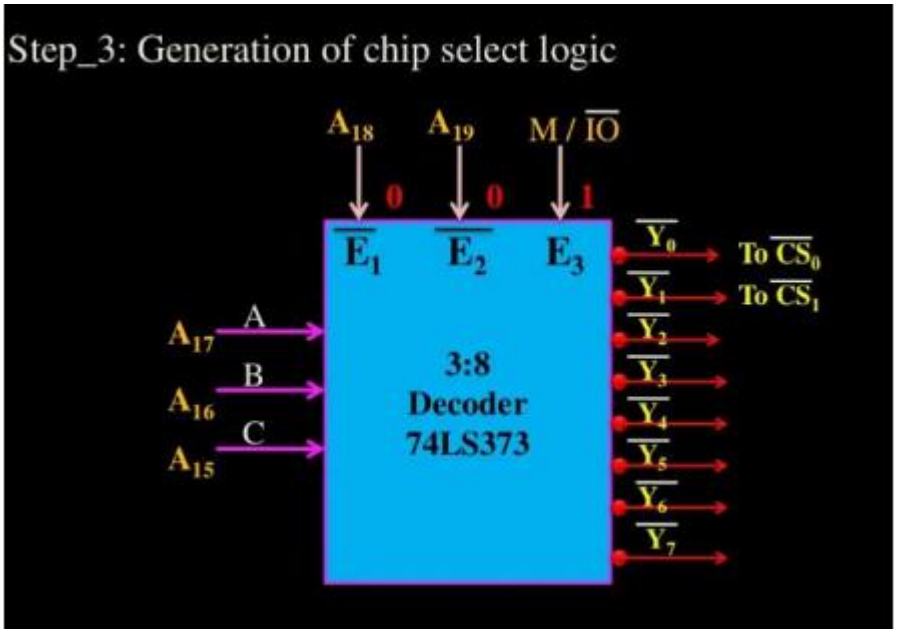
Even bank	Odd bank
RAM _1 (16K)	RAM _2 (16K)
RAM_ 3 (16K)	RAM _4 (16K)

Step_2: Number of address lines required = 15 address lines

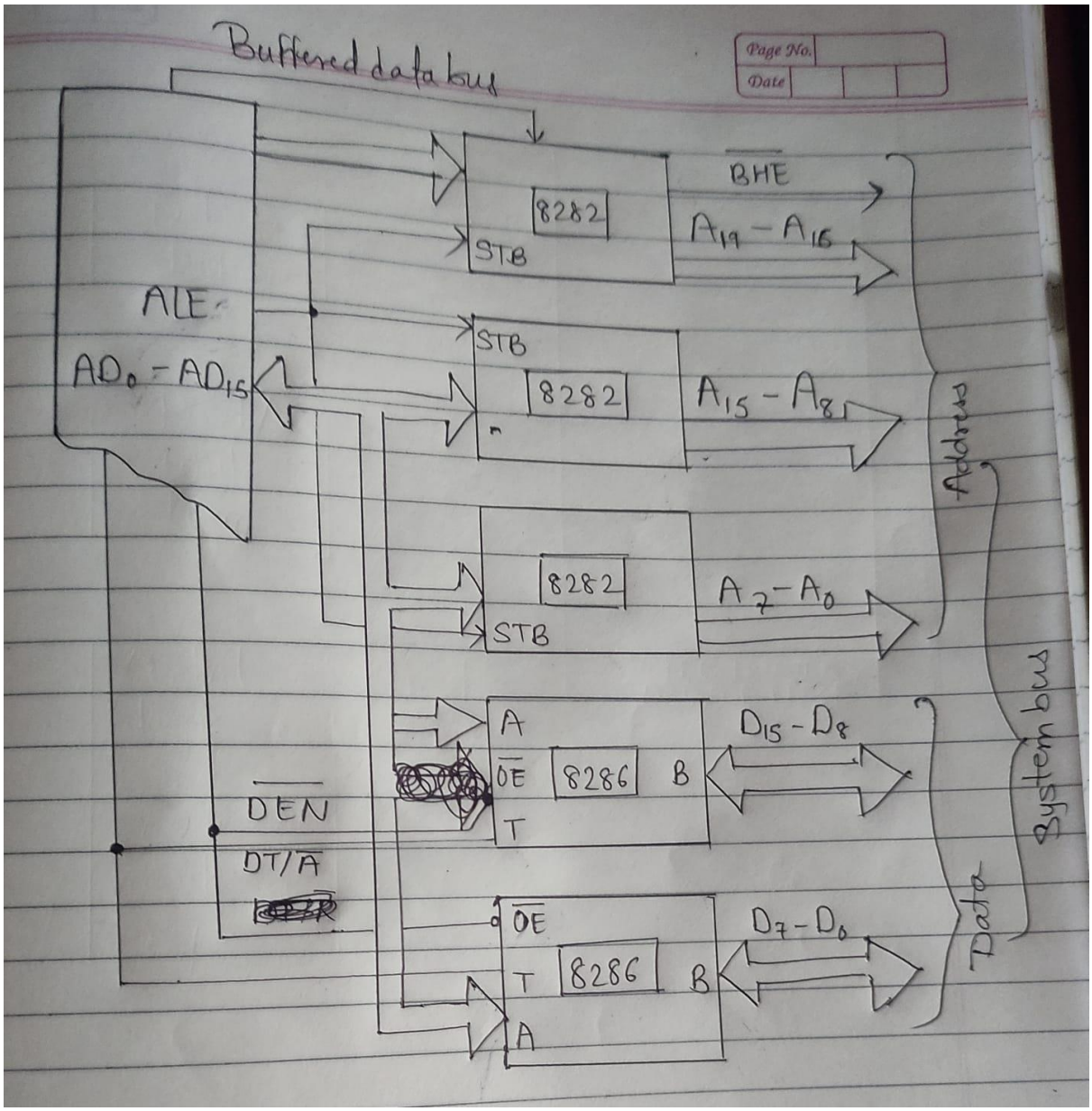
Step_3: Address decoding table

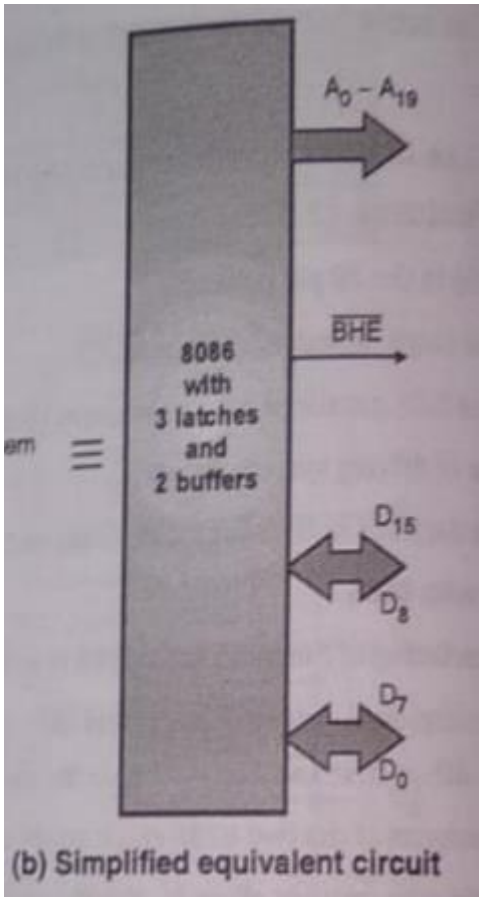
MEMORY IC	HEX ADDRESS	BINARY ADDRESS																			
		A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
16 K x 8 RAM-(1)	00000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	07FFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
16 K x 8 RAM-(3)	8000	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0FFFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0




Step_3: Generation of chip select logic
A18
A19
M / IO
0
0
1
E1
E2
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
To CS0
To CS1
A17
A16
A15
A
B
C
3:8
Decoder
74LS373

Explain address and data bus demultiplexing in 8086 with diagram.





(b) Simplified equivalent circuit

We have discussed the use of 8282 latches as address latching and 8286 as data bus buffer.

If we use them simultaneously then we can demultiplex the address data bus as shown in Fig. 3.4.3(a).

Thus we require three octal latch chips (8282) and two octal transceiver chips (8286) in order to completely demultiplex the address and data bus.

The basic diagram of Fig. 3.4.3 is often required in the memory interfacing and I/O interfacing.

The simplified equivalent of Fig. 3.4.3(a) is shown in Fig. 3.4.3(b).

(11) Discuss need for Memory banking in 8086.

Ans: → The 8086 processor provides a 16-bit data bus. So it is capable of transferring 16 bit in one cycle but each memory location is only of a byte, therefore we need two cycles to acess 16 bits from two different memory locations. The solution to this problem is Memory banking

→ The memory chip is equaly divided into two parts (banks). One of the banks contains even addresses called Even bank and other contains odd addresses called odd bank. Even bank always gives lower byte so even bank is called lower bank and ODD bank is called Higher bank.

→ This banking scheme allows to access two aligned memory location from both banks simultaneously and process 16-bit data transfer. memory banking doesn't make it compulsory to transfer 16-bit, it facilitates the 16-bit data transfer.

→ The choice between 8-bit and 16-bit transfer depends on the instructions given by the programmers.

(12) Explain Mode 0 and Mode 2 of 8255.

Sol: Mode-0:

→ This mode is also known as basic I/O mode
→ This mode provides simple input and output capability using each of the three port
→ Data can be simply read from and written to input and output ports respectively, after appropriate initialisation.

Salient features of this mode
(i) Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available
(ii) Any port can be used as an input or output port.
(iii) Output ports are latched. Input ports are not latched
(iv) A maximum of four ports are available so that overall 16 I/O configurations are possible.

Mode-2:→ This mode of operation of 8255 is also known as strobed bidirectional I/O.
→ This mode of operation provides 8255 with an additional feature for communicating with a peripheral device on an 8-bit data bus.
→ Handshaking signals are provided to maintain proper data flow and synchronization between data transmitter and reciever.
→ The interrupt generation and other functions are similar to mode 1
→ Thus, in this mode, 8255 is a bidirectional 8-bit port with handshake signals.
→ The RD and WR signals decide whether the 8255 is going to operate as an input port or output port

Salient features
→ The single 8-bit port in group A is available
→ The 8-bit port is bidirectional and additionally a 5-bit control port is available
→ Three I/O lines are available at port C, viz PC2 - PC0
→ Inputs and outputs are both latched

Explain interrupt procedure of 8086

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

When this interrupt is activated, these actions take place –

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR

These actions are taken by the microprocessor –

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location X × 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

INT

Its execution includes the following steps –

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' × 4
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

Explain integer pipeline of Pentium

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

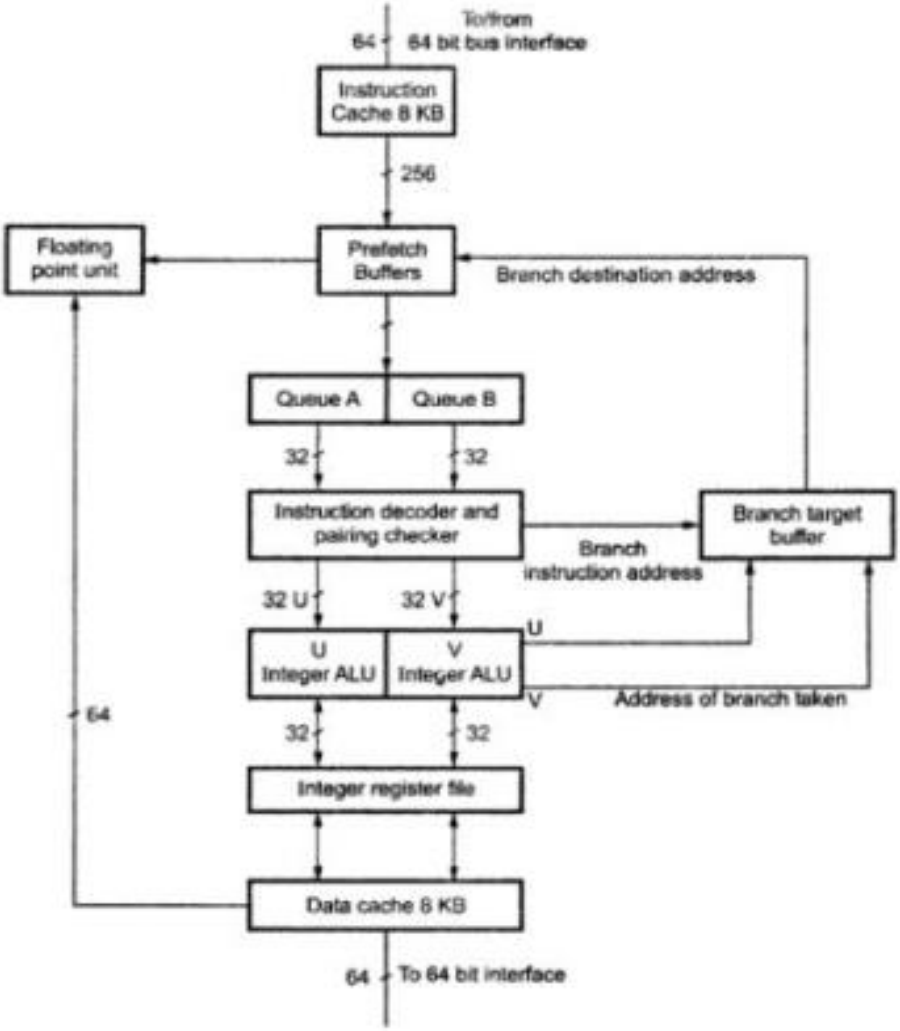
1.Prefetch stage - Pentium instructions are variable length and are stored in a prefetch buffer. There is a 256 bit path from instruction cache to the prefetch buffer.

2.Decode 1 stage - In this stage the processor decodes the instruction and finds the opcode and addressing information, check which instructions can be paired for simultaneous execution and participates in branch address prediction.

3.Decode 2 stage - Addresses for memory reference are found in this stage.

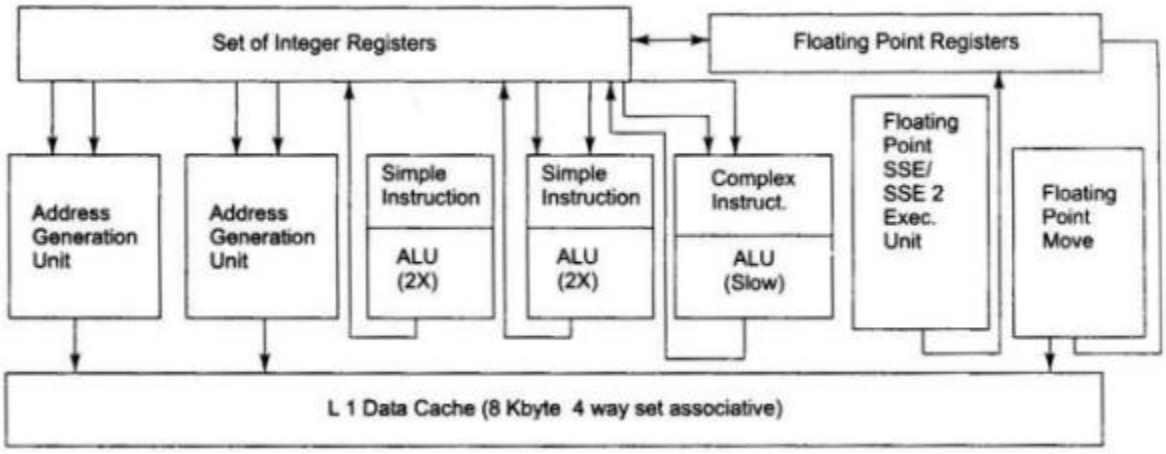
4. Execute stage - Data cache fetch or ALU or FPU operation is carried out. Two operations can be carried out at this stage.

5. Write back stage - In this stage the registers and flags are updated on the basis of the results of execution.


To/from
64
64 bit bus interface
Instruction
Cache 8 KB
256
Floating
point unit
Prefetch
Buffers
Branch destination address
Queue A
Queue B
32
32
Instruction decoder and
pairing checker
Branch target
buffer
Branch
instruction address
32 U
32 V
U
U
Integer ALU
V
Integer ALU
V
Address of branch taken
64
32
32
Integer register file
Data cache 8 KB
64
To 64 bit interface

(15) Write a note on Hyperthreading.

Ans:- → Hyperthreading used the concept of simultaneous multithreading and shows an improvement in the Intel microarchitecture development.
→ The major elegance of this architecture lies in devising appropriate resource sharing policy for each shared resource. Several resource sharing strategies have been investigated by the developers. Some of these are (a) partitioned resources (b) threshold sharing, and (c) full sharing. The choice of sharing strategy to be adopted depends on several factors. Such as, the traffic pattern, size of the resource, potential deadlock probabilities and other considerations.
→ To do this, there is one copy of the architecture state for each logical processor, and the logical processor share a single set of physical execution resources. From a software or architecture perspective, this means operating systems and user programs can schedule processes or threads to logical processors as they could on conventional physical processor in a multi-processor system. From a microarchitecture perspective, this means that instructions from logical processors will persist and execute simultaneously on shared execution resources.



Program 3.5

A program to find out the number of even and odd numbers from a given series of 16-bit hexadecimal numbers.

Solution The simplest logic to decide whether a binary number is even or odd, is to check the least significant bit of the number. If the bit is zero, the number is even, otherwise it is odd. Check the LSB by rotating the number through carry flag, and increment even or odd number counter.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DW 2357H, 0A579H, 0C322H, 0C91EH, 0CD00H, 0957H
COUNT EQU 006H
DATA ENDS
CODE SEGMENT
START:      XOR BX, BX
            XOR DX, DX
            MOV AX, DATA
            MOV DS, AX
            MOV CL, COUNT
            MOV SI, OFFSET LIST
AGAIN:      MOV AX, [SI]
            ROR AX, 01
            JC ODD
            INC BX
            JMP NEXT
ODD:        INC DX
NEXT:       ADD SI, 02
            DEC CL
            JNZ AGAIN
            MOV AH, 4CH
            INT 21H
            CODE ENDS
            END START
```

Program 3.5 *Listings*

Q18.

80386: Protected Mode

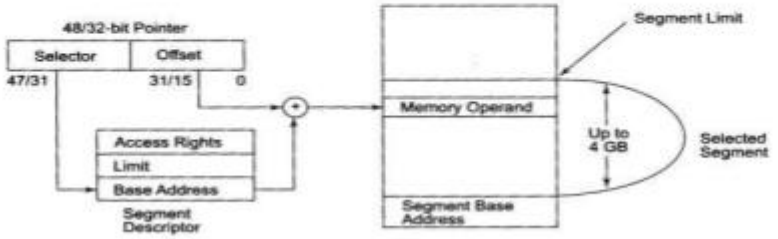
All the capabilities of 80386 are available for utilization in its protected mode of operation. In this mode, the 80386 can address 4 Gigabytes of physical memory and 64 terrabytes of virtual memory per task. The 80386 in the protected mode supports all softwares written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386. The protected mode allows the use of additional instructions, addressing modes and capabilities of 80386.

Addressing in Protected Mode

80386 support two methods
a) Paging disabled b) Paging enabled

a) Paging disabled

In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment. The effective address (offset) is added with segment base address to calculate linear address. This linear address is further used as physical address, if the paging unit is disabled. Otherwise, the paging unit converts the linear address into physical address.

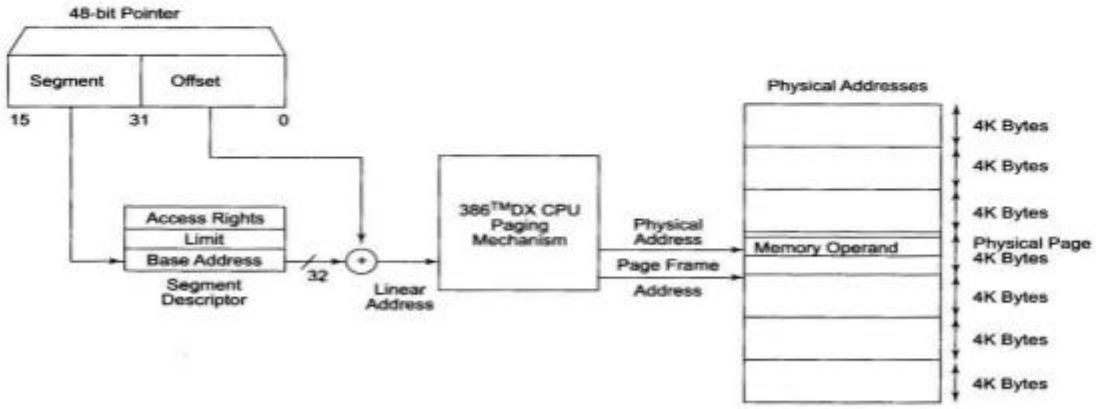


Protected Mode Addressing without Paging Unit (Intel Crop.)

80386: Protected Mode

b) Paging Enabled

The paging unit is a memory management unit enabled only in the protected mode. The paging mechanism allows handling of large segments of memory in terms of pages of 4 Kbyte size. The paging unit operates under the control of segmentation unit. The paging unit if enabled converts linear addresses into physical addresses, in protected mode.



Paging Unit Enabled in Protected Mode Addressing (Intel Corp.)

(19) Explain memory segmentation in 8086 with neat diagram.

Ans: 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.

The Bus interface unit (BIU) contains four 16-bit special purpose registers called as segment Registers.

→ Code segment Register (CS): is used for addressing memory location in 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 the segment in the memory which is another data segment in the memory.

→ Stack segment Register (SS): it is used for addressing stack segment of the memory.

Fig. 1-Q.6(a) : Computation

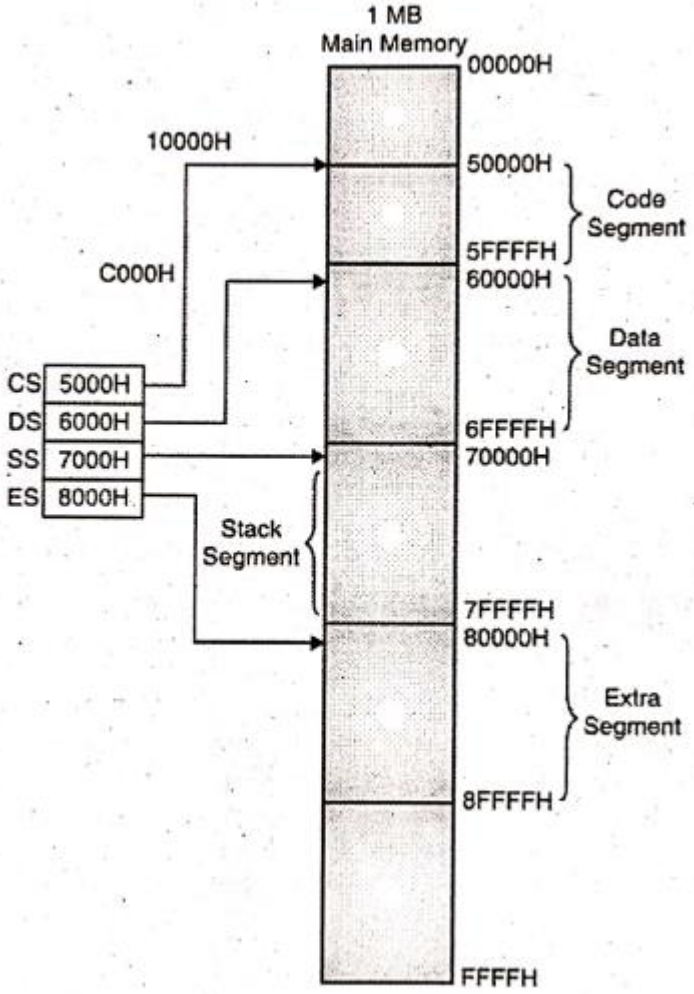


Fig. 2-Q. 6(a) : Segments in 8086

Q20) Draw timing diagram of memory read operation in minimum mode.

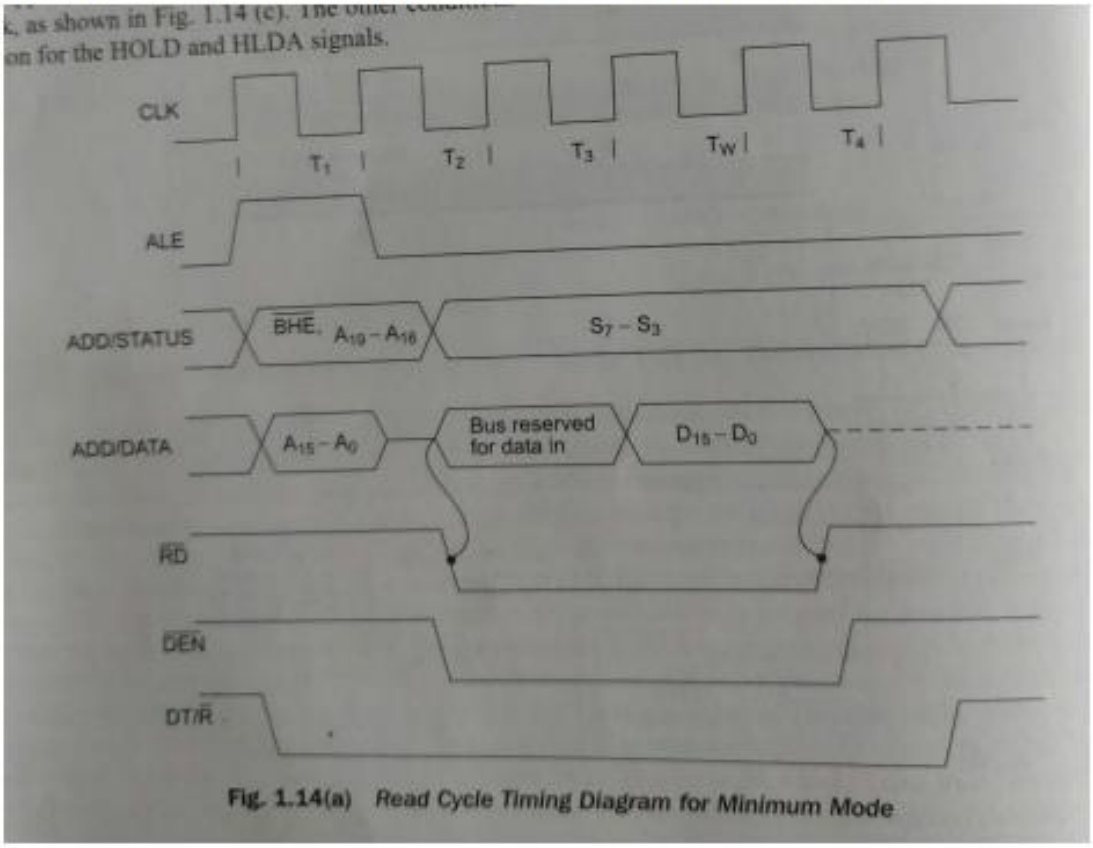
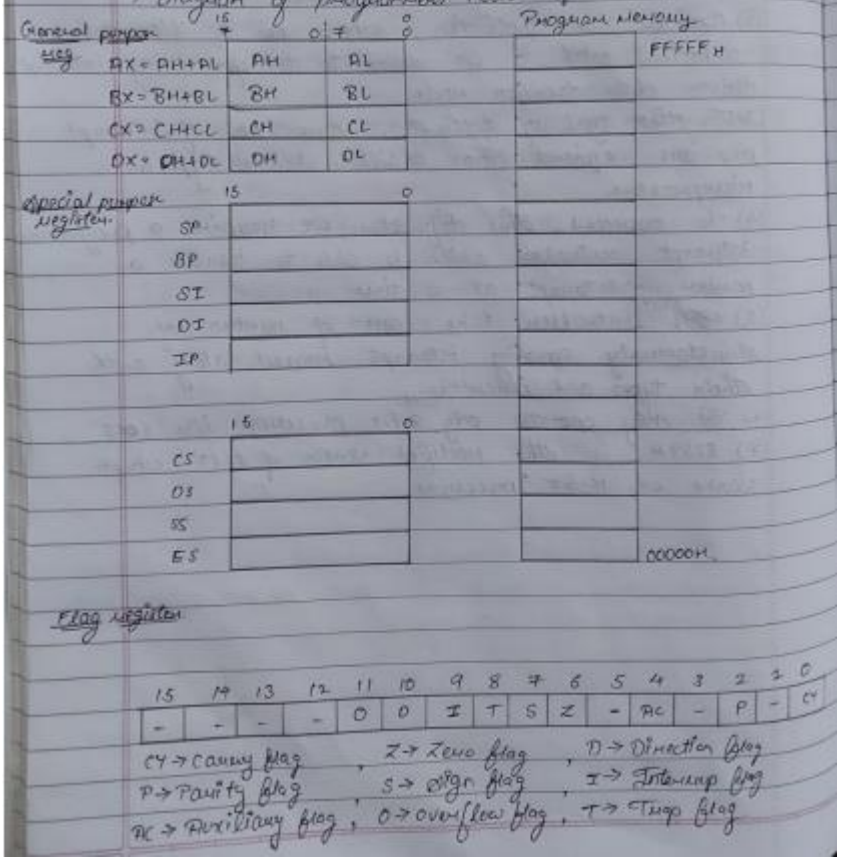


Fig. 1.14(a) *Read Cycle Timing Diagram for Minimum Mode*

Q] Explain the programming model of 8086

Sol: → The programming model for a µp shows the various internal registers that are accessible to the programmer.

→ Diagram of programming model of 8086



Flag register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	O	D	I	T	S	Z	-	AC	-	P	-	CY

CY → Carry flag, Z → Zero flag, D → Direction flag
P → Parity flag, S → Sign flag, I → Interrupt flag
AC → Auxiliary flag, O → overflow flag, T → Trap flag

In the programming model there are:
- 4 general purpose registers
- 4 segment registers
- 2 pointer registers
- 2 Index registers
- 1 instruction pointer register
- 1 flag register

→ General purpose registers
(1) AX register: This is accumulator. AX is used in arithmetic, logic and data transfer instructions.
(2) BX register: BX is a base register. BX usually contains data pointers used for based, based indexed or register indirect addressing.
(3) CX register: This is count register. Program loop constructions are facilitated by it.
(4) DX register: This is data register. Can be used as port number in I/O operations.

→ Segment registers:
(1) Code Segment: CS is used for addressing a memory location in the code segment of the memory.
(2) Data Segment: DS points to the data segment of the memory.
(3) Stack Segment: SS is used for addressing stack segment of the memory.
(4) Extra Segment: ES is another segment to the segment which essentially is another data segment of the memory.

→ Pointer registers
(1) SP Register: This is stack pointer register pointing to program stack.
(2) BP Register: This is Base pointer register pointing to data in stack segment.

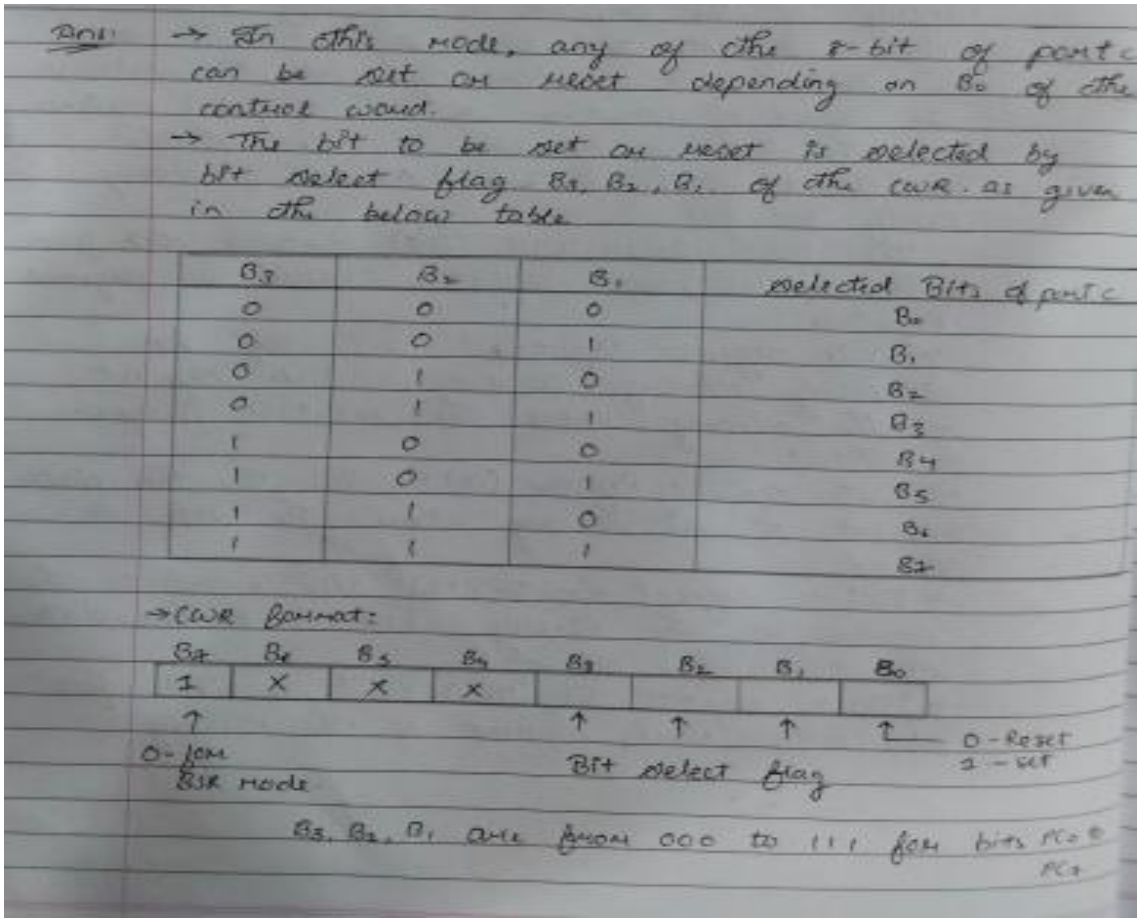
→ Index Register:
(1) SI Register (Source Index): This is used to point to memory locations in the data segment addressed by DS.
(2) DI Register (Destination Index): This register performs the same function as SI. There is a class of instructions called string instruction that use DI to access the memory locations addressed by ES.

→ Instruction pointer: The instruction pointer points to the next address of the next instruction to be executed.

→ Flag register: The 8086 flag register contents include the result of computation in the ALU.

Q.22 Explain BSR mode of 8255.

The **BSR mode** stands for **"Bit Set Reset Mode"**. The first bit, i.e. the Most Significant Bit (MSB) of the Control word decides the mode in which the 8255 IC will be. For the IC to be in the BSR mode, the MSB must be reset, i.e. it must be 0. The BSR mode works only for port C. In this mode, we can select any bit of the port C and then assign it any value: either 0 or 1.



Q.23 Same as Q.3

Q24) With neat diagram explain Net burst micro architecture of Pentium 4

14.6.2 Pentium 4 Architecture

Q. Explain Intel's Net Burst Micro architecture with neat schematic. Also highlight on hyper-pipeline concept and rapid execution engine. **(5 Marks)**

Q. Write short note on Intel's Net burst micro-architecture **(5 Marks)**

The basic blocks of the Pentium 4 processor are shown in Fig. 14.6.1.

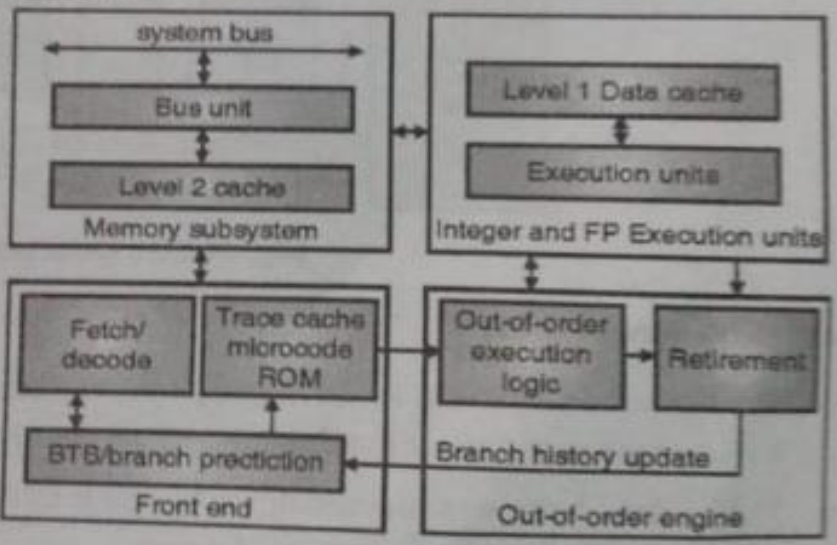


Fig. 14.6.1 : Basic Blocks of Pentium 4 processor

The basic blocks of Pentium 4, as shown in Fig. 14.6.1, consists of Memory subsystem, Front end, Out-of-order Engine and the Integer and Floating point execution units.

1. Memory Subsystem

- This unit handles the memory accesses and also consists of the L2 cache.
- The L2 cache is an Advanced Trace Cache as discussed in the features of the processor.

2. Front End

- This block consists of a Fetch / Decode unit, Trace cache along with microcode ROM and BTB along with branch prediction logic.
- The fetch decode unit fetches the instructions and decodes them.
- The trace cache, as discussed already, stores the decoded instructions and hence reduces the time for decoding in loops.
- The branch prediction logic predicts the sequence of instructions during a branch instruction with the help of branch history stored in Branch Target Buffer (BTB).

3. Out-of-order Engine

- The out-of-order engine executes the instructions out of order in case of their dependencies and hence keeps the execution units continuously busy.
- But the retirement of the instructions are in order.

4. Execution units

- There are integer execution units to execute integer instructions, floating point unit to floating point instructions and MMX unit to execute SIMD vector instructions. The detailed block diagram of Pentium 4 is shown in Fig. 14.6.2.

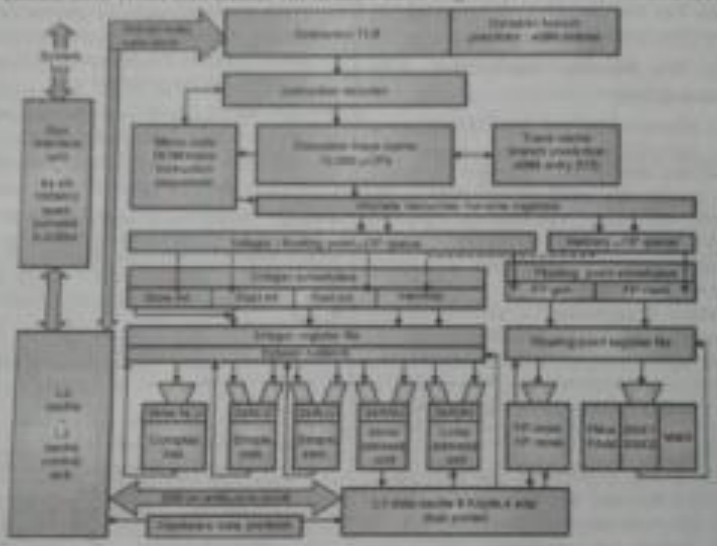
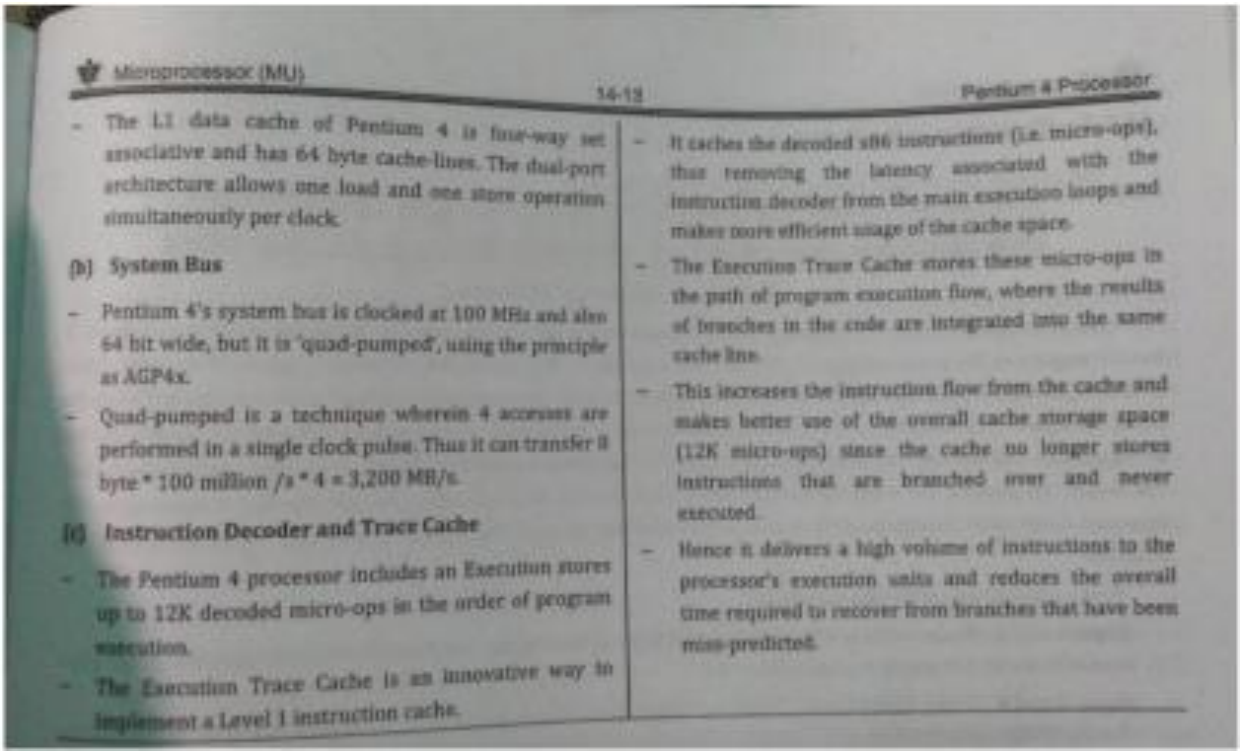


Fig. 14.6.2 : Detailed Block Diagram of Pentium 4

(a) Advanced L2 Cache

- The L2-cache also called as 'Advanced Transfer Cache' is 256 KB (or 512KB or 1 MB) and is 8-way associative. Pentium 4's L2-cache uses 128 byte cache lines, which are divided in two 64-byte pieces.
- The L1 data cache was reduced down to only 8 KB, to enable its extremely low latency of only 2 clock cycles and hence results in an overall improved read latency.



- The L1 data cache of Pentium 4 is four-way set associative and has 64 byte cache-lines. The dual-port architecture allows one load and one store operation simultaneously per clock.

(b) System Bus

- Pentium 4's system bus is clocked at 100 MHz and also 64 bit wide, but it is 'quad-pumped', using the principle as AGP4x.
- Quad-pumped is a technique wherein 4 accesses are performed in a single clock pulse. Thus it can transfer 8 byte * 100 million /s * 4 = 3,200 MB/s.

(c) Instruction Decoder and Trace Cache

- The Pentium 4 processor includes an Execution stores up to 12K decoded micro-ops in the order of program execution.
- The Execution Trace Cache is an innovative way to implement a Level 1 instruction cache.
- It caches the decoded x86 instructions (i.e. micro-ops), thus removing the latency associated with the instruction decoder from the main execution loops and makes more efficient usage of the cache space.
- The Execution Trace Cache stores these micro-ops in the path of program execution flow, where the results of branches in the code are integrated into the same cache line.
- This increases the instruction flow from the cache and makes better use of the overall cache storage space (12K micro-ops) since the cache no longer stores instructions that are branched over and never executed.
- Hence it delivers a high volume of instructions to the processor's execution units and reduces the overall time required to recover from branches that have been mis-predicted.

Q.25 Same as Q.5

Q27) Write an 8086 assembly language program to print content of flag register.

```
start:
 mov ax,Data
 mov DS,ax

 mov dx,offset msg
 mov ah,09h
 int 21h

 mov dx,offset newl
 mov ah,09h
 int 21h

 cli
 stc
 std

 pushf

 pop bx

 mov flag,bx

 mov cx,16
 mov bx,8000h

loops:
 mov ax,flag
 and ax,bx
 jz zero
 mov dl,31h
 mov ah,02h
 int 21h
 jmp space

zero: mov dl,30h
 mov ah,02h
 int 21h

space: mov dl,' '
 mov ah,02h
 int 21h

 mov ah,02h
 int 21h
 xor bx,1

 loop loops

 mov ah,4ch
 int 21h
Code ends
end start
```