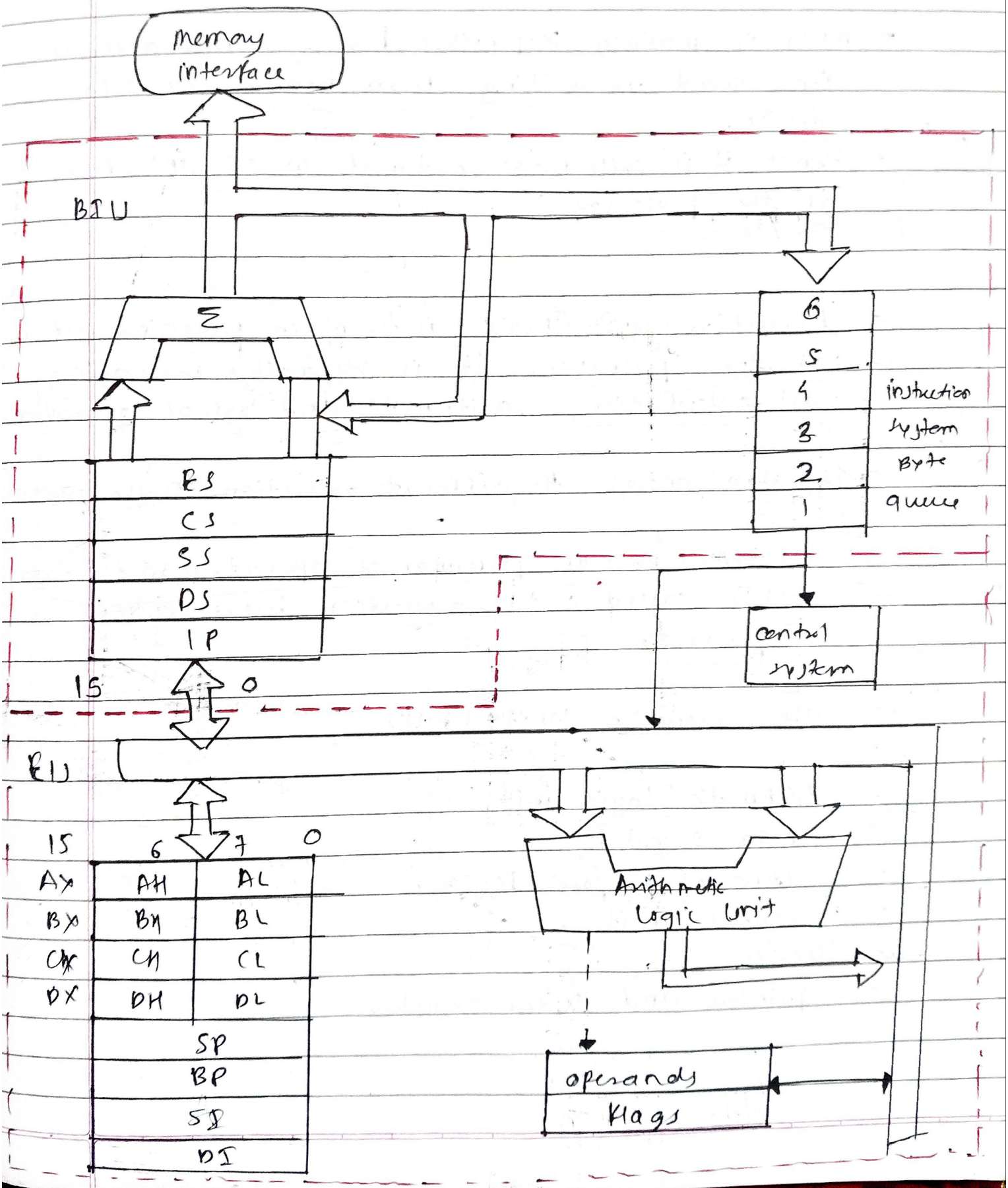
architecture

Q2 draw and explain 8086 microprocessor

Ans:



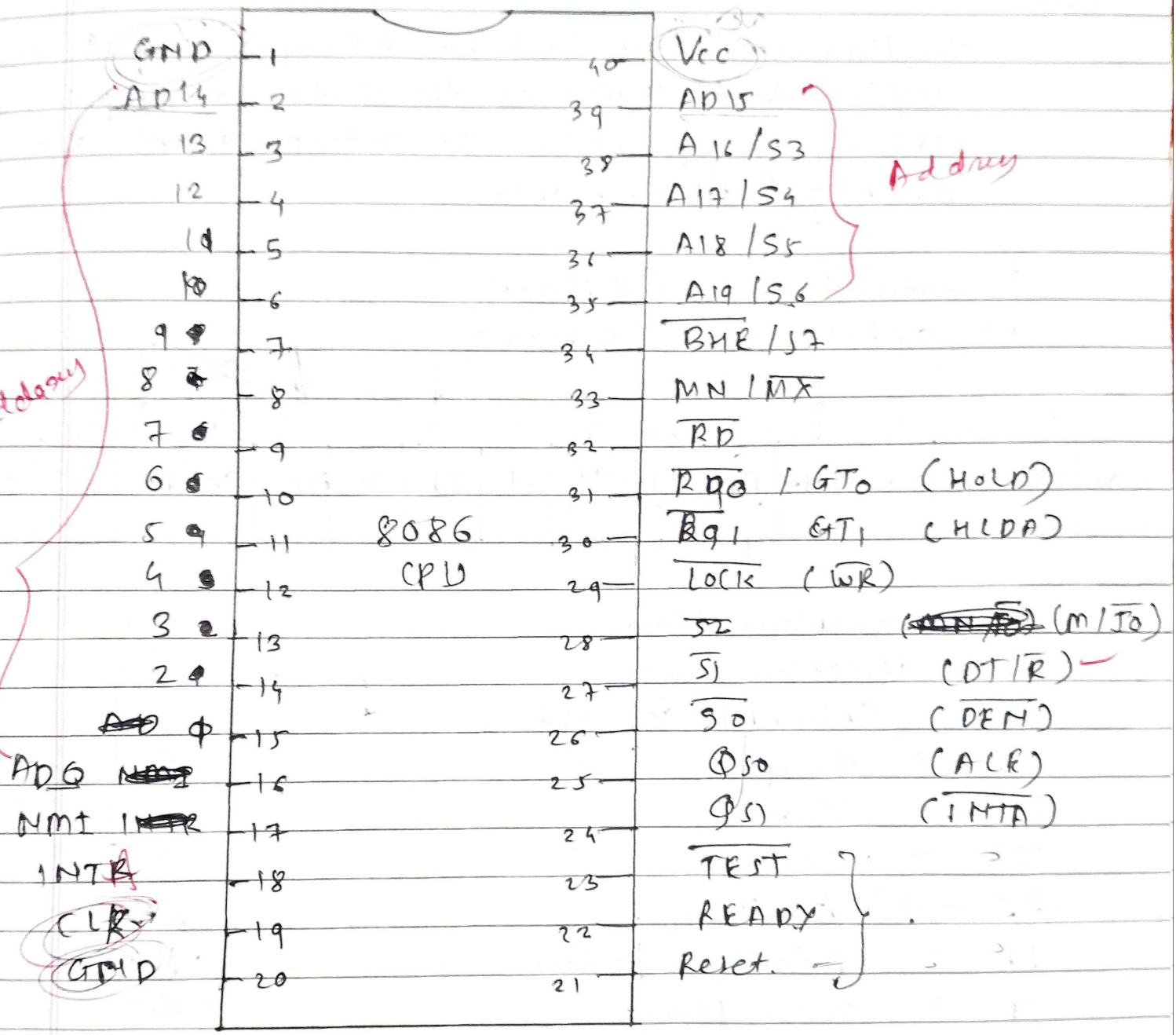
the 8086 divided into two section. namely
- The bus interfacing unit (BIU)
- The executing unit (EU)

BIU
- BIU is mainly responsible for external access i.e read or writing from memory or I/O devices.
- Hence it is called the external world interface of the processor.

EU
- Execution unit (EU) is the main processing section of the processor. it is responsible for doing all calculation, arithmetic and logical operation
- It also control the different operation in the processor
- EU takes care of performances operation on the data
- EU is called as the execution heart of the processor.

- The execution unit (EU)
  - Arithmetic logic unit
  - Flag register
  - General purpose Register
  - control unit
  - decoder
  - pointer and index register

- 8086 pin diagram



Q) explain pin configuration of 8086 MP

Ans: (draw neat slitty diagram here)

The 8086 MP has 40 pins and can operate in two modes

- minimum mode (single processor system)
- maximum mode (multi-1-1-1-1-1)

(1) Power and clock signals.

through diagram

- VCC - Pin. 40 → +5V power supply
- GND - 1, 20 → electrical ground
- CLK - 19 → provides clock signals to sychronize operations

(2) Address / Data bus

- AD0 - AD15 - ~~2-16, 39~~ 2-16, 39 → multiplexed for Address and data
- A16 - A19 / S3 - S6 - 35-38 → higher address bits and status signals.

(3) control & status signals.

- ALE — 25 → Add latch enable
- M/IO — 28 → Memory or I/O operations
- RD — 32 → Read signal
- WR — 29 → ~~write~~ " "
- BHE/S7 — 34 → Bus high enable,

(4) Interrupts & DMA signals.

NMI	17	→ Non maskable interrupt
INTA	27	→ Interrupt Aknowledge
HOLD	31	→ DMA rq.
HLDA	30	→ DMA ~~after~~ aknowledge

(5) Other signals

Rest	18	Reset Processor
Ready	22	Synchronize slow memory or I/O
DEN	25	data enable.
DT/R	24	data ~~transfer~~ transmit/Recieve
TEST	23	debugging
MN/MX	33	minimum(1), maximum(0) mode

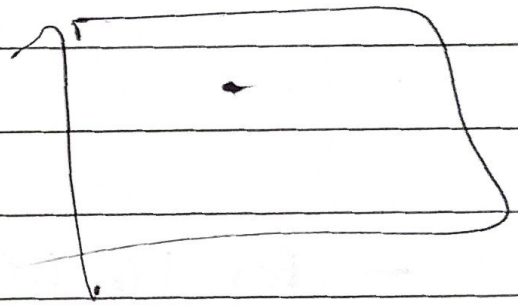
PAGE No.
DATE / /

* Minimum Mode

- ALE
- M/IO
- RD
- WR
- INTR

* Maximum Mode

- S2, S1, S0
- Locks
- RQ/GT1, RQ0/GT0

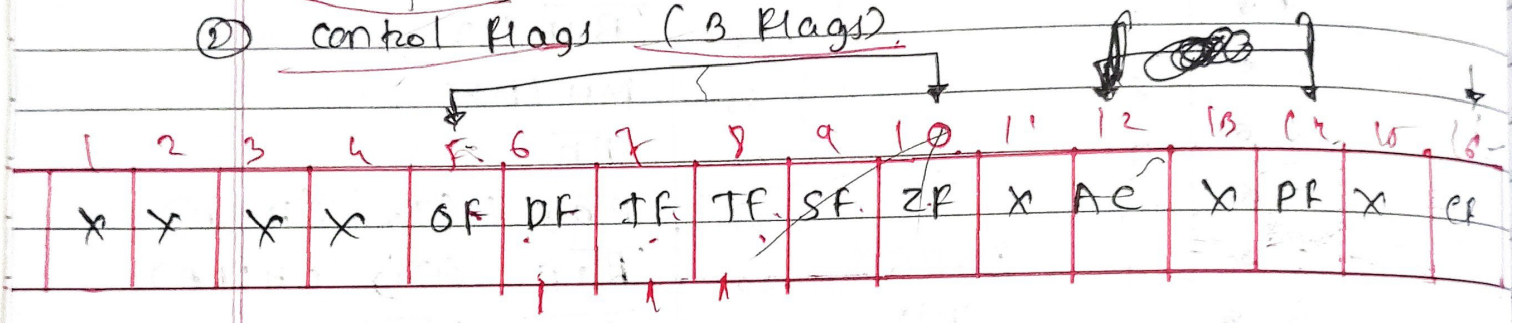


Q) explain Flag register of 8086 microprocessor (How many active flags)

Ans:

The Flag register in the 8086 microprocessor a 16-bit register that indicates the status of the processor after an operation. It contain 9 active flags which are divided into:

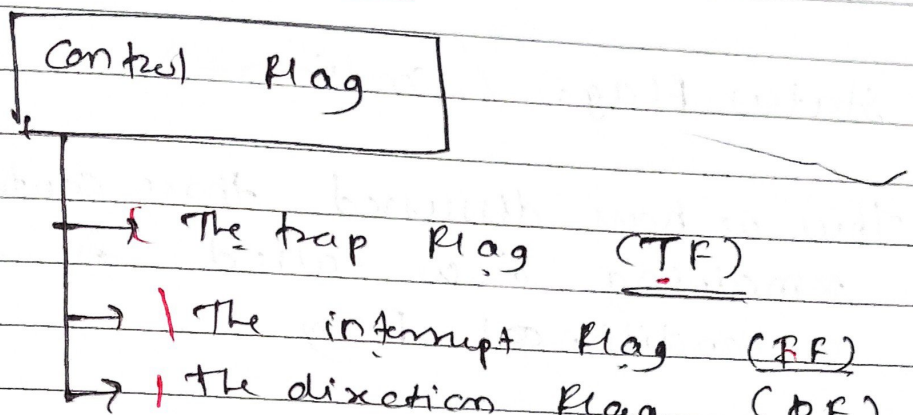
① Status Flag (6 Flags)
② control Flags (3 Flags)



OF → overflow Flag
DF → direction —1—
IF → Internal / interrupt
TF → Trap
SF → Sign
ZF → zero
AC → Auxillary
PF → ~~Priority~~ Parity
CF → carry Flag.

(1) control Flags: (3 Flags)

- As discussed, out of nine flags of 8086, three are control flags that are used to control certain operation of the processor.



example

① Trap Flag → used for debugging a program. to see execution this flag should be set to '1'.

This help the user to detect the error in the program or help programmer debug a program

TF = 1 → single step on, TF = 0 → off

② The interrupt Flag → used to maskable as well as non-maskable interrupts.

- used to enable or disable the maskable interrupt i.e INTR.

IF = 0 → disable

IF = 1 → enable

③ The direction Flag: These are special type of instruction that can works on an entire array of data.

PAGE No.
DATE / /

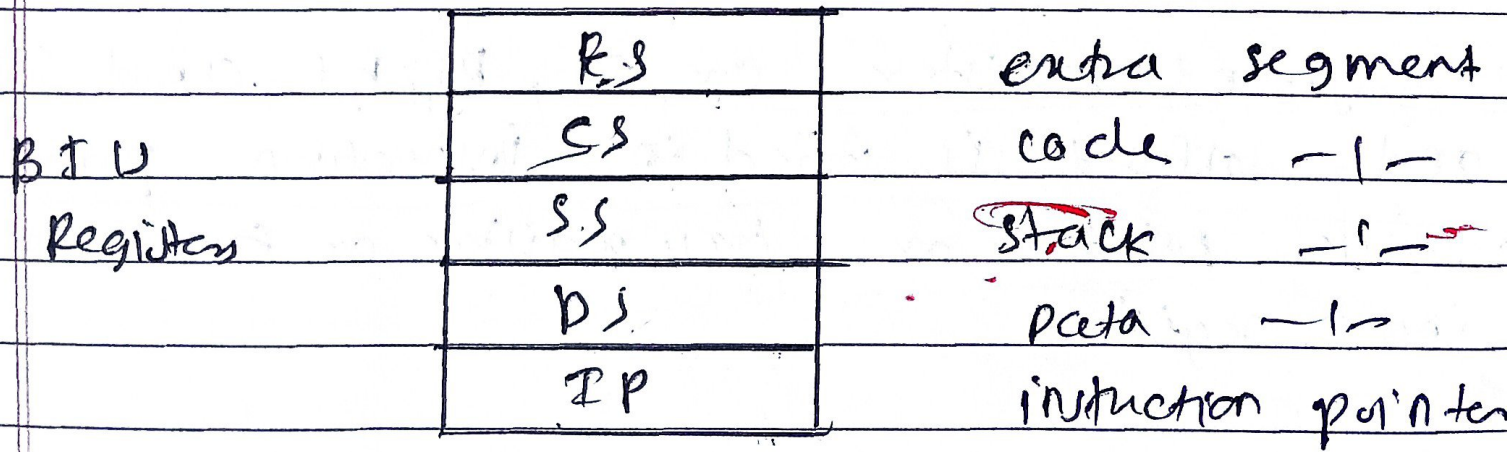
④ Zero → Set if the result is zero

⑤ Sign → Set if the result is negative (MSB = 1)

⑥ overflow → Set if the result overflows beyond the signed ranged.

Q Explain programmer model

Ans:



The 8086 programmer model is a picture of the processor as variable to the programmer. These are the registers used to hold numbers and address, as well as indicators, states and set controls.

- Data registers (16 bit each for AX, BX, CX and DX)

AX	AH	AL	Accumulator
BX	BH	BL	base register
CX	CH	CL	count —
DX	DH	DL	Data —
	SP		Stack pointer
	BP		Base —
	SI		source index Register
	DI		destination index —
	Flags		

BIU Registers

- For 16 bit data registers can be accessed by names AX, BX, CX and DX
- Individual bytes of these registers can be accessed by AH, AL, BH, CH, DH and DL
-
- The segment register are CS, DS, ES and SS and offset is stored in instruction pointer or stack pointer or base register or one of the index registers

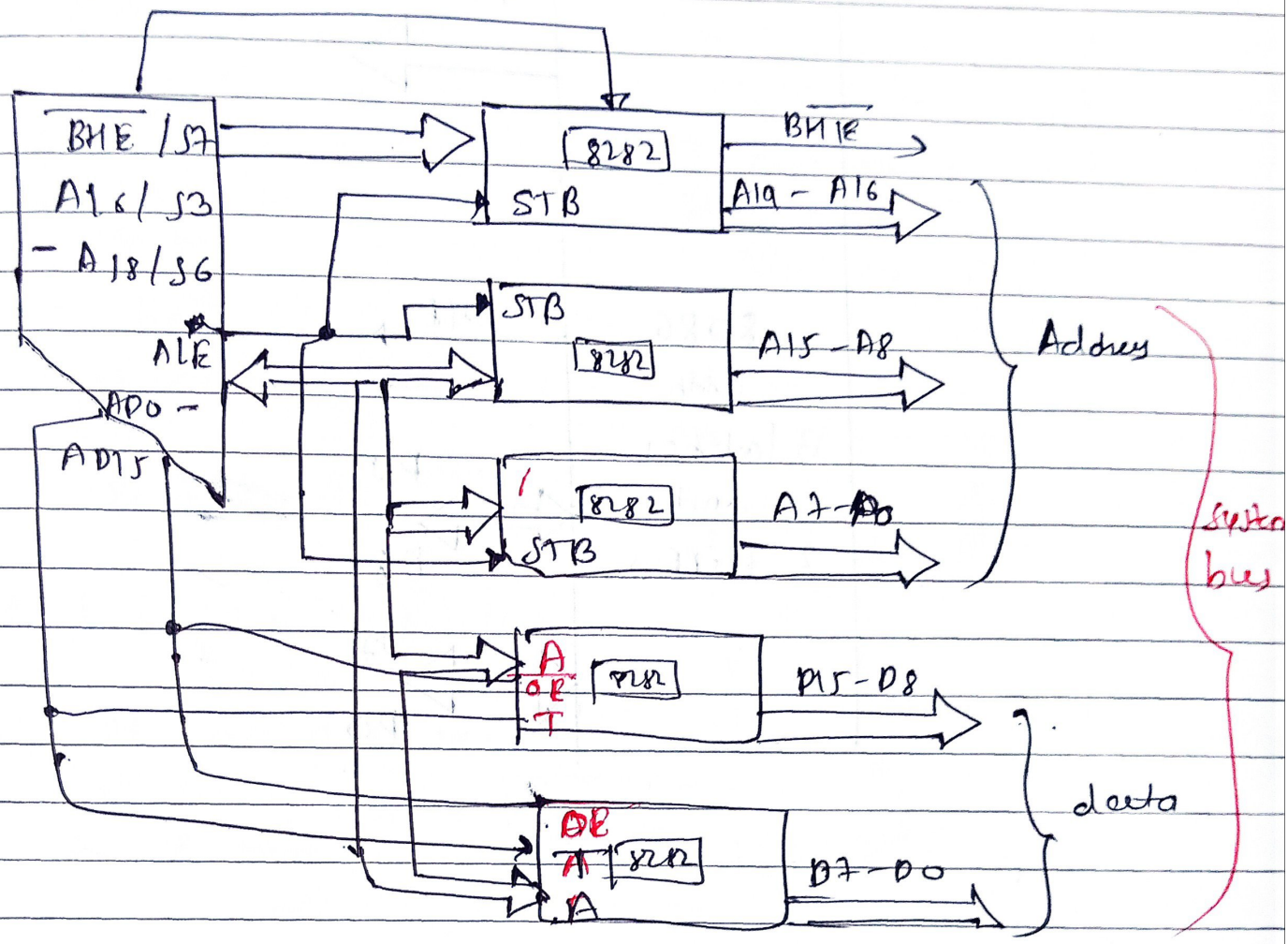
- Flag Registers (16-bit) draw diagram as previously I drew

(1) Status — CF, PF, AF, ZF, SF, OF

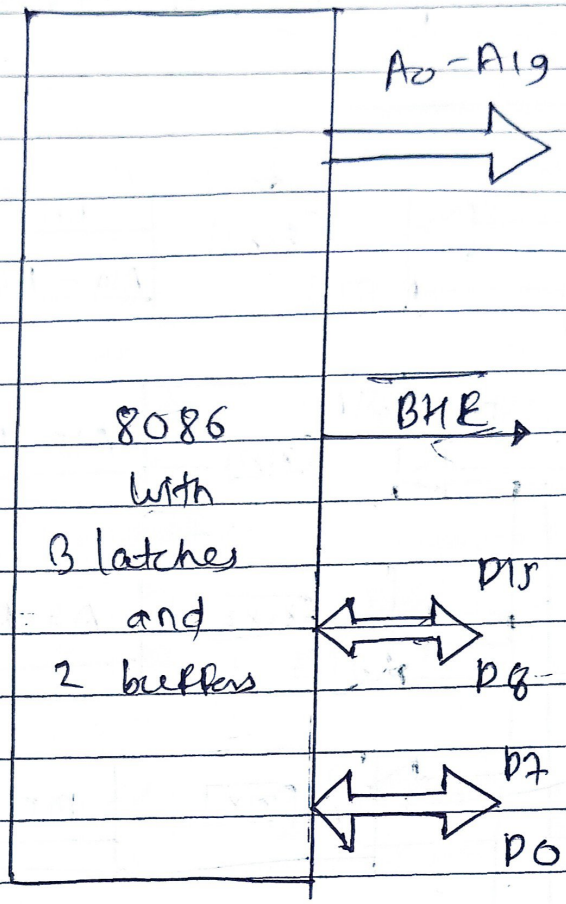
(2) control — TF, IF, DF

Q) explain demultiplexing of address and data signals.

Ans:



- We have discussed the use of 8282 latches in address latching and 8286 as data buffer
- If we use them simultaneously then we can demultiplex the address data bus as shown in Fig.
- We required three octal latch chips (OLC) and two octal transreceiver chips (OTC) in order to completely demultiplex the address and data bus.
- The simplified equivalent circuit shown below:



Q) explain features of 8086 MP

Ans:

(1) It is a 16-bit microprocessor

(2) It has 16-bit ALU so it can perform 16-bit operations simultaneously

(3) Similarly it has 16-bit registers & 16-bit internal and external buses

(4) The speed of processor depends on frequency of operation or clock speed.

PAGE No
DATE / /

(5) 8086 has 5 mega
8086 - 1 10 mega
8086 - 2 8 mega

(6) 8086 has 20 bit address line to access memory Hence it can access 1MB 2^{20}

(7) It has 16 bits address line to I/O devices so it can access 64 kb of I/O devices.

(8) It support 2 stage pipelining (fetch & execute)

(9) It has 14 16-bit registers

(10) It has 256 vector interrupt

(11) It use memory banking system (even & odd) total memory size 1MB

even bank	odd bank
512 kb	512 kb

(12) It has instruction set to perform many operations.

PAGE No.
DATE / /

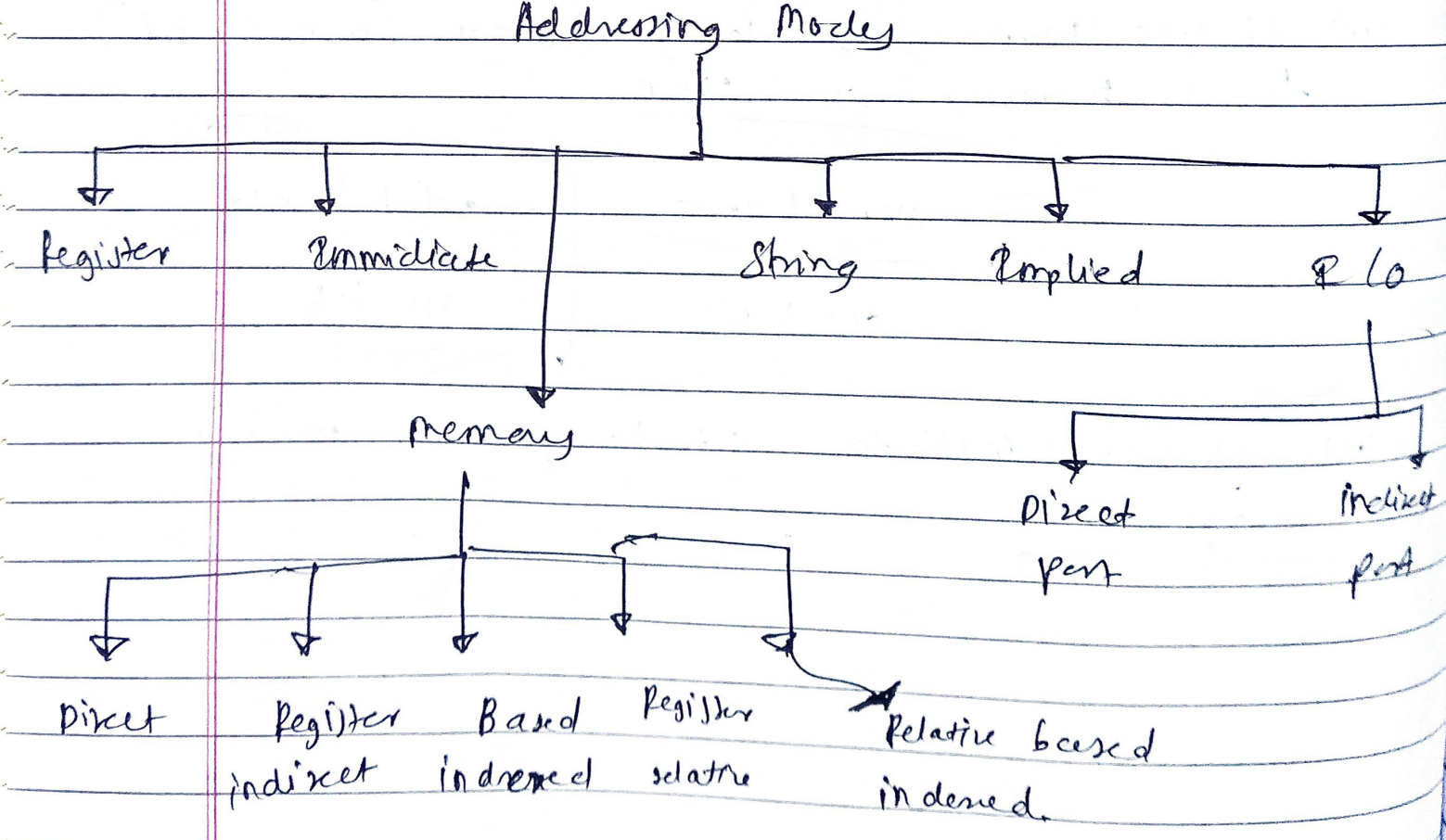
Q) explain addressing modes 8086 Microprocessor

Ans:

- Addressing modes (method) refer to the different method of addressing (selecting) the operands
- we can categorize addressing mode of 8086 as follows:

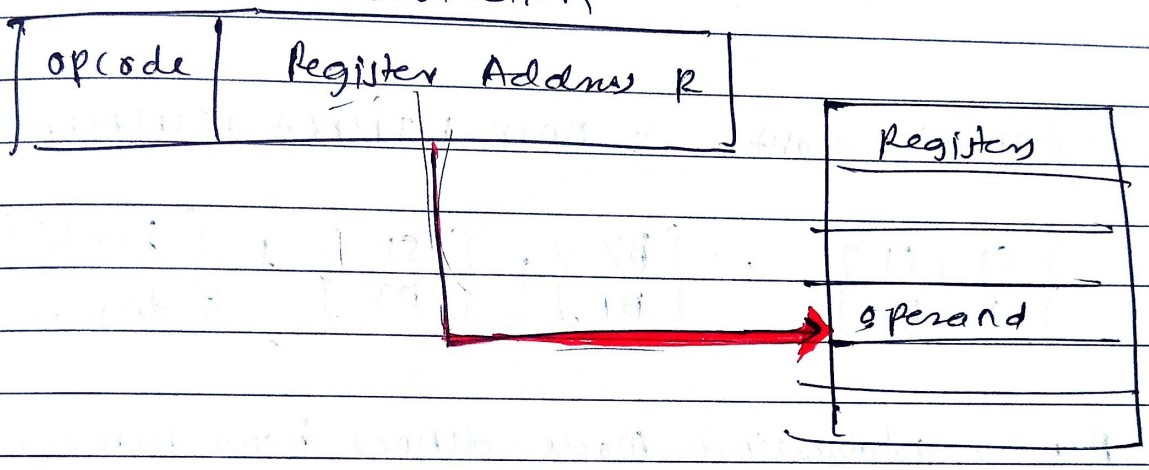
(1) Register
(2) Immediate
(3) memory
(4) String
(5) I/O
(6) Implied

} addressing modes



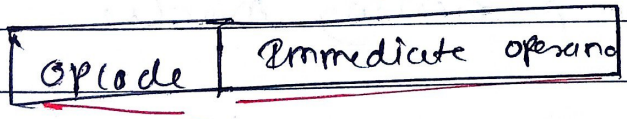
(1) Register Addressing Mode.

(1) In this mode of addressing operand is in the register, and instruction specifies the particular register as shown ~~below~~ below.



(2) The advantage of this addressing mode is that the access is faster.

(2) Immediate Operand Addressing Mode.



- in this case the operand is in the instruction itself.
- it said to be immediate addressing mode as the operand is in immediate next location of the OPCODE.

PAGE No.
DATE / /

(3) Memory Addressing Mode.

PA = Segment : Offset

Segment → Segment Register

Offset → EA

Segment register : BASE + INDEX + DISPLACEMENT

$$\left\{\begin{matrix}CS \\ SS \\ DS \\ ES\end{matrix}\right\} : \left\{\begin{matrix}BX \\ BP\end{matrix}\right\} + \left\{\begin{matrix}SI \\ DI\end{matrix}\right\} + \left\{\begin{matrix}8 \text{ or } 16 \text{ bit} \\ \text{displacements}\end{matrix}\right\}$$

① Memory addressing mode defines how memory locations are accessed in the 8086 MP

② These modes determine how the effective address (EA) of an operand is calculated

Segment register : Base + index + displacement

Types of memory Addressing modes

1. direct
2. Register indirect
3. Based indexed
4. Register relative
5. Relative based indexed.

4) String Addressing mode:

- String instruction use a different addressing mode where the pointers SI and DI along with segments register DS and ES, respectively are used to access the source and destination memory location

- These pointers are also automatically incremented or decremented according to the value of direction flag (DF)

5) I/O addressing modes

- We have (1) memory mapped I/O
(2) I/O mapped I/O

* memory mapped I/O

- As the name says, memory mapped I/O refers to an I/O location mapped in memory i.e given a memory address

* advantage of this that may instruction can access this data directly and hence giving a ease of access

- disadvantage: of such I/O location is that the access for I/O location being normally slower it makes the processors to wait.
→ make it slower or process to wait

PAGE NO.
DATE / /

* I/O mapped I/O

- in this I/O mapped IO support two different addressing modes

(1) Direct port addressing
(2) indirect port -"-"-"-

(1) direct port addressing: As the name says, direct port addressing in this case the address of I/O devices is given in instruction itself.

(2) indirect port addressing: in this case a pointer register i.e the register DX is used to give the address of the I/O location.

(3) Implied Addressing Modes:

in this case the operand or references to operand is not specified in the instruction. instead the operand is obvious in the memonic or the instruction

eg: XLAT
CMA
STC
STD

(3) Ready: The Ready signal comes 8086 processor waits for slow peripherals

(4) Synchronization (SYNC)

CSYNC & SYNC signals allow multiple 8086 MP to run synchronously in a system.

IMP

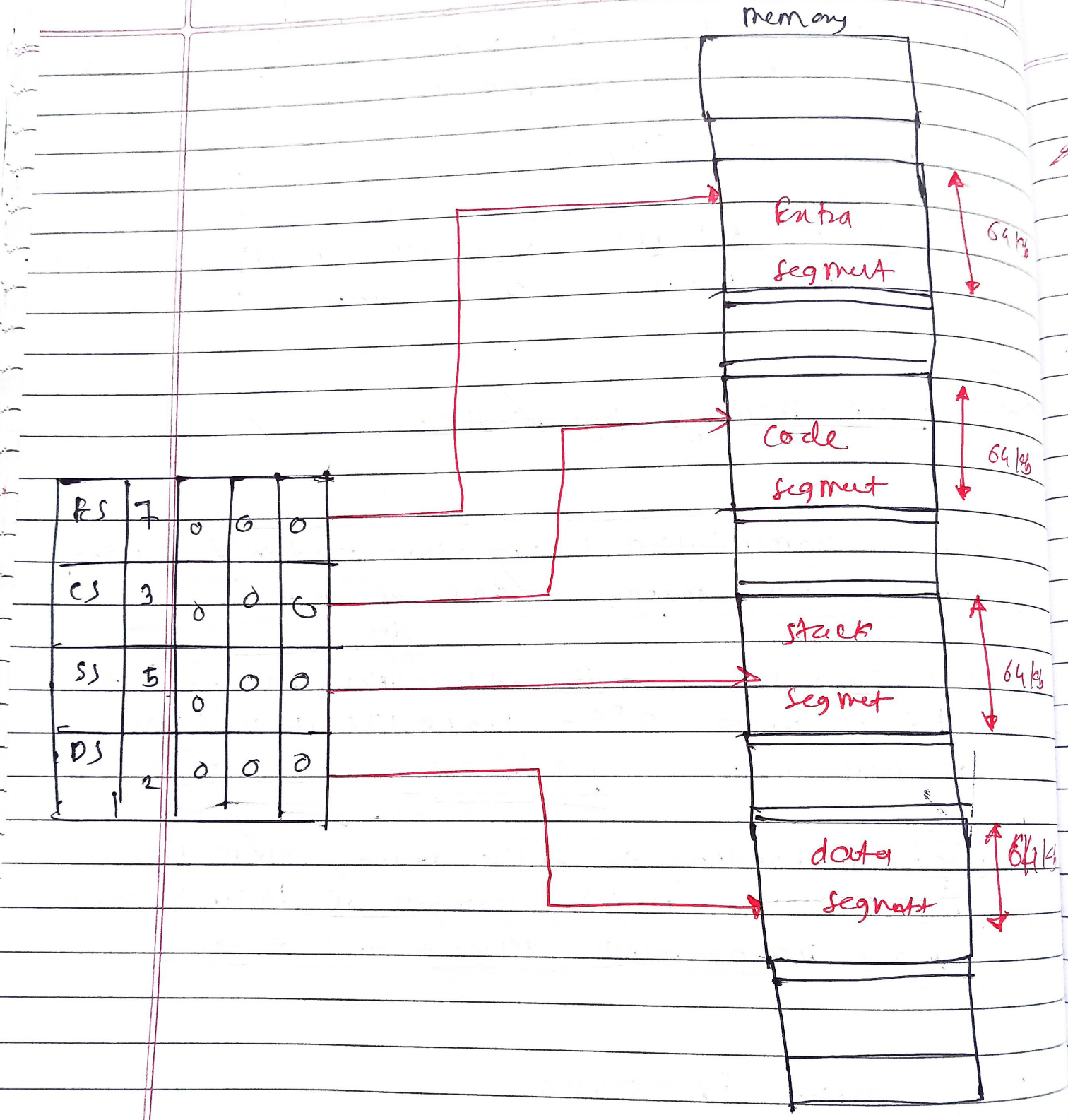
Q) explain memory segmentation:

Ans:

- Memory segmentation is a technique used in 8086 microprocessor to efficiently manage 1MB of memory.
- The 8086 has 20-bit address bus (line), allowing it to address up to 2^{20} = 1,048,576 bytes (1MB) of memory (put in calc)
- Instead of handling the entire ~~memory~~ 1MB linearly, 8086 devides it into segments of 64KB to make memory management different and structured.

- Segmentation in 8086

Segment Register	Segment Name	Purpose
CS	code segment	Store executable programs
DS	data -1-	variable and constants
SS	Stack -1-	Store stack data
ES	Extra -1-	additional data storage or string operation


Memory
Extra segment
64 kB
Code segment
64 kB
Stack Segment
64 kB
data segment
64 kB
ES 7 0 0 0
CS 3 0 0 0
SS 5 0 0 0
DS 1 2 0 0 0

PAGE No.
DATE / /

* Advantage

- The most imp advantage is that the programmer can access a memory that required 20-bit address by using 16-bit register only
- The programs, data, and stack are stored in seperate blocks in memory and hence these are organized in a modular fashion
- It also help in OOP to store data of an object
- efficient memory usages
- faster execution
- support multi-tasking

* disadvantage

- It becomes complicated for the programmer as multiple register to access a memory location
- limmited segment size
- Complex Addressing calculation

PAGE No.
DATE / /

Q) explain following instruction of 8086 mp

(1) complement Carry — CMC

- function: complements (toggle) the carry flag

if CF = 1 → CF = 0
if CF = 0 → CF = 1

Syntax: CMC

Affected flags: CF only

Use case: Used in arithmetic or logic operations where carry needs to be inverted manually.

(2) XCHG — Exchange Register / Memory with Register

- function: Swaps the content of two registers or a register and a memory location

Syntax: XCHG reg1, reg2
XCHG reg, mem

Affected flags: None

Use case: Used to exchange values without using a temporary register.

PAGE No.
DATE / /

(3) MOVSB – Move string byte

Function: Moves a bytes from [DS : SI] to [ES : DI]

Syntax: MOVSB

Affected flag: None (but direction of movement controlled by DF (direction flag))

Use case: Used for copying a string of bytes from source to destination.

Copy

PAGE No.
DATE / /

(1) LAHF (Load AH from Flags)

- Function: Copies the lower bytes (Status Flags) of the Flags Register into the AH Register
- Syntax: LAHF
- Affected Flags: None
- Use case: Used to store the status flags before modifying them or for debugging purpose

(2) XLAT (Translate byte)

- Function: Uses the AL Register as index to a lookup table located at DS:BX and replaces AL with the corresponding byte from the table. (table lookup)
- Syntax: XLAT or XLATB
- Affected Flags: None
- Use cases: Used for the lookups, such as character translations, or encryptions encryptions

(3) DAA (Decimal Adjust After Addition)

- Function: Adjust the AL Register after a BCD addition to ensure a correct BCD result

Syntax: DAA

Affected Flags: AF, CF

Use case: Used in BCD arithmetic operations

(4) JNE (Jump if Not equal)

- Function: Jumps to a specific address if zero flag (ZF) = 0
meaning is the two compared values are not equal

- Syntax: JNE

Affected Flags: None

Use case: Used in conditional branching to execute instruction based on inequality

(5) AAA (ASCII Adjust After Addition)

Function : Adjust the result in AL adding two ASCII coded decimal digits to ensure a valid ASCII decimal result

Syntax : AAA

Affected Flags : AF CF

Use case : Used in ASCII arithmetic to correct invalid result.

(4) STOSB (Store String Byte) :

Function : Stores the value in AL register into memory location pointed by DI

Syntax : STOSB

Affected Flags : None

Use Case : Used for storing strings or repeating bytes value in memory.

DATE / /

Q) dedicated interrupts in 8086 MP

→ In 8086 there are 5 dedicated interrupts that have fixed interrupt vector address and are used for specific purpose.

(1) divide by zero interrupts

- Type : 0

Trigger condn : division by zero or overflow in DIV / IDIV

Vector Address : 00000H

Use case : Prevents invalid division

(2) Single Step

Type : 1

Trigger condn : Trap flags = 1

Vector Address = 00004H

Use case : debugging (step-by-step execution)

(3) Non-Maskable Interrupts (NMI)

type: 2

trigger condn: critical hardware error

Vector Address: 0008H

Use case: Power Failure, Memory error

(4) break points.

type: 3

trigger condn: INT 3 instruction executed

Vector Address: 000CH

use case: debugging, setting breakpoints

(5) overflow (arithmetic overflow)

type: 4

trigger condn: INTO instruction and OF=1

Vector Add = 0010H

Use case: Handles arithmetic overflow

ALP

PAGE No.
DATE / /

Q) Write ALP for 8086 to exchange content of two memory blocks

Ans:

```
.model small
.stack 100h
.code

main :
    mov ax, @data
    mov ds, ax          ;   Set DS segment

    mov si, 1100h ;         block 1
    mov di , 1200h ;        block 2
    mov cx, 10    ;         no. of byte to exchange

next :
    mov al, [si] ;          get byte from block 1
    mov bl, [di] ;          —— 1 —— block 2

    mov [si], bl;           block 2 into block 1
    mov [di], al;           block 1 into block 2

    inc si;
    inc di;
    loop next ;             Repeat till cx = 0

    mov ah, 4ch ;
    int 21h

end main
```

Q) Write ALP for searching a character in given string (Consider your own string)

Ans:

string : "HELLO"
search : 'L'

```
.model small
.stack 100h
.data

    str db 'HELLO';
    ch db 'L';
    flag db 0;          0 = not found , 1 = found

.code
main:
        mov ax, @data;
        mov ds, ax

        lea si, str
        mov cx, 5

next:
        mov al, [si];
        cmp al, ch
        je found

        inc si
        loop next
        jmp end
        hlt
```

```
found :
        mov flag, 1

end :
        mov ah, 4ch
        int 21h
end main
```

Q) Write ALP for 8086 to reverse a string of 10 characters

Ans:

```
.model small
.stack 100h
.data
        str db 'ABCDEFGHIJ'
        len db 10

.code
main:
        mov ax, @data
        mov ds, ax

        lea si, str
        lea di, str+9
        mov cx, 5

reverse_loop:
        mov al, [si]
        mov bl, [di]
        mov [si], bl
        mov [di], al
```

```
inc si.
dec di
loop xchange_loop

; exit program
mov ah, 4ch
int 21h
end main
```

Q) write ALP for 8086 to transfer block of data

Ans:

```
.model small
.stack 100h
.data
    source db 10h, 20h, 30h, 40h, 50h, 60, 70h, 80h,
              90h, Ah

    dest db 10 dup(?)

.code
main:
    mov ax, @data
    mov ds, ax

    lea si, source
    lea di, dest
    mov cx, 10
```

```
transfer_loop:
        mov al, [si]
        mov [di], al
        inc si
        inc di
        loop transfer_loop

    ; Exit program
        mov ah, 4ch
        int 21h
    end main
```

(9) Write ALP for 8086 to arrange 10 numbers in ascending order.

Ans:

```
.model small
.stack 100h
.data
        numbers db 10, 2, 30, 5, 17, 40, 9, 7, 60, 11 ;

.code
main :
        mov ax, @data
        mov ds, ax

        lea si, numbers
        mov cx, 9

outer_loop :
        lea di, numbers
        mov bx, cx
```

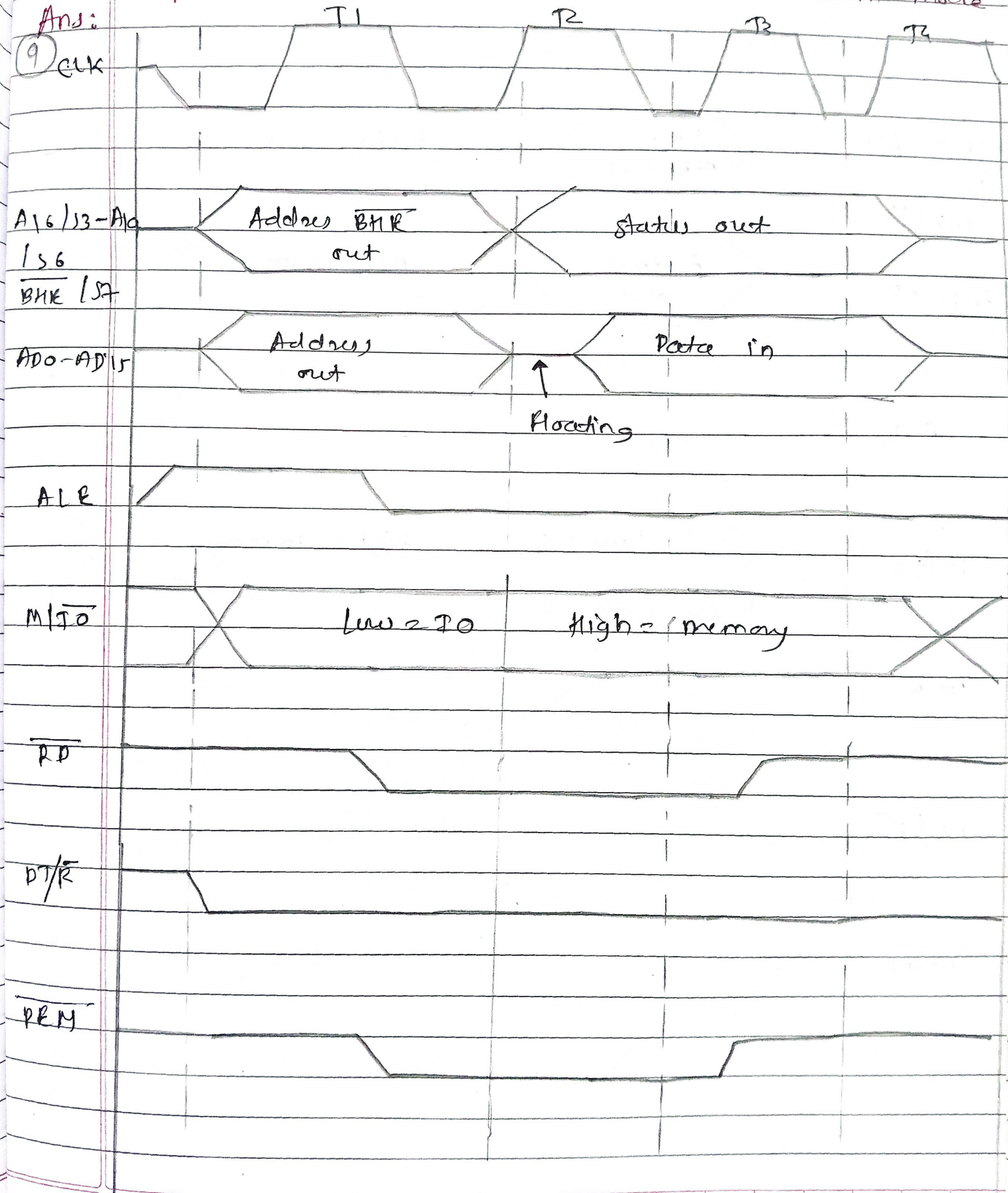
```
inner_loop:
        mov al, [di]
        mov ah, [di+1]
        cmp al, ah
        jbe no_swap

        xchg al, ah
        mov [di], al
        mov [di+1], ah

no_swap:
        inc di
        dec bx
        jnz inner_loop

        dec cx
        jnz outer_loop

        mov ah, 4ch
        int 21h
end main
```

Minimum Mode

PAGE No.
DATE / /

(9) Draw and discuss timming diagram for read operation and write operation in minimum mode

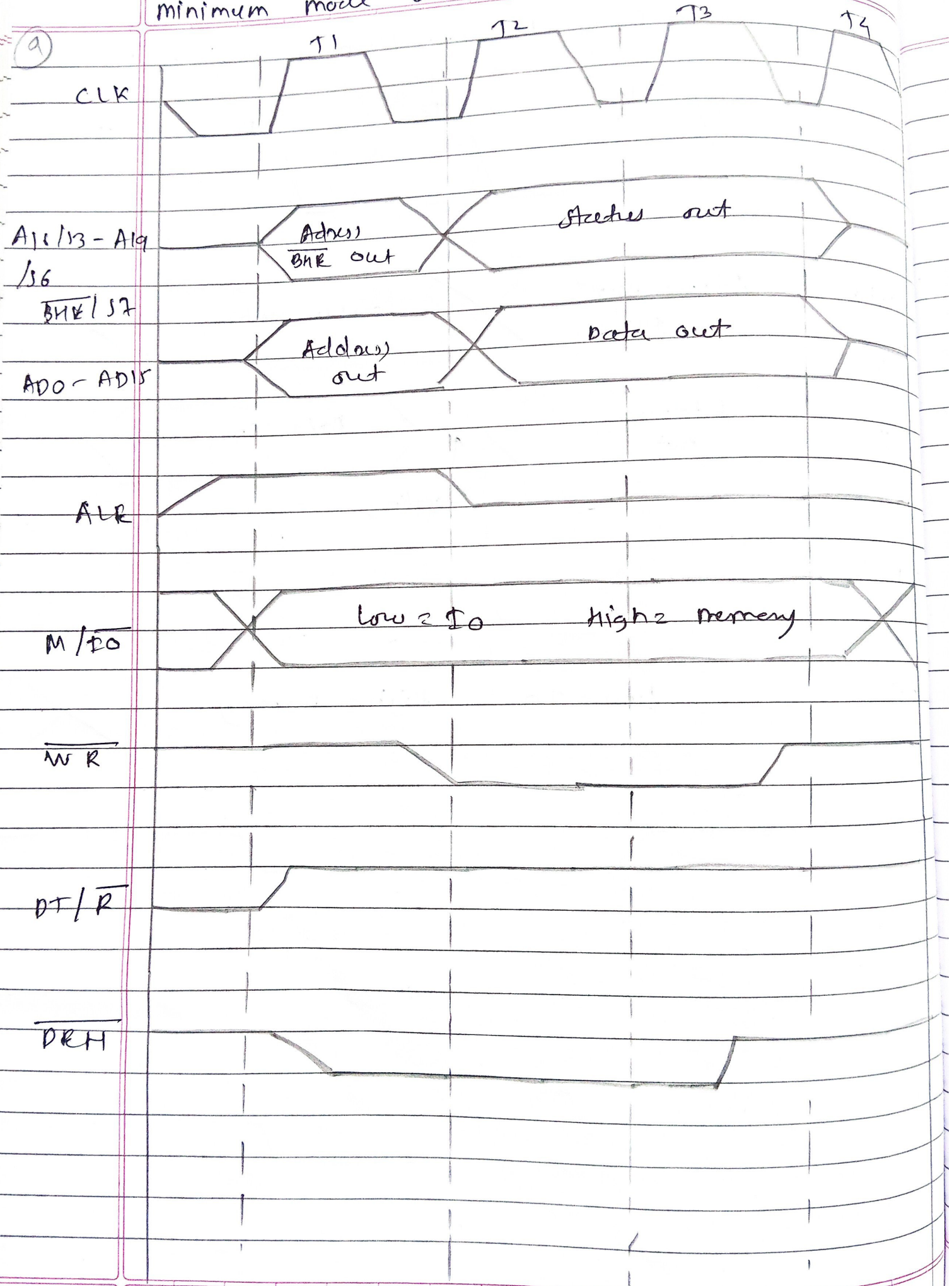
Ans:



Minimum mode read cycle

minimum mode write cycle

(9)



☆ Minimum mode read cycle:

This cycle is used when the 8086 reads data from memory or an I/O device

(1) T_1 State:
- Address placed on AD15 - AD0
- ALE (Address Latch Enable) is High
- M/$\overline{IO}$ is set
  1 = memory Read
  0 = I/O Read

(2) T_2 State:
- Address bus now used for data (bus is Floated)
- $\overline{RD}$ becomes active low - indicate 'read operation'
- DT/$\overline{R}$ is High - Processor is receiving data
- DEN is enabled - Data bus drivers are active

(3) T_3 State:
- valid data is available on data bus D15 - D0
- data is ready by the processor

(4) T_4 State:
- $\overline{RD}$, DEN signal go High - ending the cycle
- Bus is idle now

PAGE No.
DATE / /

☆ Minimum mode write cycle:

This cycle is used when the 8086 writes data to memory or I/O

(1) T_1 State:
Address placed on AD15 - AD0
ALE is High.
M/$\overline{IO}$
1 = memory write
0 = I/O write

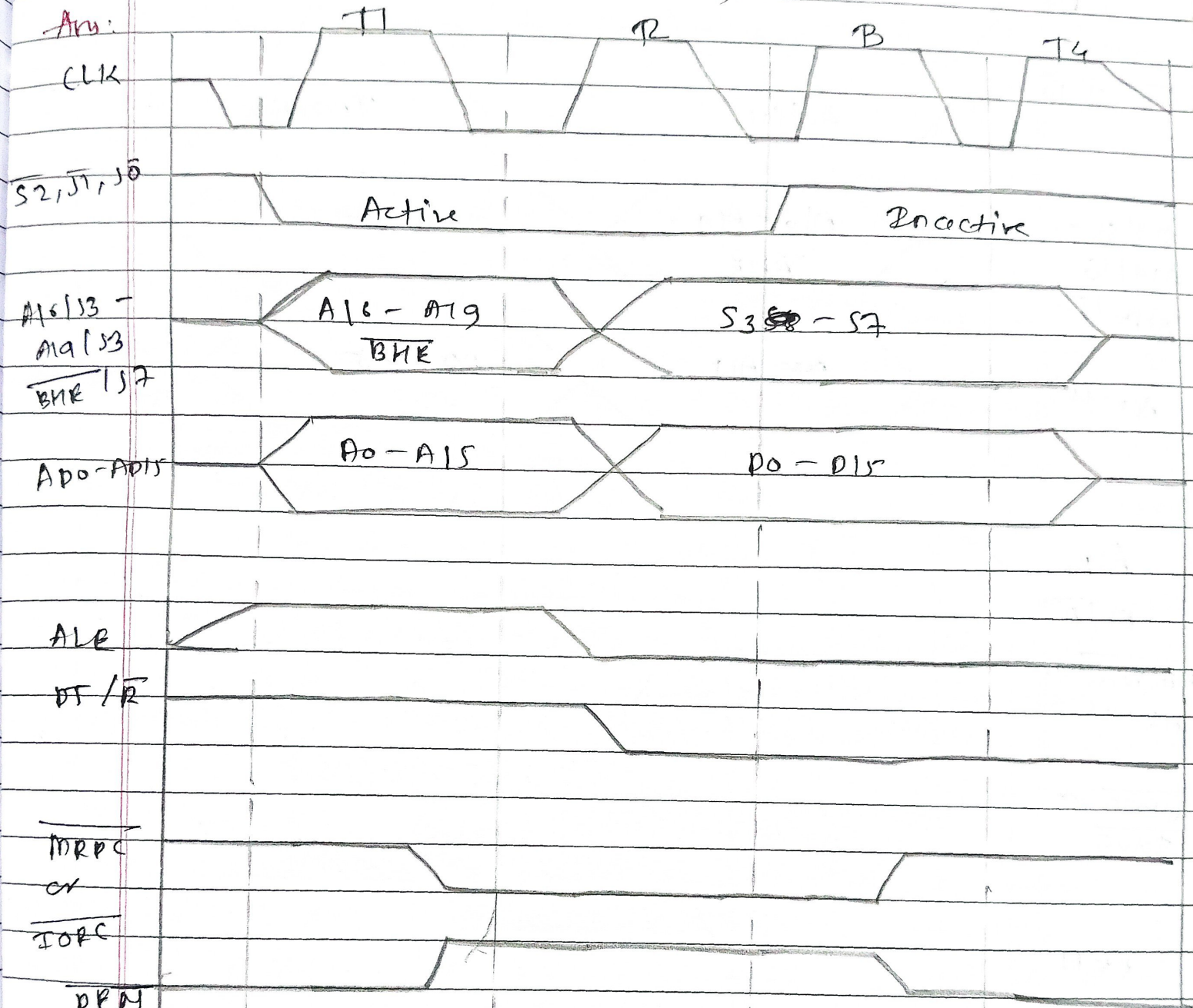
(2) T_2 State:
Address is latched, bus switches to data
$\overline{DT/R}$ is Low → processor is sending data
DEN is enabled → Bus drivers ON
$\overline{WR}$ goes Low → indicate write operation

(3) T_3 State:
Data is placed on data bus D15 - D0
data is written to memory or I/O device

(4) T_4 State:
$\overline{WR}$, DEN go High - write complete
Bus is now idle.

(8) (9) draw the timming diagram for read and write operation in Maximum mode.

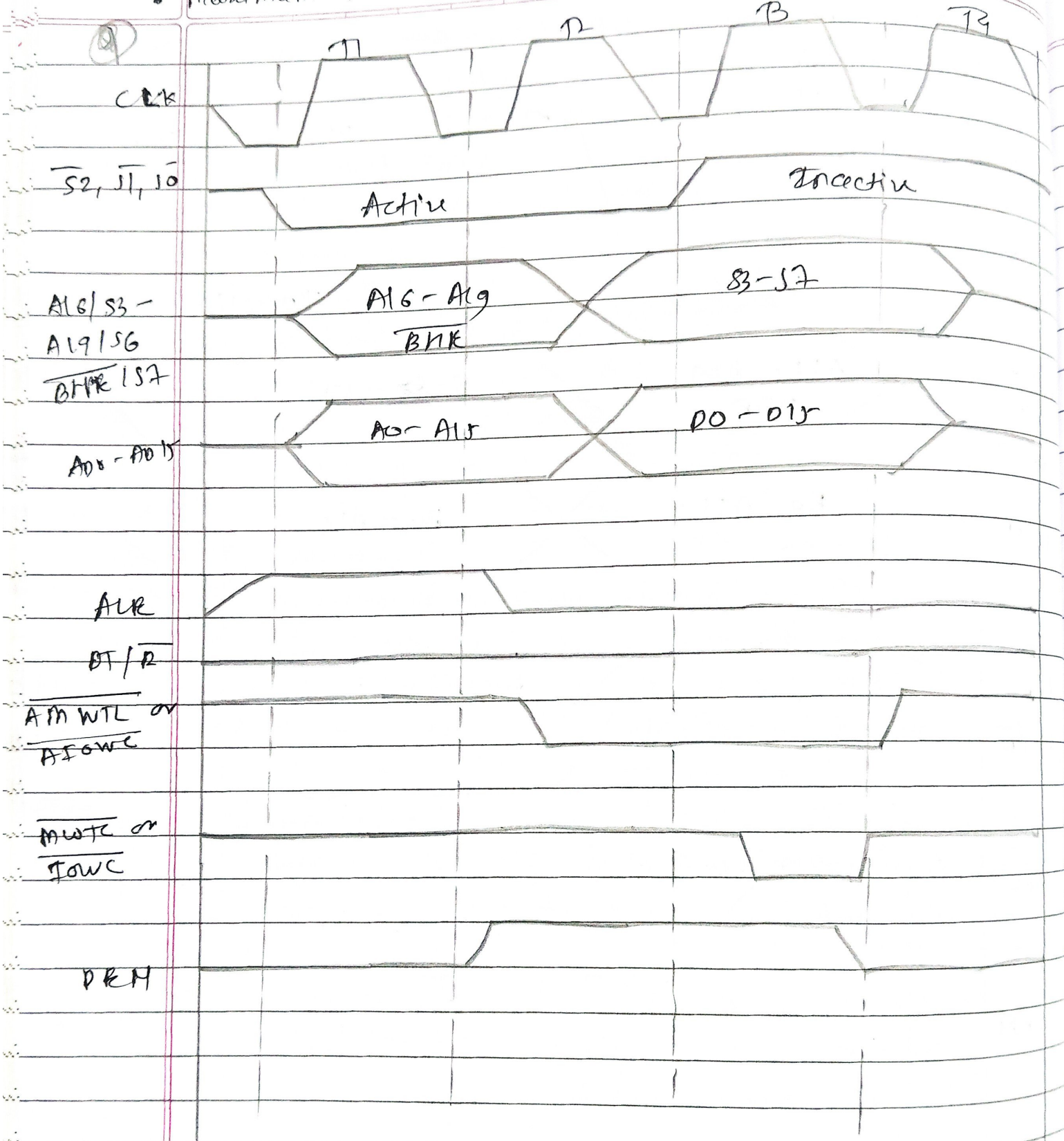
Ans:



Maximum mode read cycle

PAGE No.
DATE / /

- Maximum mode write cycle



PAGE No.
DATE / /

★ Maximum mode Read cycle:

Used when 8086 read data from memory or I/O in maximum mode.

(1) T1 (Clock cycle):
- Address line (AD15-AD0) carry address
- ALE goes High
- Status line S_0, S_1, S_2 are set
- Address is active

(2) T2:
- 8288 Bus controller use Status lines to generate $\overline{MRDC}$ (Memory Read) or $\overline{IORC}$ (I/O Read)
- $\overline{DT/R}$ is High - receiving data
- DEN (Data enable) becomes active
- Address is now latched and bus is used for data

(3) T3:
Valid data is placed on data bus by memory or I/O devices 8086 read data.

(4) T4:
$\overline{MRDC}$ / $\overline{IORC}$ and DEN go inactive
data bus is released.

PAGE No.
DATE / /

★ Maximum mode write cycle:

Used when 8086 writes data to memory or I/O in Maximum mode

(1) T1 :
Address is placed on AD15-AD0
ALE is high
S0, S1, S2 indicates type of write
Address lines are active.

(2) T2 :
- 8288 Bus controller generates:
MWTC (memory write command)
IOWC (I/O write —|—|—)
- DT/R is low — sending data
- DEN becomes active
- Data is placed on the data bus

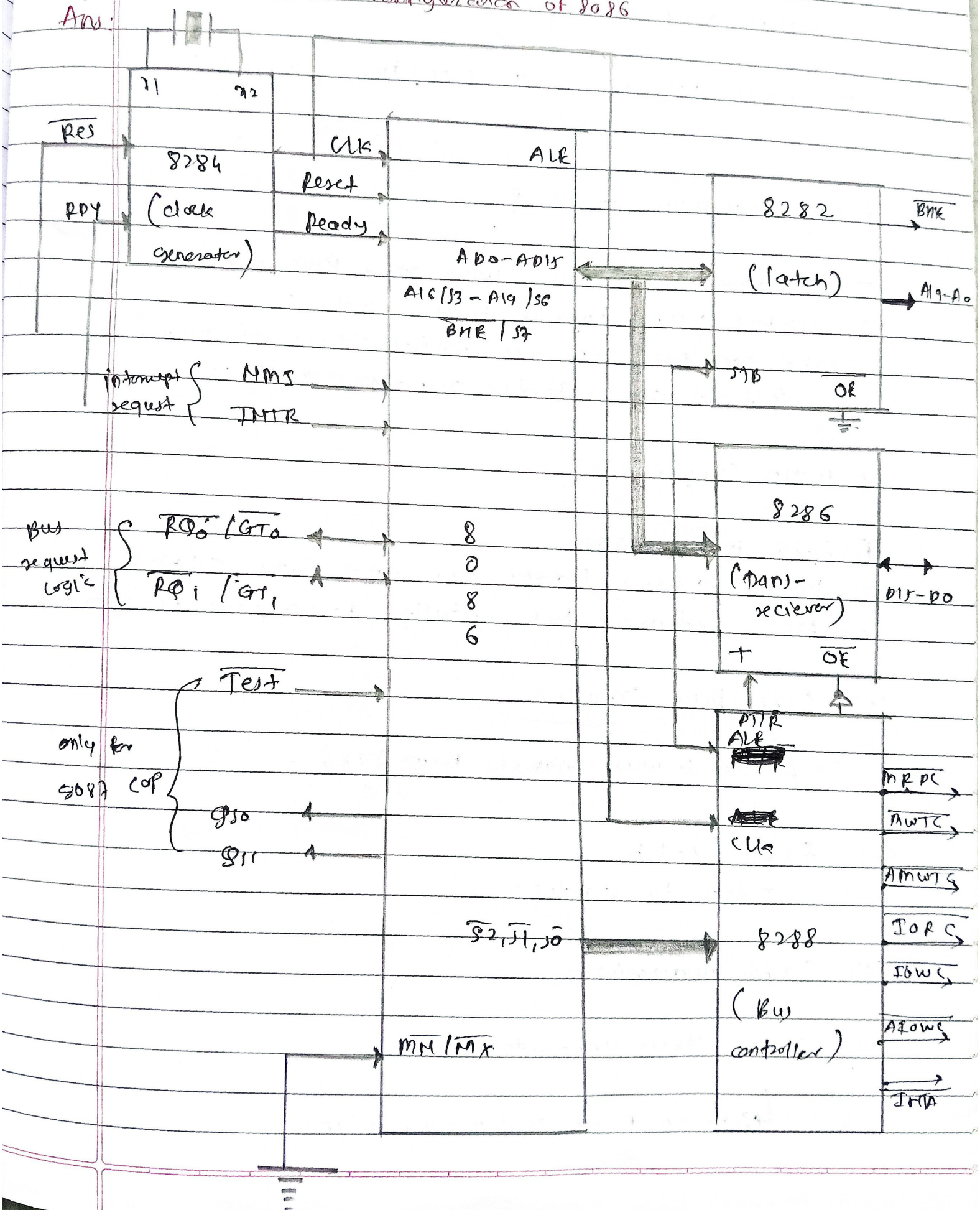
(3) T3 :
Data on D15-0 is written into memory or I/O device

(4) T4 :
- MWTC / IOWC and DEN go High — write ends
- Bus becomes idle.

Q) explain maximum mode of 8086

Q maximum mode configuration of 8086

Ans:



PAGE No.
DATE / /

- in maximum mode, the 8086 microprocessor is used in system with multiple processor or co-processor

- To enable this mode, the MN/$\overline{MX}$ pin set to 0

* Why maximum mode ?

- To share system bus with other processor
- Useful when coprocessor is used
- 8086 does not generate control signals directly, instead gives status signals. S2, S1, S0

* Main Components

(1) 8086 Microprocessor
- provide address / data bus and status signal
- need external bus controller to complete the operation

(2) 8288 Bus Controller
- Receive status signals from 8086

(3) 8282 Latch
- Store the address

(4) 8286 Transceiver

(5) 8284 Clock generator
- generate reset and ready signals.

PAGE NO.
DATE / /

* How it works:

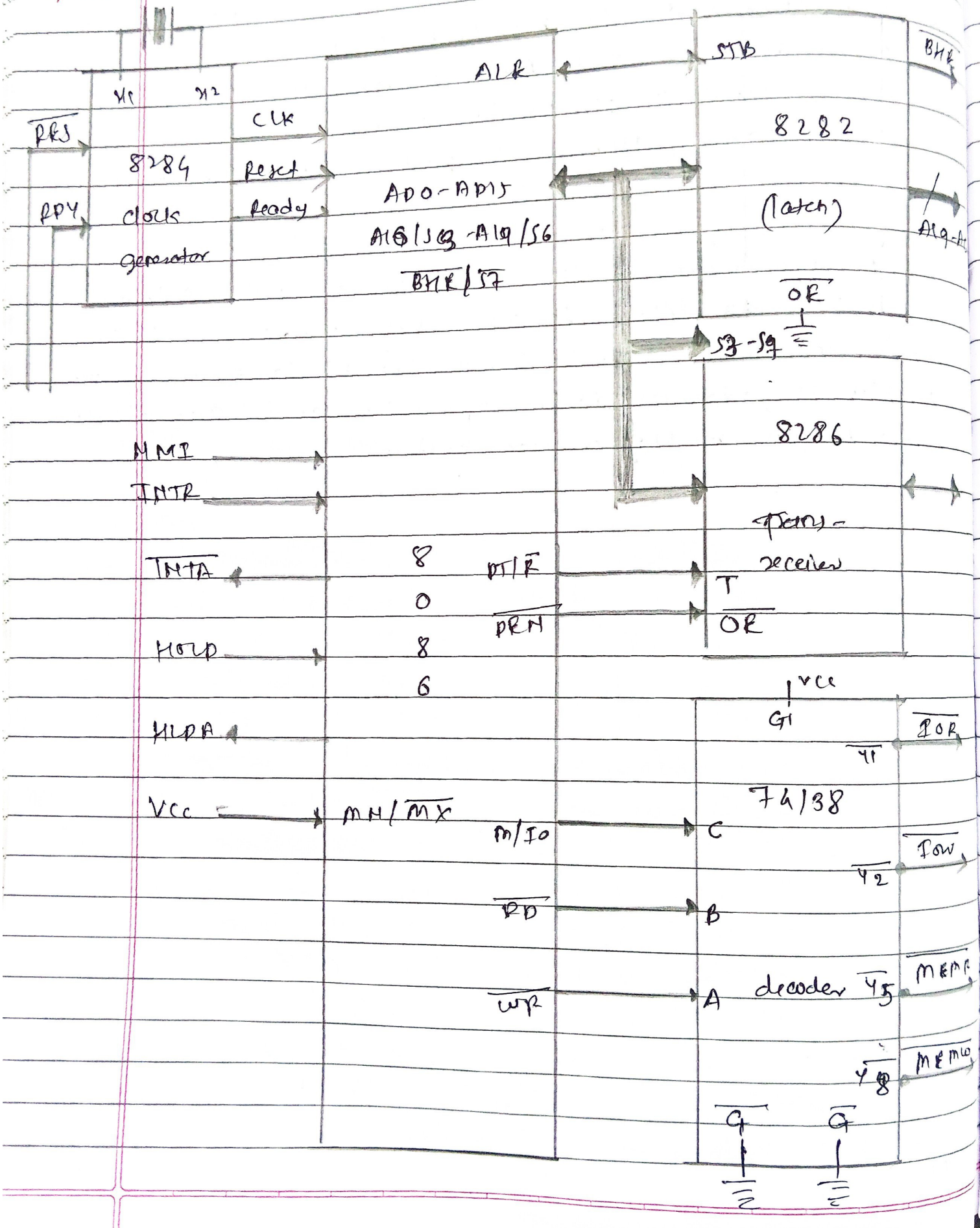
- Address is placed on AD0 - AD15 and A16 - A19 lines
- ALE enables the latch to hold the address
- 8286 manage data flow on D0 - D15
- 8086 send S2, S1, S0 to 8288
- 8288 decodes them and produces control signals

in maximum mode, 8086 works with external devices like 8288 (bus controller), 8286 (transceiver) and 8282 (latch) to manage control signals, address / data separation and communication
- used in multiprocessor systems.

PAGE NO.
DATE / /

(9) explain minimum mode of 8086

Ans:



In minimum mode, the 8086 operates as a single processor system, To enable this mode, the MN/$\overline{MX}$ pin set to 1

* Why minimum mode?

- Used for simple system with only one CPU
- 8086 generate all control signals by itself
- suitable for basic microprocessor application

* main components

(1) 8086 microprocessor
it generate all control signals

(2) 8284 clock generate
generate Reset and ready signals

(3) 8282 latch
- store the address

(4) 8286 Trans-receiver

(5) 74139 decoder
- decode status signal to generate memory I/O control signal.

PAGE No.
DATE / /

★ How it works?

- 8086 place address and ~~data~~ data on multiplexed line
- ALE enables the latch to store the address
- Control signals $\overline{RD}$, $\overline{WR}$ and ALE are generated directly by 8086
- 74138 decoder use these signals to activate memory I/O, read/write operations

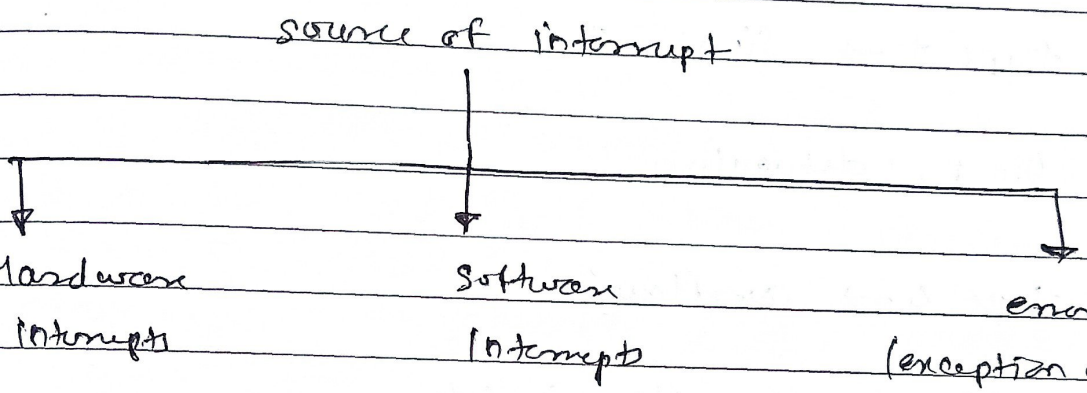
In minimum mode, 8086 handles all control signals internally, it used in single CPU system, with external support only for latching address and controlling data direction.

Q) explain the interrupt structure of the 8086 processor (IVT) and diff b/w Hardware and software interrupt

Ans:

An interrupt is a signal that temporarily stops the CPU's current execution so that it can attend to an urgent task

In 8086, we have three source of interrupts.



(1) Hardware Interrupts:

- In this type of interrupt, physical pins are provided in the chips, In 8086 we have two pins

(i) NMI (Non maskable interrupt)
(ii) INTR

(2) Software Interrupts:

- Software interrupts, in 8086 we have INT instruction
- When INT instruction is executed interrupt will occur

(3) Error condition (exception or internal interrupt)

These are automatic interrupt generate due to errors during execution

(a) Type 0 → divide by zero error

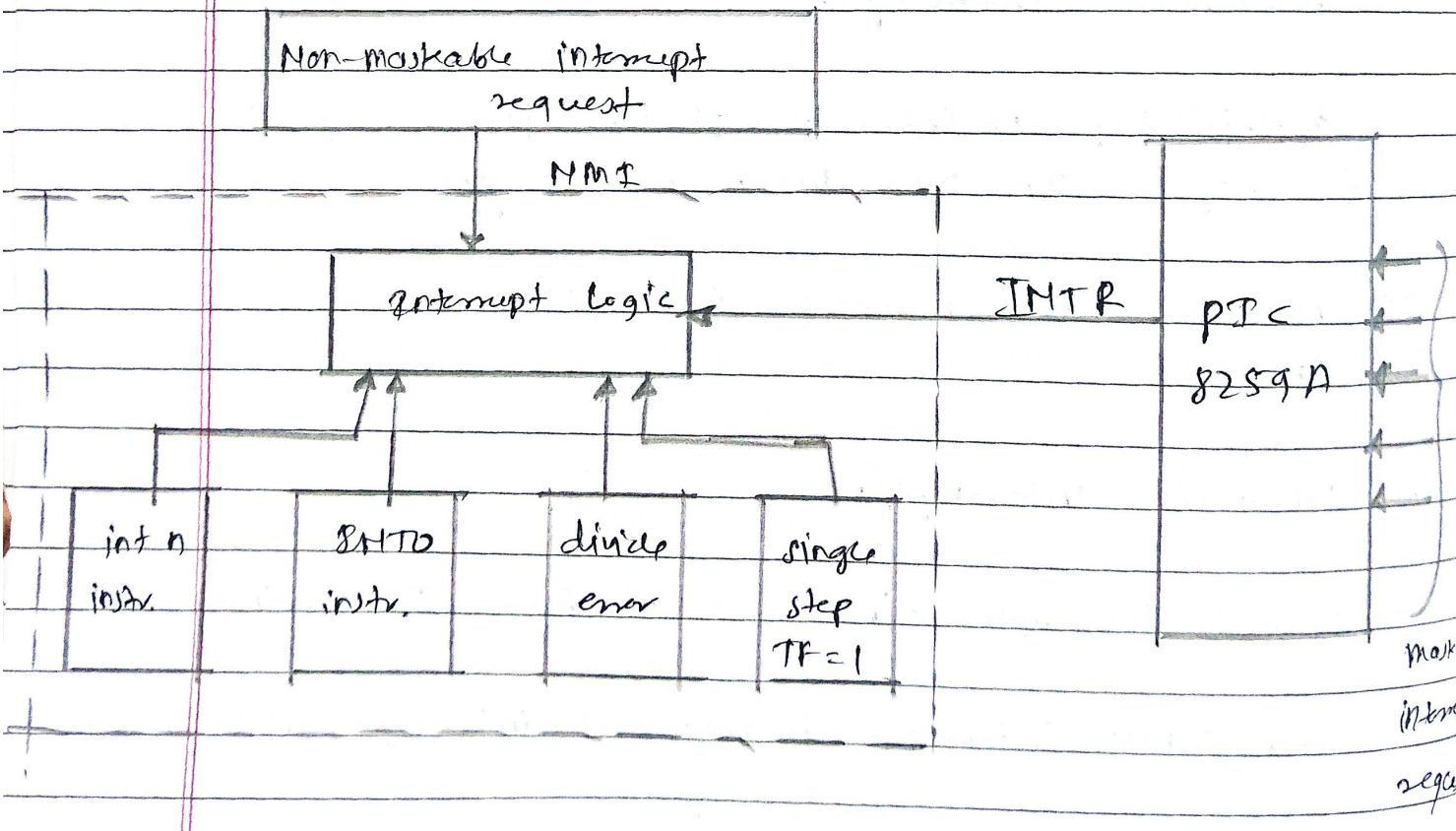
occurs when a division by zero or result overflow happen

(b) Type 1 → Single step

Use for debugging

(c) Type 4 → overflow

When overflow flag is set.



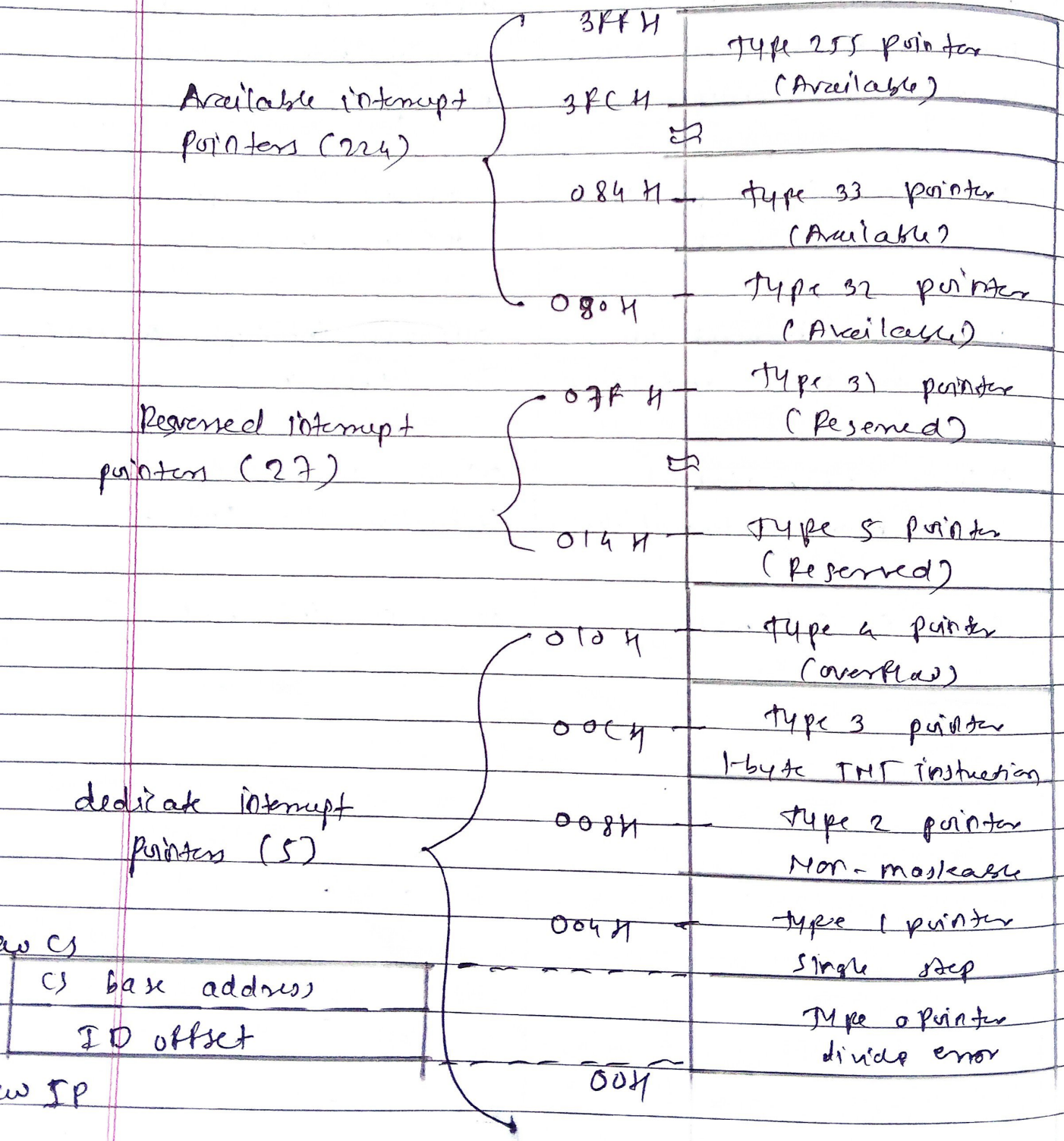
Sr. No	Hardware Interrupt	Software Interrupts
(1)	To implement hardware interrupt a pin is given to processor	to implement a software interrupt an instruction is given to the processor
(2)	Hardware interrupts are mostly maskable or non-maskable interrupts	software interrupts are mostly non-maskable interrupts
(3)	Hardware interrupts may be vectored or non-vectored	software interrupts are mostly vectored.
(4)	Hardware interrupt of 8086 are NMI and INTR	software interrupts of 8086 are INT n
(5)	Trigger when signal source from external device	Trigger when causes by instruction like Int n
(6)	Sours is external.	Source is internal

Q) explain IVT (Interrupt vector table)

Ans:

The Interrupt vector table (IVT) is a reserved area of memory used to store the address of interrupt service Routines (ISR's)

When an interrupt occurs, the 8086 uses the IVT to locate and execute the appropriate service routine.



PAGE No.
DATE / /

location of IVT

- The IVT is located at the beginning of memory from address 00000H to 003FFH
- It occupies 1KB (1024 bytes) of memory space

structure of IVT

- The IVT contains 256 interrupt vectors, one for each possible interrupt type (00H to FFH)
- each vector is 4 byte long:
  - 2 bytes the offset (IP) of the ISR
  - 2 bytes the segment (CS) -"-"-"-

purpose of IVT:

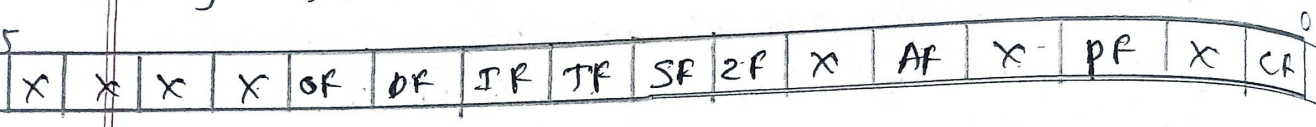
- provides a mapping blw an interrupt type and its corresponding ISR
- when an interrupt occurs, the 8086
  - (1) → multiplies the interrupt type by 4 to get the IVT address
  - (2) reads CS and IP from that location
  - (3) jumps to the ISR using CS:IP

Q) explain Flag registers of 80386 processor
Q) explain E-Flags

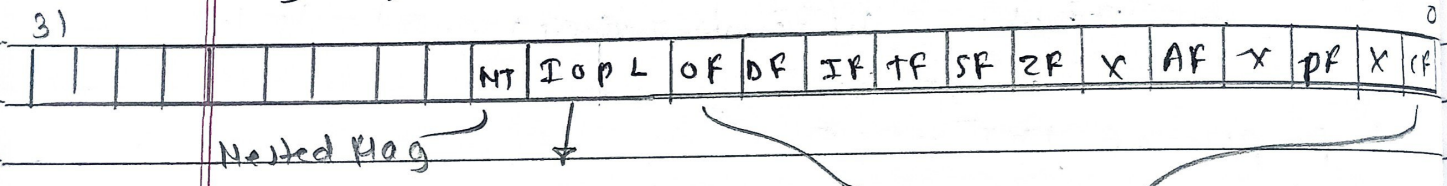
Ans:

The E-Flags (extended Flags) register is a 32 bit register in the x86 architecture used to indicate the status of the processor and control operations. it contain Status flag, control flags and system flags.

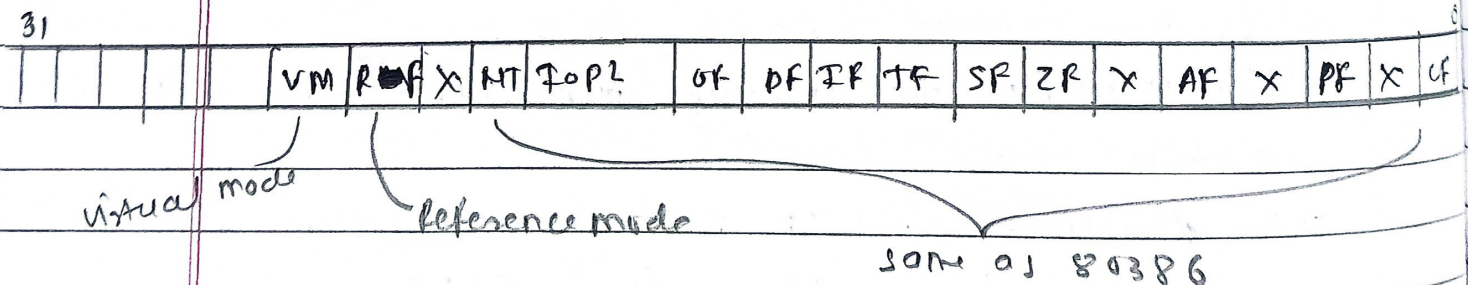
- Flag register of 8086:



- Flag register of 80386:



- Pentium



PAGE NO.
DATE / /

★ System Flags:

(1) VM → Virtual 8086 Mode
(2) RF → Resume Flags
(3) NT → Nested Flags
(4) IOPL → I/O privelege Level

★ Status Flags:

(1) OF → overflow Flags
(2) PF → Parity Flag
(3) ZF → zero.
(4) SF → Sign
(5) AF → Auxillary carry
(6) CF → carry Flag

★ Control Flags

(1) TF → Trap Flag
(2) IF → Interrupt Flags
(3) DF → direction Flags

PAGE No.
DATE / /

* System Flags:

(1) VM (virtual 8086 Mode):

enables virtual 8086 mode for running real-mode code.

(2) RF (Resume Flag):

Used with debugging to control exceptions

(3) NT (Nested Task):

Used for controlling multitasking

(4) IOPL (I/O privilege level):

privilege level for I/O operations

* Control Flags:

(1) TF (Trap Flags): enables single-step debugging

(2) IF (Interrupt Flag): If set, enables external interrupts

(3) DF (Direction Flags): control string operations (Increment/decrement)

* Status Flags:

(1) OF (Overflow Flags): Set if an arithmetic overflow occurs

(2) PF (Parity Flags): Set if the number of set bits in the result is even

(3) ZF (Zero Flags): Set if the result of an operation zero

(4) SF (Sign Flag): Set if the result is negative

(5) AF (Auxilliary Carry): Used in BCD operation (Used between 4-bit to higher bits)

(6) CF (Carry Flag): Set if there's carry or borrow in arithmetic operations.

PAGE No.
DATE / /

Q* compare b/w 8086, 80386, Pentium 1, 2, 3

Ans:

Attribute	8086	80386	Pentium 1	Pentium 2	Pentium 3
Processor size	16 bit	32 bit	32 bit	32 bit	32 bit
Data bus size	16 bit	32 bit	64 bit	64 bit	64 bit
Address bus size	20 bit	32 bit	32 bit	36 bit	36 bit
Physical memory	1 MB	4 gb	4 gb	64 gb	64 gb
Segmentation	Yes	Yes	Yes	Yes	Yes
Paging	No	Yes	Yes	Yes	Yes
Protection	No	Yes	Yes	Yes	Yes
Pipeline level	2 stage	3	5	14	14
Branch Prediction	No	No	Yes	Yes	Yes
Cache	No	Yes	Yes	Yes	Yes
Operating Frequency	5 MHz	16 MHz	66 MHz	233 MHz	450 MHz
Out of order execution	No	No	No	Yes	Yes
Year of Release	1978	1985	1993	1997	1999

Q) diff b/w real mode, virtual mode and protected mode

Ans:

Sr. No	Real Mode	Protected Mode	Virtual mode
(1)	It support 1 MB + 64 KB memory	it support maximum 4gb of memory	it support maximum 4gb of memory
		↳ same	↑
(2)	In real mode, it execute only single task at an instance.	It support multi-tasking at a particular instance	It support multi-tasking at a particular instance.
		↳ same	↑
(3)	In real mode, processor behave like a 8086 or 8080 microprocessor	In protected mode it provides memory protection, multitasking and extended memory access	In virtual mode, it is a combination of Real and protected mode.
(4)	The switching between protected and real mode is not possible	The switching between protected and virtual mode is possible	The switching between real and protected mode is possible.
(5)	It allow direct access to hardware and memory	It provides privilege level (0 to 3) to control access to system resources	Running multiple virtual 8086 environments.

PAGE No.
DATE / /

Q) explain Floating point pipelines of pentium Processor

Ans:

- Floating point unit is heavily pipeline, hence allowing several instruction to be executed simultaneously under certain conditions

- most of the floating point instruction have to be in U pipeline only and cannot be paired with integer instruction.

- The first four (4) stage of the floating point pipeline are shared with the integer pipeline unit

- The 8 pipeline stages of the pipeline unit are as follows:

(1) Prefetch: instruction is fetched from memory into the pipelines

(2) Instruction Decode one: Basic decoding of the instruction format begins.

(3) Instruction Decode two: Completes decoding and identifies operations type and operands.

(4) Execution Stage: Perform the actual floating-point operations.

(5) Floating point execution 1 (FPE 1):

- In this stage, read the information from register and memory and moving into floating point register
- It is also converted to FP format

(6) Floating point Execution 2 (FPE 2):

- In this stage FP operation is performed.

(7) Write FP Result:

The result is rounded off and return in FP register

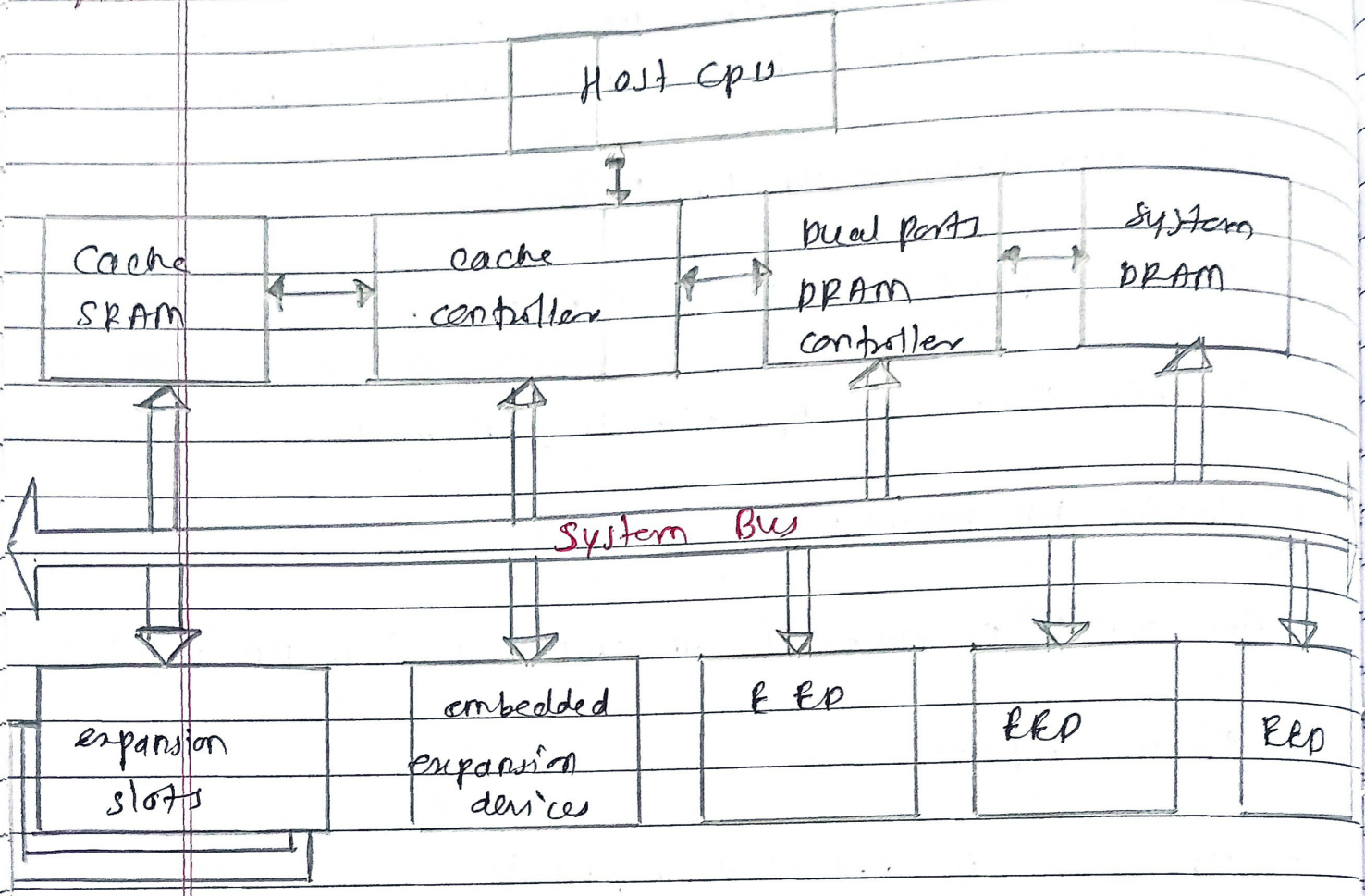
(8) Error Reports:

- If error is detected, report it.

Architecture

Q) Explain the cache ~~organization~~ of Pentium Processor.

Ans



The figure illustrates the typical memory and cache architecture ~~organization~~ in pentium-class processor, focusing on high-speed access and efficient system performances, through multiple levels and controllers.

PAGE NO.
DATE / /

* components of cache organization

(1) Host CPU: Central processing Unit that executes instruction and processes data.

(2) Cache SRAM (Static RAM): High speed memory used for cache.
- stores frequently accessed data to reduce CPU wait time.

(3) Cache Controller: manage all cache operations. (read/write, replacements)
- interface b/w CPU and cache SRAM

(4) Dual port DRAM controller: Allows simultaneous access from both cache controller and system
- enhanced memory access efficiently by supporting concurrent operations

(5) System DRAM: Main memory of the system
- Accessed when data is not found in the cache.

(6) System Bus: data communication path b/w CPU, memory and other components.
- transfer data, address and control signals.

(7) expansion slots: physical interfaces for adding extra hardware.
eg: CPU, network cards

- connected via system bus to communicate with CPU and memory

(8) embedded expansion devices:

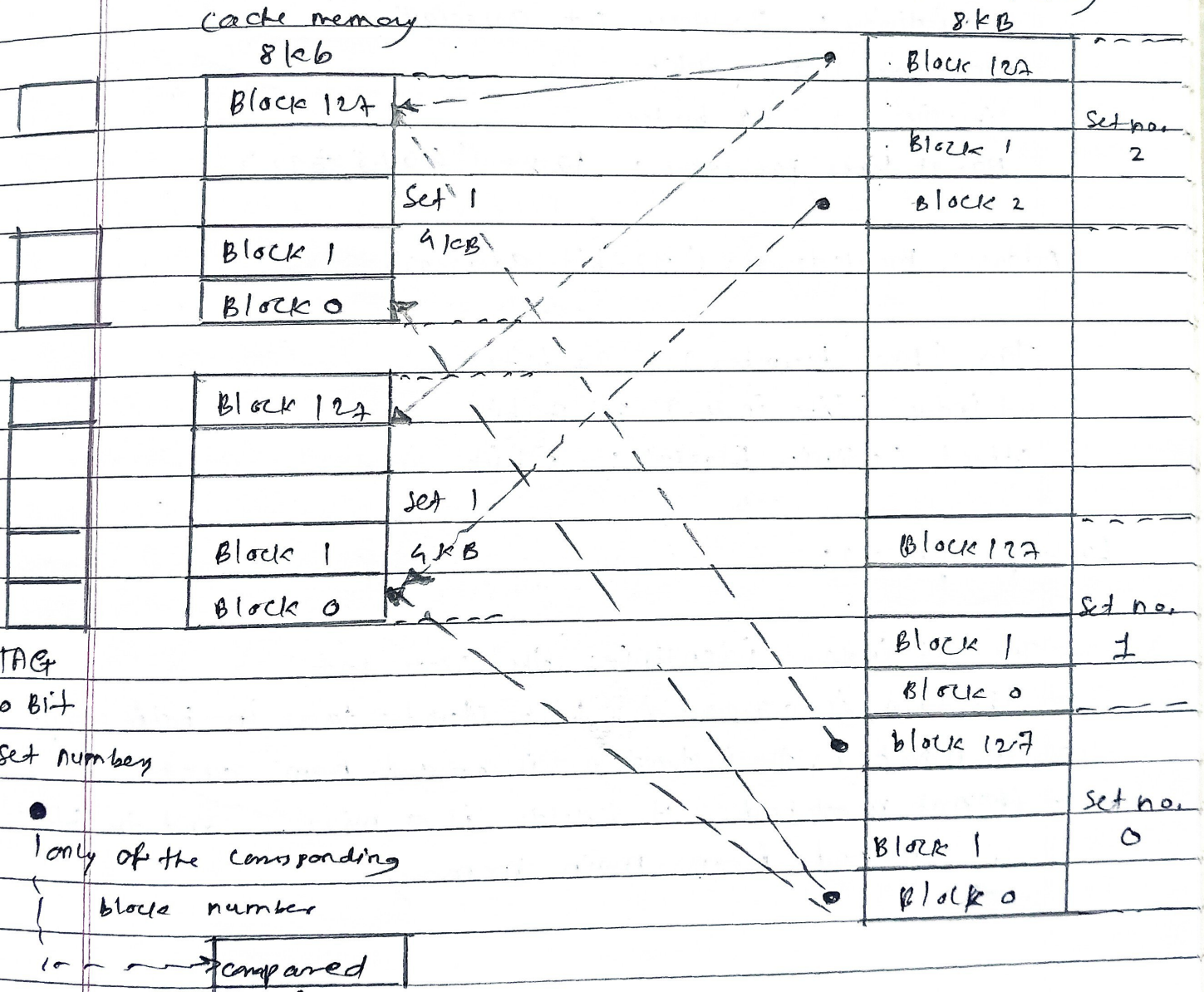
Built-in ~~to~~ additional hardware components
eg: wifi, graphics

- Integrated to motherboard and communicate with system bus

Q) explain in brief cache organization of pentium processor also called code cache organization and explain data cache organization too

Ans:

The pentium processor uses a two-way set associative cache organization for both code and data cache each of size 8 kB. The code cache specially stores the instruction data to be executed by the processor, enhancing the instruction fetch performances.



Main memory address	20 bit	7 bit	5 bit
	Set no.	Block no.	Location within block

The pentium processor uses two separate level 1 (L1) cache

- code cache (instruction)
- data cache

① Code cache organization

purpose: stores instructions to be executed by the CPU

total size: 8kb
Associativity: 2-way set associative
each way size: 4kb
Line size: 32 bytes
No. of lines per way: 128 (4096 ÷ 32)

Address Breakdown (32-bit address)

Tag (Page Number): 20 bits
set index (line index): 7 bits
offset (within block): 5 bit

Lookup process:

- The set index identifies the cache set
- the tag is compared with stored tags in both ways
- if matched, the instruction is read from cache
- if not matched, it results is a miss, and the block is fetched from main memory

② Data cache organization

purpose: Stores data operands used by instruction

total size : 8 Kb
Associativity : 2-way set associative (way 0 and way 1)
each way size : 4 Kb
Line size : 32 bytes
No. of lines per way : 128

Directory Entry for each line:
20 bit Tag
2-bit MESI states:
00 : invalid
01 : exclusive
10 : Modified
11 : Shared

- Parity bit : error detection for each byte.

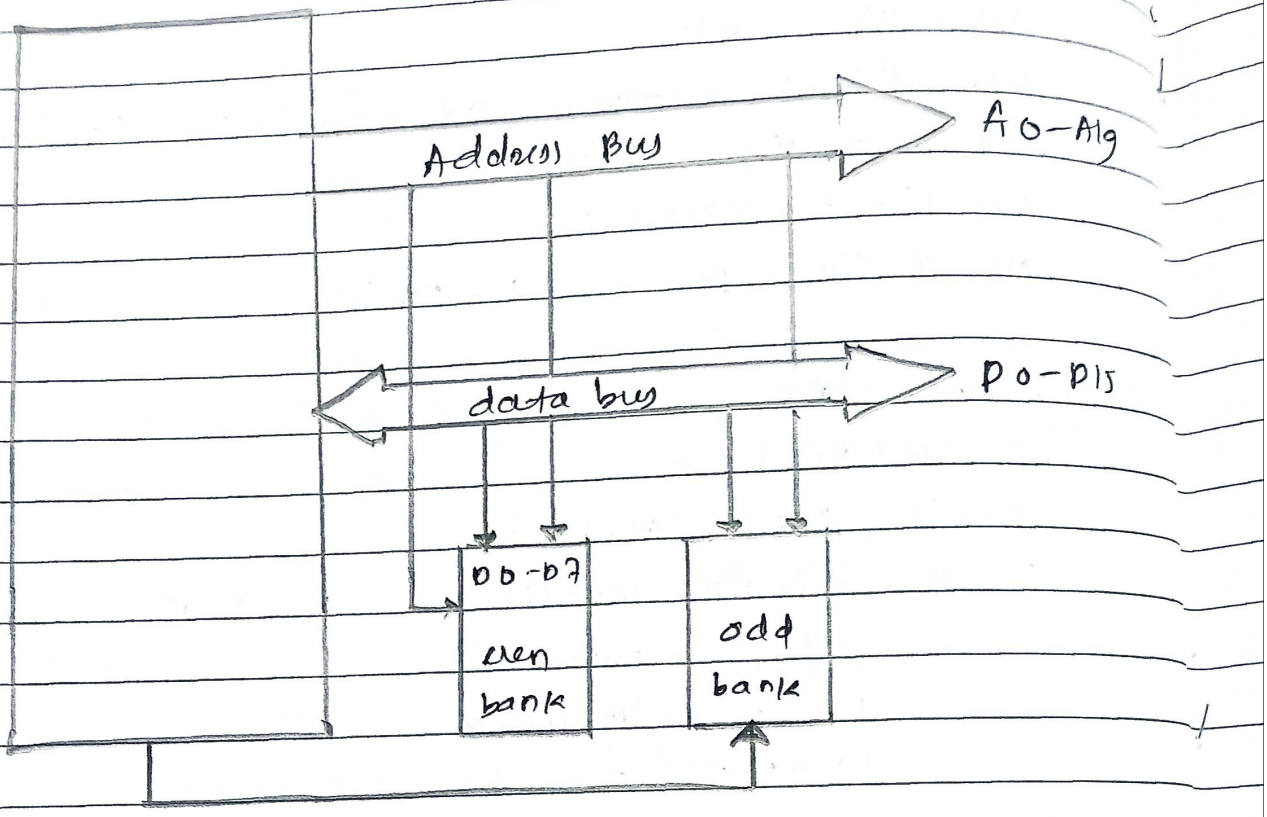
Cache Access:

- When data is accessed, the set index and tag are used
- if data is present, it's read quickly
- if not, its fetched from system memory and stored in cache

PAGE No.
DATE / /

Q) explain memory banking? what is advantage of memory banking and justify with example.

Ans:



- The 8086 microprocessor is a 16-bit processor with 20-bit address bus (A0-A19) and 16-bit data bus (D0-D15) allowing it to access up to 1MB of memory and process 16-bit data efficiently

- However, memory is organized as 8-bit wide, So to handle 16-bit data, 8086 uses a technique called memory banking.

PAGE NO.
DATE / /

✱ Memory banking structure:

- Address Bus – (A0-A19): Used to specify memory address
- data bus (D0 - D15) : 16-bit wide data bus
- memory is divided into two banks:

  even - bank: for even address bytes (D0 - D7)
  odd - bank: for odd address bytes (D8 - D15)

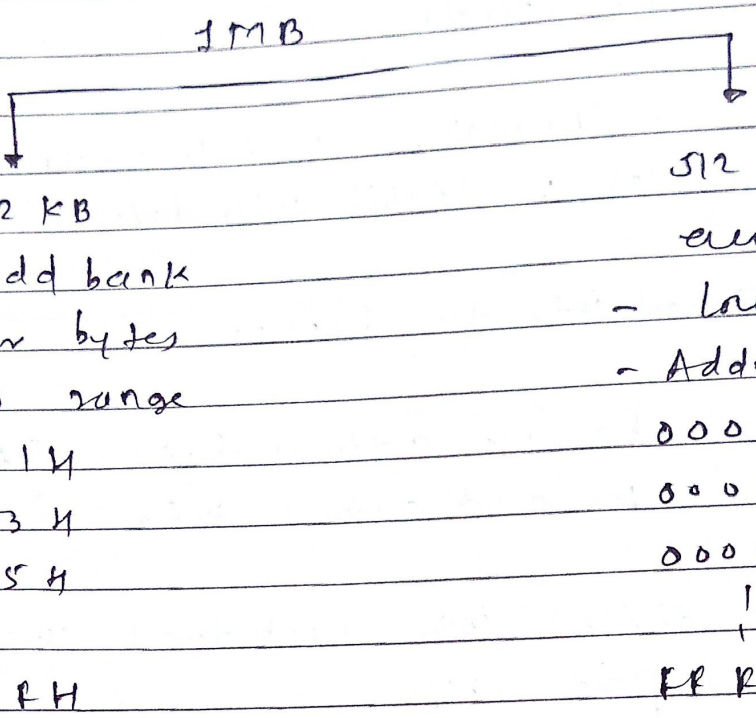
- memory is divided into, such that consecutive location fall into two parts chips

- So one chip has even locations 0, 2, 4, 6 ... etc called even bank

- So one chip has odd location 1, 3, 5 ... etc called odd bank.

Generally for any 16-bit operations, even bank provides the lower bytes and ~~high~~ odd banks provides the higher bytes.

Hence the even bank is also called lower bytes and the odd bank is also called higher bytes.



Select when $\overline{BHE}$ 0 — Select when A0 = 0

$\overline{BHE}$	A0	Operation
0	0	Read or write 16-bits from both banks
0	1	Read or write 8 bits from higher bank
1	0	Read or write 8 bits from lower bank
1	1	No operation

★ Advantage:

- efficient for 16-bit data handeling
- Faster execution
- Better performance
- Flexibility
- Simplified memory design

PAGE No.
DATE / /

eg:- Accessing a word at even bank (eg: 1000H)

$A0 = 0$, $\overline{BHE} = 0$ → both banks accessed

entire 16-bit word fetched in ~~one~~ one cycle

- Accessing a word at Odd address (eg: 1001H)

Require two memory accesses - one to 1001H
one to 1002H

takes two cycles reducing efficiency.

Q) explain hyper threading Technology and its use in pentium 4.

Ans:

- Hyper threading (HT) Technology is intel's hardware based simultaneous multithreading (SMT) technology.

- it improves the performance of IA-32 processor by allowing better utilization of CPU resource when running multi-threaded operating system and applications.

★ Key features of HT:

(1) concurrent execution
(2) simulated multi-processing
(3) IA-32 support
(4) Logical processor design
(5) APIC support (Advance Programmable Interrupt Controller)
(6) OS View:

★ Use in Pentium 4:

- Pentium-4 was the first intel processor series to implement Hyper-threading Technology

- it improved performance in:

  - multitasking
  - threaded application

PAGE No.
DATE / /

- Allowed one physical Pentium 4 processor to appear as two CPUs to the O.S.

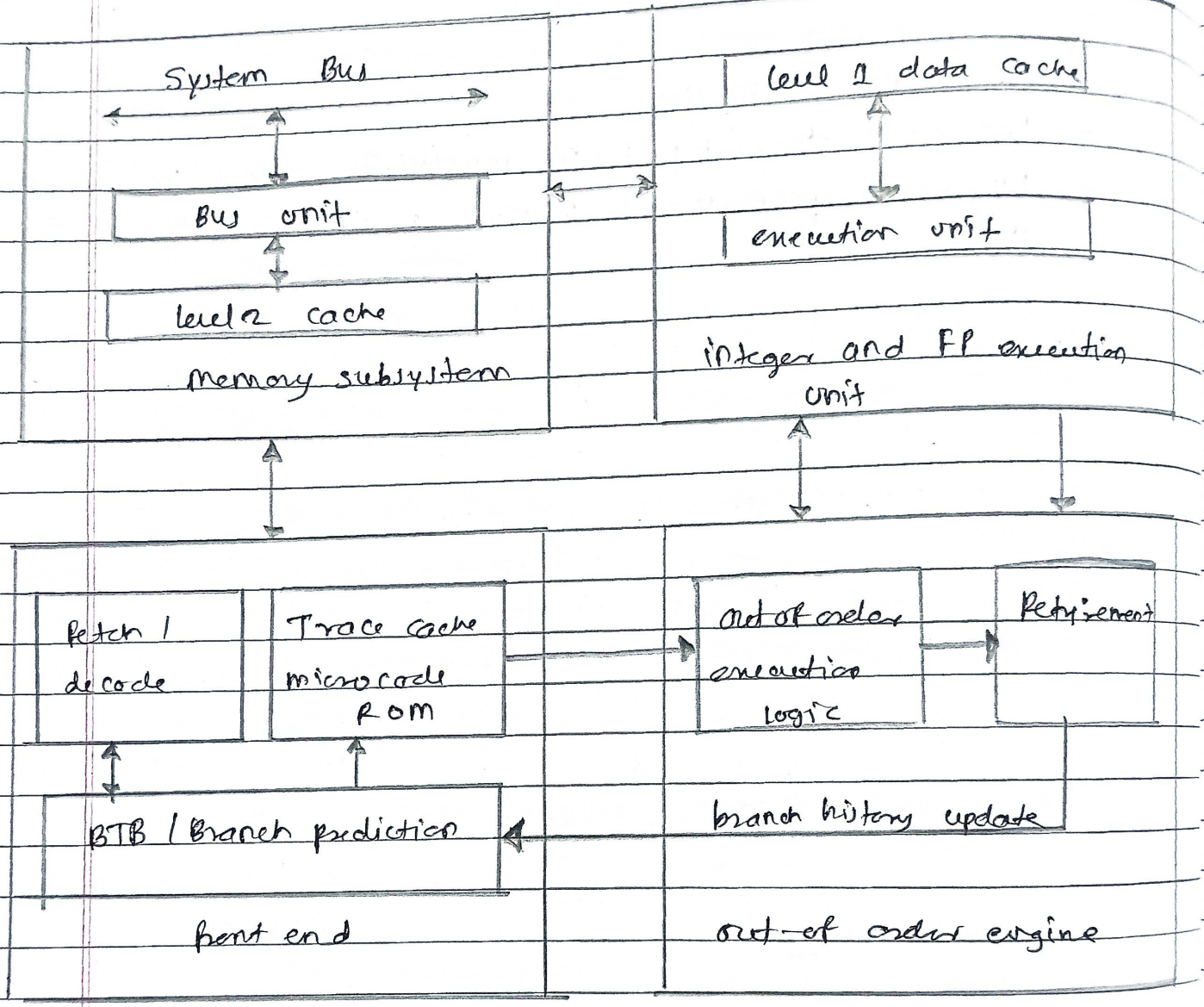
* Advantage:

- Better CPU utilization
- Increased throughput in multi-threaded workloads
- Improved performance without needing extra physical cores.

PAGE No.
DATE / /

Q) draw and explain Pentium 4: Net burst micro-architectures

Ans:



PAGE No.
DATE / /

* Front-end Module:

- instruction fetch & decode unit: fetches and decodes instruction
- branch prediction unit: predicts the next instruction to reduce delays.
- Trace cache & microcode ROM: stores decoded instructions for fast access

* out-of-order execution unit:

- executes instructions out of order to improve speed
- Retirement Unit: Reorder results to maintain correct program order.

* execution unit:

- performs actual operation
- connected to level 1 data cache for fast data access

* Memory subsystem:

- BUS interface unit (BUI): connect CPU to the system bus
- ~~Ind~~ includes optional Level 3 cache for better memory performances

Q) explain (ICW) initialization command words and (OCW) operational command words of the 8259 PIC

Ans:

The 8259 programmable Interrupt controller (PIC) is used to manage hardware interrupts and send them to the CPU in a prioritized way.

To control its operations, it uses two sets of control words:

(1) Initialization command words (ICWs)

ICW are used to initialize the 8259A programmable interrupt controller. They are issued once during system setup to configure the operation mode of the PIC

- There are 4 ICWs

(1) ICW 1 : Starts initialization and sets basic operation

(2) ICW 2 : Sets the basic address for interrupt routine

(3) ICW 3 : Specifies master/slave configuration

(4) ICW 4 : Specifies additional control information like operating mode

(2) OCWs (operational command words)

OCWs are used to control and manage 8259A PIC during normal operation.

These command can be issued at any time after ~~Reinitialization~~ initialization.

- There are 3 OCWs

(1) OCW 1 : Used to mask or unmask interrupt request lines

(2) OCW 2 : controls how interrupts are acknowledge

(3) OCW 3 : Control interrupts polling, special mask mode and display the status

- The ICWs set up the 8259 PIC initially, while OCWs manage its behaviour during normal operation

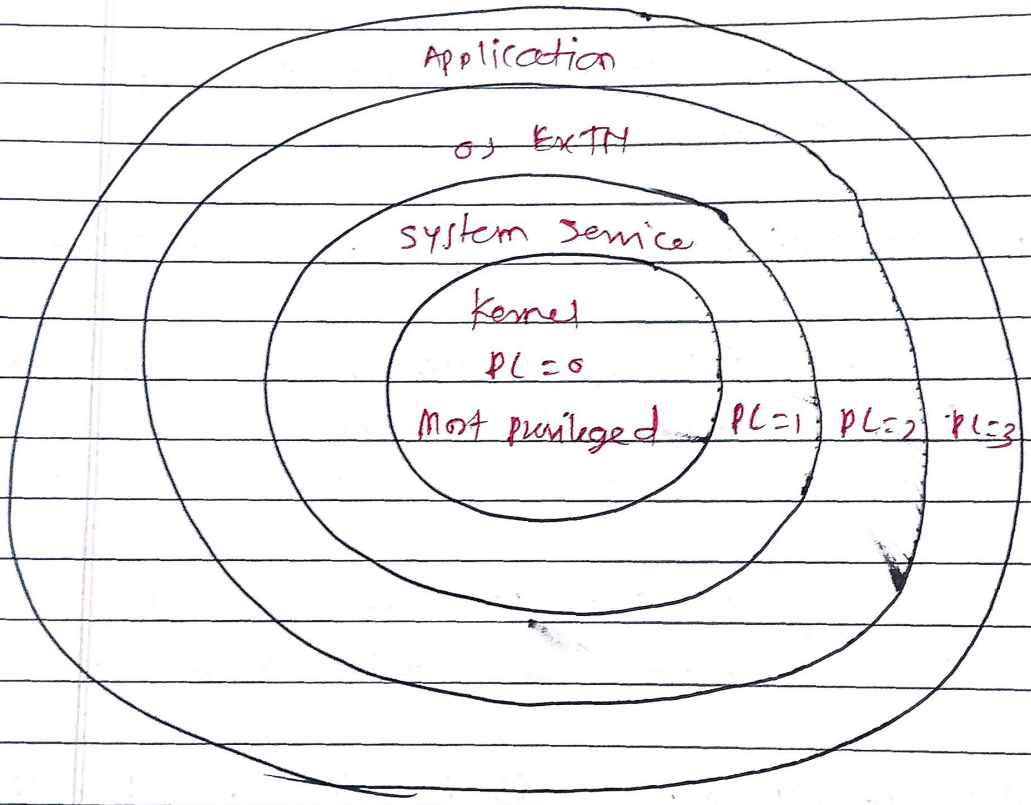
- These control words provide flexible interrupt handling and efficient CPU interaction in system using 8259.

Q) discuss in brief protection mechanism of 80386 DX

Ans:

The 80386 DX processor provides a powerful protection mechanism to support multi-tasking and memory protection in modern operating system.

These mechanism are based on segment-level and task level protection and help prevent unauthorized access between programs and the O.S



- Privilege level (PL)

The x86 architecture supports 4 privilege level; known as privilege levels (PL) to isolate tasks.

PAGE No.
DATE / /

- PL = 0 → most privilege (kernel level)
- PL = 1 → system services
- PL = 2 → OS extension
- PL = 3 → least privilege (user application)

Lower PL = Higher privilege

A task at lower level can access higher level segment but not vice versa.

- five key protection mechanism in 80386 DX

(1) Privilege Check:

- Stops lower level tasks from using higher level code
- compare privilege levels before allowing access.

(2) Limit Check:

- each memory segments has a size limit
- Access outside the limit causes an error

(3) A (Available) Bit:

- shows if a segment is already being used by another task
- Prevents sharing of active segments.

(iv) Type check:

- Makes sure segments are used properly.
  Code segments → for instruction
  data -1- → for data
  system segments → only by system

(v) User / Supervisor Check:

- each memory page has U/S bit

  U/S = 1 → User level access

  U/S = 0 → Supervisor Only access

- Stops user programs for accessing system memory

⊙ Task Management::

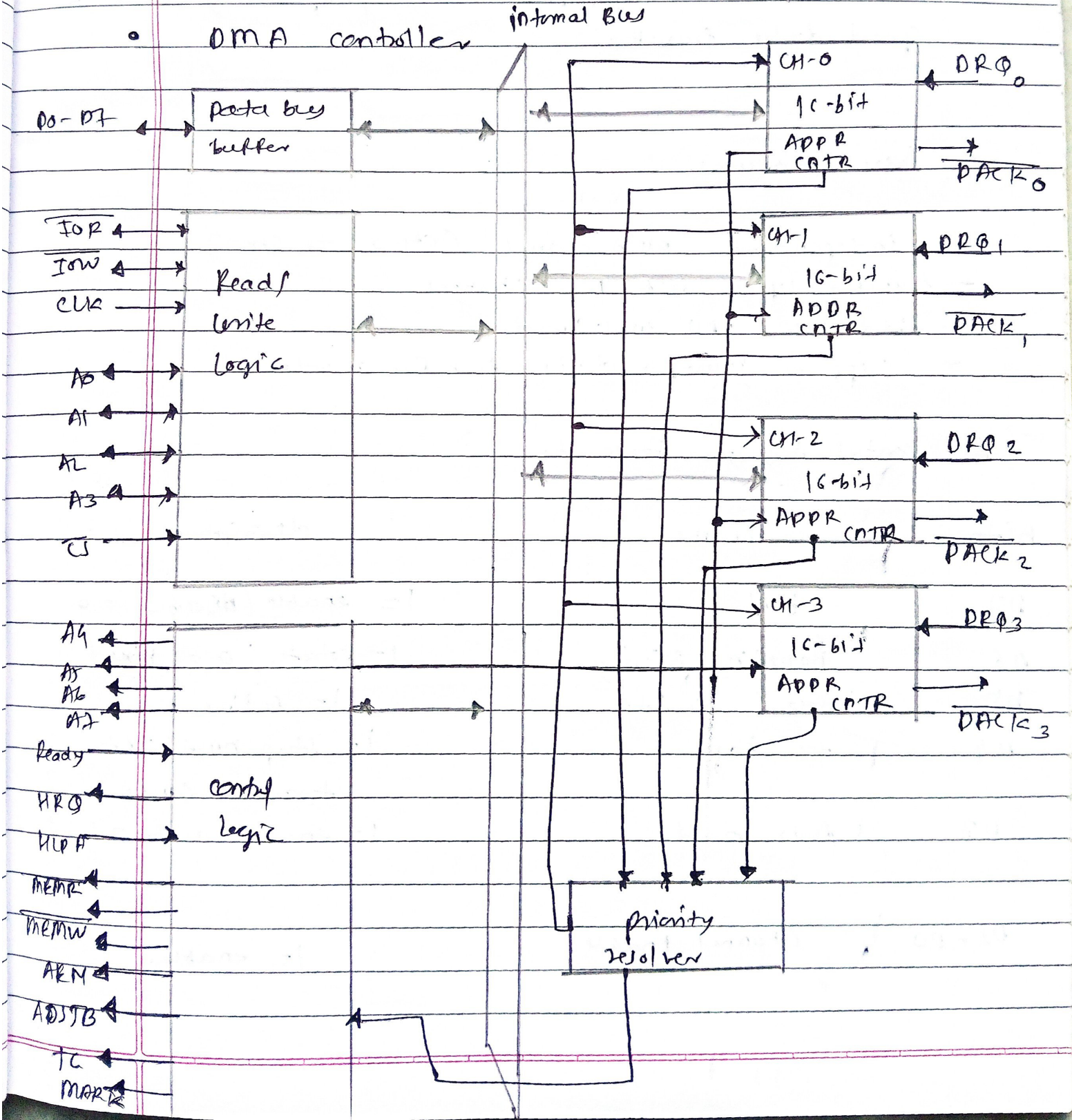
- Support hardware-based task switching with protection
- each task has a TSS (Task state segments)
- TSS contain register state, stack pointers and I/O permissions
- during a task switch:
  old tasks content is saved in its TSS
  New tasks content is loaded from its TSS
  All protection checks are performed

PAGE NO.
DATE / /

8) explain the 8257 DMA controller with a neat diagram and its control register format

9) Interface DMA controller 8257 with 8086 mp explain different data transfer modes of 8257 DMA control.

Ans:



- The intel 8257 is a 4-channel direct memory Access (DMA) controller
- it allows peripherals devices to directly access system memory without involving the CPU
- this improves systems performance, especially for high speed data transfer.

* Key features.

- 4 independent DMA channel (CH-0 to CH-3)
- transfer upto 16KB per channel
- Reduces CPU workload
- supports multiple data transfer modes.

* Control Register Format:

Bit	Name	description
D7	EN	1= enable / disable = 0
D6	Rotating priority	1= Rotate ; 0 = Fixed
D5	extended write	1= Active
D4	TC stop	1= stop DMA after terminal count
D3	Auto load	1= enable auto reload
D2 - D0	Channel Enable	1= enable.

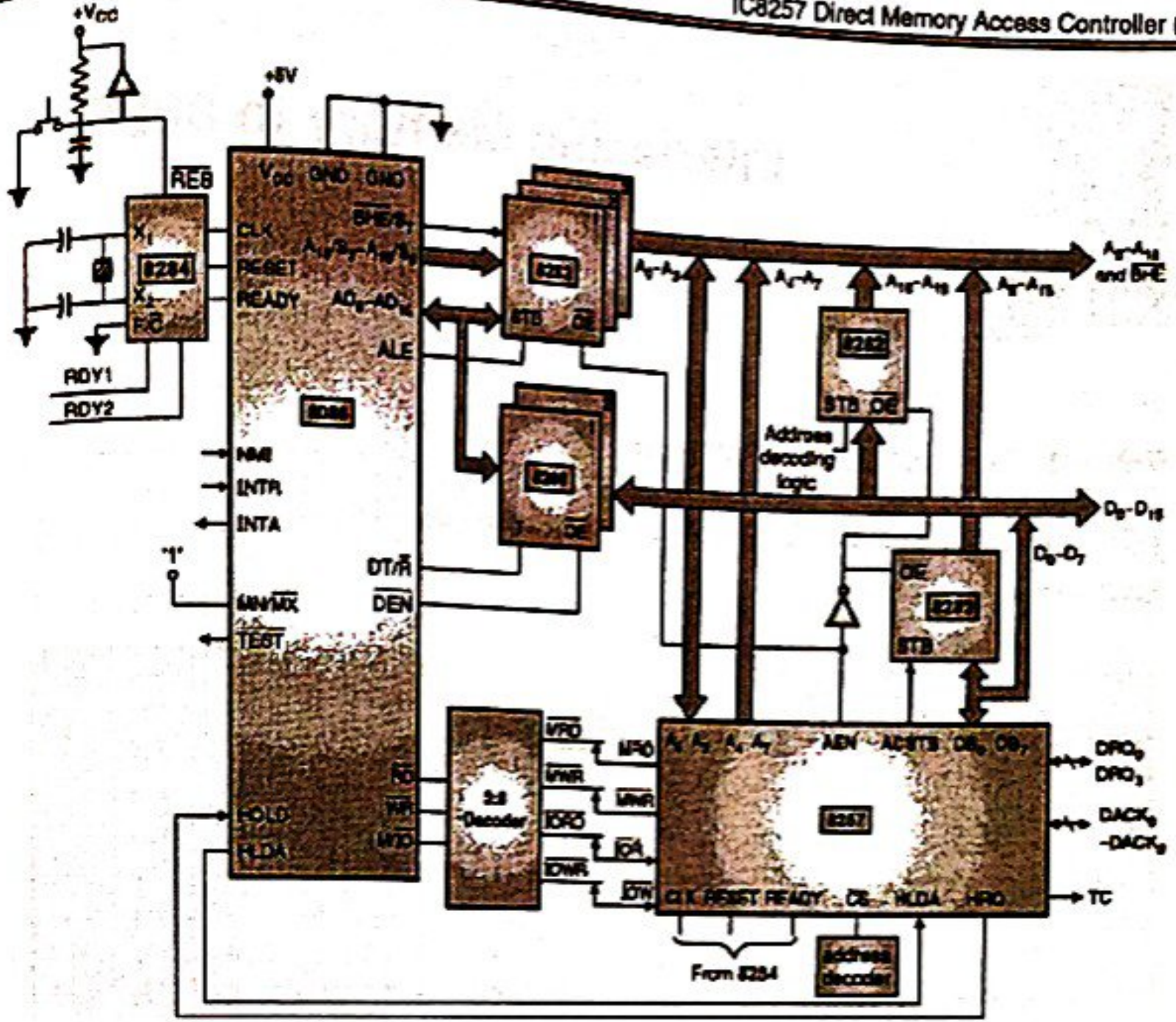


Fig. 10.7.1 : Interfacing of 8086 with 8257

8257

Interfacing with 8086

To use 8257 with 8086 microprocessor, proper signal connection are needed.

(1) Address Bus:

8086 A_0-A_3 are connected to 8257 to access its internal registers.

(2) Data Bus:

Connect 8257 D0-D7 to the lower 8 data lines of 8086

(3) Control signals:

- $\overline{RD}$, $\overline{WR}$, $\overline{CS}$ signals control Read/write access to 8257
- HRQ (Hold Request) from 8257 connect to HOLD 8086
- HLDA (Hold acknowledge) from 8086 connects to HLDA of 8257

(4) DMA Channels:

- Peripherals send DMA requests via DRQ0 - DRQ3
- 8257 respond with DACK 0 - DACK 3 to initiate data transfer.

PAGE No.
DATE / /

★ Data transfer modes :

(1) Burst Modes :

- transfer a complete block of data continuously without CPU interruption
- CPU is kept in hold state during the entire transfer
- suitable for high-speed devices

(2) Cycle stealing mode :

- DMA controller takes one bus cycle at a time from the CPU
- CPU and DMA work alternately.
- Reduces CPU performances slightly but allows concurrent operations.

(3) Demand Modes :

- DMA transfers data only when the peripheral request it
- transfer pauses automatically if DREQ becomes inactive
- Useful when peripheral is not ready continuously.

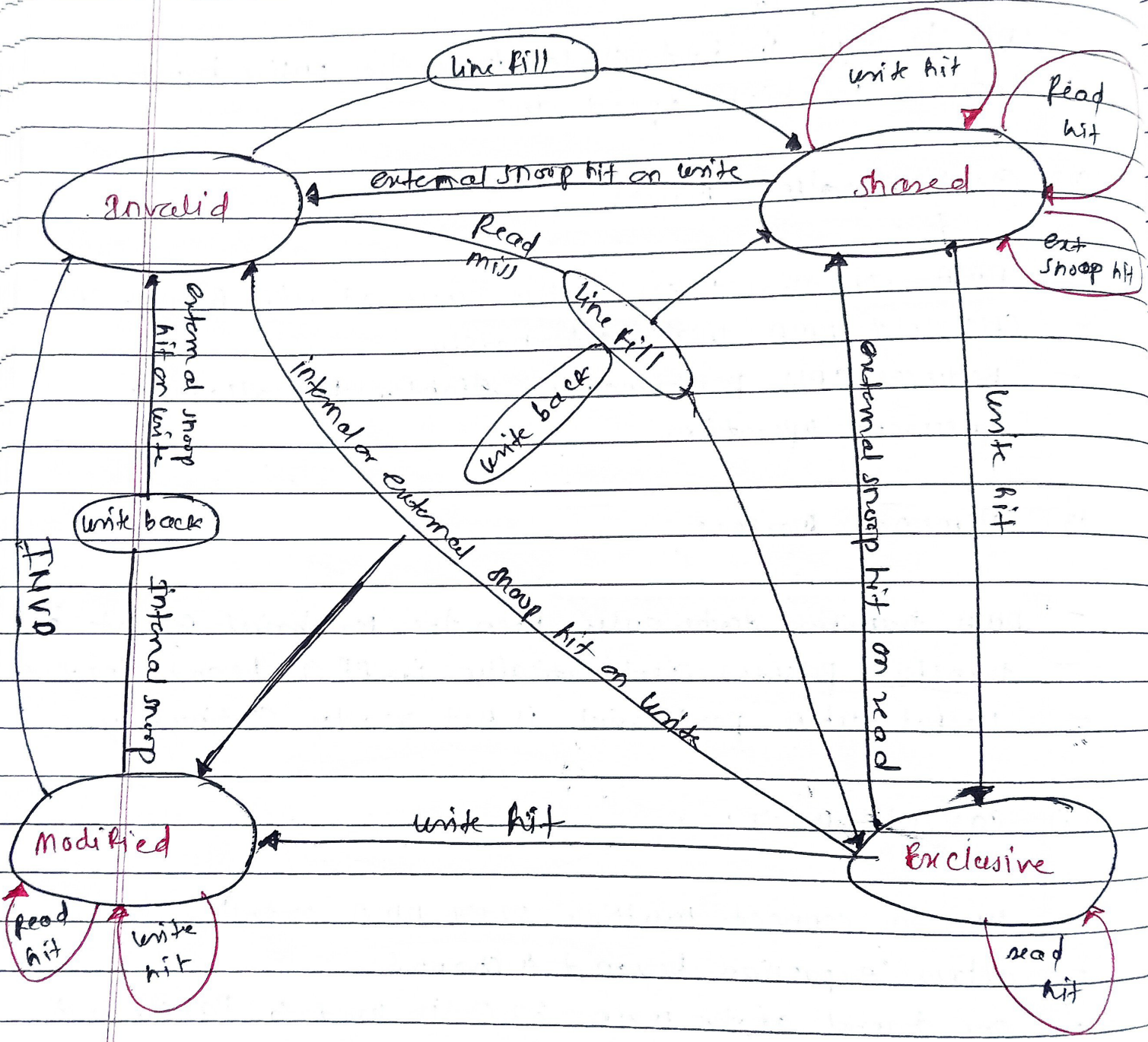
(4) Cascade mode :

- Used to connect multiple 8257 DMA controller together
- allows expansion beyond 4 channel
- one channel of the main 8257 is used to cascade a slave 8257
- The main controller passes control signals to the slave

Q) explain MESI protocol

Ans:

The MESI protocol is cache coherence protocol used in multiprocessor systems to ensure that all caches have a consistent view of memory.



PAGE No.
DATE / /

★ MESI Stand for:

(1) M – Modified
The Cache line has update data.
the data is different from the main memory
The only cache that has this data
Must write back to memory before being replaced

(2) E – Exclusive
The Cache line is same as main memory
it exists only in this cache
can be modified without telling others

(3) S – Shared
data is same as main memory
may exist in multiple caches
Read-only state.

(4) I – Invalid
The Cache line is not valid
Cannot be used.

★ How it works?

Cache communicate with each other to update states
When:
A CPU reads or write data
a Cache line is replaced or invalidated.

PAGE No.
DATE / /

* Why MESI is useful:

- Maintains data consistency across multiple caches
- Reduce unnecessary memory access
- Improve performance in multi-core processor.

Q) explain segment descriptor of 80386 processor?

Ans:

A segment descriptor is a 8-byte (64-bit) entry that provides detailed information about a memory segment.

it is used in protected mode to define the properties of code, data, stack segments.

7	Base (B31-B24)	G	D	0	AV	Limit (L19-L16)
5	Access right byte	Base address (B23-B16)				
4	Base address (B15-B16)					
3	Limit (L5-L0)					
1						

it is 8 byte (64 bit), divided like this:

(1) Base Address (32 bits) - starting address of segment
- spread across bytes 2, 3, 4 and 7

(2) Limit (20 bits) - size of segment
- spread across bytes 0, 1, and part of 6

(3) Access byte (1 byte) - tell if it's code/data, read/write allowed.
- in byte 5

(4) Flags (G, D, AV):

G - Granularity
D - Default size
AV - Available for system
- in byte 6

PAGE No.
DATE / /

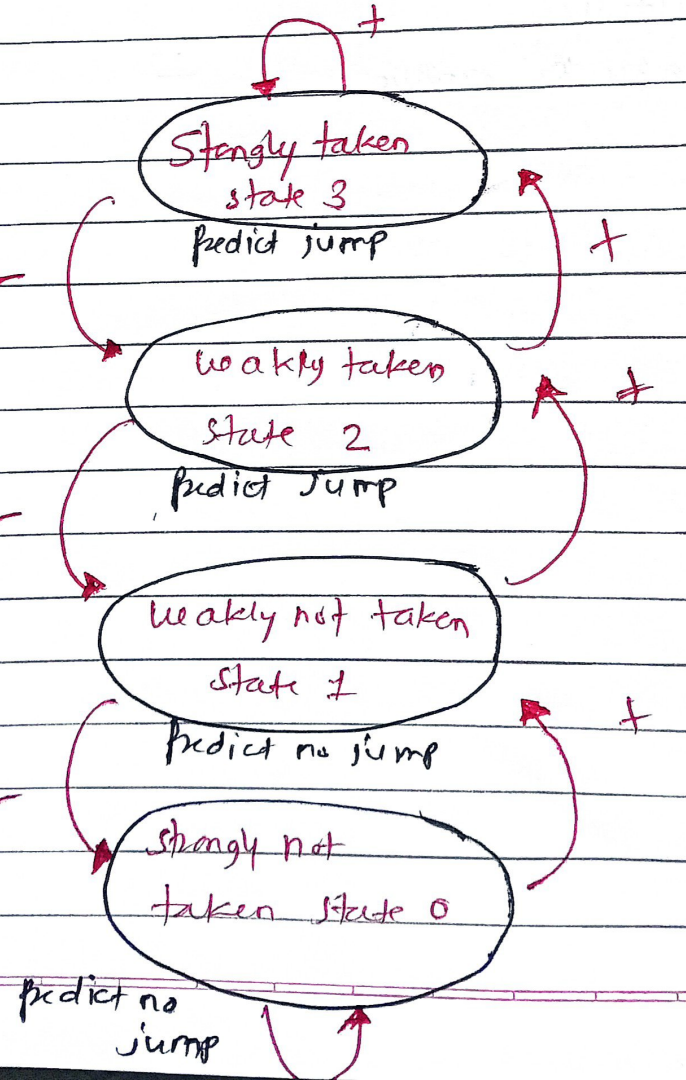
Q) explain branch prediction mechanism of pentium processor

Ans:

The pentium processor uses a dynamic branch prediction method to improve performance by predicting the outcomes of branch instruction. (like jumps or loops) before they are executed. This helps avoid delays in the instruction pipelines.

* Why is it needed?

In pipeline CPU like pentium, if a branch (eg. jump) is ~~misplaced~~ mispredicted, the pipeline must flush (clear instruction), which wastes time. So predicting correctly improves speeds.



+ → branch was taken
− → Branch was not taken

Pentium uses a 4-state branch prediction system based on a 2-bit counter.
It avoid changing prediction on a single incorrect guess.

State	Name	Prediction
0	Strongly not taken	Predict no jump
1	Weakly not taken	Predict no jump
2	Weakly taken	Predict jump
3	Strongly taken	Predict jump

- if predict is correct, stay in the same state
- if a prediction is wrong:
  - move one step toward the opposite prediction
  - This makes prediction stable and accurate.