Date
Page

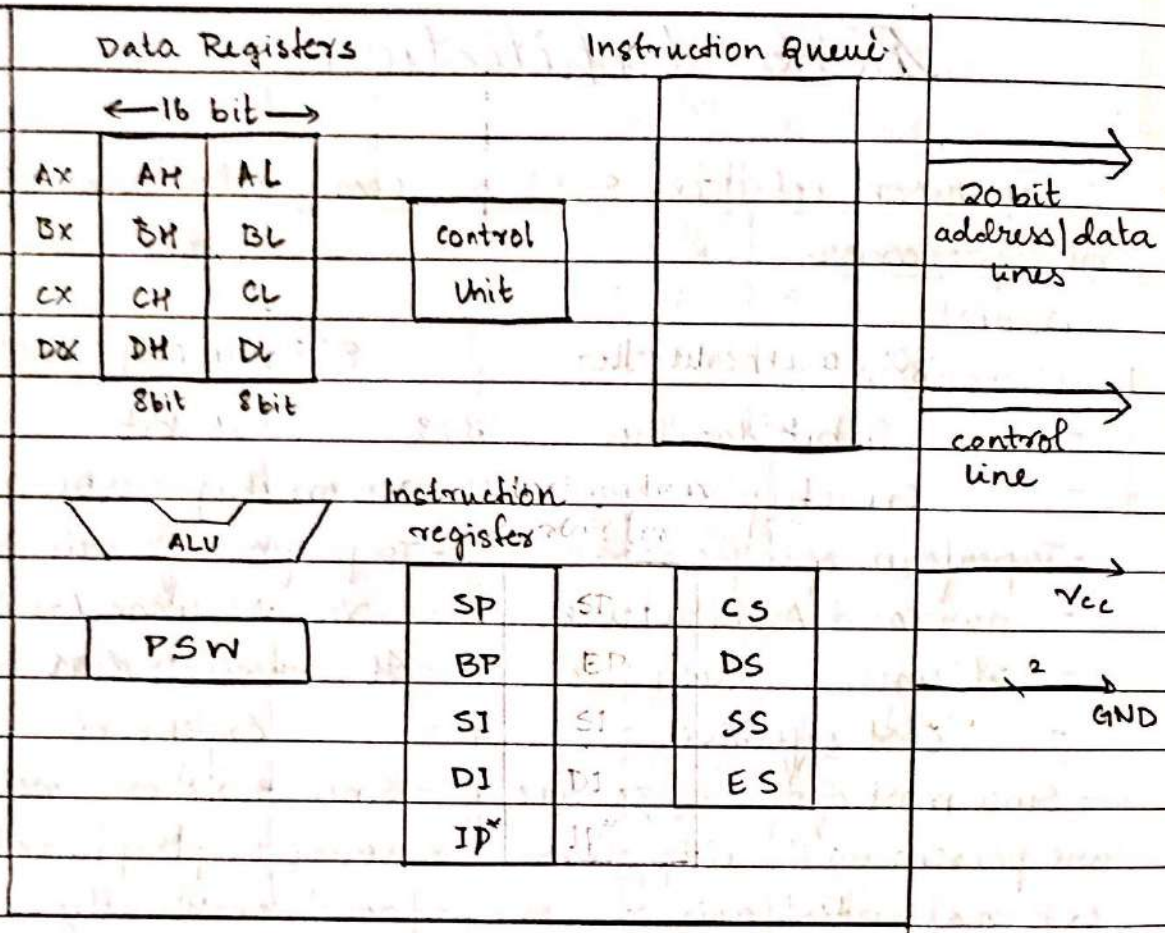
Unit - 01

8086 Architecture

* Comparison between 8051 microcontroller and 8086 microprocessor :

8051 microcontroller	8086 microprocessor
- 8 bit data bus	- 16 bit
- On chip	- No on chip resources
- To perform specific tasks	- To perform generic tasks
- Harvard Architecture	- Van Neuman Architecture
- It consumes less power	- It consumes more power
- Cost effective	- Expensive
- Since most of the resources are present on the chip, less external interfacing.	- Since there are no on chip resources, peripherals are interfaced externally.
Specifications:	Specifications:
64 kB RAM	20 bit address lines
4kB ROM	16 bit data lines
5 interrupts	2^{16} = 65535 bytes

★ Architecture of 8086 microprocessor:

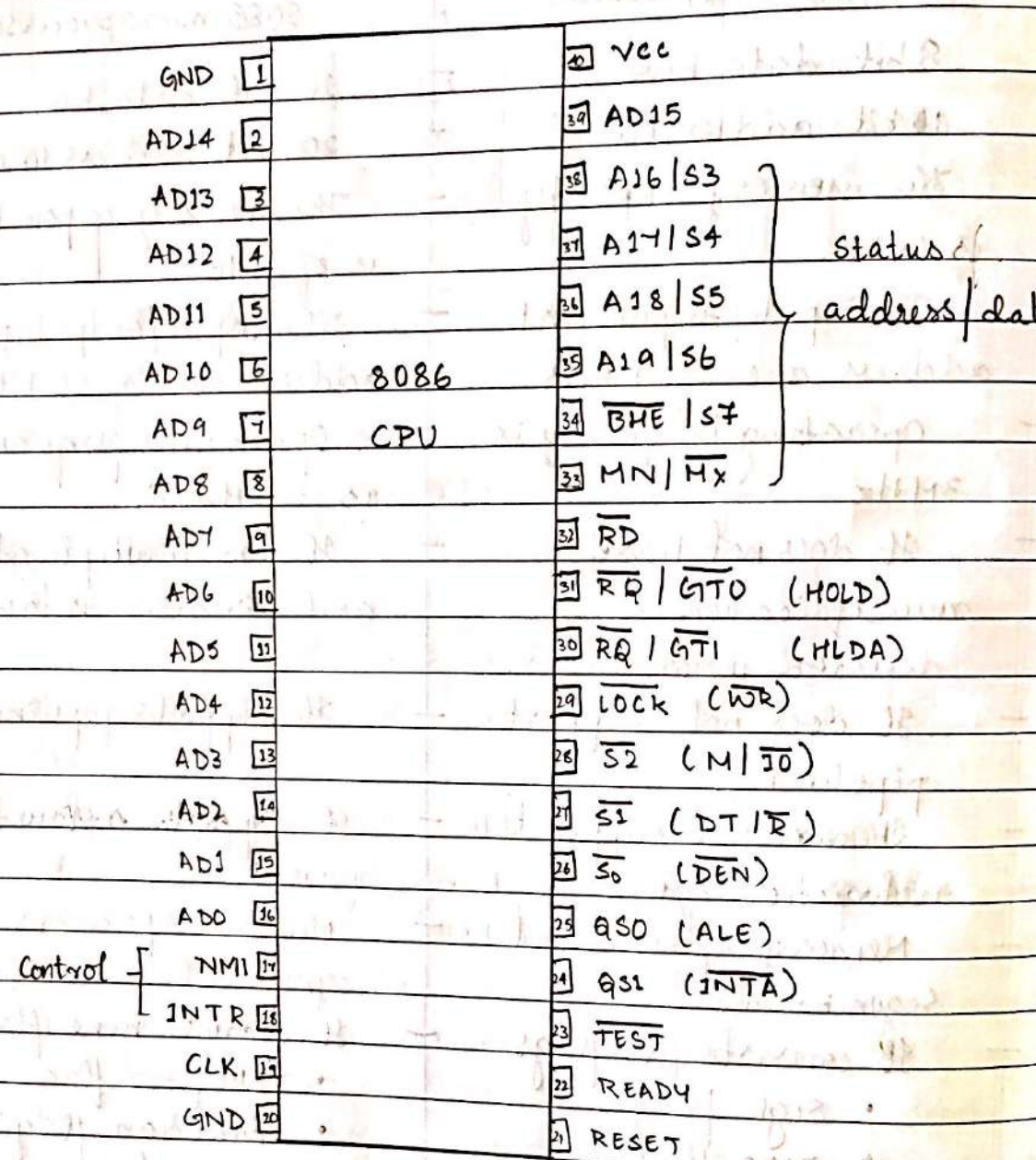


AX - 16 bit accumulator divided into two 8 bit registers
BX - 16 bit base register divided into two 8 bit registers
CX - 16 bit counter register divided into two 8 bit registers
DX - 16 bit data register divided into two 8 bit registers.
SP - Stack pointer : points the topmost item of stack.
BP - Base pointer : accesses parameters passed by the stack.
SI - Source Index : used in pointer addressing of data.
DI - Destination Index ; used in pointer addressing of data.
PSW - Program Status word : 16 bit
IP - Instruction pointer.
8086 consists 3 buses : A-bus, B-bus, C-bus

* Difference between 8085 microprocessor and 8086 microprocessor.

8085 microprocessor	8086 microprocessor.
– 8 bit data bus	– 16 bit data bus
– 16 bit address lines	– 20 bit address lines.
– The memory capacity is 64kB.	– The memory capacity is of 1MB
– The input/output port address are of 8 bits	– The input/output port address are of 16 bits.
– Operating frequency is 3MHz	– Operating frequency is 5MHz
– It does not have multiplication and division instructions.	– It has multiplication and division instructions
– It does not support pipelining.	– It supports pipelining.
– It does not support instruction queue.	– It supports instruction queue.
– Memory space is not segmented.	– Memory space is segmented.
– It consists five flags • sign flag • zero flag • auxillary flag • parity flag • carry flag.	– It consists nine flags. • overflow flag • direction flag] control flags • interrupt flag] • trap flag • sign flag • zero flag • auxillary carry flag • parity flag • carry flag

* Pin configuration of 8086 microprocessor:



QS0	QS1	Operation
0	0	No operation
0	1	First byte operand from queue
1	0	Empty queue
1	1	Subsequent byte from queue

S2	S1	S0	
0	0	0	Interrupt Queue
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive.

There are two grounds, one is internal and one external ground.

NMI : Non maskable interrupt.

INTR : Interrupt

$\overline{BHE}$: Bus High Enable (edge triggered)

MN/$\overline{MX}$: High - minimum mode } After 4 clock cycles
Low - maximum mode

$\overline{RD}$: Read (active low)

$\overline{RQ}$: Request

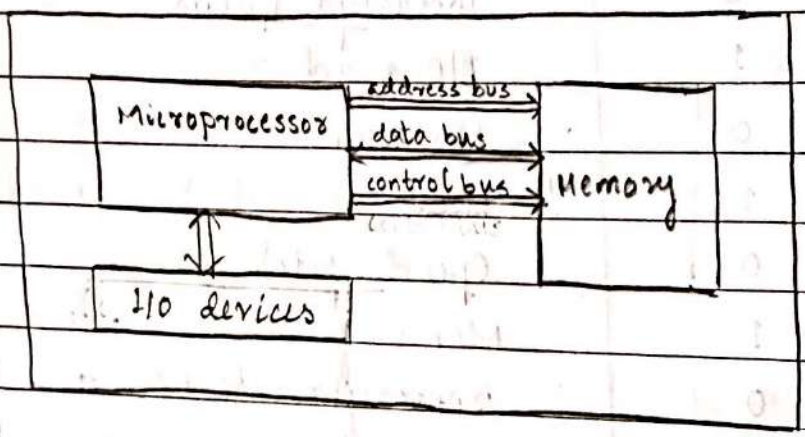
$\overline{LOCK}$: When a process is occurring no other command is sent to the CPU

ALE : Address Latch Enable

$\overline{TEST}$: To test the read and write signal

READY : If ready to execute or not.

* Computer System



write $\overline{WR}$
address → data → control
read RD
address → control → data

Memory

a. primary : RAM, ROM
temporary permanent
b. secondary

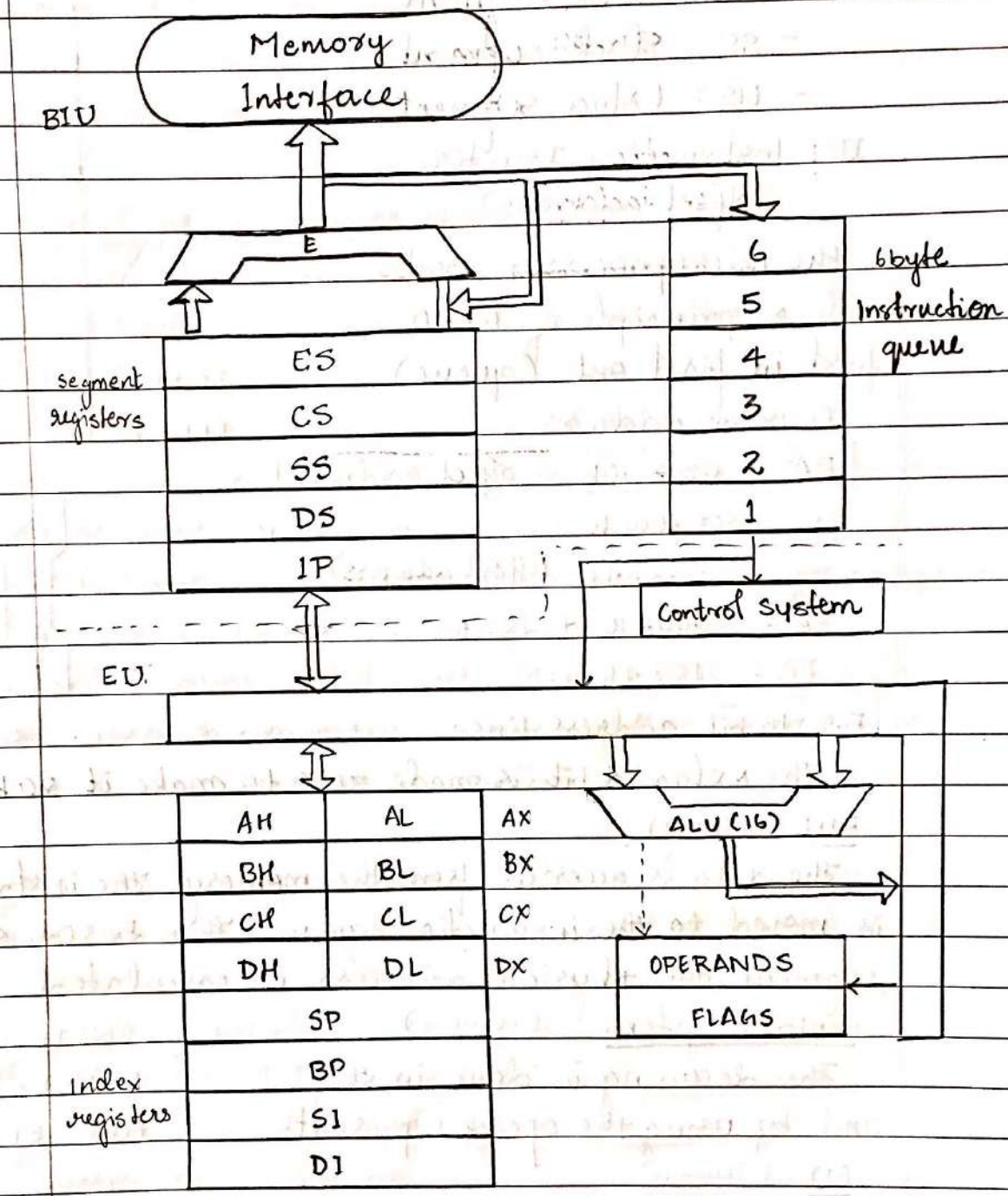
SRAM : Static RAM : flipflops : faster
DRAM : Dynamic RAM : capacitors : cheaper
SDRAM : Synchronous Dynamic RAM

High level Language : HLL : a+b=c : compiler
Assembly Language : ASM : add B, C : assembler
Low level Language : LLL | Bin : 0110111 :
opcode

Storage

2^{10} = 1KB
2^{20} = 1MB
2^{30} = 1GB
2^{40} = 1TB

* **Programming model of 8086 with architecture**



The processor consists of two parts : BIU and EU :
Bus Interfacing unit — Execution unit

Pipelining : 1. Fetching
→ Decoding (control system)
2. Execute

There are four segments, they are accessed by segment registers
- CS : code segment
- DS : data segment
- SS : stack segment
- ES : extra segment

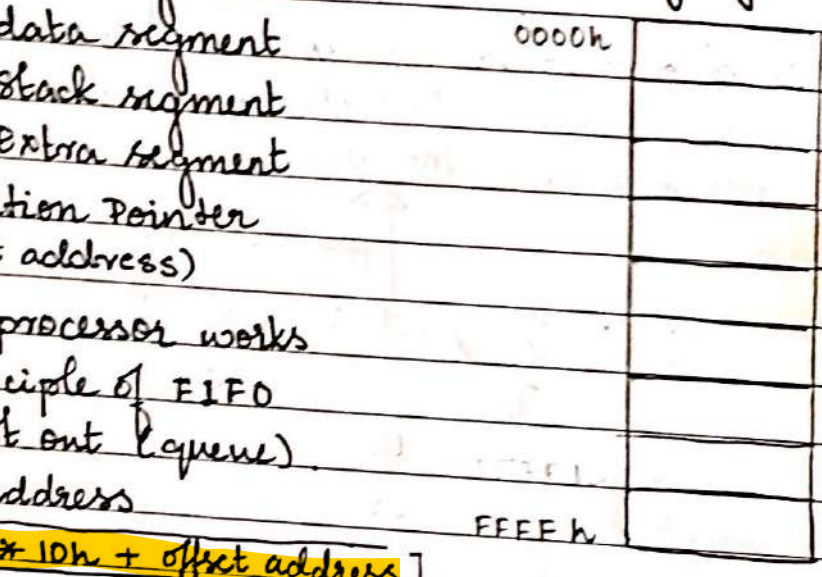
IP : Instruction Pointer (offset address)

The microprocessor works with a principle of FIFO (first in first out queue).

Physical address

[PA = CS * 10h + offset address]

Ex : CS = 1000h
IP = 1234h (offset address)
PA = 1000 * 10 + 1234
PA = 11034h



For 16 bit address line:
The extra 4 bit is made zero to make it 20 bit.

BIU (Fetching)
The data is accessed from the memory. The instruction is moved to the instruction queue. Then based on the segment the physical address is calculated.

control system (decoding)
The decoding is done in this unit by using the opcodes present.

EU (Execution)
The instruction is executed respectively by the ALU.

Flags : PSW : program status word.

NOTE
MOV BX, CX : 16 bit
MOV BL, 05h : 8 bit

NOTE: To find the physical address we multiply the segment address by 10h because it is the hexadecimal equivalent of the decimal number 16. This is because if we multiply with 16 it will misinterpret as 16h (22 in decimal) but all the operations in 8086 is done in hexadecimal.

* **Advantages of using segment registers:**

1. Allow the memory capacity to be 1MB even though the addresses associated with the individual instructions are only 16 bit wide.
2. It allows instruction, data or stack portion of a program to be more than 64 kB long by using more than 1 code data or stack segment.
3. It fecilitates the use of separate memory area for program its data and stack
4. It permits a program and/or its data to be put into different areas of memory each time the program is executed.

* **Flags:**

There are 9 flags. 6 conditional flags and 3 control flags.

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

← 16 bit →

OF : Overflow flag
DF : Direction flag
IF : Interrupt flag
TF : Trap flag
SF : Sign flag
ZF : Zero flag
AF : Auxillary flag
PF : Parity flag
CF : Carry flag

NOTE: For a branch instruction the architure of pipelining does not work effectively.
If takes one cycle to execute even address and two cycles to execute odd address.

* **Addressing Modes:**

The way in which an operand is specified is called its addressing mode. There are two types of addressing modes:

a. Data Address
b. Branch Address

- The Data Addressingare

i. Immediate: The datum is either 8 bit / 16 bit long and is part of the instruction.

ii. Direct: The 16 bit effective address of the datum is part of an instruction.

iii. Register: The datum is the register, that is specified in the ~~register~~ instruction.

Ex: A 16 bit operand: A register maybe SP, SI, DI, AX, BX, CX, DX

A 8 bit operand: A register maybe AH, AL, BH, BL, CH, CL, DH and DL.

iv. Register Indirect: The datum is the effective address present in SI, DI or BX

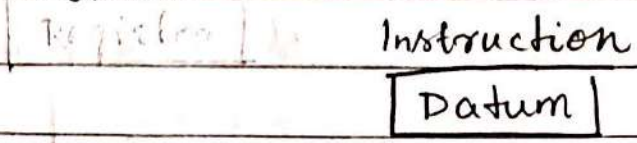
v. Register relative: The datum is at an effective address formed by adding displacement with content, the registers BX, BP, SI, DI in the default (DS or ES) segment.

vi. Based Indexed: The effective address of data is formed by adding content of base register BX to content of index regis

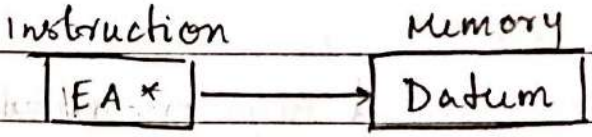
vii. Relative based Indexed

Keerthana Ashok

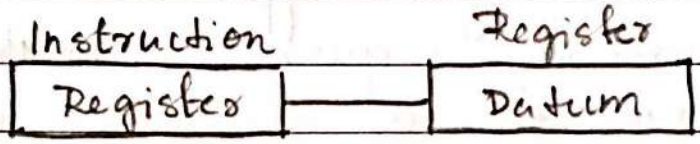
a. Immediate



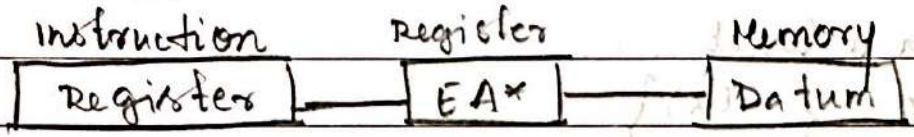
b. Direct



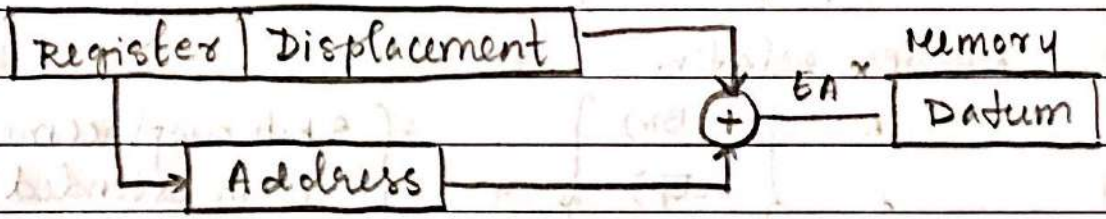
c. Register



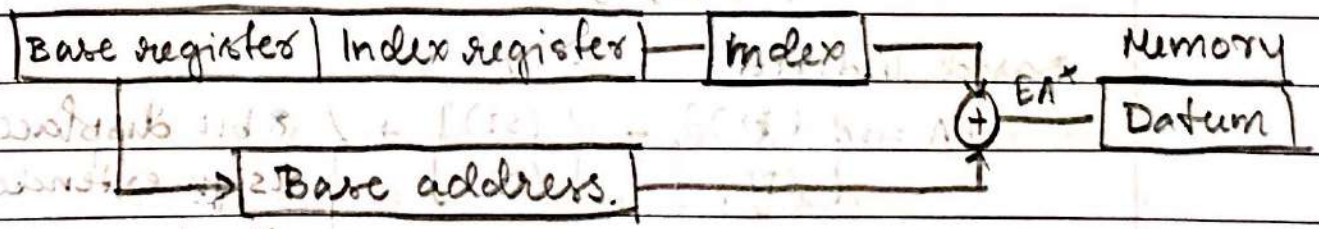
d. Register indirect



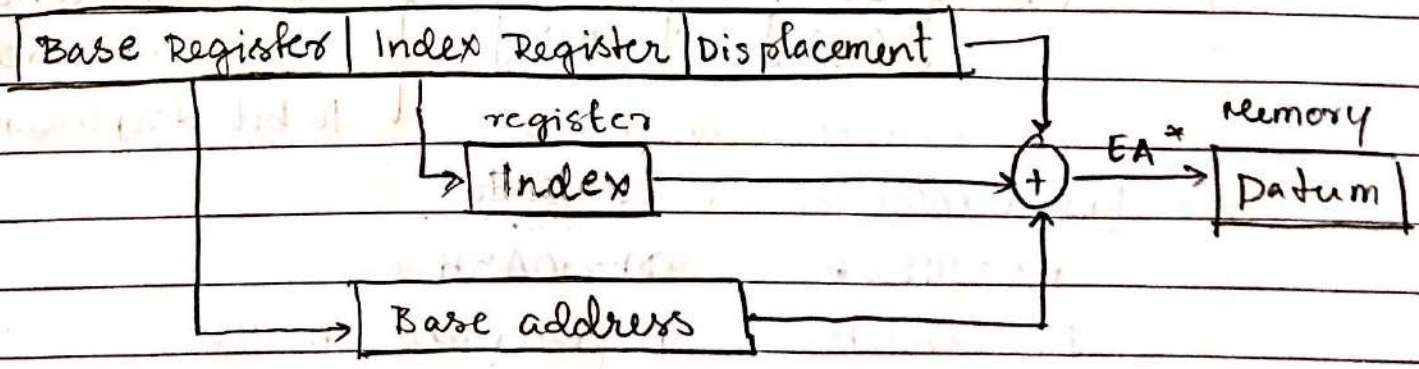
e. Register relative



f. Based Index



g. Relative Based Index



Q: Given: BX = 0158H
DI = 10A5H
Displacement = 1B57H
DS = 2100H

Consider DS as the segment register by using various addressing modes. calculate effective and physical address produced by the quantites of various addressing modes:

NOTE:

Register indirect

$$EA = \begin{Bmatrix} (BX) \\ (DI) \\ (SI) \end{Bmatrix}$$

Register relative

$$EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{(sign extended)} / \\ \text{16 bit displacement} \end{Bmatrix}$$

Based Indexed

$$EA = \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix}$$

Relative Based Indexed

$$EA = \begin{Bmatrix} (BX) \\ (IP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{(sign extended)} \\ \text{16 bit displacement} \end{Bmatrix}$$

— Direct mode:

BX = 0158H DI = 10A5H
DS = 2100H Displacement = 1B57H

Effective address

EA = 1B57H

Physical address:

PA = segment * 10H + EA

PA = DS * 10H + EA

PA = 2100H * 10H + 1B54H

PA = 22B54H

Register Relative

Assuming relative register address as BX = 0158H

Effective address

EA = BX + displacement

EA = 0158H + 1B54H

EA = 1CAFH

Physical address

PA = segment * 10h + EA

PA = DS * 10h + EA

PA = 2100H * 10H + 1CAF

PA = 22CAFH

Register Indirect

Assuming register relative address as BX = 0158H

Effective address

EA = BX

EA = 0158H

Physical address

PA = DS * 10H + EA

PA = 200H * 10H + 0158H

PA = 21158H

Based Indexed

Effective address

EA = BX + DI

EA = 0158H + 10A5H

EA = 11FDH

Physical address

PA = DS*10H + EA

PA = 2100H * 10H + 11FDH

PA = 221FDH

Relative Based Indexed

Effective address.

EA = BX + DI + displacement

EA = 0158H + 10A5H + 1B57H

EA = 2D54H

Physical address:

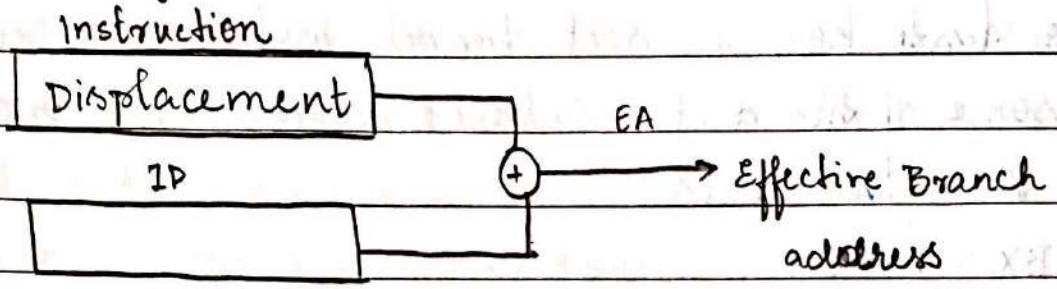
PA = DS*10H + EA

PA = 2100H * 10H + 2D54H

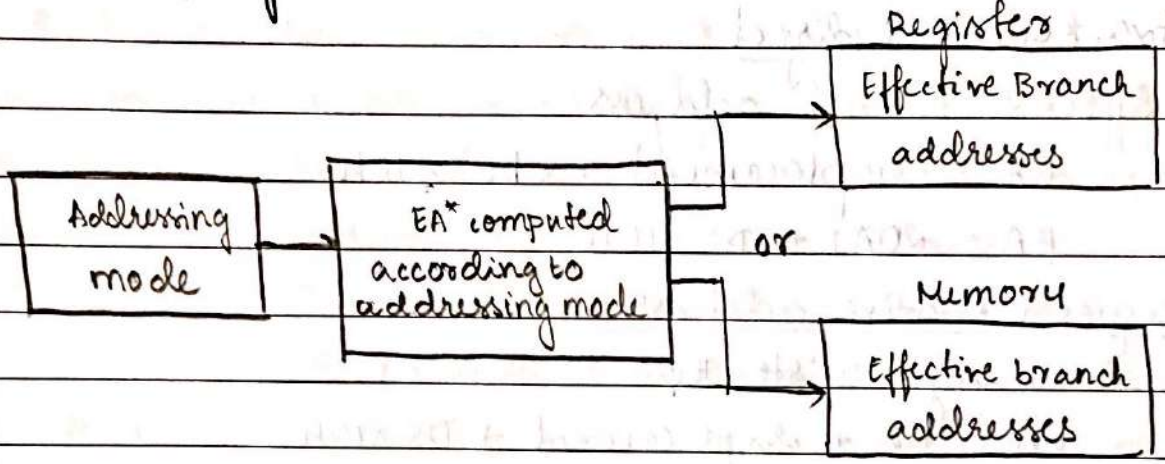
PA = 23D54H

- The Branch addressing are:
  i. Intra segment direct
  ii. Intra segment indirect
  iii. Intersegment direct
  iv. Intersegment indirect

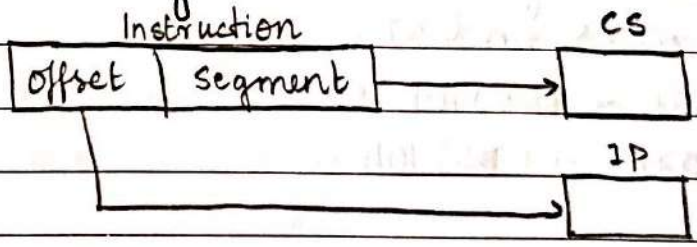
i. Intra segment direct



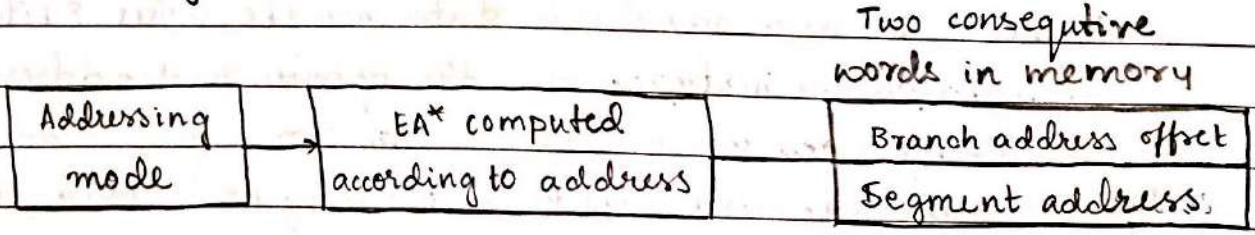
ii. Intersegment indirect



iii. Intersegment direct



iv. Intersegment indirect



Q: Demonstrate how indirect branch instructions work with some of the data related addressing modes by using the given data.

BX = 1256H

SI = 528F H

displacement : 20A1 H

- Direct Addressing

Effective Branch address

EA = Displacement + (DS) x 10h

EA = 20A1 + DS x 10h

- Register Relative addressing

Assuming register BX

EA = BX + displacement + DS x 10h

EA = 1256 + 20A1 + DS x 10h

- Based indexed addressing

Assuming registers BX and SI

EA = BX + SI + (DS) 10h

EA = 1256 + 528F + (DS) 10h

* Instruction Formats:

The instructions vary from 1 to 6 bytes in length. Displacements and immediate data may be either 8 bits or 16 bits long based on the instruction. The opcode and addressing mode designations are in the first or 1 or 2 bytes of an instruction. The opcode / addressing mode bytes may be followed by

- No additional bytes
- A 2-byte effective address (direct addressing)
- A 1 or 2 byte immediate operand
- A 1 or 2 byte displacement followed by a 1 or 2 byte immediate operand
- A 2 byte displacement and a 2 byte segment address (direct intersegment addressing)

Special One byte Indicators

- **W bit :** If an instruction can operate on either a byte or a word, the op code includes a W-bit which indicates whether a byte or a word is being accessed.

  W=0 : byte

  W=1 : word

- **D bit :** For double operand instructions one of the operands must be a register specified by a REG field. For these instructions the D-bit is used to indicate whether the register specified by REG is the source or the destination operand

  D=0 : source operand

  D=1 : destination operand.

- **S bit :** An 8-bit 2's complement number can be extended to a 16 bit 2's complement number by setting all the bits in the high-order byte equal to the MSB in the lower order byte. This is referred to as sign extension. The S bit appears with the W-bit in the immediate to register/memory add, subtract and compare instructions

S	W	
0	0	8 bit operation
0	1	16 bit operation with 16 bit immediate operand.
1	0	
1	1	16 bit operation with a sign extended 8-bit immediate operand.

If there are two op code/addressing mode bytes then the second byte is of one of the following two forms:

MOD	OP CODE	R/M

MOD	REG	R/M

First form : single operand instructions
second form: double operand instructions.
in which REG specifies a register where a source or a destination operand based on value of D-bit.

Register Address	W=1	W=0	Register Address	segment register
000	AX	AL	00	ES
001	CX	CL	01	CS
010	DX	DL	10	SS
011	BX	BL	11	DS
100	SP	AH		
101	BP	CH		
110	SI	DH		
111	DI	BH		

Instruction Execution Timing:

The execution time of an instruction can be determined by multiplying the number of clock cycles needed to execute the instruction by the clock period.

It can be expressed as the sum of a basic execution time plus the time required to calculate the effective address if a memory operand is involved.

It is assumed that the instruction to be executed has already been prefetched and stored in the instruction queue otherwise an additional time necessary to fetch the instruction must be added.

UNIT - 02.

Assembly Language and Instruction Set

* **Assembly Language Instruction Format:**
  Label : mnemonics operands, operands ; comments.

* **Types of Instructions:**
1. Data transfer instructions
2. Arithmetic instructions
3. Branch instructions
4. Loop Instructions
5. NOP and HALT instructions
6. Flag Manipulation instructions
7. Logical Instructions.
8. Shift Rotate Instructions.

* **Data Transfer Instructions:**
  - MOV DST, SRC — (DST) ← (SRC) : Move
  - LEA REG, SRC — (REG) ← SRC : Load Effective Address
  - LDS REG, SRC — (REG) ← (SRC) : Load DS with pointer
    (DS) ← (SRC+2)
  - LES REG, SRC — (REG) ← (SRC) : Load ES with pointer
    (ES) ← (SRC+2)
  - XCHG OPR1, OPR2 — (OPR1) ↔ (OPR2) : Exchange
  - XTRANS : Translate.

Instruction MOV, which is used for moving the data from source to distination or destination to source using different addressing modes where its data can be moved in the following ways:

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Segment Register	Register
Register	Segment Register
Memory	Segment Register

MOV instruction cannot be used to move from

a. memory → memory
b. immediate → segment registers
c. segment register → segment register.

- XLTA / XLTAB
- IN
- OUT
- PUSH } Decrement Stack
- POP } Increment
- POPF
- LAHF : Load
- SAHF : Store

NOTE:
By data transfer instruction no flags are effected.

* Arithmetic Instructions:

1. Signed : add, sub, mul, div
2. unsigned : add, sub, mul, div
3. Packed BCD : add (DAA), sub (DAS)
1. Unpacked BCD : add, sub, mul, div

- ADD : ADD AL, 15H | (AL) ← (AL+15H).
- ADC : add with carry.
- SUB : SUB AL, 15H (AL) ← (AL-15H)
- SUBB : subtract with borrow.

NOTE :

The flags affected are : Parity flag, Zero flag, carry flag, Sign flag, Overflow flag, Auxillary flag.

- MUL : MUL BL → AL x BL and result is stored in AL.
- IMUL : signed multiplication
- DIV : DIV BL → AX / BL : 16bit / 8bit | DIV BX → DX AX / BL = 32/16 Q: DX R = AX
- IDIV : signed division.
- INC : Increment data / register pointer by 1
- DEC : Decrement data / register pointer by 1.
- DAA : Decimal Adjust after BCD Addition.
- DAS : Decimal Adjust after BCD Subtraction
- AAA : ASCII Adjust for addition.
- AAD : Adjust AX register for division.
- AAM : Adjust AX register for multiplication.
- AAS : ASCII Adjust for subtraction.
- CBW : convert signed byte to signed word
- CWD : convert signed word to signed double word.
- NEG : Obtain 2's complement of the content of an 8 bit / 16 bit specified register or memory location
- CMP : Compare

Q: let AL = 59 bcd and BL = 35 bcd. Add AL and BL

AL = 59 bcd

BL = 35 bcd

ADD AL, BL

DAA

```
  59 H
+ 35 H
------
  8E H
+ 06   ; DAA
------
  94   BCD
```

Q: AL = 74 bcd and BL = 92 bcd.

SUB BL, AL
DAS

```
 92 H
-74 H
 1E
-06     DAS
 18 BCD
```

* Logical Instructions:

- AND : bitwise logical AND
- OR : OR AH, CL : bitwise logical OR
- XOR : bitwise logical XOR
- NOT : NOT BX
- TEST : Performs logical AND operation of specified operand with another specified operand. There will be no change in the distination or source. After execution the status can be checked at
  - → Sign flag
  - → Parity flag
  - → carry flag
  - → Zero flag

* Rotate Instructions:

- RCL : Rotate with carry left
  RLC dst, count
- RCR : Rotate with carry right
  RCR dst, count
- ROL : Rotate left without carry
  ROL dst, count
- ROR : Rotate right without carry
  ROR dst, count

Date
Page

* **Shift Instructions:**

SHR : Shift Right

SHL : Shift Left

NOTE:

Swapping of higher and lower nibble is not possible in 8086 as no instruction is present. Then this can by done by rotation.

In shift left / right zero is appended. where as in arithmetic shift left / right we append one.

* **Transfer control Instructions** : (Branch Instructions)

Conditional branch instructions

- JZ : Jump on zero or equal (ZF = 0)
- JNZ: Jump on non zero or not equal (ZF = 1)
- JS : Jump on sign set (SF = 1)
- JNS : Jump on sign clear (SF = 0)
- JO : Jump on overflow (OF = 1)
- JNO : Jump on no overflow (OF = 0)
- JP : Jump on parity set / even parity (PF = 1)
- JNP : Jump on parity clear / odd parity (PF = 0)
- JB : Jump on below
- JNB : Jump on not below
- JBE : Jump on below or equal
- JNBE : Jump on not below or equal
- JL : Jump on less
- JNL : Jump on not less
- JLE : Jump on less or equal
- JNLE : Jump on not less or equal

Unconditional branch instructions.

- JMP SHORT : Intrasegment direct short jump
- JMP NEAR PTR : Intrasegment direct near jump.

- JMP : Intrasegment / Intersegment indirect jump
- JMP FAR PTR : Intersegment direct far jump

★ **Assembler Directives**

- ASSUME

  ASSUME DS : code is written in data segment
  ASSUME CS : code is written in code segment
  ASSUME SS : code is written in stack segment

- ORG
- SEGMENT
- END
- DB : Define data as byte (8 bit)
- DW : Define data as word (16 bit)
- DD : Define data as double word (4 bytes)
- DQ : 8 byte
- EQU
- MACRO
- PROC
- START
- LABEL
- POINTER

Ex :

```
ORG 1000H
.DATA SEGMENT
DB 50H
DB 30H
.END DATA
.CODE SEGMENT
MOV AH, 50H
MOV BH, 30H
ADD AH, BH
.END CODE
```

* String Instructions

	Instruction	Mnemonic Operands	Descriptions
1.	Move String	MOVS DST, SRC MOVSB	((D1)) ← ((S1)) Byte operand (S1) ← (S1) ± 1 (D1) ← (D1) ± 1
	no flags are affected	MOVSW	word operand (S1) ← (S1) ± 2 (D1) ← (D1) ± 2
2.	compare String	CMPS DST, SRC CMPSB	((D1)) ← ((S1)) Byte Operand (S1) ← (S1) ± 1 (D1) ← (D1) ± 1
		CMPSW	word Operand (S1) ← (S1) ± 2 (D1) ← (D1) ± 2
3.	Scan String	Scan DST scan B	Byte Operand (AL) − ((D1)), (D1) ← (D1) ± 1
		Scan W	word Operand (Ax) − ((D1)) (D1) ← (D1) ± 2

	Instruction	Mnemonic Operands	Descriptions
4.	Load String	LODS SRC	
		LODSB	Byte Operand (AL) ← ((SI)) (SI) ← (SI) ± 1
		LODSW	Word Operand (AX) ← ((SI)) (SI) ← (SI) ± 2
5.	Store String	STOS DST	
		STOSB	Byte Operand ((DI)) ← (AL) (DI) ← (DI) ± 1
		STOSW	Word Operand ((DI)) ← (AX) (DI) ← (DI) ± 2

* Advantages of Move string (MOVS) and compare string (CMPS) over move and compare (MOV and CMP)

- They are only one byte long
- Both operands are memory operands.
- Their auto indexing property need for separate incrementing or decrementing instructions. Thus, decreasing overall processing time

(Advantages of string instructions over data transfer instructions)

Q: Develop an ALP to move the content of a block of memory to another area in memory using MOV and MOVS instructions. (using data transfer and string instructions)

— Using MOV instructions

```
MOV  SI, OFFSET  STRG1
MOV  DI, OFFSET  STRG2
MOV  CX  LENGTH  STRG1
MOVE :  MOV AL,[SI]
        INC SI
        DEC DI
        LOOP MOVE
```

— using MOVS instructions

```
MOV  SI, OFFSET, STRG1
MOV  DI, OFFSET  STRG2
MOV  CX, LENGTH  STRG1
CLD      # clear direction flag : DF = 0
MOVE:  MOVS  STRG1, STRG2
       loop  MOVE
```

* For source operand, the address of the operand is
$(SI + DI) \times 16_{10}$
For destination operand, the address of the operand is
$(DI + ES) \times 16_{10}$

* Differences between Macros and Procedures

MACRO	PROCEDURE
- Requires more memory	- Requires less memory
- Does not require call or return instructions.	- Requires call or return instructions.
- Machine code is generated every time the macro is called.	- Machine code is generated only once
- It executes faster.	- It executes slower
- Used for less instructions	- Used for more instructions

Keerthana Ashok

UNIT - 03

Memory Interfacing

- Pin 33 : MN/$\overline{MX}$
  0: maximum mode : multiprocessing
  1: Minimum mode : single processor.
- Pin 38 and Pin 39.

S_4	S_3	
0	0	: ES
0	1	: DS
1	1	: CS
1	0	: SS

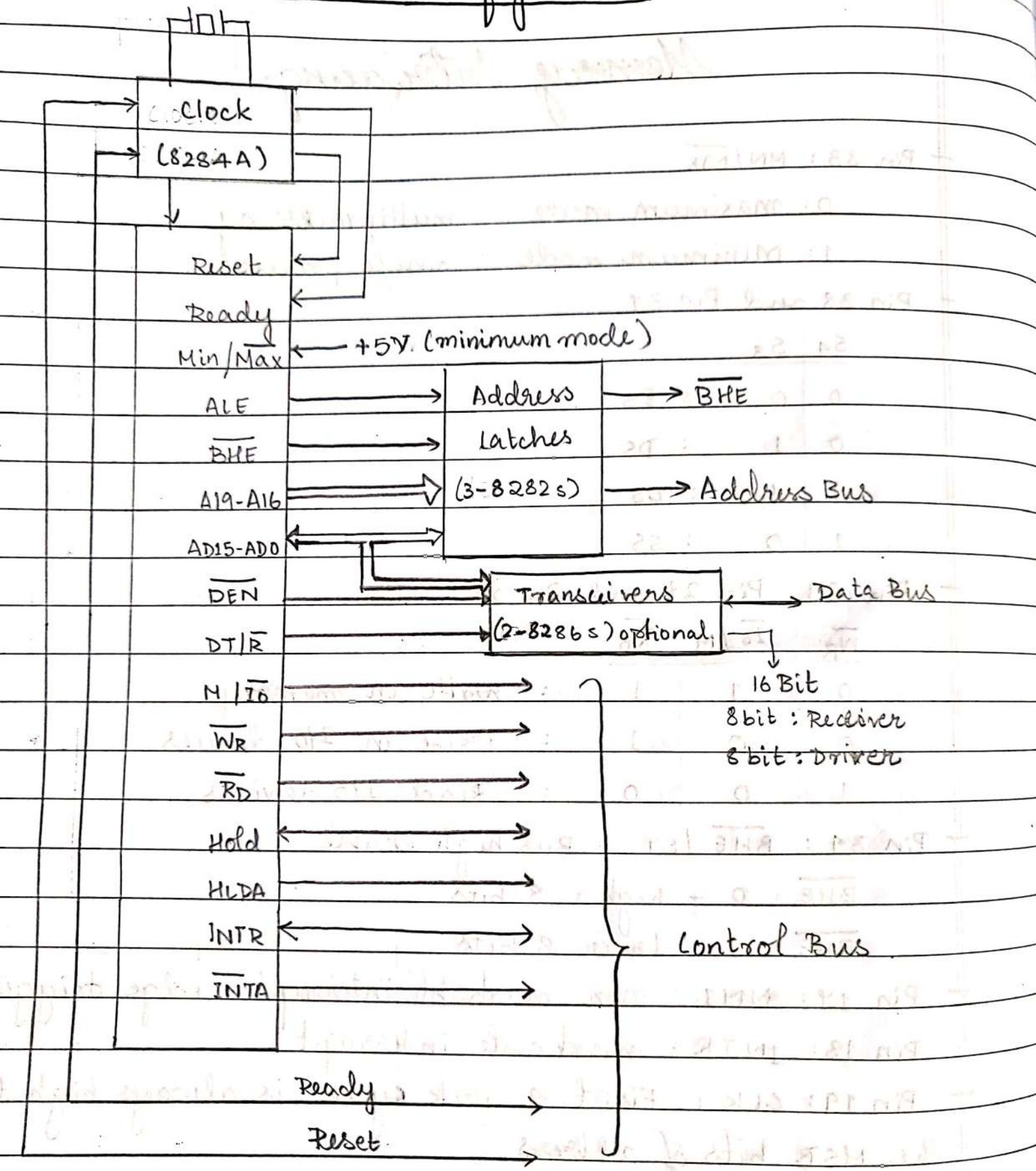
- Pin 29, Pin 28 and Pin 27

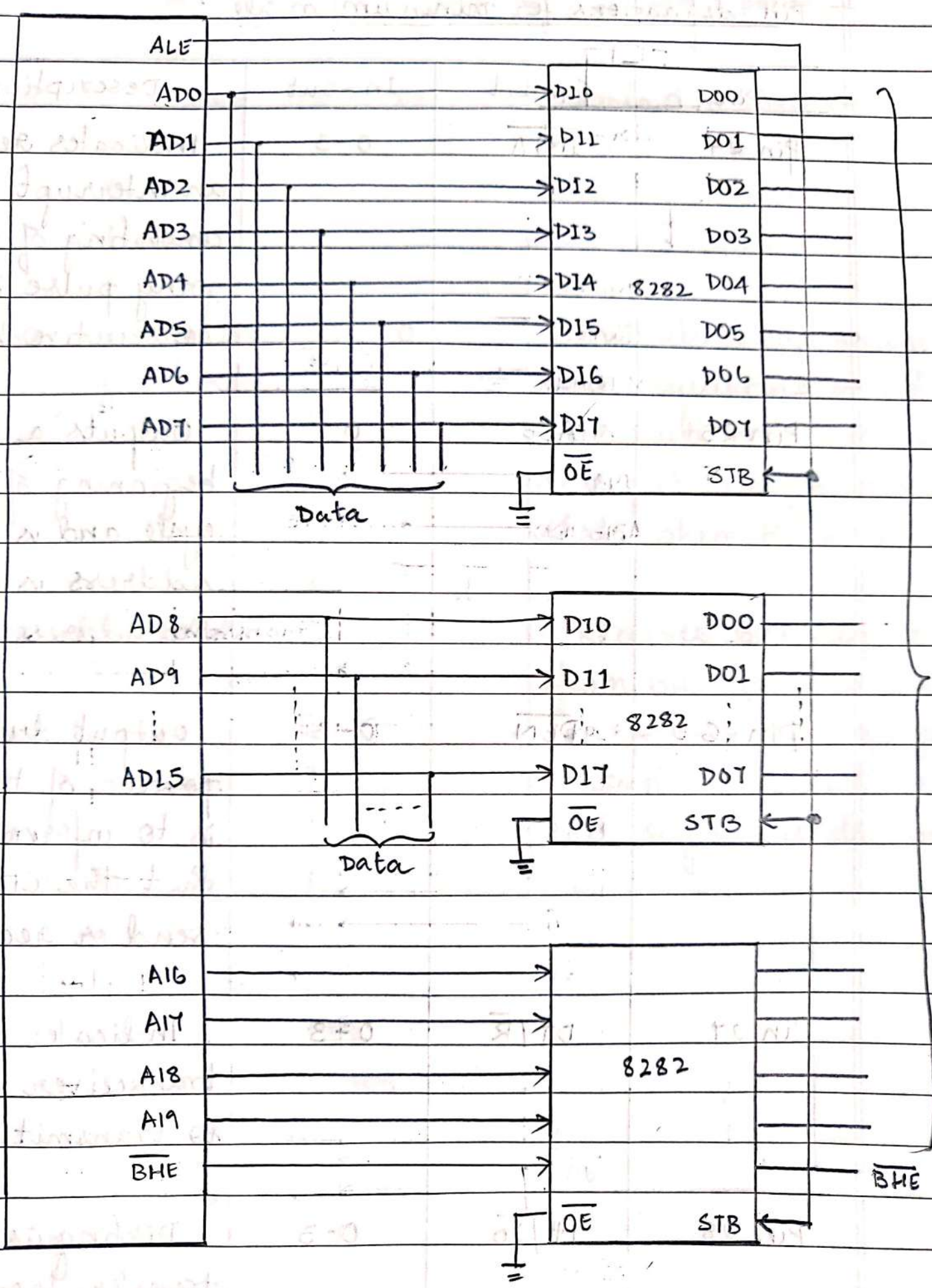
$\overline{W_R}$	$\overline{I_O}/M$	$\overline{R_D}$	
0	1	1	: Write in memory
0	0	1	: Write in I/O devices
1	0	0	: Read I/O devices.

- Pin 34 : $\overline{BHE}$ /S7 : Bus high enable
  $\overline{BHE}$: 0 - higher 8 bits
  $\overline{BHE}$: 1 - lower 8 bits.
- Pin 17: NMI : non maskable interrupt : edge triggered
  Pin 18: INTR : maskable interrupt.
- Pin 19: CLK : First 4 clock cycles is always high to enable the MSB bits of address

8086 minimum mode configuration

Keerthana Ashok



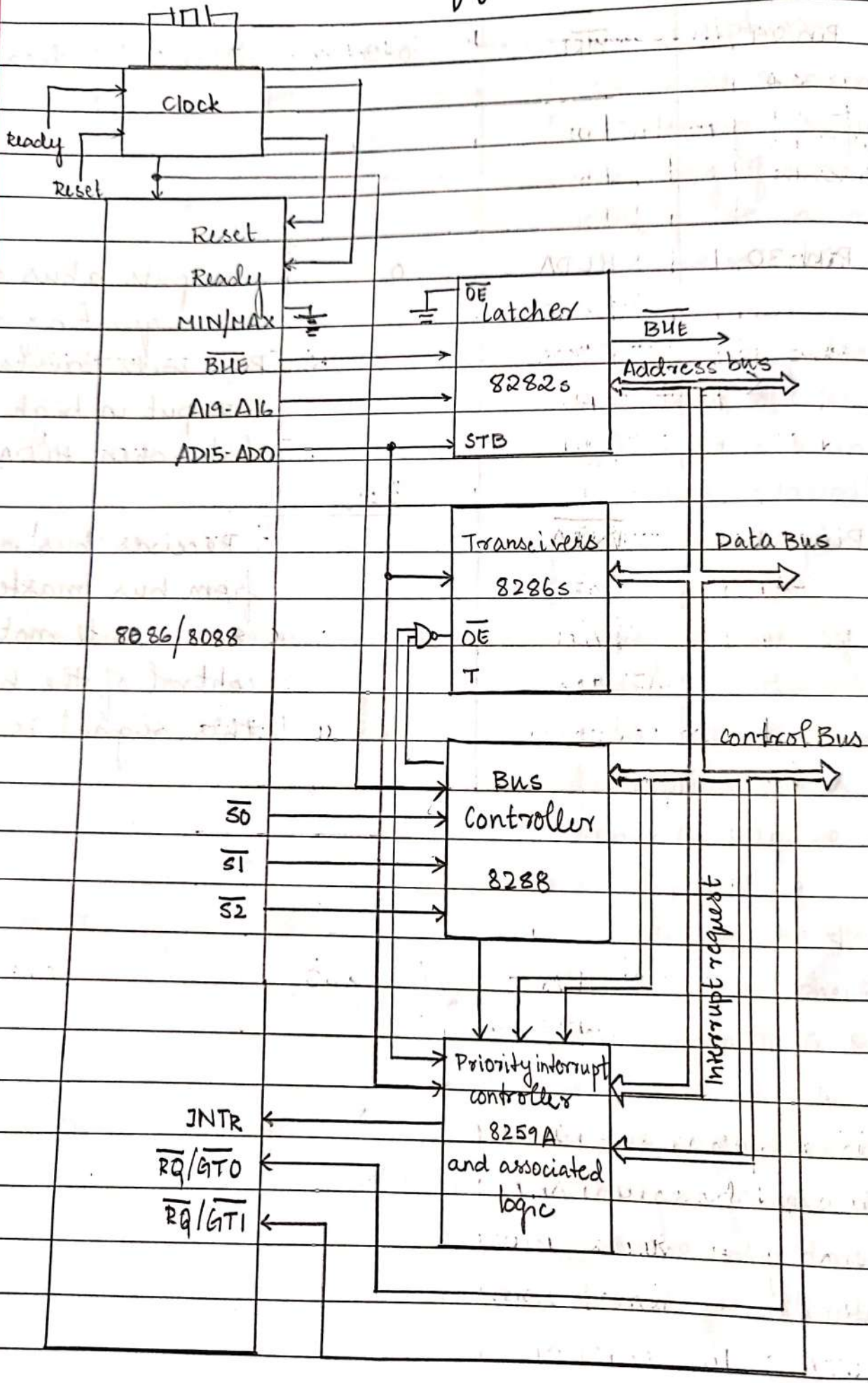


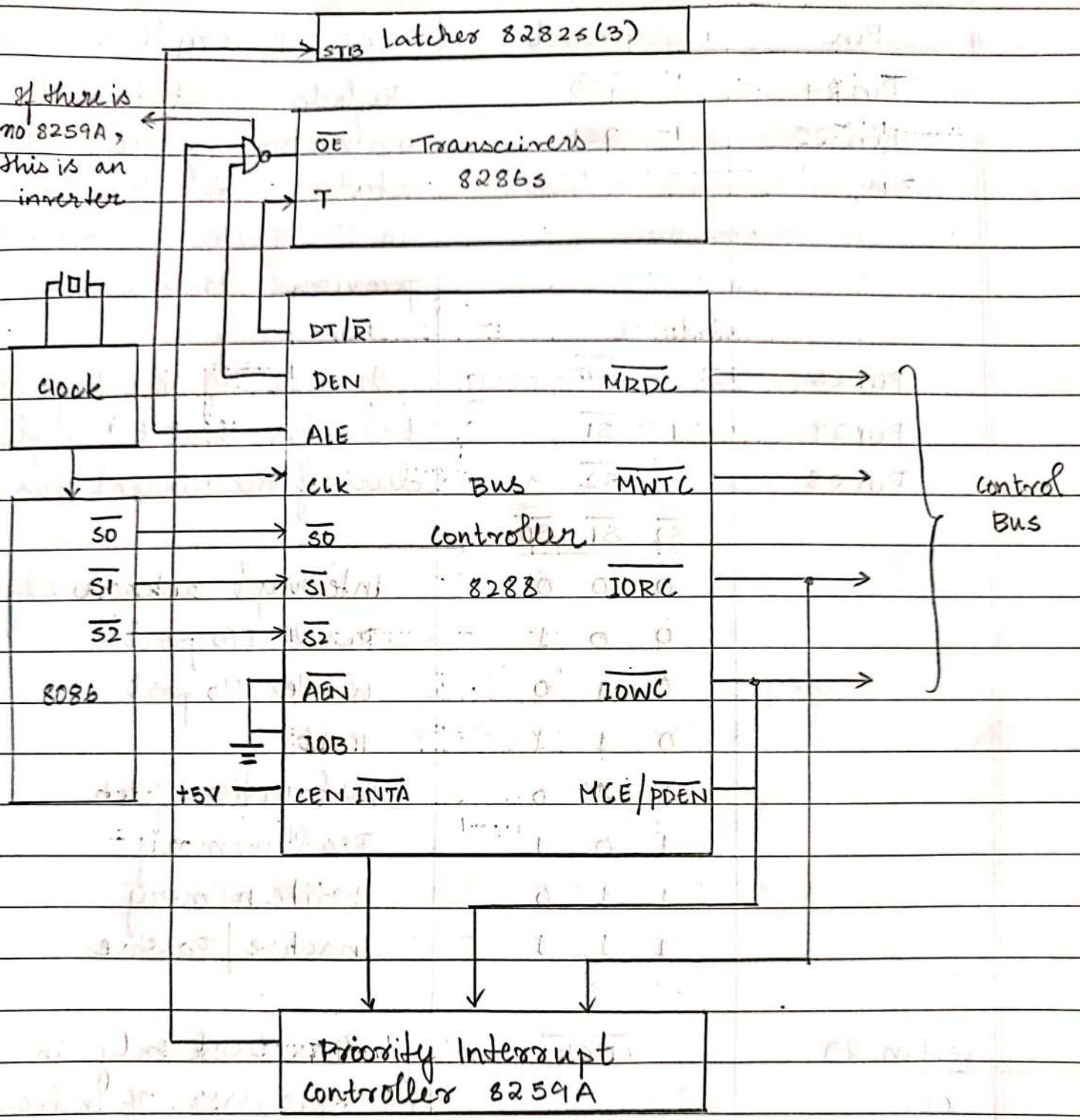
Pin definations for minimum mode

Pin	Symbol	In-out	Description
Pin 24	$\overline{INTA}$	0-3	Indicates recognision of an interrupt request consisting of two negative going pulse in two consecutive bus cycles
Pin 25	ALE	0	Outputs a pulse at the beginning of the bus cycle and is to indicate an address is available on the address pins.
Pin 26	$\overline{DEN}$	0-3	output during the later portion of the bus cycle and is to inform the transceiver that the CPU is ready to send or receive data
Pin 27	DT/$\overline{R}$	0-3	Indicates to the set of transceiver whether they are to transmit or receive data.
Pin 28	M/$\overline{IO}$	0-3	Distinguishes a memory transfer from I/O transfer For memory transfer it is 1 and for I/O transfer it is 0.

Pin	Symbol	In -Out	Description
Pin 29.	$\overline{WR}$	0-3	when 0 it indicates
Pin 30	HLDA	0	outputs a bus grant to a requesting master. Pins with tristate gates are put in high impedance state when HLDA = 1
Pin 31	$\overline{INTA}$		Receives bus request from bus master. The 8086 will not gain control of the bus until this signal is dropped,

* 8086 maximum mode configuration:





$\overline{MRDC}$: Memory read controller

$\overline{MWTC}$: Memory write controller

$\overline{IORC}$: I/O read controller

$\overline{IOWC}$: I/O write controller

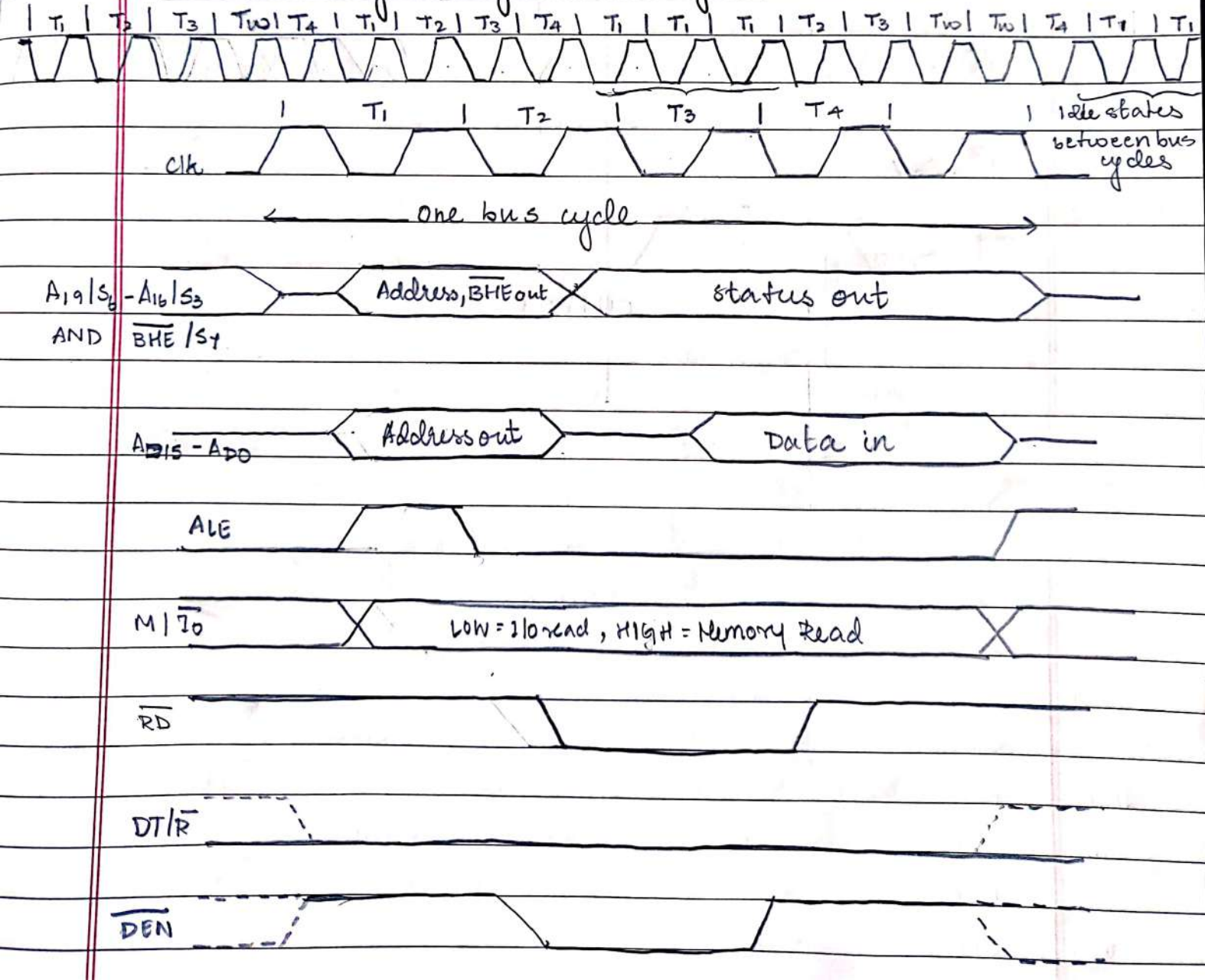
DT/$\overline{R}$: Transmit/Receive

Pin definations for maximum mode

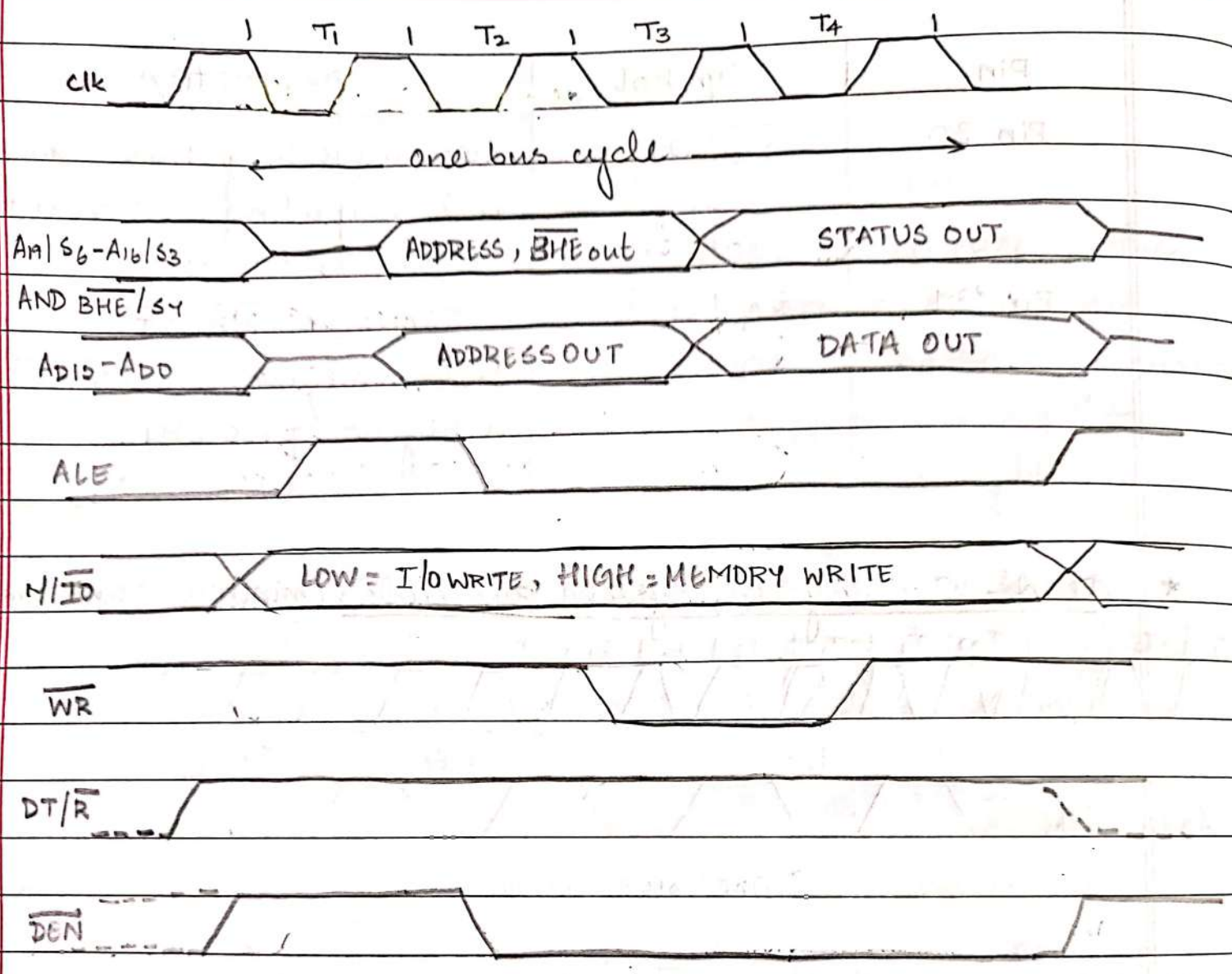
Pin	Symbol	Description
Pin 24	QS0	Reflects the status of the instruction queue. This status indicates the activity in the queue during the previous clock cycles.
Pin 25	QS1	
Pin 26	$\overline{S0}$	Indicating the type of transfer that takes place during the current bus cycle.
Pin 27	$\overline{S1}$	
Pin 28	$\overline{S2}$	
	$\overline{S2}$ $\overline{S1}$ $\overline{S0}$	
	0 0 0	Interrupt acknowledgment
	0 0 1	Read I/O port
	0 1 0	Write I/O port
	0 1 1	Halt
	1 0 0	Instruction fetch
	1 0 1	Read memory
	1 1 0	Write memory
	1 1 1	Inactive | Passive.
Pin 29	$\overline{LOCK}$	It is used only in multiprocessor. It indicates the bus is not to be relinquished to other potential bus masters. It is indicated by a lock instruction prefix and is maintained until the end of the instruction. It is also active during and between two $\overline{INTA}$ pulses.

Pin	Symbol	Description
Pin 30	$\overline{RQ}$ / $\overline{GT1}$	for inputting bus request and outputing bus grants.
Pin 31.	$\overline{RQ}$ / $\overline{GT0}$	same as $\overline{RQ}$ / $\overline{GT1}$ except that a request on $\overline{RQ}$ / $\overline{GT0}$ has a higher priority.

* **Read - Write cycle, system Bus cycle** : (minimum and maximum mode)

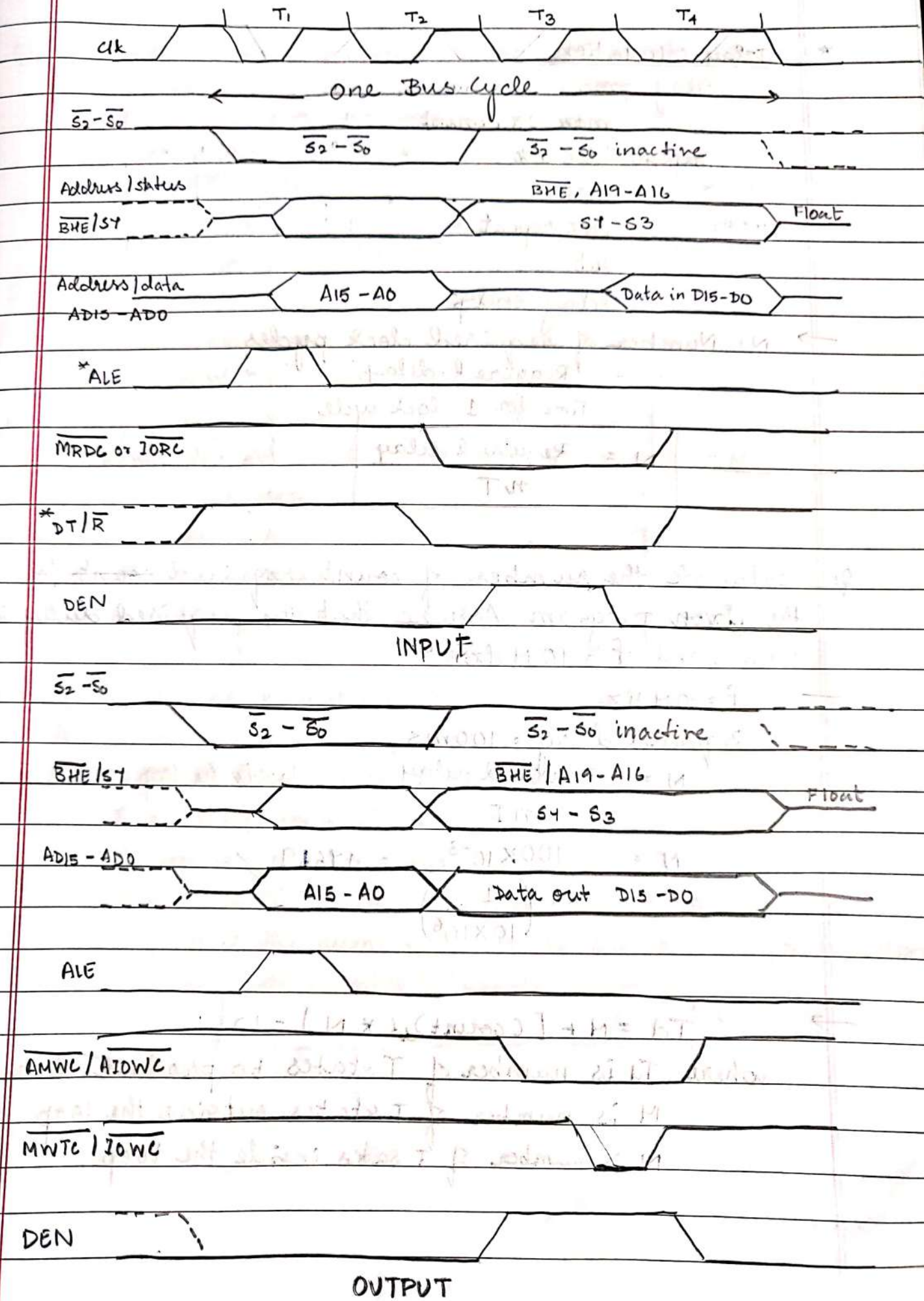


INPUT



OUTPUT

Maximum mode:



* **Delay calculation:**

Delay proc:

```
        mov cx, count    - 4
repeat: dec cx           - 2   } 21   Inside the loop
        NOP              - 3   }
        JNZ repeat       - 16  }
        ret              - 8
```

Outside the loop 12

delay endp

→ N: Number of required clock cycles

$$= \frac{\text{Required delay}}{\text{Time for 1 clock cycle}}$$

$$N = \frac{\text{Required delay}}{nT}$$

Q: calculate the number of count required ~~count~~ for the given program. Assume that the required delay is 100ms and f = 10 MHz

— f = 10 MHz

Required delay = 100 ms.

$$N = \frac{\text{Required delay}}{nT}$$ Inside the loop $n = 21$

$$N = \frac{100 \times 10^{-3}}{21\left(\frac{1}{10 \times 10^{6}}\right)} = 47619$$ //

→ $$Td = M + [(count)_d * N] - 12$$

where Td is number of T states to produce delay

M is number of T states outside the loop

N is number of T sates inside the loop

Q: Find the number of T state to produce a delay for the given delay program, when count = FFFFh

count = FFFFh = 65535

M = 12 (outside the loop)

N = 21 (inside the loop)

$T_d = [12 + (65535 \times 21)] - 12$

$T_d = 1376235$

```
delay proc:
        mov bx, count2      - 4        → P = 4
loop2:  mov cx, count1      - 4
loop1:  dec cx              - 2  } 18 = N   22 = M
        JNZ loop1           - 16 }
        dec bx              - 2
        JNZ loop2           - 16
        delay endp
```

Q: Calculate the number of T state required to produce a delay for a given delay program. Assume that count 1 = 65535d and count 2 = 4132d

$T_d = [P + [(count2)_d \times T_d\ loop1] - 12]$

$T_d\ loop1 = [M + [(count1)_d \times N] - 12]$

where P is the number of T states outside loop 2

M is the number of T states inside loop 2, outside loop 1

N is the number of T states inside loop 1

Here P = 4

M = 22

N = 18

$T_d\ loop1 = [22 + (65535 * 18) - 12] = 1179640$

$T_d = [4 + (4132 * 1179640) - 12] = 4874272472$

Q: Calculate count2 value for the given delay. Assume count 1 = FFFFh and delay = 60 sec. f = 1MHz

T_d loop1 = M + [(count1)$_d$ * N] − 12

T_d loop1 = 22 + [65535 * 18] − 12

T_d loop1 = 1179640

T_d = P + [(count2)$_d$ * T_d loop1] − 12

$$T_d = \frac{60}{1/1M} = 60 \times 10^6$$

$$60 \times 10^6 = 4 + [count2 * 1179640] - 12$$

$$count2 = \frac{60 \times 10^6 + 8}{1179640}$$

=

UNIT -03

* Memory Interfacing:

Q: Find the address space or starting address and ending address of the given memory map.

— $1k = 2^{10} = 1024$

$\therefore 2k = 2048$

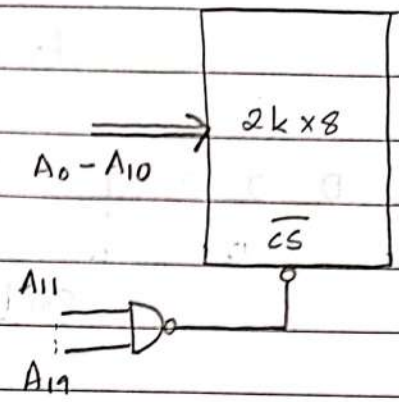
$2^n = N$

$2^n = 2048$

$n = \frac{\log(2048)}{\log 2}$

$n = 11$ (address lines)

$(A_0 - A_{10})$



For the signal at chip select to be active low, the input to the NAND gate should all be high.

Chip select								Address lines											
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
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
F				F				8				0				0			

Starting address = FF800 h

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F				F				F				F				F			

Ending address = FFFFFh

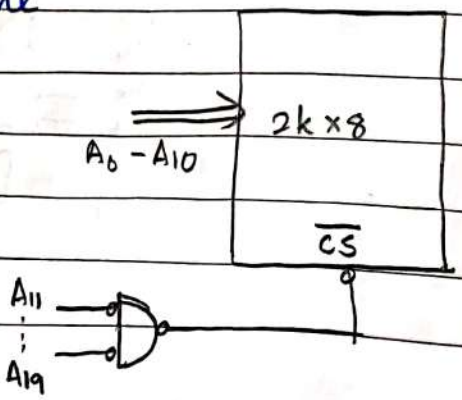
Q: Find the address space or starting address and ending address of the given memory map.

— $2k = 2048$

$2^n = 2048$

$\therefore n = 11$ address lines

$(A_0 - A_{10})$



For the signal at the chip select to be active low, the input to the gate ($A_{11}-A_{19}$) should all be lows.

Chip select | Address lines

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				0				0				0				0			

Starting address = 00000h //

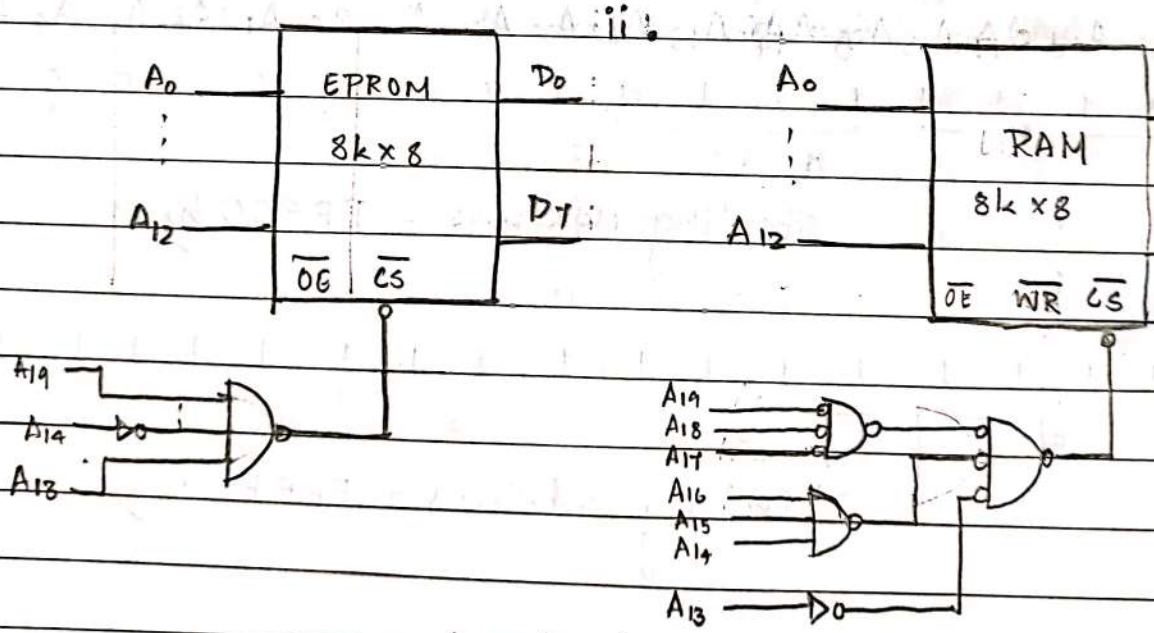
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0				0				1				F				F			

Ending address = 001FFFh //

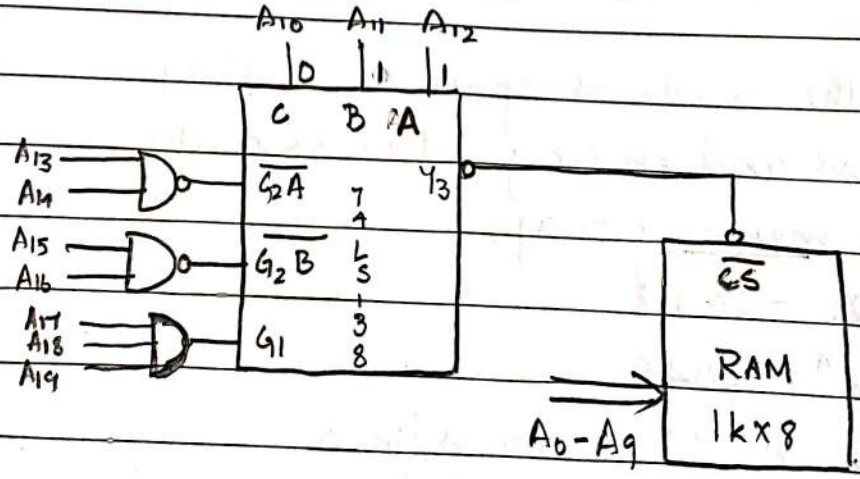
Q: Find the address sequence for the following chips which are the address decoder circuit as shown in figure.

i.

ii.



iii.



i.

Chip select							Address lines												
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
1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
F				A			0					0				0			

Starting address = FA000H

1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F				B			F					F				F			

Ending address = FBFFFH

ii.

Chip select							Address lines												
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
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1				D			0					0				0			

Starting address = 1D000H

0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1				F			F					F				F			

Ending address = 1FFFFH

iii.

Address lines																			
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
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
F				F			8					0				0			

Starting address = FF800H

1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
F				F			B					F				F			

Ending address = FFBFFH

Q: Design an addressing decoding circuit to interface two RAM block and a ROM block of each 4kB with a starting address at 4000H.

2 RAM] Each of 4kB
1 ROM] $4 \times 1024 = 4096$ bytes

STEP1: calculate the number of address pin required

$2^n = N$

n_1 - RAM I n_2 - RAM 2 n_3 = ROM.

$2^n = 4096$

$\therefore n = 12$ address lines
$(A_0 - A_{11})$

STEP2: Memory mapping

To compute ending address

End address = Base address + $(2^{10} \times 4 - 1)_d$

RAM 1 = $4000H + (1024 \times 4 - 1)_d$

= $4000H + (4095)_d$

= $4000H + FFFH$

= $4FFFH$

RAM 2 = $5000H + FFFH$

= $5FFFH$

ROM = $6000H + FFFH$

= $6FFFH$

Memory	Address
RAM 1	Starting address = 4000H
	Ending address = 4FFFH
RAM 2	Starting address = 5000H
	Ending address = 5FFFH
ROM	Starting address = 6000H
	Ending address = 6FFFH

* Interfacing techniques

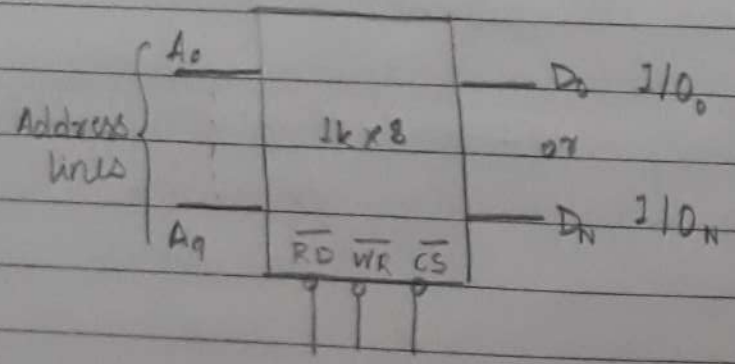
Microprocessor needs to access memory to get instruction (code and data).

Microprocessor unit will initiate read (MEMR) signal and write (MEMW) signal according to operation.

To do the above job it is necessary to have a device or a circuit which performs this task and is known as interfacing device.

To do the interfacing three different tasks have to be performed:

a. To select the required chip.
b. Identify the required register.
c. It must enable the appropriate buffer that is data buffers, address buffers, read or write signals.



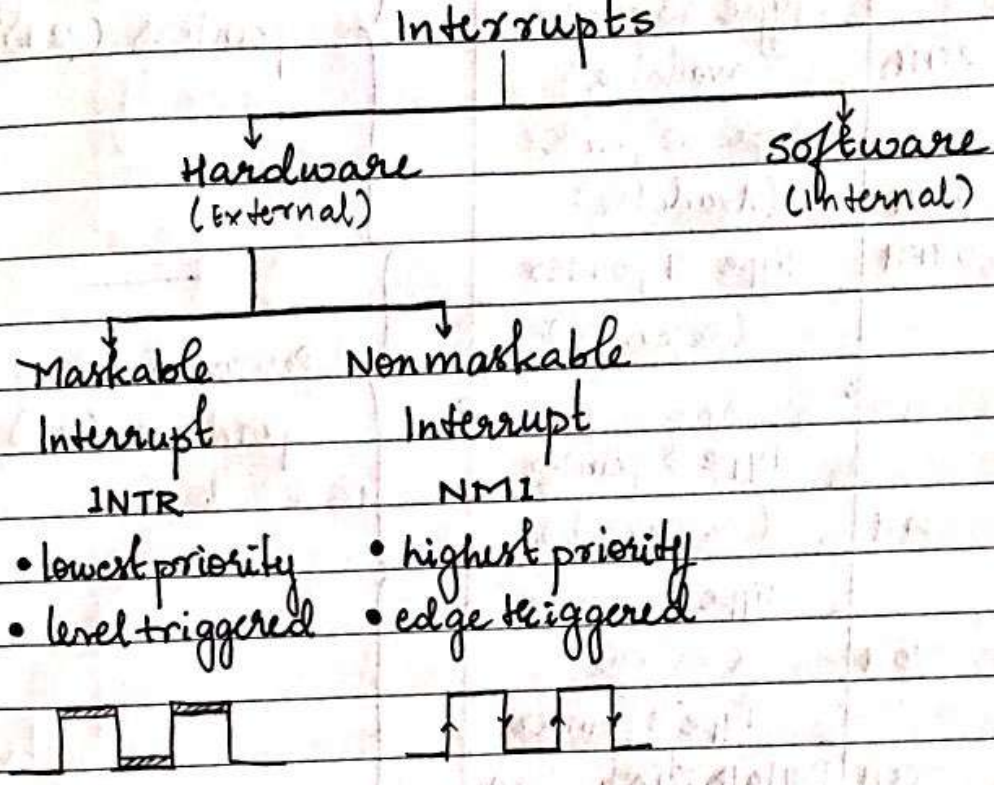
n: number of address lines

N: Memory

$$n = \frac{\log N}{\log 2}$$

UNIT - 04

Interrupt Structure and Interface



The microprocessor services an interrupt by executing a subroutine called Interrupt Service Routine (ISR).

Hardware interrupt

These interrupts are caused by signals on the external pins of the microprocessor. 8086 has two pins to accept hardware interrupts NMI and INTR.

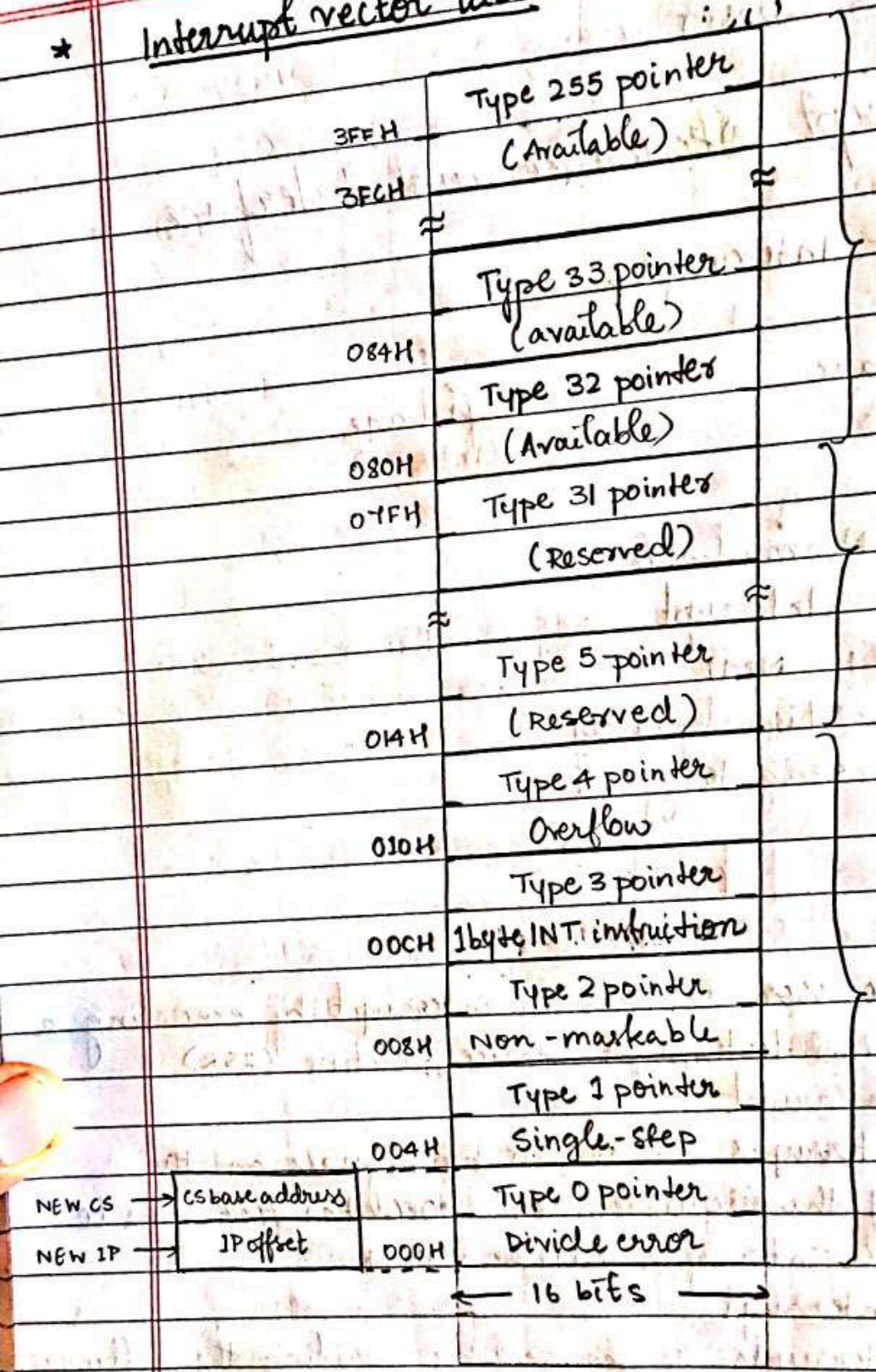
Software interrupt

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.

Error conditions

8086 is interrupted when some special conditions occur while executing certain instructions in the program.

* Interrupt vector table:



The interrupt vector table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code. For each type it has to reserve four bytes. This double word pointer contains the address of that procedure that is to service interrupts of that type.

The higher addressed word of the pointer contains the base address of the segment containing the procedure which is reffered as NEW CS.
The lower addressed word of the pointer contains the procedure's offset from the beginning of the segment which is referred as NEW IP.
Thus the NEWCS and NEW IP provides the new physical address from where user ISR routine will start.

For each type, four bytes (2 for NEW CS and 2 for NEW IP) are required. Therefore interrupt pointer table occupies up to the first 1k bytes ($256 \times 4 = 1024$ bytes) of low memory.
The total interrupt vector table is divided into three groups:

a. Dedicated interrupts (INT 0 - INT 4)
b. Reserved interrupts (INT 5 - INT 31)
c. Available interrupts (INT 32 - INT 225)

- Dedicated Interrupts:

1. INT 0 - Divide Error

This interrupt occurs whenever there is division error i.e., when the result of a division is too large to be stored. This condition normally occurs when the divisor is very small as compared to the dividend or the divisor is zero.

2. INT 1 - Single Step

The microprocessor executes this interrupt after every instruction if the trap flag (TF) is set. It puts the microprocessor in single step mode i.e., the microprocessor pauses after executing every instruction. This is very useful during debugging.

3. INT 2 - Non Maskable Interrupt
The microprocessor executes this ISR in response to an interrupt on the NMI (Non mask-able Interrupt) line.

4. INT 3 - Breakpoint Interrupt
This interrupt is used to cause breakpoints in the program. It is useful in debugging large programs.

5. INT 4 - Overflow Interrupt
This interrupt occurs if the overflow flag is set. It is used to detect overflow error in signed arithmetic operations.

- Reserved Interrupts: (INT 32 - INT 225)
These levels are reserved by Intel to be used in higher processors and are not available to the users

- Available Interrupts:
These are user defined software interrupts. The ISRs for these interrupts are written by the user to service various user defined conditions. They are invoked by INT n instruction.

- Hardware Interrupts

1. NMI (Non maskable Interrupt)
This is a non maskable, edge triggered, high priority interrupt. On receiving an interrupt on NMI line the microprocessor executes this interrupt.

2. INTR
This is a mask-able, level triggered, low priority interrupt. It is masked by making IF = 0 by software through CLI instruction and unmasked by making IF = 1 by software through STI instruction.

Date
Page

* Interrupt Instructions:

	Mnemonic	Meaning	Format	operation	Flags affected
1.	CLI	Clear Interrupt flag	CLI	IF = 0	Interrupt flag
2.	STI	set Interrupt flag	STI	IF = 1	Interrupt flag
3.	INTn	Type n software interrupt	INTn		Trap flag Interrupt flag
4.	IRET	Interrupt return	IRET		All
5.	INTO	Interrupt on overflow	INTO	INT4 if O=1	Trap flag Interrupt flag
6.	HLT	Halt	HLT	wait for an external interrupt or reset to occur	None
7.	WAIT	Wait	WAIT	wait for test input to go active	None

	Mnemonic	Meaning	Format	Operation	Flag
8.	Int 80			It allows to execute interrupt service routine of interrupt 80 in a program which is similar to subroutine call but calling an ISR.	

* The Interrupt Flag:
- If IF is set to 1 (IF=1) then the external hardware can initiate an interrupt via INTR input of the microprocessor.
- If IF is cleared (IF=0) then the external device cannot initiate an interrupt.
- During the initiation sequence of an interrupt service routine (ISR) the 8086 automatically clears the Interrupt flag. This disable or masking of the occurance of any additional external hardware interrupt.
- The IF flag should be reenabled at the end of the service routine.

Q: If INT 50 occurs what are the addresses storing information for CS 50 and IP 50.

— CS and IP : 4 byte

$4 \times 50 = 200 = C8H$

CS		CBH CAH
IP		C9H C8H

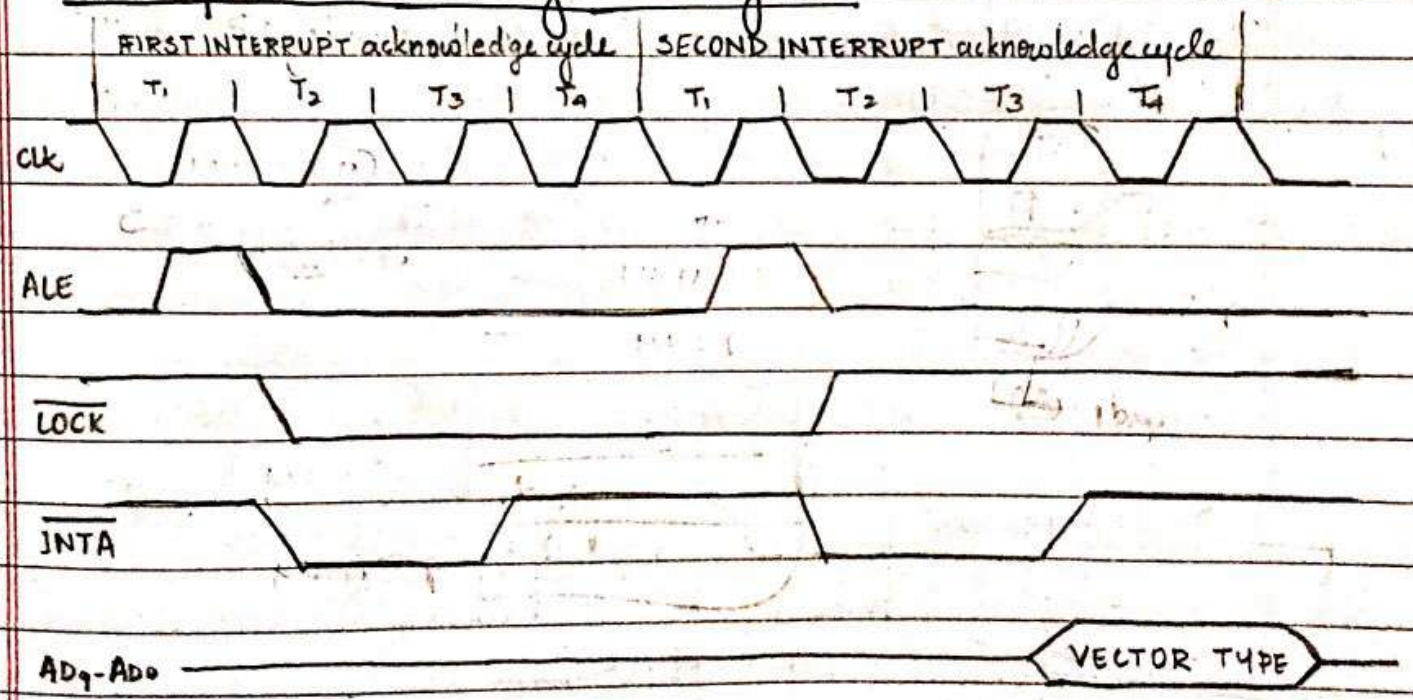
Q: If INT 56 occurs what are the addresses storing information for CS 56 and IP 56.

— CS : 2 byte IP : 2 byte

$4 \times 56 = 224 = E0H$

CS		E3H E2H
IP		E1H E0H

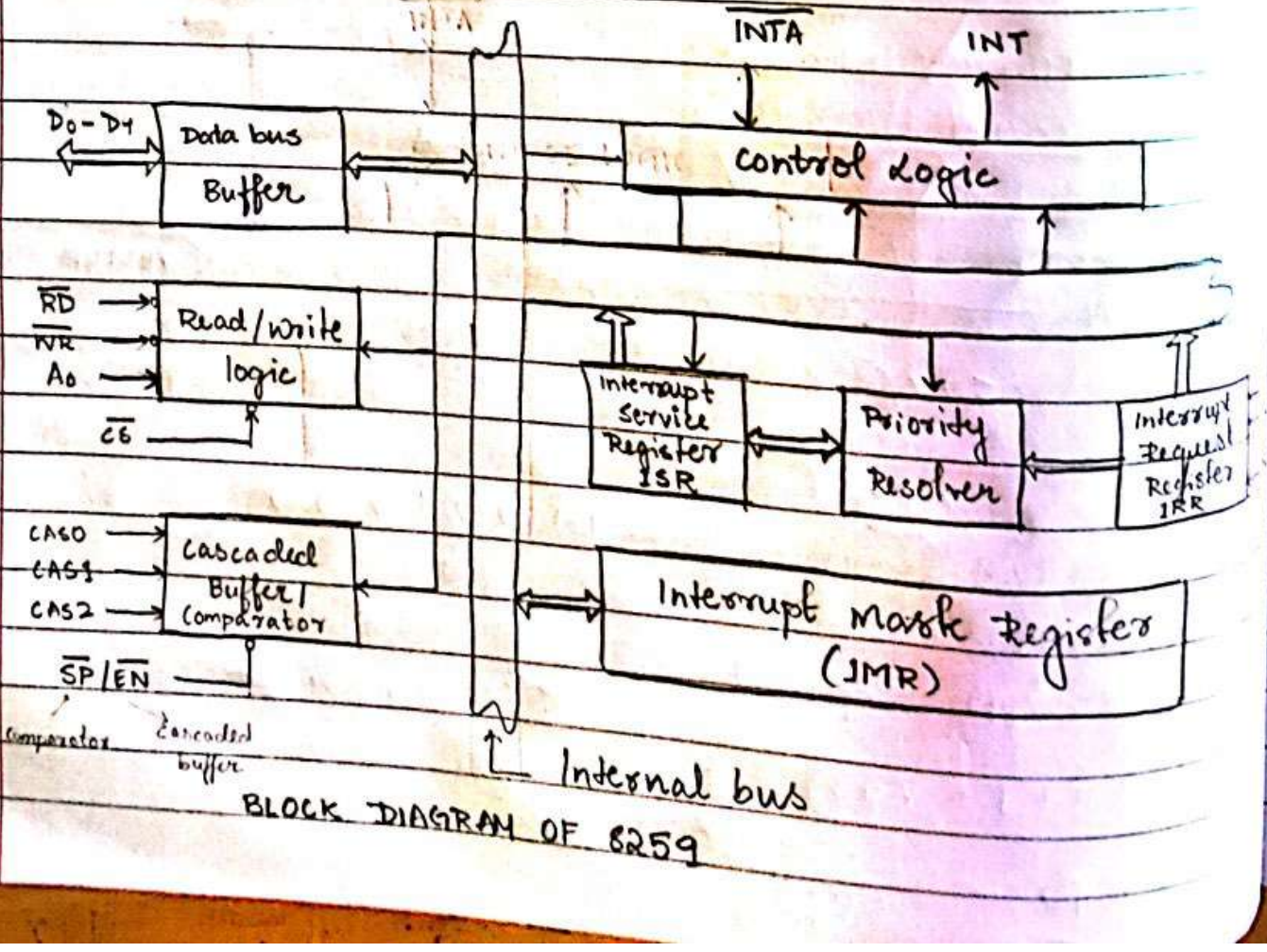
* Interrupt acknowledge bus cycles :



8259 Interrupt

$\overline{CS}$ →	1		28	← Vcc
$\overline{WR}$ →	2		27	← A_0
$\overline{RD}$ →	3		26	← $\overline{INTA}$
D1 ↔	4		25	← IR7
D6 ↔	5		24	← IR6
D5 ↔	6	8	23	← IR5
D4 ↔	7	2	22	← IR4
D3 ↔	8	5	21	← IR3
D2 ↔	9	9	20	← IR2
D1 ↔	10		19	← IR1
D0 ↔	11		18	← IR0
CAS0 ↔	12		17	← INT
CAS1 ↔	13		16	↔ $\overline{SP}/\overline{EN}$
GND →	14		15	↔ CAS2

PIN CONFIGU
OF 825



BLOCK DIAGRAM OF 8259

- **V_{cc} and Ground**

It is the power supply and ground pin. +5V power supply is issued in this pin.

- **$D_0 - D_7$**

For communication with processor there are 8 bidirectional data pins.

- **$\overline{RD}$**

It is an active low input pin activated by the processor to read the information status from the 8259.

- **$\overline{WR}$**

It is an active low input pin activated by the processor to write the control information to 8259.

- **$\overline{CS}$**

For selecting the chip. It is an active low input pin.

- **A0**

An address input pin used along with $\overline{RD}$ and $\overline{WR}$. It is used to identify various commands.

- **IR0 - IR7**

There are 8 asynchronous interrupt request pins. These interrupt request can be programmed for level triggered or edge triggered mode.

- **INT**

A strong active high output pin which interrupts the processor. Always connected to the INTR interrupt input of 8085 or 8086. The INT output is only activated when all the given conditions are satisfied

- **CAS0 - CAS2**

They are cascaded lines used only when there are multiple 8259 in the system. The interrupt control system might have a master 8259 and

maximum 8 slave 8259s.

— $\overline{SP}/\overline{EN}$

It stands for slave program or enable buffer. This pin serves dual function when it is used as $\overline{EN}$ pin. It provides an active low output that controls the buffer transceivers in the buffer mode

UNIT - 5

Common Peripheral Controllers

Serial Transmission : sequential transmission of data
Parallel Transmission : simultaneous transmission of data

* PIO 8255 : Peripheral input-output / Parallel input-output port.

- I/O Modes

a. Mode 0 : Basic I/O mode
- data is simply read and written to the input and output ports respectively.
- Two 8bit ports : port A and port B
  Two 4 bit ports : port C upper and lower - third 8 bit port
- Any port can be used as input or output port
- Output ports are latched but input ports are not latched.
- 16 I/O configurations are possible

b. Mode 1 : Strobed I/O mode
- Hand shake control the input and output action of the specified port.
- Port A and Port B are available for strobed data transfer
- 8 bit data port + 4 bit control / data port : Group.
- 8 bit port can be used as input-output port
- The inputs and outputs are both latched.
- PC0 - PC2 generate control signals for port B
  PC3 - PC5 generate control signals for port A
  PC6 - PC7 : independent data lines.

- STB: Strobe input:
  0: 8 bit input port is loaded into input latches.
- IBF: Input buffer full
  1: data has been loaded into latches
- INTR: Interrupt Request.
- OBF: Output Buffer Full
  0: CPU has written data to specified output port.
- ACK: Acknowledge input
  0: informs CPU data transfered has been recived by the output device

c. Mode 2: Strobed bidirectional I/O
- bidirectional 8 bit port with handshake signal.
- Rd and Wr decide to operate as input or output port.
- Single 8 bit port in group A.
- 8 bit bidirectional port + 5 bit control port.
- There I/O lines at port C : PC2 - PC0
- Both inputs and outputs are latched.
- 5 bit control port C: PC3 - PC7 : used for generating and accepting handshake signals. on port A

All these modes are selected by programming control word Register: CWR

D7	D6	D3	D4	D3	D2	D1	D0

D7 : 0 : I/O mode
1 : BSR mode

D6-D5 Mode selection
00 : Mode 0
01 : Mode 1
10 : Mode 2

D4 : Port A
1 : Input
0 : Output

D3 : Port C upper
1 : Input
0 : Output

D2 : Mode selection
0 : Mode 0
1 : Mode 1

D1 : Port B
1 : Input
2 : Output

D0 : Port C lower

SYNCHRONOUS TRANSMISSION	ASYNCHRONOUS TRANSMISSION
- same clock pulse is applied to both Tx and Rx simultaneously.	- Different clock pulses are applied to Tx and Rx separately.
- Only hardware is required	- Both hardware and software is required
- group or a set of characters can be transmitted at a time.	- Only one character is transmitted at a time.
- synchronous pulses are required.	- synchronous pulses are not required but uses start and stop bits.
- Used for high speed Tx.	- Used for low speed Tx

* 8251A : USART

Accepts parallel format data from CPU and converts to serial data stream for transmission.

It can receive serial data streams and convert them into parallel data characters for the CPU.

- Data bus buffer : 3 state bidirectional buffer used to interface 8251A to system data bus. command words and status information are transfered through it.
- Read/Write logic and Registers:

RESET : 1 : forces 8251A to idle mode

CLK : generates internal device timing.

$\overline{WR}$: 0 : informs 8251A that the CPU is writting data or control words to it.

$\overline{RD}$: 0 : informs 8251A that the CPU is reading data or status from it.

C/$\overline{D}$: control / Data : 1 : control / status word on the data bus
0 : data word on the data bus.

$\overline{CS}$: chip select : 0: selects 8251A

- Modem control:
  $\overline{DSR}$: Data Set Ready
  $\overline{DTR}$: Data Terminal Ready
  $\overline{RTS}$: Request to send.
  $\overline{CTS}$: Clear to send
- Transmitter Buffer: It accepts parallel data from Data Bus Buffer converts it to a serial bit stream, inserts the appropriate characters or bits.
- Transmitter control: Manages all activities associated with the transmission of serial data.
  TxRDY : Transmitter ready.
  TxE : Transmitter empty
  TxC : Transmitter clock.
- Receiver Buffer: It accepts serial data & and converts serial input to parallel format.
- Receiver control: Manages all activities associated to the receiver.
  RxRDY: Receiver ready
  $\overline{RxC}$: Receiver clock
  SYNDET (SYNC detect / BRKDET Break detect)
  BREAK : output is 1 when the receiver remains low through two consecutive stop bit sequences.

NOTE:

Modes of serial data transmission.

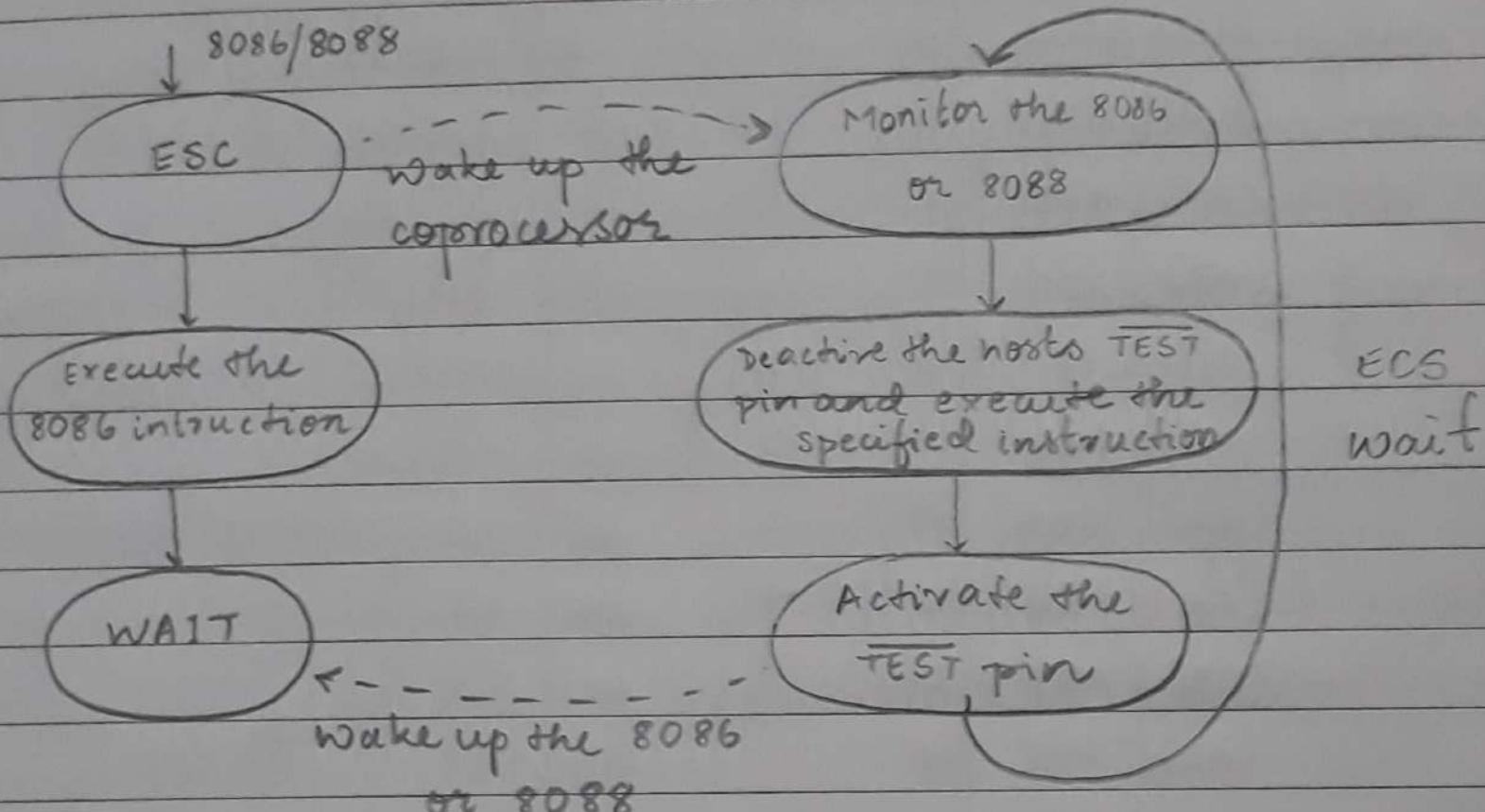
Simplex : data is transmitted in one direction only

Half duplex : data transfer is two-way, but only in one direction at a time.

Full duplex : data can be sent and received on both ends simultaneously.

UNIT - 6

Numeric Data Processor



SYNCHRONIZATION BETWEEN THE 8086 AND ITS COPROCESSOR (8087)

GND	1		40	Vcc
(A14)/AD14	2		39	A15
(A13)/AD13	3		38	A16/S3
(A12)/AD12	4		37	A17/S4
(A11)/AD11	5		36	A18/S5
(A10)/AD10	6		35	A19/S6
(A9)/AD9	7		34	$\overline{BHE}$/S7
(A8)/AD8	8		33	$\overline{RQ}$/$\overline{GT1}$
(A7)/AD7	9	8	32	INT
(A6)/AD6	10	0	31	$\overline{RQ}$/$\overline{GT0}$
(A5)/AD5	11	8	30	NC
(A4)/AD4	12	7	29	NC
(A3)/AD3	13		28	$\overline{S2}$
(A2)/AD2	14		27	$\overline{S1}$
(A1)/AD1	15		26	$\overline{S0}$
(A0) AD0	16		25	QS0
NC	17		24	QS1
NC	18		23	BUSY
CLK	19		22	READY
GND	20		21	RESET

PIN CONFIGURATION OF 8087

8086:

pin 17 : NMI
pin 18 : INTR
pin 23 : TEST
pin 24 : $\overline{INTA}$
pin 25 : ALE
pin 26 : $\overline{DEN}$
pin 27 : DT/$\overline{R}$
pin 28 : M/$\overline{IO}$
pin 29 : $\overline{WR}$

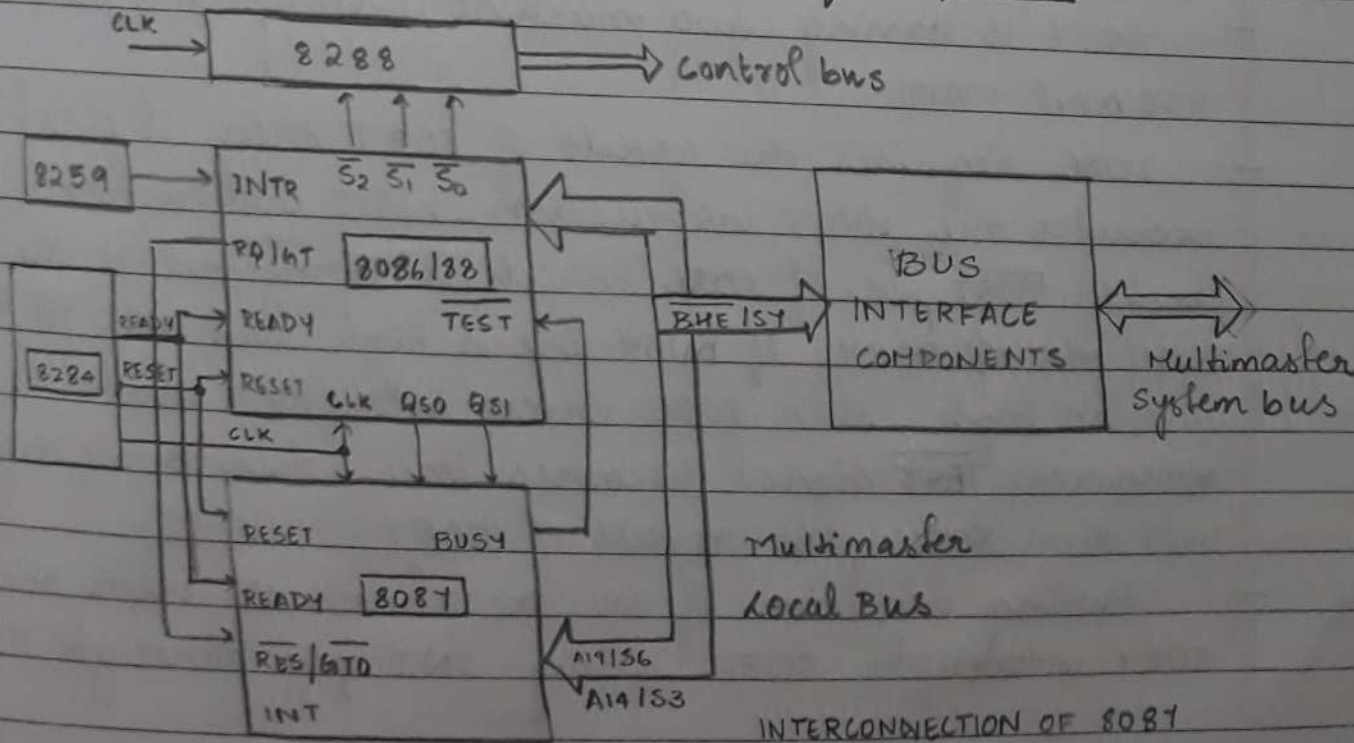
pin 30 : HOLDA
pin 31 : HOLD
pin 32 : $\overline{RD}$
pin 33 : MN/$\overline{MX}$

* Features of 8087:

- 8087 is designed to perform arithmatic operations efficiently.
- It can operate on data of the integer, decimal and real types which lengths from 2 - 10 bytes.
- The instruction set not only includes various forms of addition, subtraction, multiplication and division but also provides many useful functions such as taking the square root, exponential, taking the tangent and so on.
- The 8087 provides a simple and effective way to enhance the performance of an 8086 based system particularly when an application is primarily computational in nature.

NOTE: The NDP cannot fetch its own instruction and therefore it must operate with either 8086 or 8088 which acts as the host in the coprocessor configuration.

* Multiprocessor configuration and interfacing 8087:

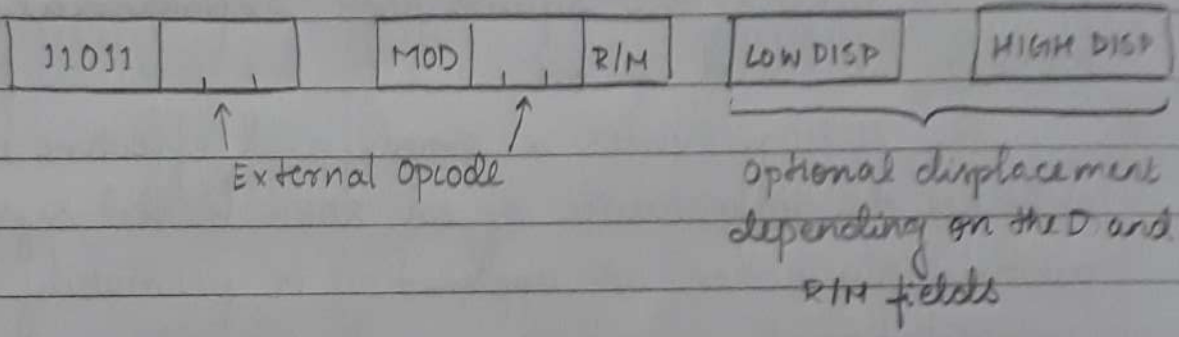


INTERCONNECTION OF 8087

8086/8088 : Host processor
8087 : coprocessor
8284 : clk, Ready and Reset
8282 : Latch the address
8288 : Generate control signal from S_2, S_1 and S_0.
8259 : PIC

* **Machine Code Format:**

a) Memory operand is used



- 8087 is connected as a coprocessor to 8086.
- 8087 uses I/O system, bus system, clock inputs and memory of 8086. 8086/8088 are known host processor and 8087 is known as coprocessor because it is associated ~~as a copro~~ with a host processor.
- 8087 is having two machine instructions that is ESC and WAIT
- 8086 requires the result of 8087 then it first executes the WAIT instruction. WAIT instruction checks $\overline{TEST}$ pin of 8086 which is connected to the BUSY pin of 8087. If BUSY line is high and the $\overline{TEST}$ line is also high, then 8086 enters into WAIT state. Whenever $\overline{TEST}$ signal becomes low when BUSY signal is low then 8086 gets result of 8087.
- During execution if an exceptions or error occurs, 8087 interrupts 8086 using INT pin through 8259

* NDP data types:

1. Word integer - 2 bytes
2. Short integer - 4 bytes
3. Long integer - 8 bytes
4. Packed BCD - 10 bytes
5. Short real - 4 bytes
6. Long real - 8 bytes
7. Temporary real - 10 bytes