1. Explain the levels of the programming language

↳ Answer:

Programming language can be categorized into these three different levels.

(a) High level Language

- Designed to be more humon-readable and programmer-friendly.
- Provides more abstraction and encapsulation.
- Usually platform-independent, requiring an interpreter or compiler.
- Example:

  C, C++, jave etc.

(b) Assembly level language

- Consists of binary code and mnemonic instructions representing specific operations.
- Platform dependent.
- Slower than machine language.
- Assemble converts into machine language

(c) Machine level Language

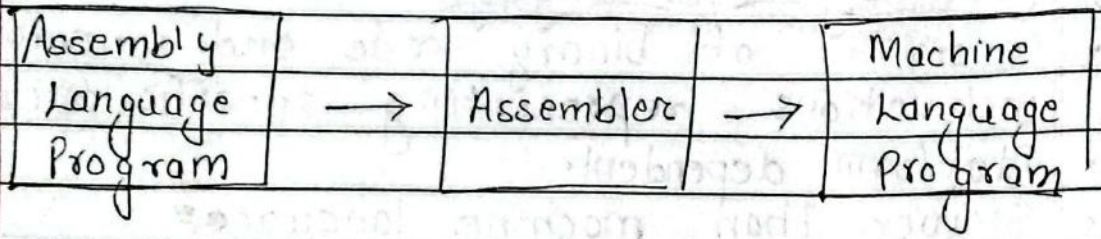
- Also called binary language because they contain only 0s and 1s.
- Have the lowest level of abstraction.
- Difficult to read, write and debug.
- Faster execution time.

2. Explain assembly language and compiling programming languages process with suitable examples.

↳ Answer:

Assembly language process

- Assembling converts source program into object program if syntactically correct and generates an intermediate .obj file or module.
- Assembler is used in assembling process which converts assembly code into the machine code.
- GAS, GNU is an example of assembler.



Source Program — One by one Pass. — Object Program

The Assembly process

i) Assembling the source code into an object code.

ii) Linking the object- file with other modules or libraries into an executable program.

iii) Loading the program into memory.

iv) Running the program.

Assembly Language Program

↓

Assembler

↓

Object: Machine Language Module Object: Library routine

↓

Linker

↓

Executable: Machine Language Program

↓

Loader

↓

Memory.

Fig: Assembly Language Process

Example:

Mnemonics	Operands
MOV	AX, BX

The assembler reads a line like this one from the source code files and writes the equivalent machine instructions to an object code files.

⇩

[8BH 0C8H]

Compiling Process

- The compilation is a process of converting the source code into an object code.
- Stages of compilation
  - lexical analysis
  - symbol table construction
  - syntax analysis
  - semantic analysis
  - code generation
  - optimization

- Compilation makes sure that the source code follows syntax.
- Compilation error will be given if any code doesn't follow syntax.
- It converts whole code into machine language at a time.

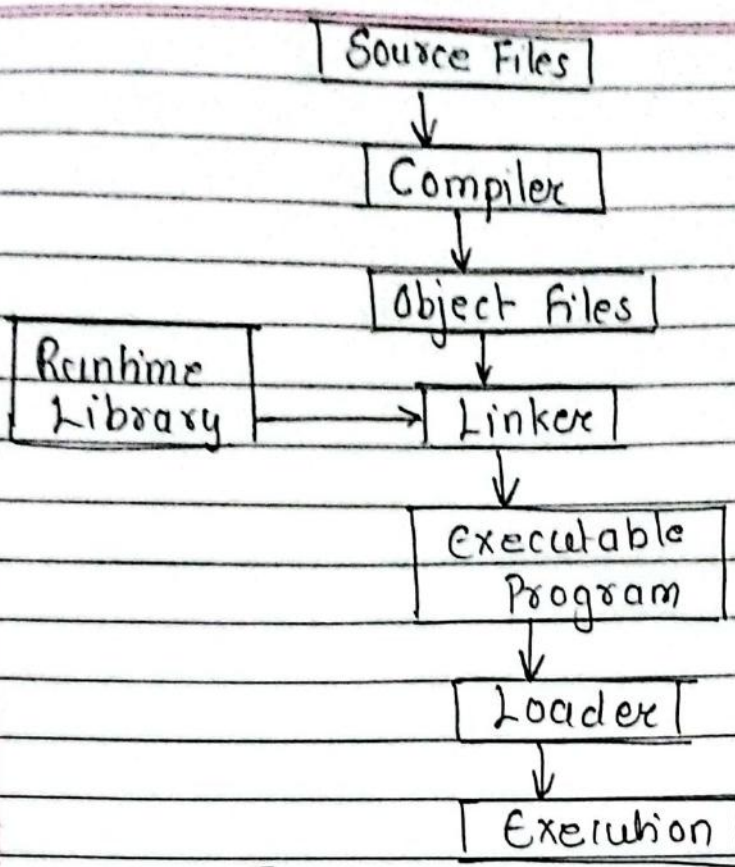


Fig: Compilation Process.

- Eg of purely compiled language are C, C++, Erlang, Rust, Haskell etc.

(3) Explain different types of addressing modes with examples.

↳ Answer:

It is the way in which operands are specified in an instruction. Types of addressing modes are:

(a) Direct Mode

- contains memory addresses that CPU accesses.
- The 16-bit effective address of the data is part of the instruction.

Example:

ADD AL, [0801]

(b) Indirect Mode

- The address in the instruction is not the address of operand, rather it contains the address of memory location where operand resides.
- Example:

LDAC (5)

c. Register Direct and Register Indirect Mode

- works same as direct and indirect modes, except they do not specify memory address, they specify a register.
- Example:

LDAC (R) or
LDAC @R or
LDAC R

(d) Immediate Mode

- Operand specified is not an address, it is actual data to be ~~stored~~ used.

Example:

```
LDAC #5
```

(e) Implied Mode

- Operand is specified itself in the instruction.

Example:

```
CLAC   (Clears the Accumulator)
```

(f) Relative Mode

- Operand ~~itself~~ specified is an offset, not the actual address which is added to the content of the program counter register to generate required address.

Example:

```
LDAC $5
```

(If next instruction is at location 12, the instruction reads data from 12+5=17 location and stores in accumulator.)

(g) Inden and Base address Mode

- Inden mode work like relative mode, except the address supplied by instruction is added to the content of ~~regis~~ inden register.

Example:

```
LDAC 5(X) ;   Reads from 10+5 = 15
```

(Index Register)

- Base mode is same as index mode except inden register is replaced by base register.

(4) What do you mean by instruction set in the processor? How are they different from micro instructions; Explain the machine cycles associated with the instruction cycles with example.

↳ Answer:

Instruction Set

- An instruction set refers to the collection of instructions that processor can understand and execute.
- It defines the operation that the processor can perform such as arithmetic, logical and control operations.

Micro Instructions

- Micro instructions are the lower level instructions that directly control the internal components of a processor.

Differences

↳ Instruction set define the higher level instructions a processor can execute while micro instructions control the hardware components and execute the instruction at a lower level of abstraction.

↳ Instruction set defines the interface between the processor's hardware and software whereas micro instructions controls the hardware implementation of the processor.

Instruction Machine Cycle

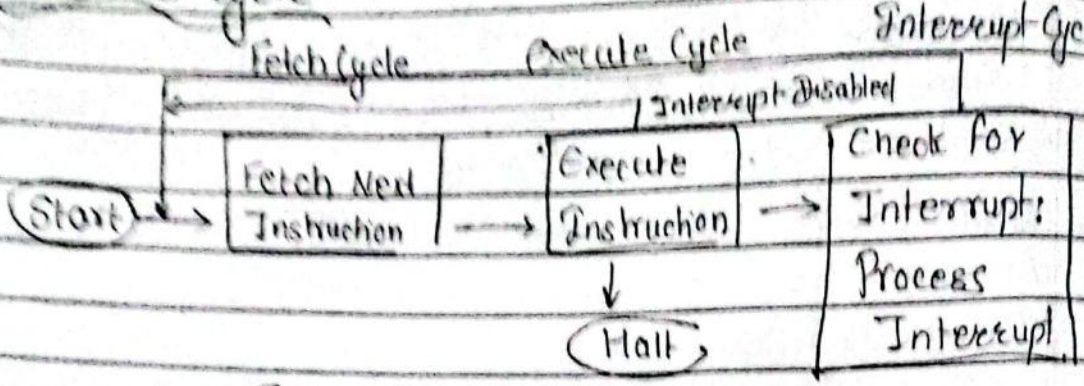


Fig:

↳ Instruction cycle is the procedure a microprocessor goes through to process an instruction.

Suppose we have the instruction:
"INC R1"

a. Fetch

- Microprocessor fetches the instruction from memory.
- PC holds the address of the next instruction to be fetched.
- CPU retrieves the address from PC and outputs it onto the address bus.
- Then memory subsystem places the corresponding instruction into the system data bus.
- The CPU reads the instruction code from the system data bus, which in this case is "INC R1".

b. Decode

. The CPU decodes the instruchon, determining that it is an incremen operation.

@ Execute

. The CPU performs the operation necessary to execute the instruchon.

. It retrieves the value from register R1.

. The CPU increments the value by 1.

. The resulting value is stored back in register R1.

(5) Write about ISA. What are the factors to be considered while designing ISA ? Explain.

↳ Answer:

- Instruction Set Architecture
- Crucial computer system that defines the interface between hardware and software.
- It provides the information needed to interact with the microprocessor.

↳ Factors to be considered while designing ISA architecture

(a) What should the instruction set and its processor be able to do ?

↳ The ISA acts as an interface between the hardware and software, specifying both what the processor is capable of doing as well as how it gets done.

(b) Whether the instruction set is complete or not ? (Completeness)

. For a processor for general purpose computing it requires rich ISA, large instruction set, many registers. So it must be complete.

(c) Orthogonality
A good ISA should provide a rich set of independent and flexible instructions that can be combined to perform complex operations

(d) Support for High-level language
It must provide instructions that map directly to high level language constructs.

(e) Are interrupts needed?
Many tasks in the computer can be performed efficiently if the interrupts are incorperated. So, designer must consider the interrupts.

(f) Types and sizes of the data
↳ While designing ISA, designer must consider what type and size of data will the microprocessor deal with.

↳ Some other factors to be considered are:
- Simplicity
- Scalability
- Security
- Code Density
- Performance
- Forward Compatibility
- Backward Compatibility

⑥ Explain basic computer organization with a block diagram.

↳ Answer:

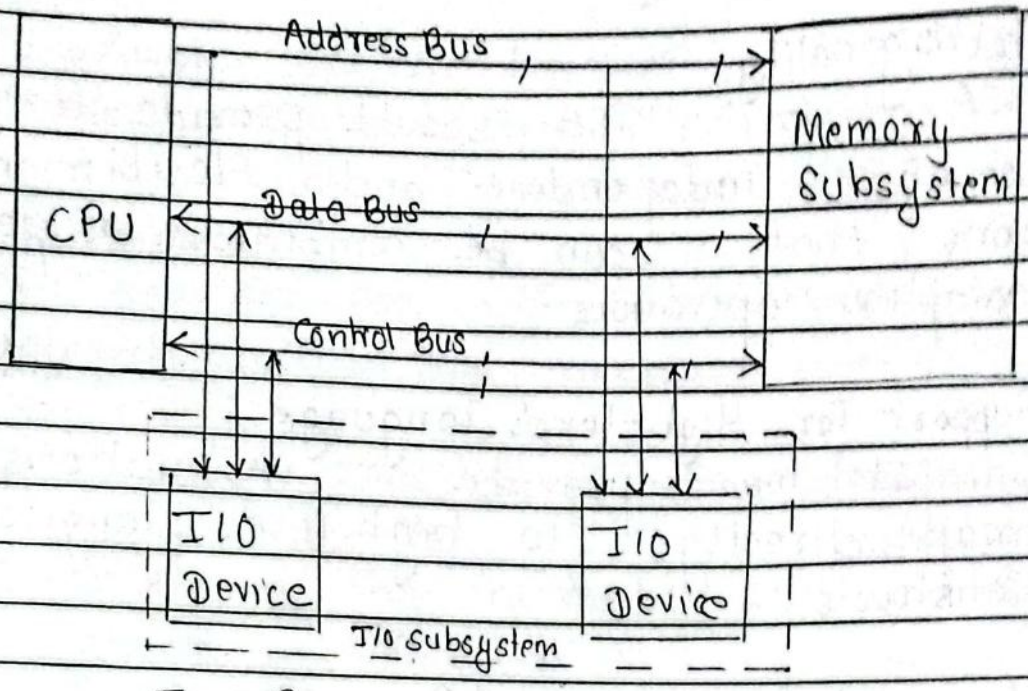


Fig: Basic CPU organization

The block diagram of basic CPU organization consists of 4 ~~three~~ main components.

① CPU
② Memory subsystem
③ I/O subsystem
④ System Buses

① CPU Operations

Here, the CPU perform 4 operations:

i) Fetch

Retrieving an instruction from program memory.

(1) Design the two dimensional chip organization of 16 x 8 ROM chips.

ii) Decode
Fetched instruction is broken into parts.

iii) Execution
The decoded instruction is then executed.

iv) Store
CPU writes back the result of execution to the computer.

(2) System Buses

1. A bus is a set of wires.
There are three types of bus:

i) Control Bus

a. The control bus carries that control signals which is generated by CU of the CPU.

b. Control bus signal is used for controlling and coordinating the various activities across the computer.

ii) Address BUS

a. It connects the CPU and another peripheral and carries only the memory address.

b. The address bus carries 8-bit data at a time, the CPU could address only ($2^n 8$) i.e 256 bytes of RAM.

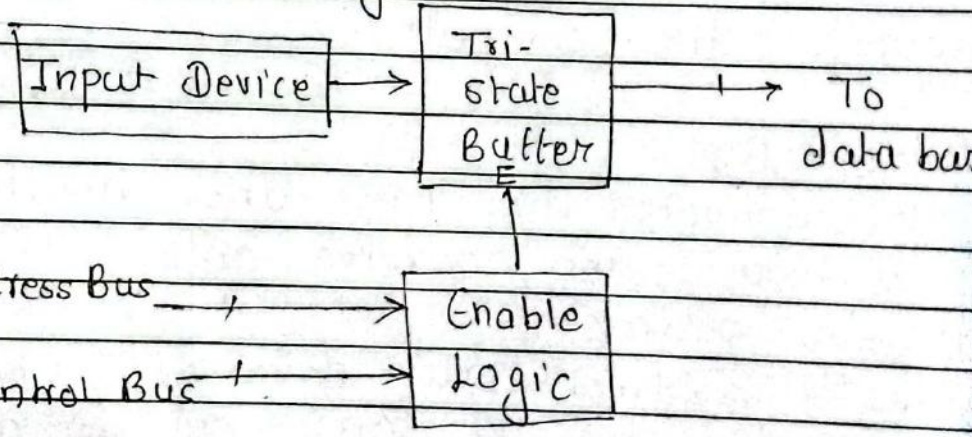
iii) Data Bus
- Bidirectional.
- Transfers data from one location to another across the computer.
- It may be 16 bit or 32 bit data bus.

(3) Memory Subsystem
- The memory subsystem is made up of hardware and software components.
- It consists of RAM, ROM where there is chip enable (CE) and output enable (OE) logic exists.

(4) I/O Devices

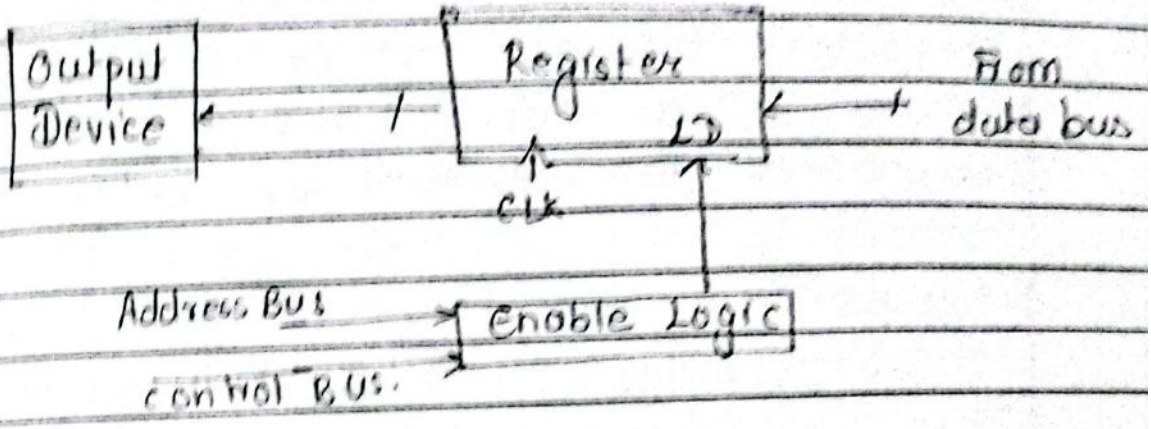
~~It cons~~

1) Input Device
- It consists of Tri-state buffer and Enable logic.



(d) Output Device

- It consist of Register and Enable logic.



(7) Explain the linear chip organization with an example.

→ Answer:

Linear chip organization

It refers to the arrangement of components on a chip or integrated circuit in a linear fashion to create a function unit or subsystem.

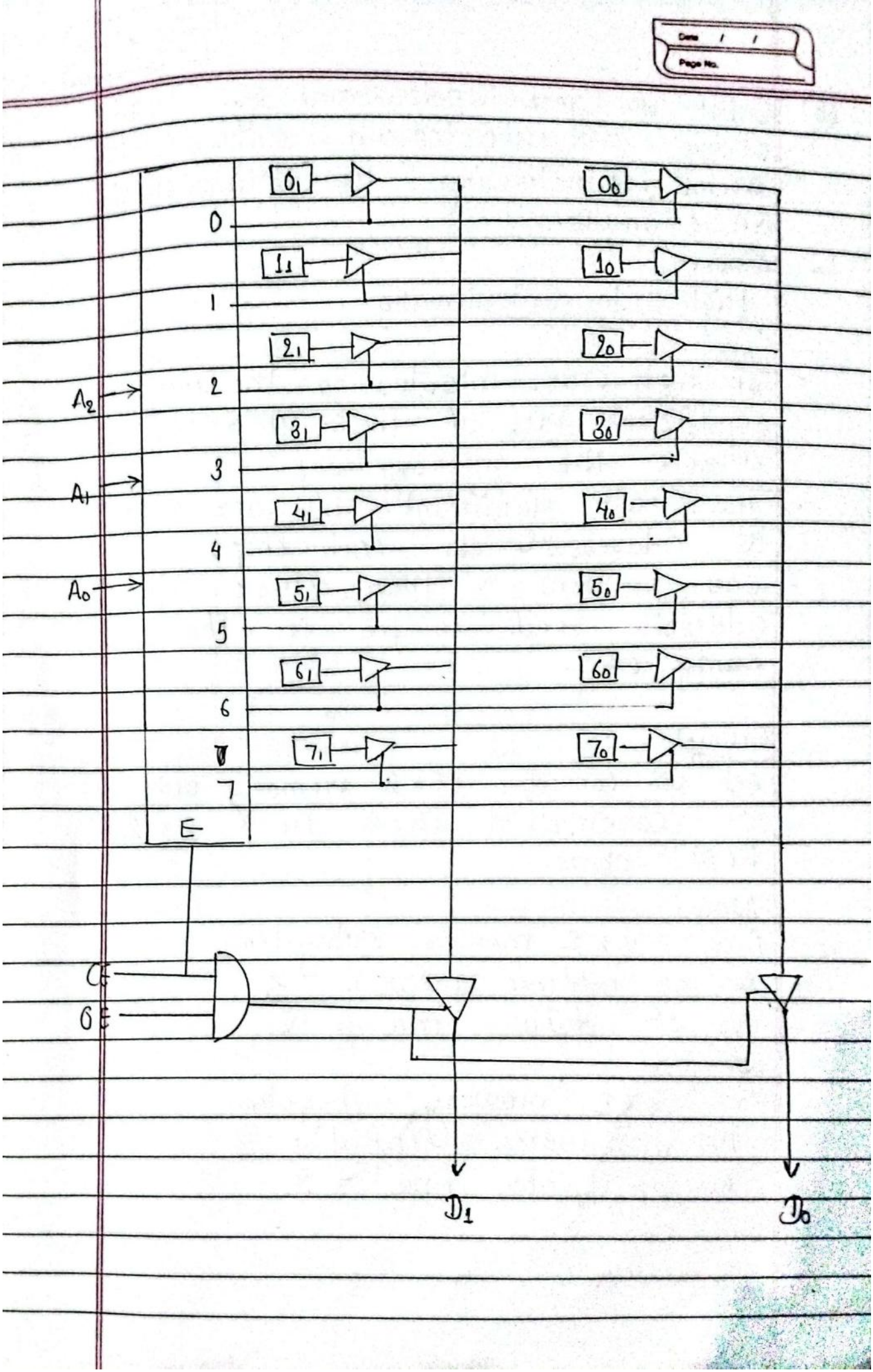
Example:

Let us consider an 8 X 2 ROM chips.

Then:

- Comparing with $2^n \times m$ i.e $2^3 \times 2$
  ∴ ~~16 bit~~
  - No Of Address input = 3 = n
  - No. of data output = 2 = m
  - 16 bits internal storage arranged as eight 2-bit locations.
- If CE = 0, decoder is disabled and no location is selected.
- If CE = 1 and OE = 1, tri-state buffer for that location cell is enabled and data is output from the chip.

↳ The size of decoder is 3X8 for an 8X2 ROM chips.


O_1
O_0
0
I_1
I_0
1
2_1
2_0
A_2
2
3_1
3_0
A_1
3
4_1
4_0
4
A_0
5_1
5_0
5
6_1
6_0
6
7_1
7_0
7
E
CE
OE
D_1
D_0

(8) What do you understand by high order interleaving and low order interleaving? Explain with an example.

↳ Answer:

High Order Interleaving

- In high order interleaving, the most significant bits of the address selects the memory chip.
- The least significant bits are sent as addresses to each chips.
- One problem is that consecutive addresses tends to be in the same chip.

Example:

Let us consider 16 x 2 memory subsystem constructed from two 8 x 2 ROM chips.

Now,

For, 16 x 2 memory subsystem:

- No of address inputs = 4
- No of data lines = 2

So,

For 8 x 2 memory subsystem:

No of address input = 3

No of data lines = 2

For High order interleaving:
A_3 selects memory chip.
A_2, A_1, A_0 are sent as address bits.

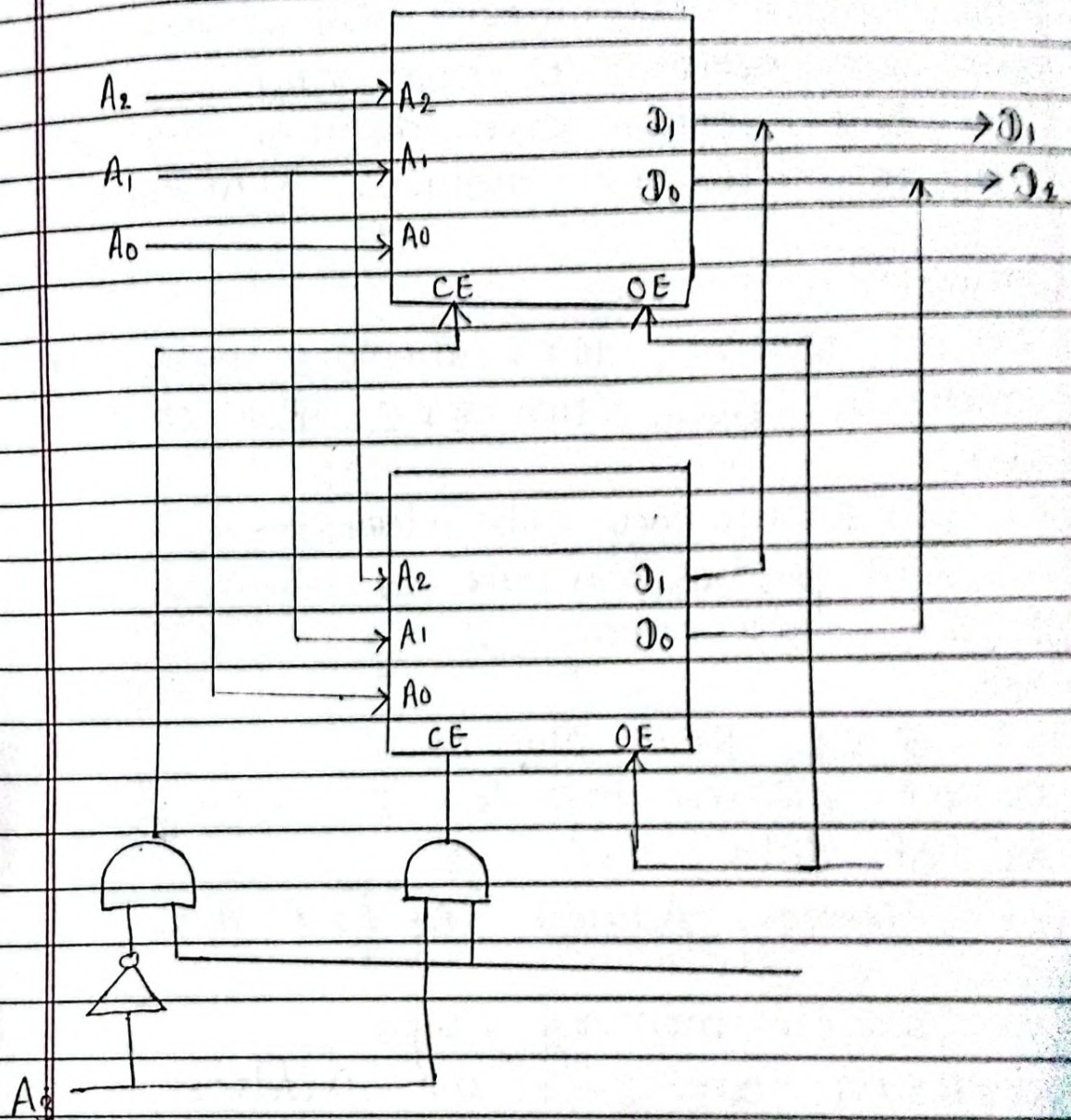


Fig: High order interleaving

Low order interleaving

- In the low order interleaving, the least significant bits of the address select the memory chips.
- The most significant bits are sent as address to each chip.
- In this, consecutive memory address are in different memory modules.

Example:

Let us consider 16 x 2 memory subsystem constructed using two 8 x 2 ROM chips.

Then,

↳ For 16 x 2 memory subsystem:

- no. of address inputs = 4
- no of data lines = 2

And,

↳ For 8 x 2 ROM chips,

- no of address inputs = 3
- no of data lines = 2

↳ For, Memory Address $A_3 A_2 A_1 A_0$,

A_0 selects memory chips.

$A_3 A_2 A_1$ are sent as address.

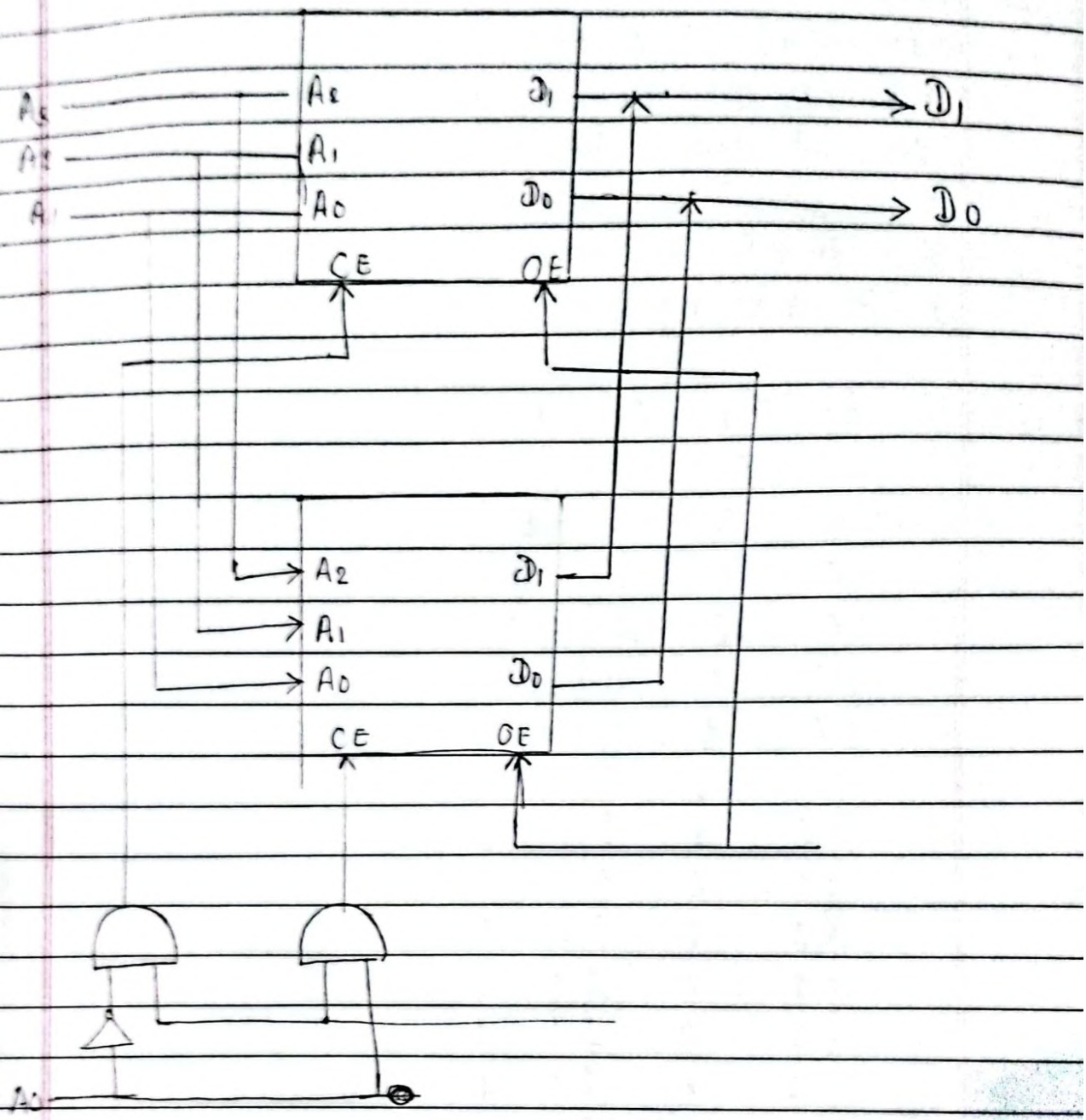


Fig: Low Order Interleaving

Date / /
Page No.

(9) Design two dimensional chip organization of 16 X 2 ROM chips.

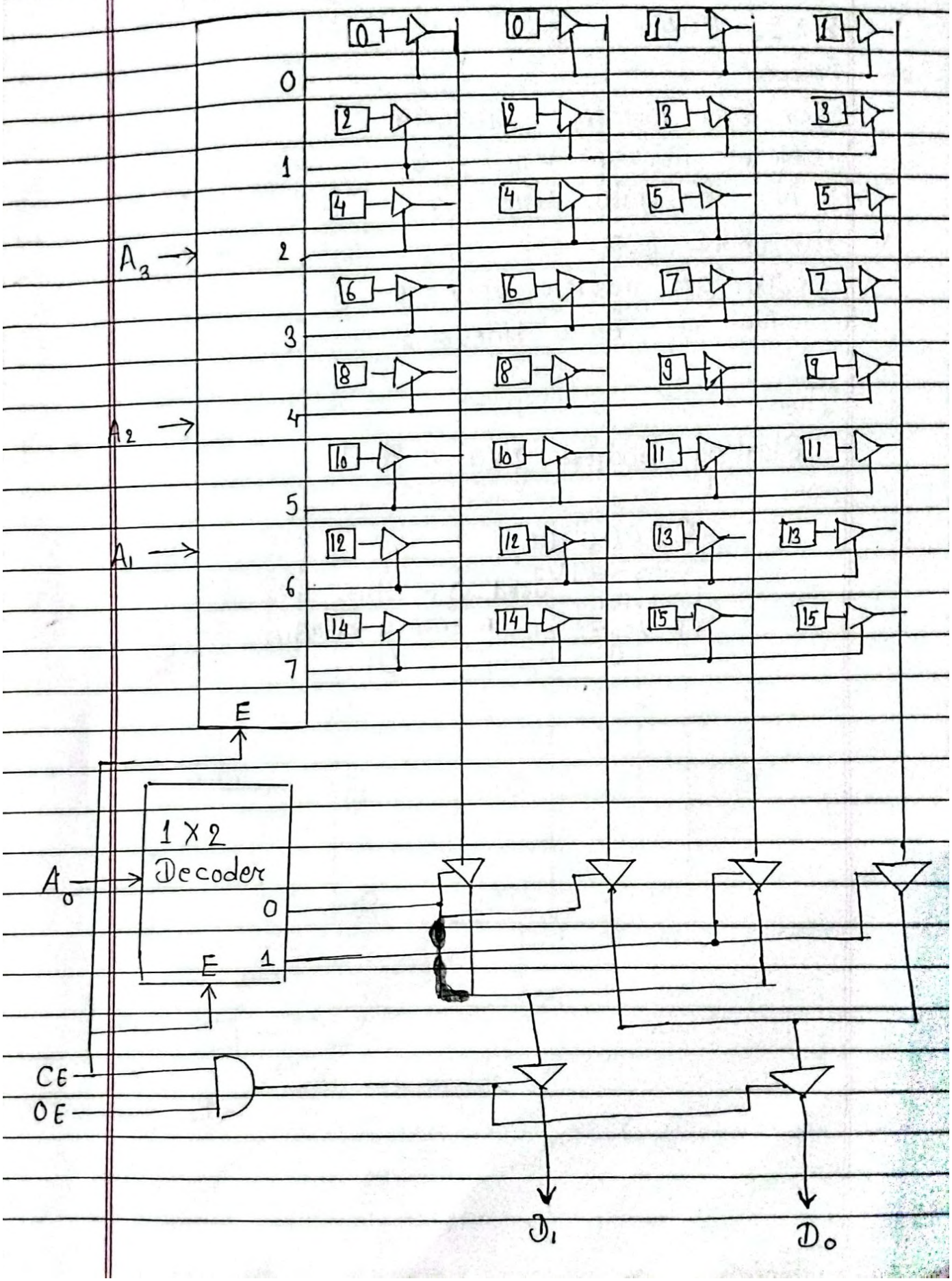
↳ Answer:

For 16 X 2 ROM chips,
No of address inputs = 4 (Say $A_3 A_2 A_1 A_0$)
No of data lines = 2 (Say D_0, D_1)

For 16 X 2 memory chip organization 2-D arrangement is given as:

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

↳ In the above arrangement, for 3 higher order bits the chip value is same and low order is 0,1 alternatively. Then,


A3
A2
A1
A0
E
1 X 2
Decoder
0
E
1
CE
OE
D1
D0

(11) Design 8x4 memory subsystem at address 80H constructed using 8x2 ROM.

↳ Answer:

For 8x4 Memory subsystem,
No of Address input = 3
No. of data lines = 4

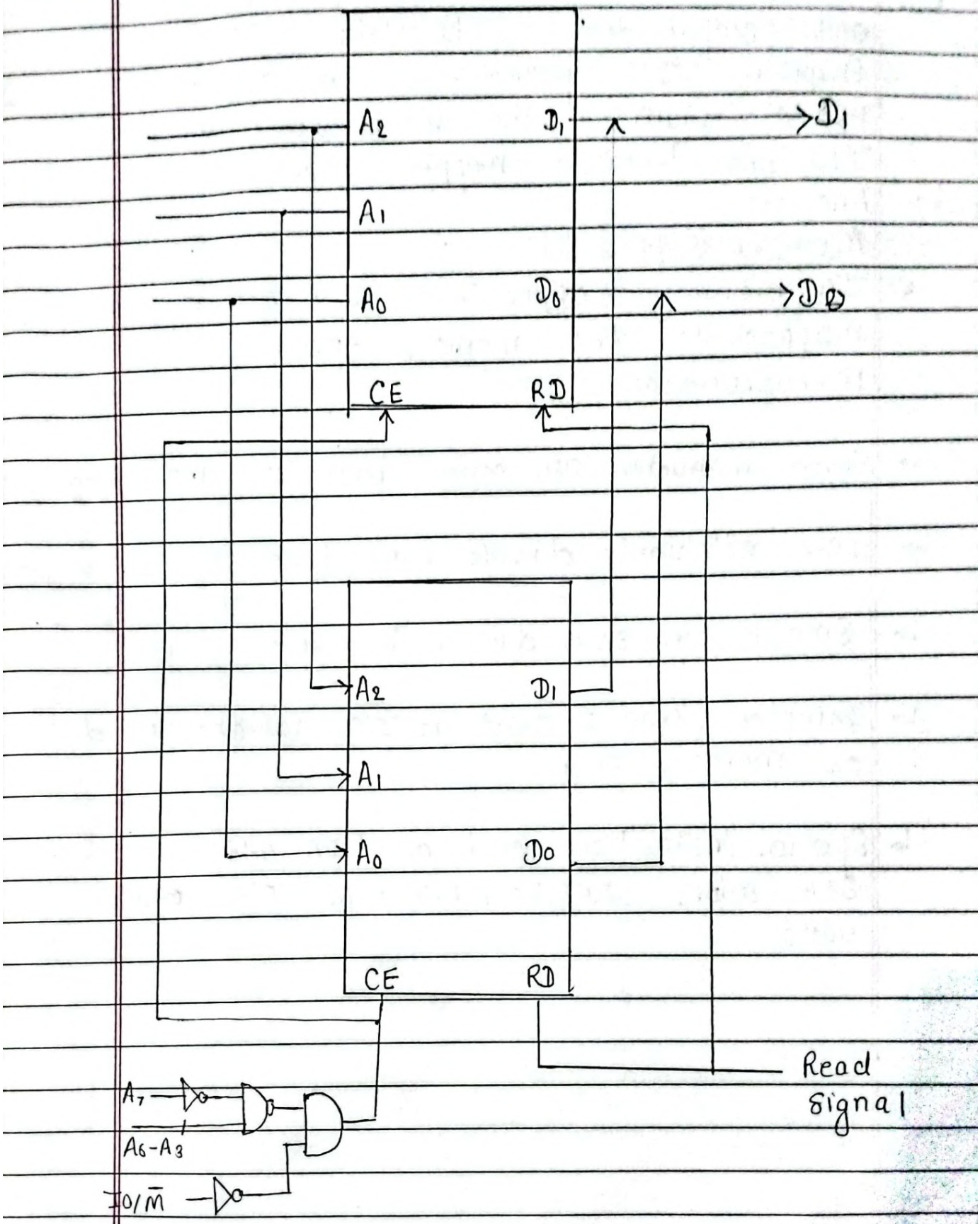
For 8x2 ROM,
No of address input = 3
No. of data lines = 2

Then,
Starting Address given = 80H

So,

10000000

Used for CE Logic. | Used for the memory selection


A2
A1
A0
D1
D0
CE
RD
D1
D0
A2
A1
A0
D1
D0
CE
RD
Read signal
A7
A6-A3
IO/M̄

(12) A computer is connected with multiple input and output devices to the CPU through I/O processor via a common bus. Explain the memory mapped I/O and Isolated mapped I/O.

↳ Answer:

Memory Mapped I/O

↳ In memory mapped I/O, I/O devices are mapped to the memory space of the microprocessor.

↳ Same instruction can control both I/O and memory

↳ Less efficient due to same buses.

↳ Smaller in size due to less buses.

↳ Simpler logic is used as I/O is also treated as memory only.

↳ Special instruction such as LDA addr, STA addr, LDAX Rp, MOV R,M etc are used.

Isolated Mapped I/O

↳ In Isolated Mapped I/O, input/output devices are mapped to separate I/O address space that is different from the memory address space.

↳ Separate instruction control read and write operation in I/O and memory.

↳ More efficient due to ~~more~~ separate buses.

↳ Larger in size due to more buses.

↳ It is complex due to separate logic is used for controlling both memory and I/O.

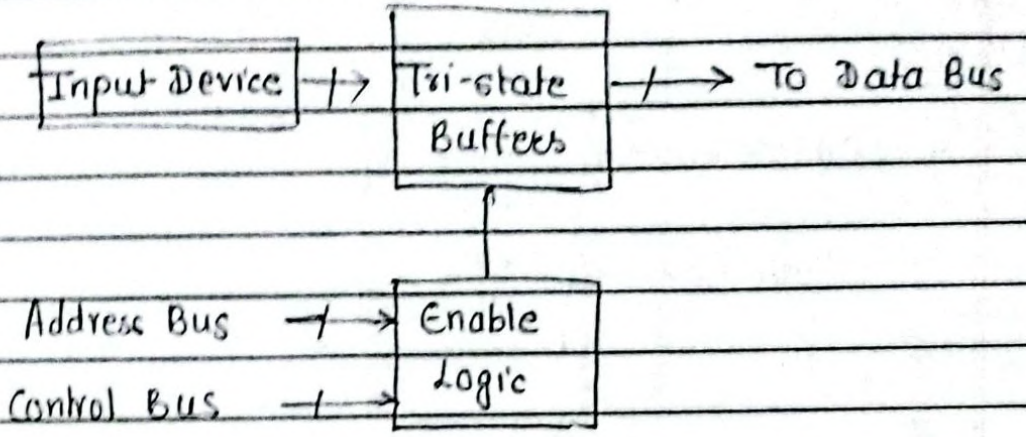
↳ Limited instruction such as IN/OUT.

(13) Write about IO system organization and interfacing for input device and output device with necessary diagram.

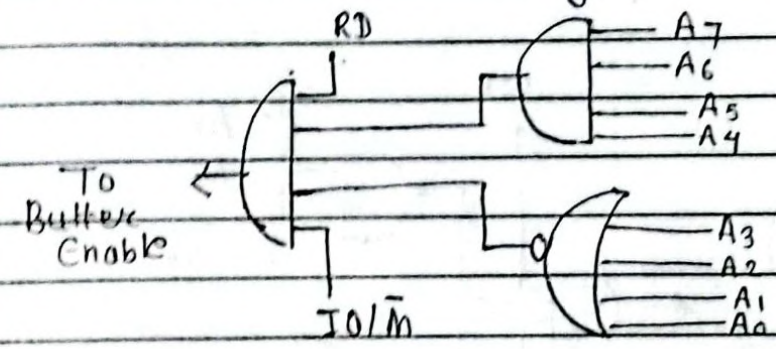
↳ Answer:

(A) Input Device

a) With its interface



b) With the enable logic for the tri-state buffer



Explanation: The data from the input device goes to the tri-state buffer.

↳ When the values on the address bus and control bus are correct, the buffers are enabled and data passes on the data bus.

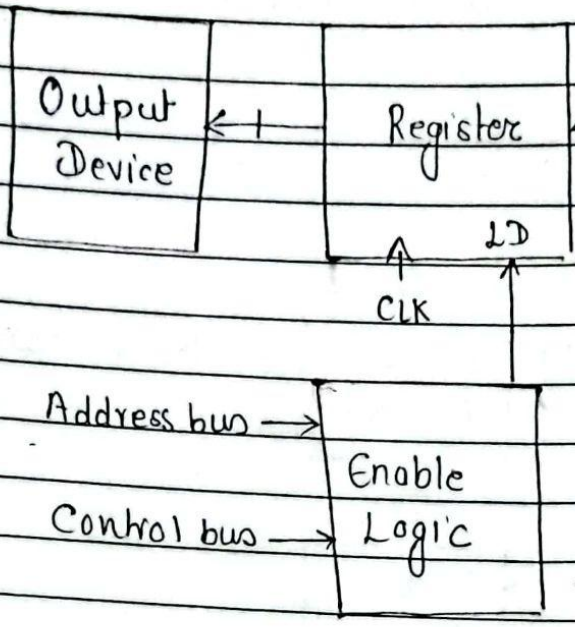
↳ CPU then can read these data.

↳ The key to ~~these~~ this design is enable logic.

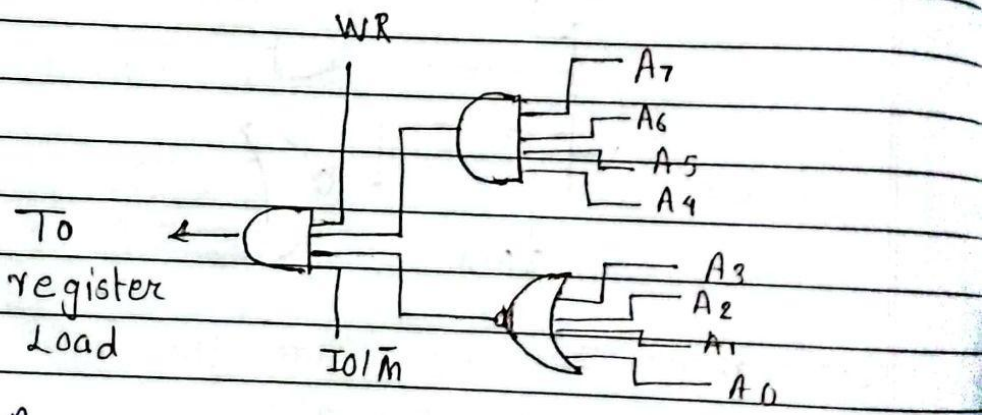
- For the input device, the read signal should be asserted
- Fig(b) shows the enable logic at address 11110000 wi 8-bit address and control signals RD and IO/M̄.

(B) Output Device

a) With its interface



b) The load Logic for the Register



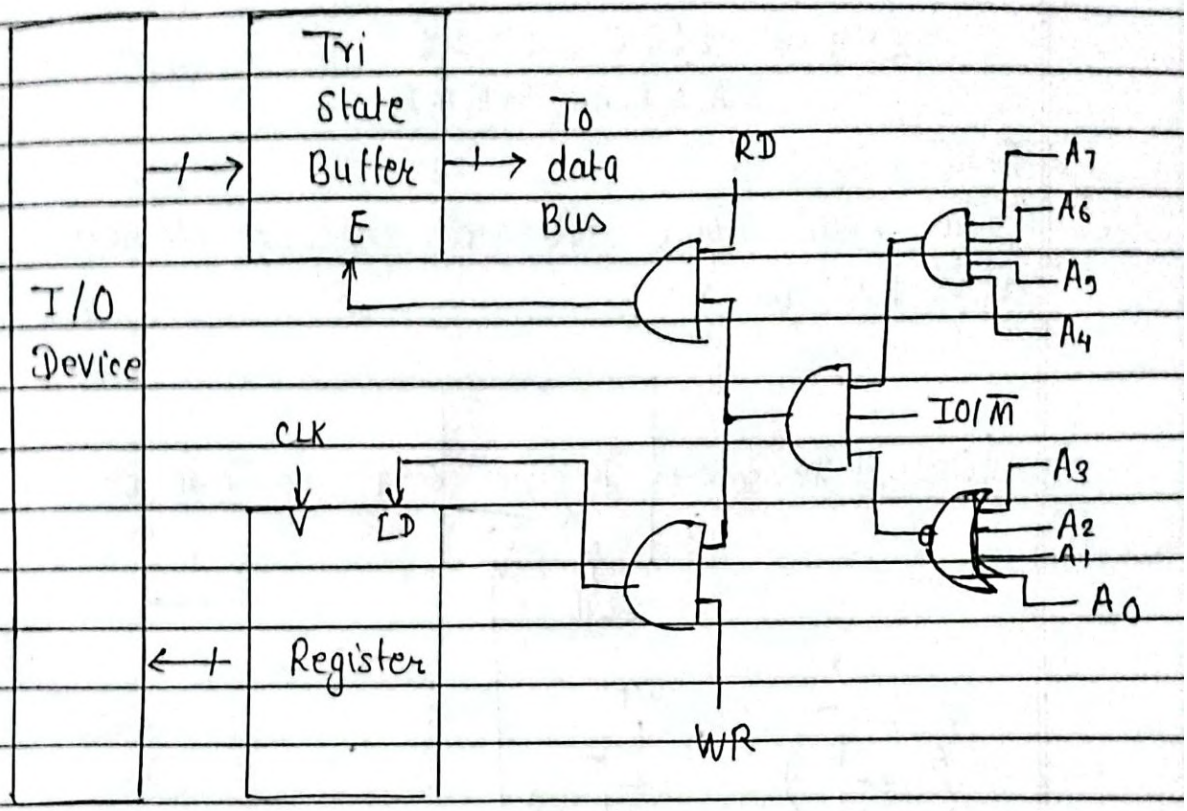
Explanation

↳ For the interface circuit for an output device, the tri-state is replaced by register.

↳ The tri-state buffers are used in input device make sure that no more than one device write data to the bus at any time.

↳ Since, the o/p devices read data from the bus, they do not need buffers so that data can be made available to all output devices.

(c) A bidirectional I/O device with its interface and enable / Load logic.



(14) Generate a load logic for an output device at CFH. Assume that the system is being operated in isolated mode.

↳ Answer

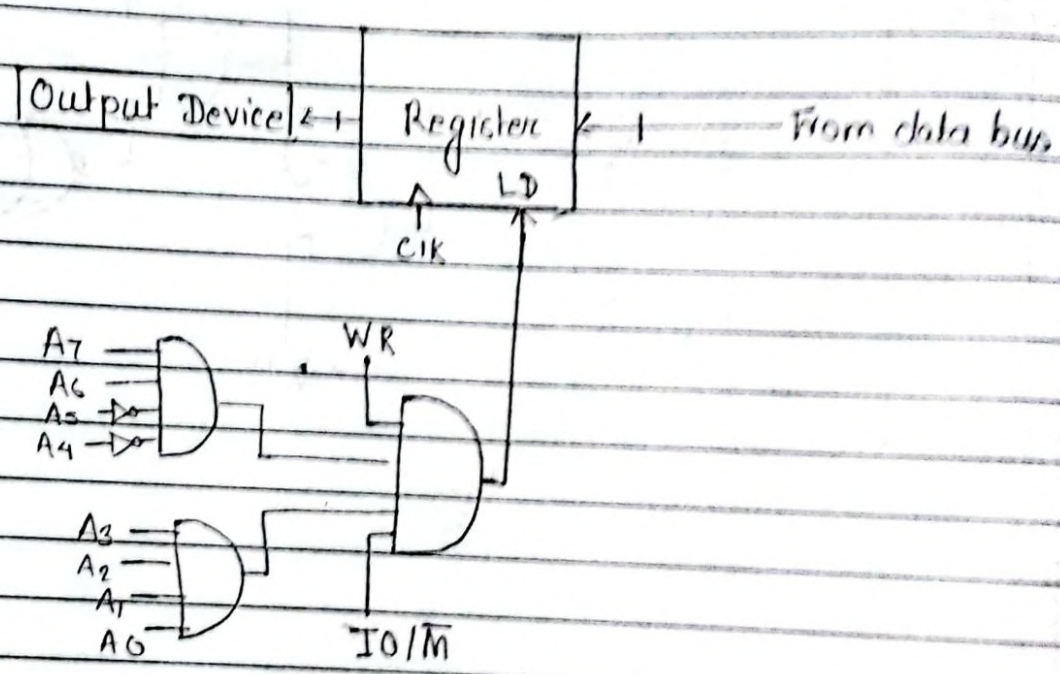
At CFH,

CFH = 1100 1111

$A_7 A_6 A_5 A_4 \; A_3 A_2 A_1 A_0$

Now,

The load logic for an output device at CFH is:



(15) There is an input device at address 00H and an output device at 20H. Illustrate the design with necessary logic.

→ Answer

Address of input device = 00H

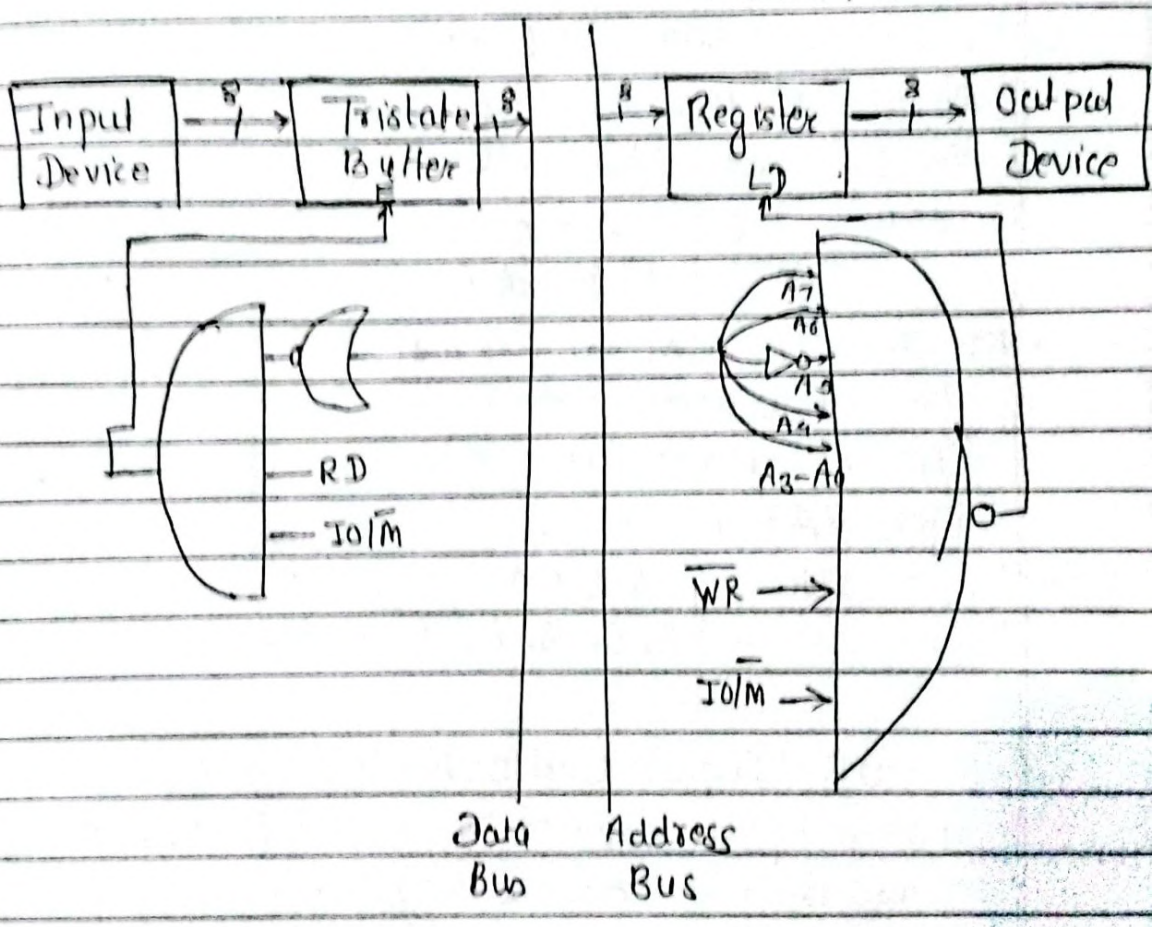
= 0000 0000

$A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

Address of output device = 20H

= 0010 0000

$A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$



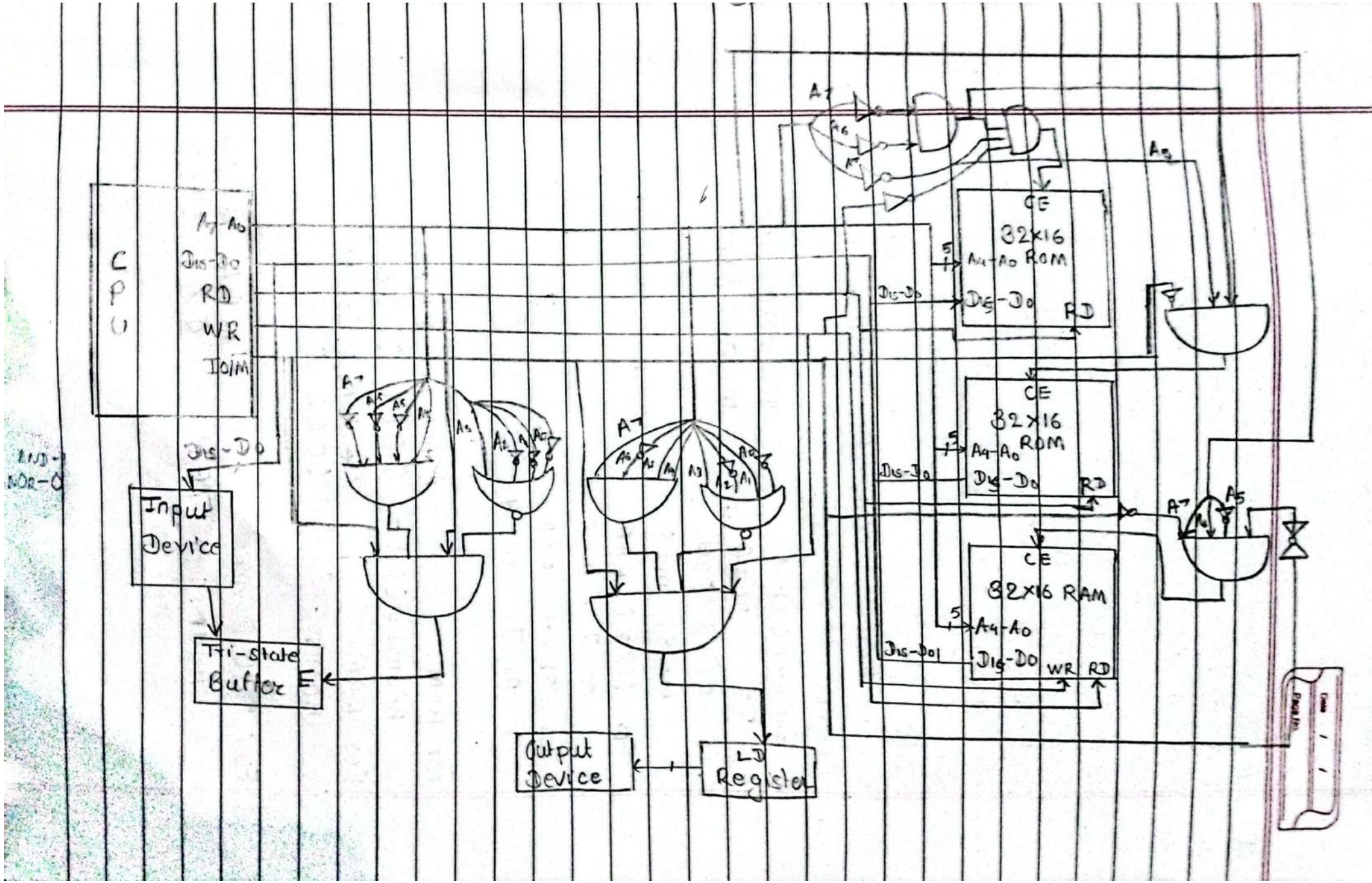
(16) A computer has a CPU with an 8 bit address bus and 16 bit data bus. The computer uses isolated IO. It has 64 x 16 ROM at 00H constructed using two 32 x 16 ROM chips. It also has 32 x 16 of RAM at C0H. The system has an input device at 15H and the output device at 75H. Show the design for the required system including all necessary logic.

↳ Answer:

↳ For 64 x 16 of ROM, ($2^6 \times 16$)
No of address input = 6
No of data lines = 16
00H = 0000 0000
$A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

↳ For 32 x 16 ROM and RAM
No. of address input = 5
No of data lines = 16
For RAM:
C0H = 1100 0000
$A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

For an Input and output Device,
For input device = 17H = [0001 0111]
For output device = B5 = [1011 0101]
$A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$


CPU
A15-A0
D15-D0
RD
WR
IO/M
D15-D0
Input Device
Tri-state Buffer E
Output Device
LD Register
CE
32×16 ROM
A4-A0
D15-D0
RD
CE
32×16 ROM
A4-A0
D15-D0
RD
CE
32×16 RAM
A4-A0
D15-D0
WR
RD
5
D15-D0

17. A computer system has an 8-bit data bus and 8 bit address bus. The system operates in isolated IO mode. It has 64 byte of ROM at 00H, constructed using two 32 byte of ROM. It also has ROM of 128 bytes. The RAM is constructed using 2 128 x 4 RAM chips at address 80H. The system also has an IO device at 7E H. Show the necessary design for the system.

→ Answer:

For 64 x 8 ROM,
n = 6, m = 8

n = Address line
m = data lines

For 32 x 8 ROM,
n = 5, m = 8

For 128 x 8 RAM,
n = 7, m = 8

For 128 x 4 RAM,
n = 7, m = 4

For ROM:

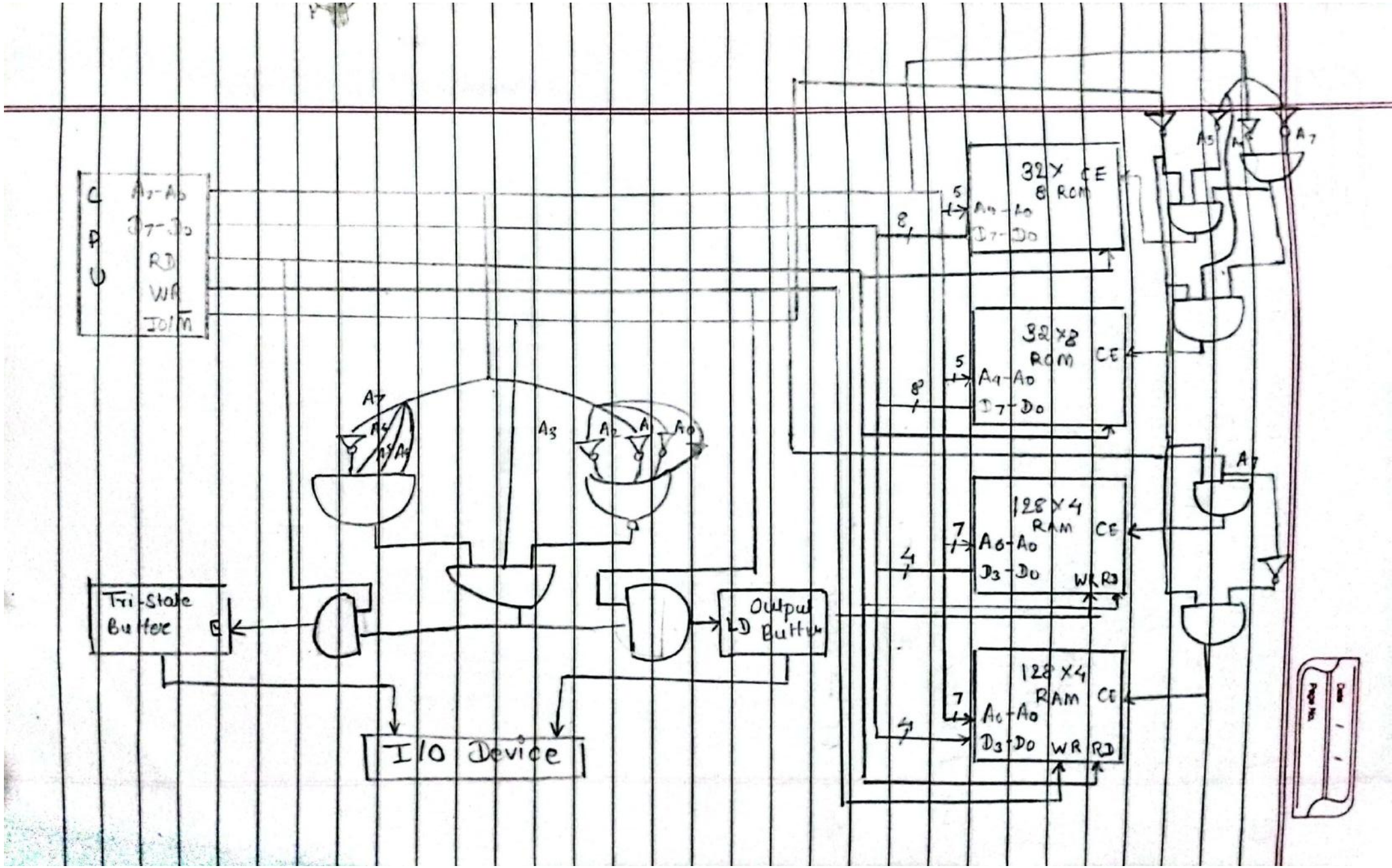
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Address = 00H =	0	0	0	0	0	0	0	0

For RAM,

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Address = 80H =	1	0	0	0	0	0	0	0

For I/O Device:

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Address = 7EH =	0	1	1	1	1	1	1	0


CPU
A7-A0
D7-D0
RD
WR
IO/M
32 X 8 ROM
CE
A4-A0
D7-D0
5
8
32 X 8 ROM
CE
A4-A0
D7-D0
5
8
128 X 4 RAM
CE
A6-A0
D3-D0
WR RD
7
4
128 X 4 RAM
CE
A6-A0
D3-D0
WR RD
7
4
A7
A6
A5
A4
A3
A2
A1
A0
Tri-State Buffer
Output LD Buffer
I/O Device

(19) A computer system with an 8-bit address/data bus using Isolated I/O. It has 16X8 ROM starting at the address 00H constructed using 8X8 chips. 64X8 of RAM starting 80H constructed using 64X4 chips. There is an I/O device at 40H.

↳ Answer:

Same.

20, 21, 22 → Same.

(23) Design an 8X4 ROM chip using 4X4 low level ROM chips. Illustrate using interleaving.

↳ Answer:

For 8X4 ROM chip:

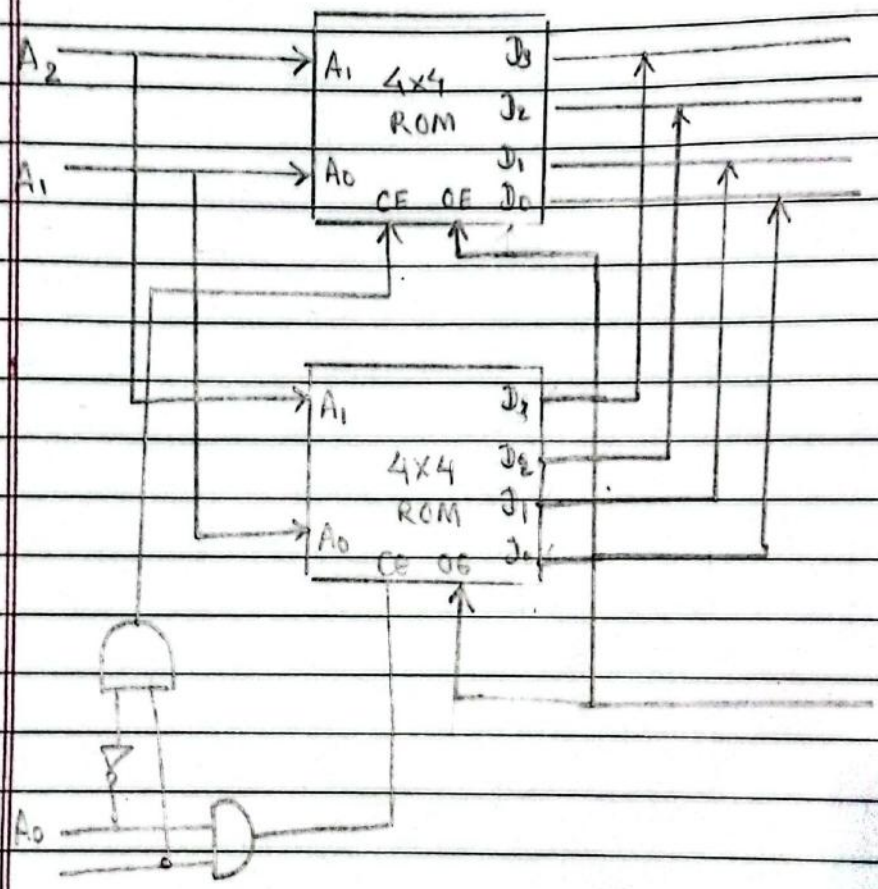
No of address input = 3

No. of data lines = 4

For 4 x 4 ROM chip,

No. of address input = 2

No. of data lines = 4



(25) What do you mean by RTL? In how many ways RTL are implemented. Explain with an example.

↳ Answer:

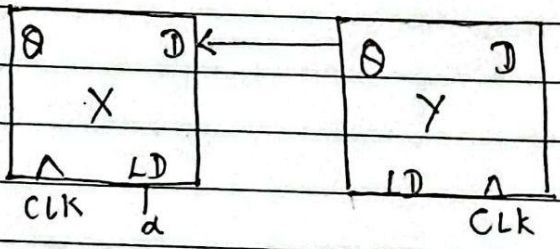
- RTL stands for Register Transfer Language.
- It is a symbolic representation of notation used to specify the sequence of the micro-operations.
- Notation used by RTL is: ~~conditions:~~
  conditions: micro-operations

↳ Example: Consider that, the transfer $X \leftarrow Y$ occurs when control signal α is high in RTL.

$$\alpha : X \leftarrow Y$$

RTL Implementation

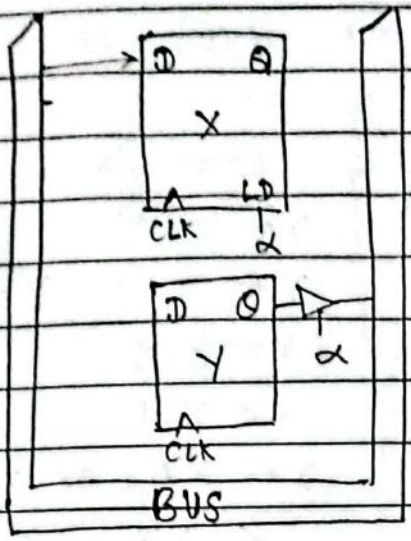
(a) Direct Connection



↳ In this method, the components in the designs are connected directly to each other.

↳ Each component has its own individual connections to other components forming a point-to-point connection.

(b) Bus Connection.



↳ A bus is a shared ~~commia~~ communication pathway that connects multiple component in design.

↳ Bus acts as a communication medium, allowing the components to transmit and receive data.

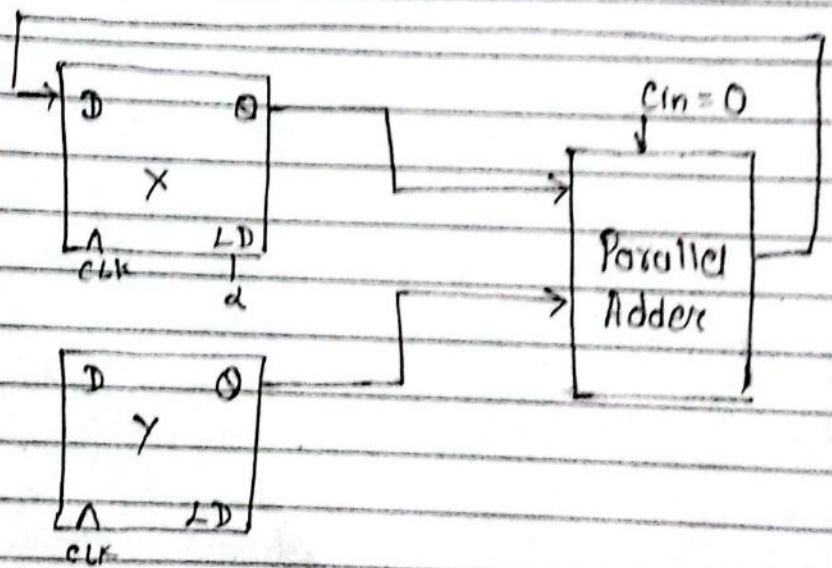
(26) List out the arithmetic and logical microoperations and, illustrate the implementation of each of them.

↳ Answer:

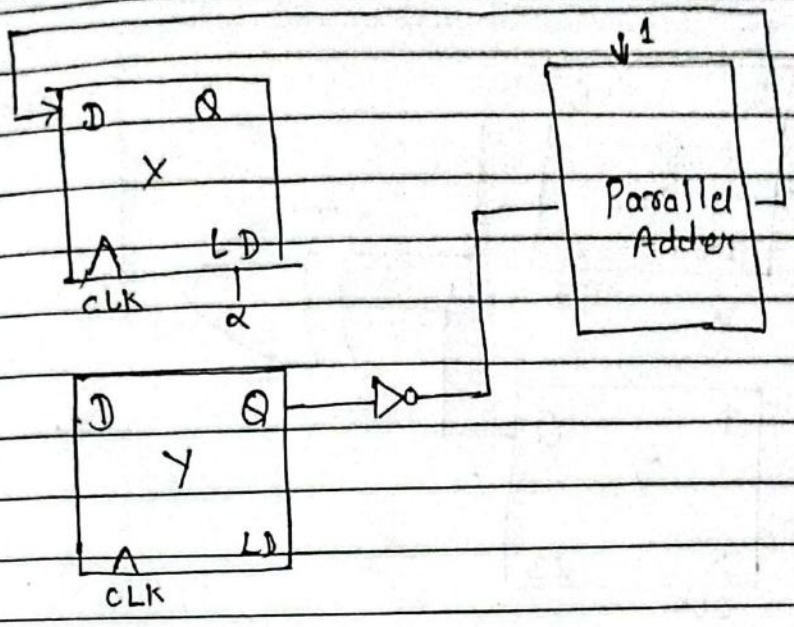
Operations	Examples
Add	$X \leftarrow X+Y$
Subtract	$Y \leftarrow X-Y$; $X \leftarrow X+Y'+1$
Increment	$X \leftarrow X+1$
Decrement	$Y \leftarrow X-1$
AND	$X \leftarrow X \wedge Y$ or $X \leftarrow XY$
OR	$Y \leftarrow X \vee Y$
XOR	$X \leftarrow X \oplus Y$
NOT	$X \leftarrow X'$

Implementation

(a) Addition ($d: X \leftarrow X+Y$)

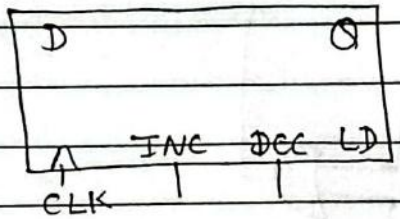


(b) Subtraction : ($\alpha \rightarrow X - Y$ or $X \leftarrow X + \overline{Y} + 1$)

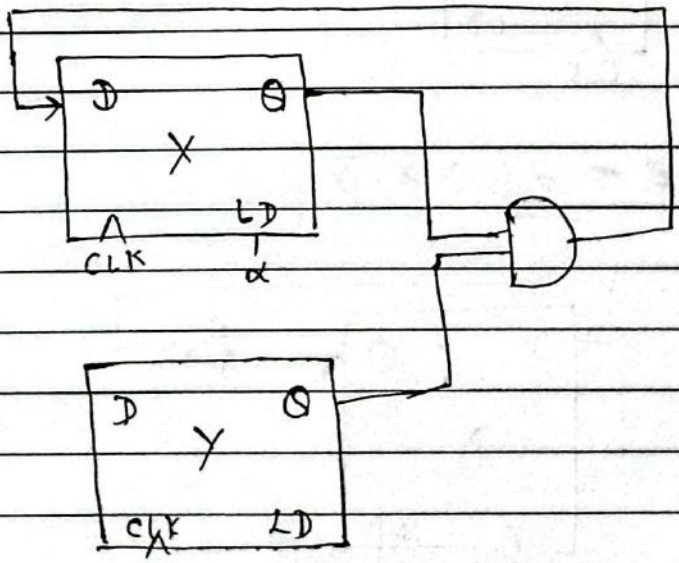


(c) Increment & Decrement

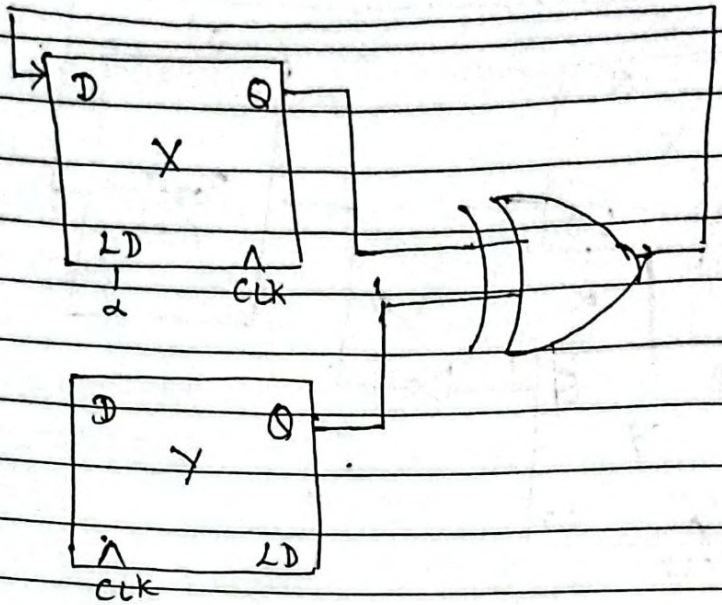
α : $X \leftarrow X + 1$, β : $X \leftarrow X - 1$



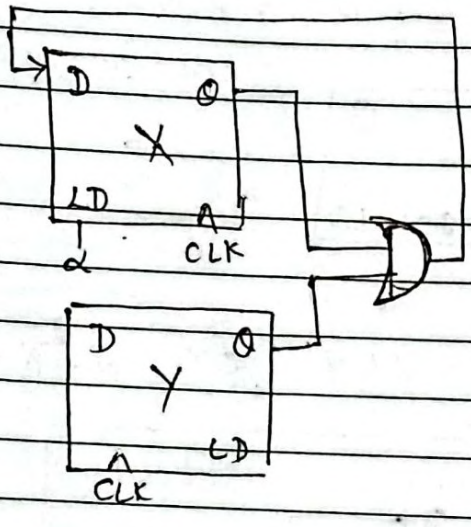
(d) AND α : $X \leftarrow X \wedge Y$ or α : $X \leftarrow X.Y$



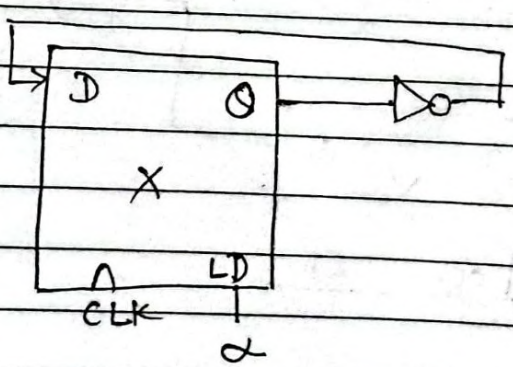
(e) XOR



(f) OR $\alpha : X \leftarrow X \vee Y$



(g) NOT: $\alpha : X \leftarrow \overline{X}$



(27) Draw the state diagram of the modulo 6 counter and implement it using registers.

↳ Answer:

State Table of Modulo-6 Counter

Present State	U	Next State	C	$V_2V_1V_0$
S_0	0	S_0	1	000
S_0	1	S_1	0	001
S_1	0	S_1	0	001
S_1	1	S_2	0	010
S_2	0	S_2	0	010
S_2	1	S_3	0	011
S_3	0	S_3	0	011
S_3	1	S_4	0	100
S_4	0	S_4	0	100
S_4	1	S_5	0	101
S_5	0	S_5	0	101
S_5	1	S_0	1	000
S_6	X	S_0	1	000
S_7	X	S_0	1	000

Here:

States = $S_1, S_2, S_3, S_4, S_5, S_6, S_7$

States of counter = $V_2V_1V_0$

U is used to control the counter.

Rough:

↳ Here, when the counters value is '000' to '100' and its 'up' signal 'U' is asserted, the output of the counter is incremented. This condition can be expressed as $(S_0+S_1+S_2+S_3+S_4)U$.

↳ Under this condition, C is also set to '0'.

↳ Under V to represent the value $V_2V_1V_0$, these two actions can be expressed as:

$$V \leftarrow V+1 \;,\; C \leftarrow 0$$

⇒ The resulting RTL is:

$$(S_0+S_1+S_2+S_3+S_4)\,U : V \leftarrow V+1, C \leftarrow 0$$

Combining with counter in state S_5 ($V = 101$) and $U=1$, the counter must be reset to 000 and C must be set to 1.

$$S_5 U : V \leftarrow 0, C \leftarrow 1.$$

This leaves only the condition

$$(S_0+S_1+S_2+S_3+S_4+S_5)\,\bar{U}$$

unaccounted.

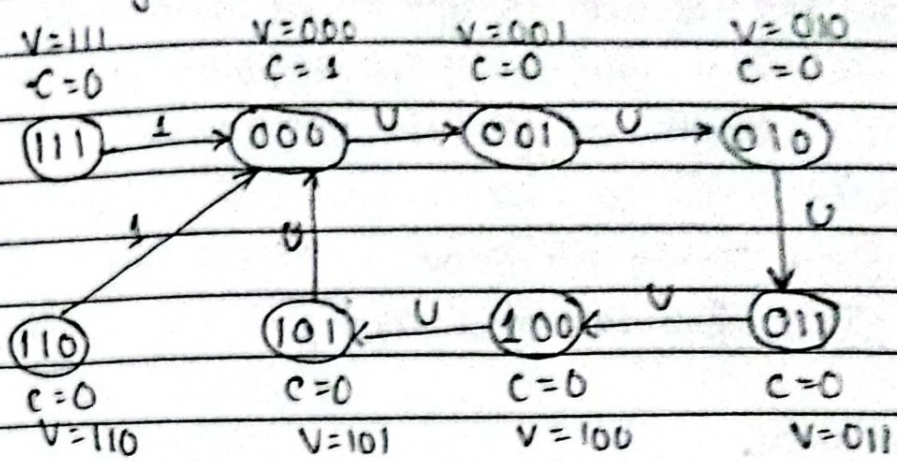
Since $S_5 U$ and S_0+S_7 trigger the same microoperation, they can be combined.

Thus the final behaviour of the module 6 counter can be expressed by:

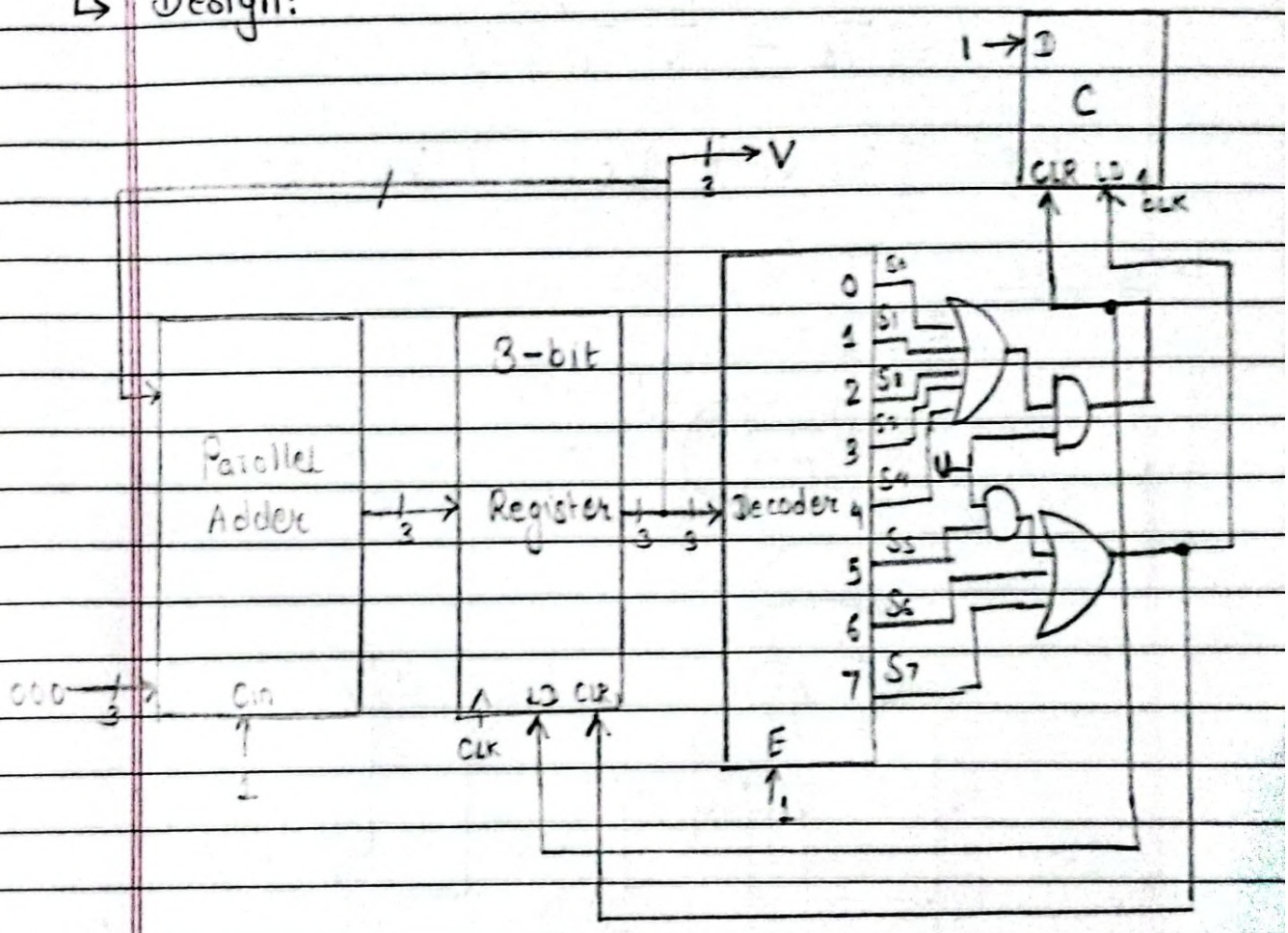
$$(S_0+S_1+S_2+S_3+S_4)\,U : V \leftarrow V+1, C \leftarrow 0$$

$$(S_5 U + S_6 + S_7) : V \leftarrow 0, C \leftarrow 1.$$

↳ State diagram for the modulo 5 counter:



↳ Design:



(28) Draw the state diagram of the modulo 6 counter and implement it using ~~regi~~ a 8 bit counter.

↳ Answer:

State Table → same.

State Diagram → same.

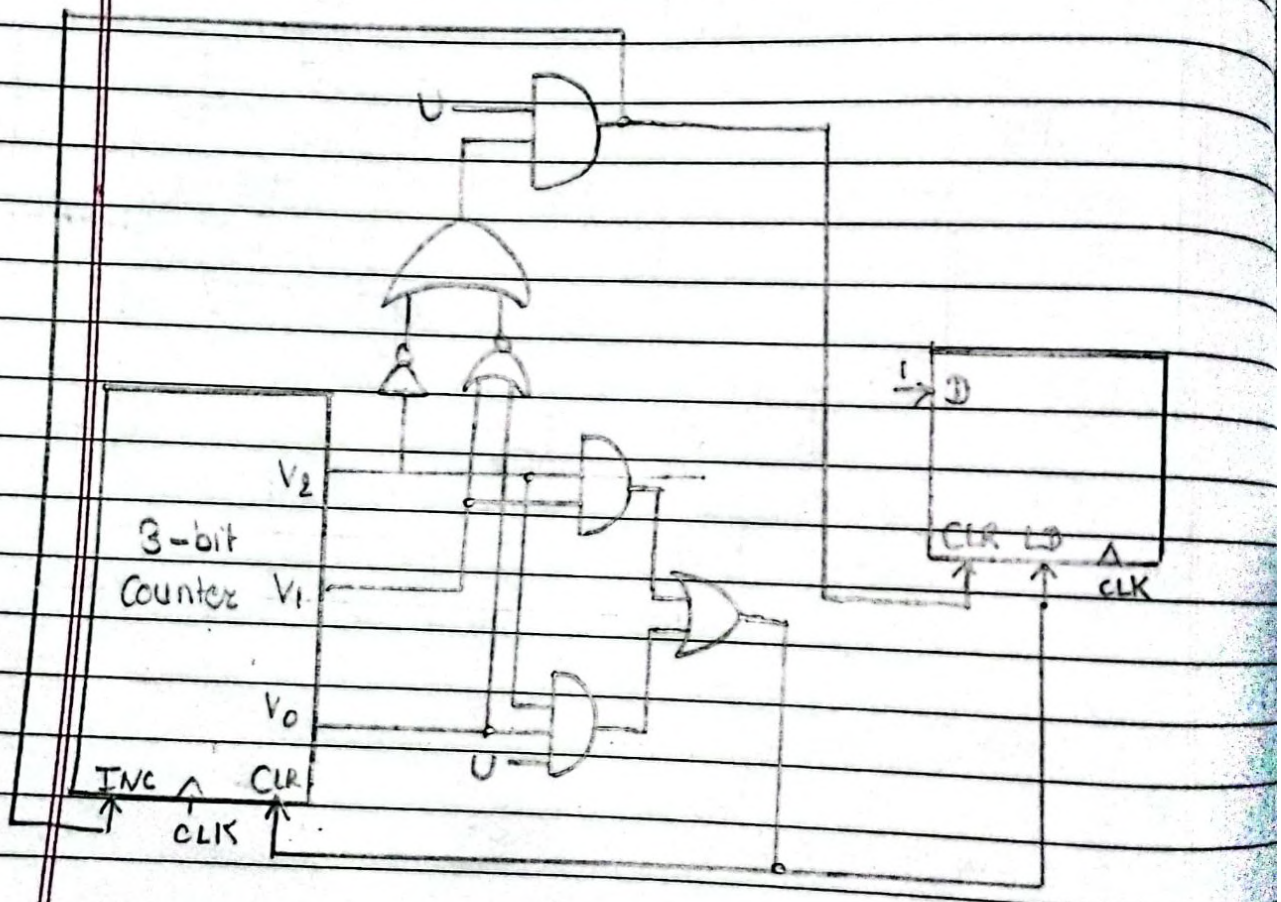
Behaviour:

Clear: $(\overline{V_1} + (\overline{V_1 + V_0})\, U$

$= (\overline{V_2} + \overline{V_1} \cdot \overline{V_0})\, U$

Load: $(V_2 \cdot V_1 + V_2 V_0 U)$: $V \leftarrow 0, C \leftarrow 1$

↳ Design



(2.9) Explain different types of shift micro operations with examples.

↳ Answer:

There are four types of shift microoperations:

a. Arithmetic Shift
b. Decimal Shift
c. Linear Shift
d. Circular Shift

a. Arithmetic Shift

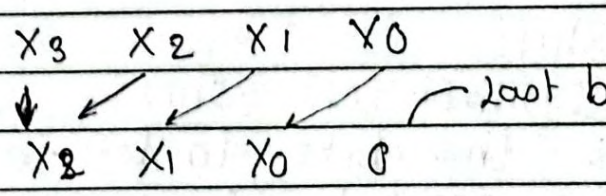
↳ (i) work with number that have signed formats.

↳ Leftmost bit is sign bit.

(i) Arithmetic Shift Left (ashl)

- All bits are shifted to left except the bit at the begining, which is the sign bit and is discarded.
- Bit 0 is loaded into last position.

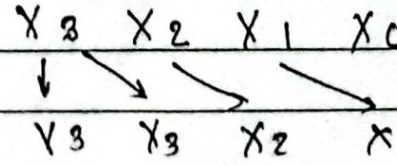
Eg:



(ii) Arithmetic Shift Right (ashr)

- Each bit is moved to the right one by one and the least significant (LSB) bit is rejected and the empty most significant bit (MSB) is filled with the value of the previous MSB.

Eg:



(b) Decimal Shift

- For a BCD representation, each decimal digit is represented by 4 bits. Similar to linear shift but it shifts (one digit) inside of one bit

(i) Decimal shift left (dshl)

A_1 A_0 = A_0 0000

Eg:

1101 0011

↙

A_0 0000

(ii) Decimal shift right (dshr)

Eg:

A_1 A_0

1101 0011

↓

0000 A_1

(c) Linear Shift

(i) Linear shift left (shl)

- Shifts all the bits to left
- MSB is discarded at beginning and we load zero at LSB a vacant position

a_3 a_2 a_1 a_0

↙ ↙ ↙

a_2 a_1 a_0 0 ← vacant position at last.

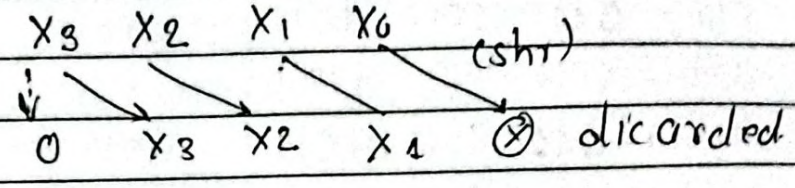
(ii) Linear shift Right (shr)

↳ Shifts all the bits at right.

↳ Bit at last position is discarded.

↳ The vacant position at begining is loaded.

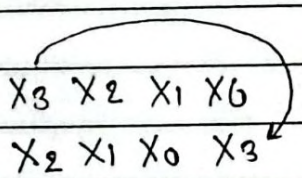
Example:



(d) Circuar Shift

i) Circular shift left (cil)

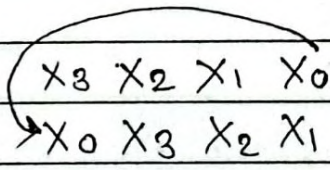
↳ All bits are shifted to left and the bit at the begining is circulated back at the end position.



ii) Circular shift Right (Cir)

↳ All bits are shifted right and the bit at last is circulated at the 1st position.

Eg:



(30) Perform eight different shift operations on X = 1101 0101 0110

↳ Answer: 1101 0101 0110

shl = 1010 1010 1100

shr = ~~0101 0101 0110~~ 0110 1010 1011

cil = 1010 1010 1101

cir = 0110 1010 1011

ashl = 1010 1010 1100

ashr = 1110 1010 1011

dshl = 1011 0000 0000

dshr = 0000 0000 1101

(31) Repeat the above on 1010 1100

↳ Answer: 1010 1100

shl = 0101 1000

shr = 0101 0110

cil = 0101 1001

cir = 0101 0110

ashl = 0101 1001

ashr = 1101 0110

dshl = 1100 0000

dshr = 0000 1010

(32) Write about VHDL code and explain different sections of VHDL with its advantages and disadvantages.

↳ Answer:

VHDL

: Very High-speed Integrated Circuit Hardware Description Language.

• Programming language used to model the digital system by dataflow, behavioural and structural style of modeling.

↳ VHDL is used for following purposes:

. For describing hardware and for the simulation of hardware.

• As a modeling language and for early performance estimation of system architecture.

↳ Sections of VHDL

@ Entity Declaration

. It defines the names, input-output, signals and modes of a hardware module.

Syntax :-

```
entity enhity_name is
        port declaration;
        end enhity_name;
```

In port declarations, reserved words are:

in :- port can be Read

out :- port can be written

inout :- can be read and written
Buffer :- port can be read and written, it can have only one source.

(b) Architecture

- Architecture can be described using structural, dataflow, behavorial and mixed style.

Syntax :-

```
architecture architecture_name of entity_name;
architecture_declarative_part;
begin
    statements;
    end architecture_name;
```

↳ A buit-in or user defined signal type
eg:- bit, characters, Boolean, std_logic etc.

Advantages

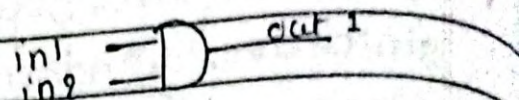
i. It supports various design methodologies like Top-down and Bottom-up approach.
ii. It supports all CAD tools.
iii. It provides tight coupling to low levels of design.
iv) Supports multi-level abstraction.

Disadvantages

i) It requires specific knowledge of the structure and syntax of the language.
ii) More difficult to visualize and troubleshoot a design.
iii) Some VHDL programme can't be synthesized.
iv) It is more difficult to learn.

(32) Write down VHDL code for the following:

a. AND gate



↳
```
library ieee;
use ieee.std_logic_1164.all;

entity AND_Gate is
  Port ( in1, in2 : in std_logic;
         out1 : out std_logic);
end AND_Gate;

architecture behavioral_and of AND_Gate is
begin
    out1 <= in1 and AND in2;
end behavioral_and;
```

b. OR Gate

```
library ieee;
use ieee.std_logic_1164.all;

entity OR_GATE is
  Port ( in1, in2 : in std_logic;
         out1 : out std_logic);
end OR_GATE;

architecture behavioral_or of OR_GATE is
begin
    out <= in1 or in2;
end behavioral_or;
```

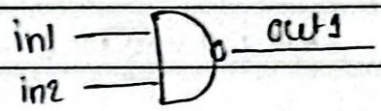
© NOT gate

↳
```
library ieee;
use ieee.std_logic_1164.all;

entity NOT_GATE is
    Port ( A: in std_logic;
           Y: out std_logic);
end NOT_GATE;

architecture behavorial_NOT of NOT_GATE is
begin
    Y <= not(A);
end behavorial_NOT;
```

(d) NAND Gate

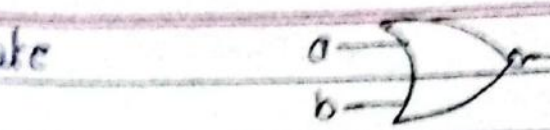


↳
```
library ieee;
use ieee.std_logic_1164.all;

entity NAND_GATE is
    Port( in1, in2 : in std_logic;
          out1 : out std_logic );
end NAND_GATE;

architecture behavioral_NAND of NAND_GATE
begin
    out <= in1 nand in2;
end behavioral_NAND;
```

(e) NOR gate

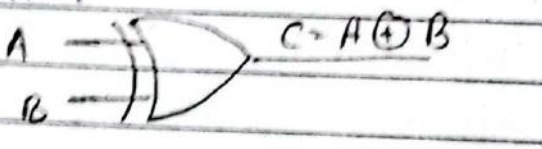



```
library ieee;
use ieee.std_logic_1164.all;

entity NOR_GATE is
    port (a, b : in std_logic;
          c : out std_logic);
end NOR_GATE;

architecture behavioral_NOR of NOR_GATE is
  begin
      c <= a nor b;
end behavioral_NOR;
```

(f) XOR GATE

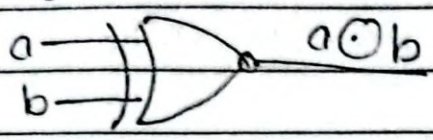



```
library ieee;
use ieee.std_logic_1164.all;

entity XOR_GATE is
    port ( A,B : in std_logic;
           C : out std_logic);
end XOR_GATE;
```

```
architecture behavioral_XOR of XOR_GATE IS
  begin
      c<= A xor B;
end behavioral_XOR;
```

(9) XNOR GATE



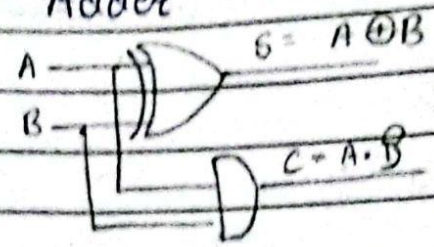
↳
```
library ieee;
use ieee.std_logic_1164.all;

entity XNOR_GATE IS
      Port (a, b : in std_logic;
            c : out std_lo
end XNOR_GATE;

architecture behavioral_XNOR OF XNOR_GATE IS
 begin
      c<= a xnor b
end behavioral_XNOR;
```

h. Half Adder

↳



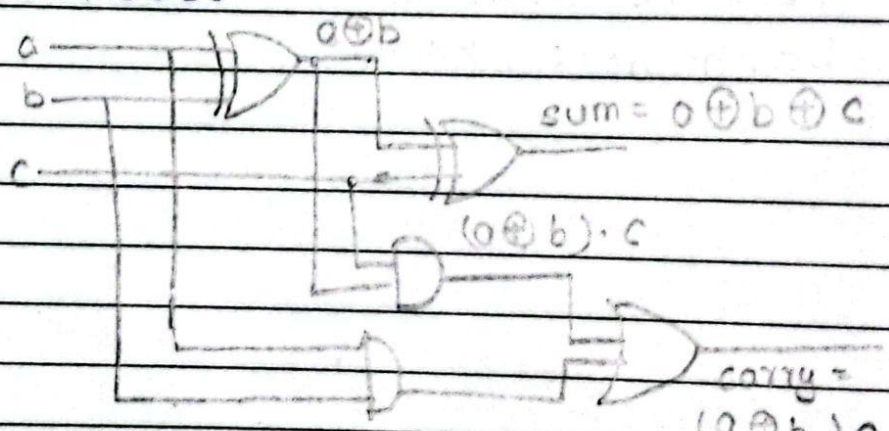

```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
    Port ( a, b : in std_logic;
           sum, carry : out std_logic);
end half_adder;

architecture behavioral_half of half_adder is
begin
    sum <= A xor B;
    carry <= A and B;
end behavioral_half;
```

i. Full Adder

↳

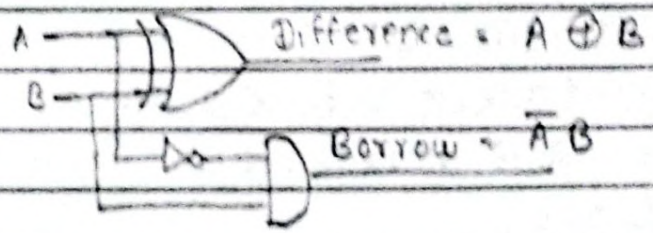



```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity full_adder is
    Port (a,b,c : in std_logic;
          sum,carry : out std_logic);
end full_adder;

architecture behavorial_full of full_adder is
begin
    sum <= (A xor B) xor C
    carry <= ((a and b) or (b and c) or (a and c));
end behavorial_full;
```

ii) Half Subtractor

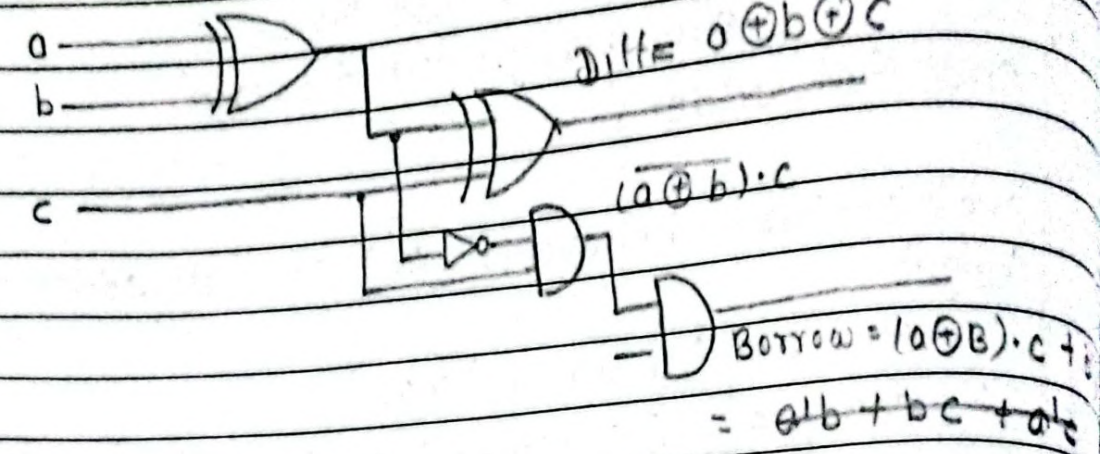



```
library ieee;
use ieee.std_logic_1164.all;

entity half_subtractor is
    Port ( A,B : int std_logic;
           Difference, Borrow : out std_logic);
end half_subtractor;

architecture h_subtractor of half_subtractor is
begin
    Difference <= A xor B;
    Borrow <= ((not A) and B);
end h_subtractor;
```

k) Full subtractor




```
library ieee;
use ieee.std_logic_1164.all;
entity full_sub is
    Port (a,b,c : in std_logic;
          Diff, Borrow : out std_logic);
end full_sub;

architecture full of full_sub is
begin
    Diff <= (a xnor b) xnor c
    Borrow <= ((((not (a xnor b)) and c) or
                (not(a) and b) ))

end full;
```

(m) 4x1 Multiplexer

```
Library ieee;
use ieee.std_logic_1164.all;

entity MUX is
    Port (S1, S0, D0, D1, D2, D3 : in std_logic;
          Y : out std_logic);
end MUX

architecture mux of MUX is
begin
    Y <= ((not S0) and (not S1) and D0) or
         (S0 and (not S1) and D1) or
         ((not S0) and S1 and D2) or
         (S0 and S1 and D3);
end mux;
```

n) 1x4 demultiplexer

```
Library ieee;
use ieee.std_logic_1164.all;
entity DEMUX is
   Port (S1, S0, D : in std_logic;
         Y0, Y1, Y2, Y3 : out std_logic);
end MUX;
architecture demux of DEMUX is
begin
    Y0 <= ((not S0) and (not S1) and D);
    Y1 <= ((not S0) and S1 and D);
    Y2 <= (S0 and (not S1) and D);
    Y3 <= (S0 and S1 and D);
end demux;
```

m) 8 x 3 encoder

```
Library ieee;
use ieee.std_logic_1164.all;
entity encode is
    Port (i0, i1, i2, i3, i4, i6, i7 : in std_logic;
          o0, o1, o2 : out std_logic);
end encode;
architecture encoder of encode is
begin
    o0 <= i4 or i5 or i6 or i7;
    o1 <= i2 or i3 or i6 or i7;
    o2 <= i1 or i3 or i5 or i7;
end encoder;
```

o) 3 x 8 decoder

```
Library ieee;
use ieee.std_logic_1164.all;
entity decode is
    Port (i0, i1, i2 : in std_logic;
          o0, o1, o2, o3, o4, o5, o6, o7 : out std_logic);
end decode

architecture decoder of decode is
begin
    o0 <= (not i0) and not(i1) and not(i2);
    o1 <= (not i0) and (not i1) and i2;
    o2 <= (not i0) and i1 and (not i2);
    o3 <= (not i0) and i1 and i2;
    o4 <= i0 and (not i1) and (not i2);
    o5 <= i0 and (not i1) and i2;
    o6 <= i0 and i1 and (not i2);
    o7 <= i0 and i2 and i2;
end decoder
```

(2) SR flip-flop

```
Library ieee;
use ieee std_logic_1164.all;

entity SR is
  Port ( s, r, clk : in std_logic;
  r, qb : inout std_logic);
end SR;

architecture data of SR is
  signal s1, r1 : std_logic;
  begin
      s1 <= s nand clk;
      r1 <= r nand clk;
      q <= s1 nand qb;
      qb <= r1 nand q;
end data;
```

(9) JK Flipflop

```
Library ieee;
use ieee.std_logic_1164.all;

entity JK is
 Port (J, K, clk, reset: in std_logic;
       q, qbar : out std_logic);
end JK;

Architecture data of JK is
 signal s1, s2, s3, s4: std_logic;
 begin
    s1 <= not (J and clk and qbar);
    s2 <= not (K and clk and q);
    s3 <= s1 nand q_bar;
    s4 <= s2 nand q;
    q <= s3 and (not (reset));
    qbar <= s4 and reset;

end data;
```

① T flipflop.

```
Library ieee;
use ieee.std_logic_1164.all;

entity T_flip is
    port( T, clk, clr: in std_logic;
          Q: out std_logic);
end T_flip;

architecture T_flipflop of T_flip is
    signal temp = std_logic;
    begin
     process (clk, clr)
       begin
       if clr = '1' then
          temp <= '0';
       else if ( clk 'event' and clk = '1' ) then
          if T = '0' then
           temp <= temp;
       else if T = '1' then
           temp <= not (temp);
       end if;
          end if;
     end process;
     Q <= temp;
end T_flipflop;
```

5. D - Flip flop

```
library ieee;
use ieee.std_logic_1164.all;

entity D_flip
    Port ( d, clk: in std_logic;
           q, qb : inout std_logic);

end D_flip;

architecture data of D_flip is
  signal d1, s1, r1 : std_logic;
  begin
     s1 <= d nand clk;
     d <= d nand d;
     r1 <= d nand d;
     q <= s1 nand qb;
     qb <= r1 nand q;
end data;
```

34) Write down the VHDL code for Modulo 6 counter using a low level of abstraction.

↳

```
Library ieee;
use ieee.std_logic_1164.all;

entity mod6 is
  Port ( U, clk : in std_logic;
         q2, q1, q0 : buffer std_logic;
         V2, V1, V0, C : out std_logic);
end mod6;

architecture modulo6 of mod6 is
begin cct_mod6 : process (q2, q1, q0, U, clk)
begin if rising_edge(clk) then
    q2 <= (q2 and (not q1) and (not q0)) or
          (q2 and (not q1) and (not U)) or
          ((not q2) and q1 and q0 and U);

    q1 <= ((not q2) and q1 and (not q0)) or
          ((not q2) and q1 and (not U)) or
          ((not q2) and (not q1) and q0 and U);

    q0 <= ((not q2) and (not q0) and U) or
          ((not q1) and (not q0) and U) or
          ((not q2) and q0 and (not U)) or
          ((not q1) and q0 and (not U));
. end if;
    V2 <= q2;
    V1 <= q1;
    V0 <= (q2 and q1) or q0;
```

```
    c <= not (q2 or q1 or q0);
  end process cct_mod6;
end modulo6;
```

35) Write a VHDL code for the following combinational circuits.

a. $F = AB + \overline{A}\,\overline{B}$

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit is
  Port (A, B, C : in std_logic;
        Y : out std_logic );
end circuit;

architecture data of circuit is
begin
  Y <= (A and B) or ((not A) and (not B));
end data;
```

b. $F(X,Y,Z) = \Sigma(1,3,4,6)$

c. $F(A,B,C) = \Sigma(1,3,6,7)$

d $F(A,B,C) = \Pi(0,5,7)$

↳ Truth table:

	Input Variables			SOP (Σ)	POS (Π)
	A	B	C		
0	0	0	0	$\bar{A}\,\bar{B}\,\bar{C}$	$A+B+C$
1	0	0	1	$\bar{A}\,\bar{B}\,C$	$A+B+\bar{C}$
2	0	1	0	$\bar{A}\,B\,\bar{C}$	$A+\bar{B}+C$
3	0	1	1	$\bar{A}\,B\,C$	$A+\bar{B}+\bar{C}$
4	1	0	0	$A\,\bar{B}\,\bar{C}$	$\bar{A}+\bar{B}+\bar{C}$
5	1	0	1	$A\,\bar{B}\,C$	$\bar{A}+B+\bar{C}$
6	1	1	0	$A\,B\,\bar{C}$	$\bar{A}+\bar{B}+C$
7	1	1	1	$A\,B\,C$	$\bar{A}+\bar{B}+\bar{C}$

b) $F(X,Y,Z) = \Sigma(1,3,4,6)$

↳ It can be written as: $F(A,B,C) = \Sigma(1,3,4,6)$

Now, From truth table,

$F = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}BC + A\,\bar{B}\,\bar{C} + AB\,\bar{C}$

VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit is:
        Port (A, B, C : in std_logic;
              F : out std_logic);
end circuit;
```

```
architecture data of circuit is
begin
    F <= ((not A) and (not B) and C) or
         ((not A) and B and c) or
         (A and (not B) and (not C)) or
         (A and B and (not C));
end data;
```

b. $F(A,B,C) = \Sigma(1,3,6,7)$

↳ From truth table:

$F = \bar{A}\bar{B}C + \bar{A}BC + ABC$

VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit is
    Port (A,B,C: in std_logic;
          F: out std_logic);
end circuit;

architecture data of circuit is
begin
    F <= ((not A) and (not B) and C) or
         ((not A) and B and C) or
         (A and B and (not C)) or
         (A and B and C);
end data;
```

$F(A,B,C) = \pi(0,5,7)$

From the above table,

$F = (A+B+C) \cdot (\bar{A}+B+C) \cdot (\bar{A}+\bar{B}+\bar{C})$

VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit is
  Port ( A,B,C : in std_logic;
         F: out std_logic);
end circuit;

architecture data of circuit is
 begin
    F <= (A or B or C) and
         ((not A) or B or (not C)) and
         ((not A) or (not B) or (not C));
end data;
```