COA LAB - Coa lab practicals for aktu

Computer Organization & Architecture Lab (Dr. A.P.J. Abdul Kalam Technical University)

Project/Topic

PAGE

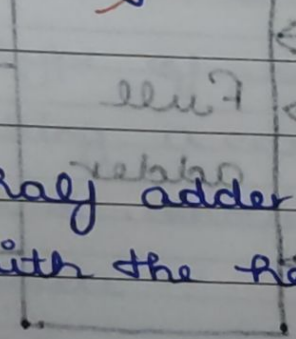
DATE

Experiment 1

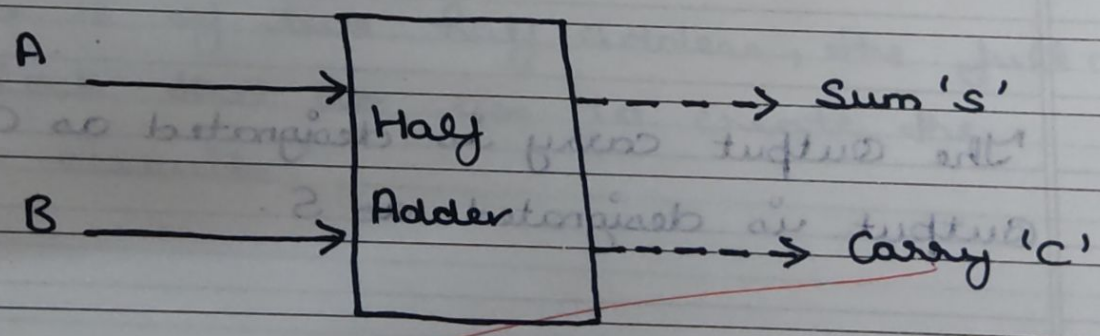
Aim: Implementation of half adder, full adder using basic logic gates.

Theory:

An adder is a digital circuit that performs addition of numbers. The half adder adds two binary digits called as augend and addend and produces two outputs as sum and carry; XOR is applied to both inputs to produce sum and AND gate is applied to both inputs to produce carry. The full adder adds 3 one bit numbers, where two can be referred to as operands and one can be referred to as bit carried in. And produces 2-bit output and these can be referred to as output carry and sum.



By using half adder, you can design simple addition with the help of logic gates.



Half adder:

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 10

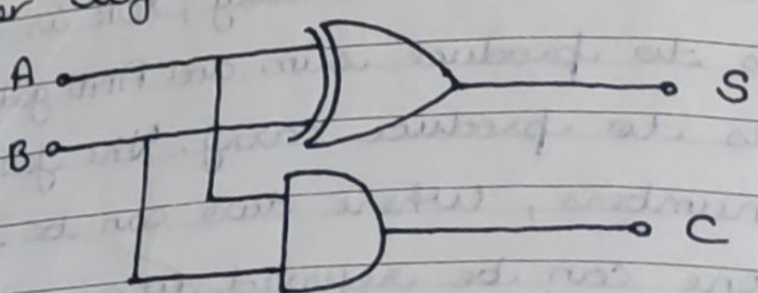
Signature.....

Project Topic

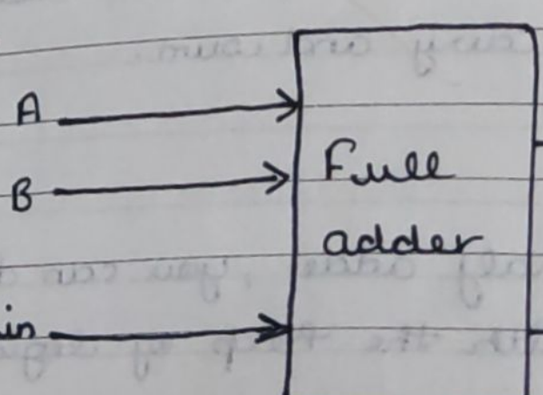
Half adder truth table

Input		Output	
		Sum	carry
A	B		
0	0	0	0
0	1	1	0
0	1	1	0
1	0	0	1
1	1		

Half adder logic circuit



Full adder



The output carry is designated as C-out and the normal output is designated as S.

Full adder truth table

We can see that the output S is an XOR between the input A and the half adder, Sum output with B and C-IN inputs, we take C-out will only be true if

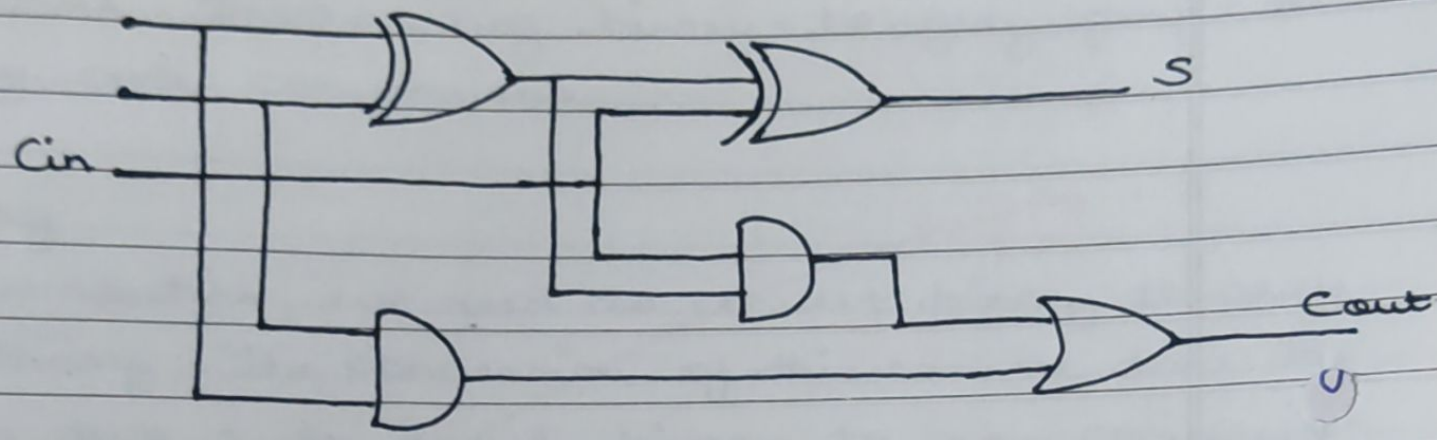
Signature.....

Project/Topic

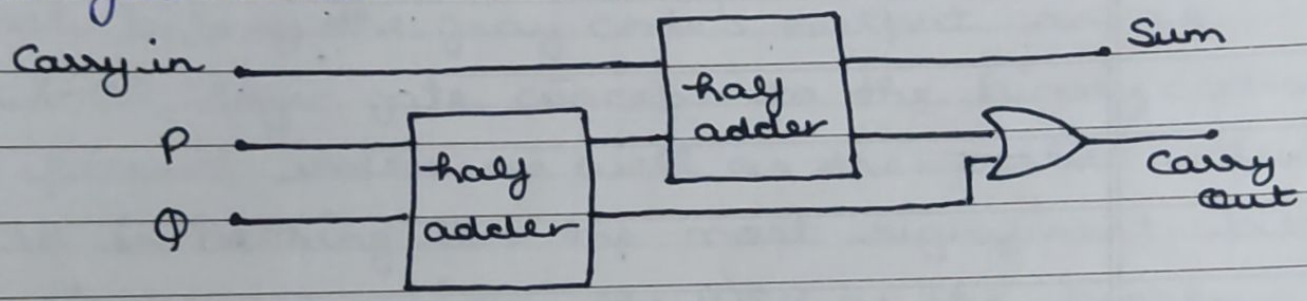
PAGE

DATE

any of the two inputs out of the three are High. So, we can implement a full adder circuit with the help of two half adder circuits.



Given below is the schematic representation of a one bit full adder.



With this type of symbol, we can add two bits together, taking a carry from the next lower magnitude and sending a carry to next higher order magnitude.

Full adder is of two half adders, the full adder is actual block that we use to create the arithmetic circuits.

Signature.....

Experiment no : 2

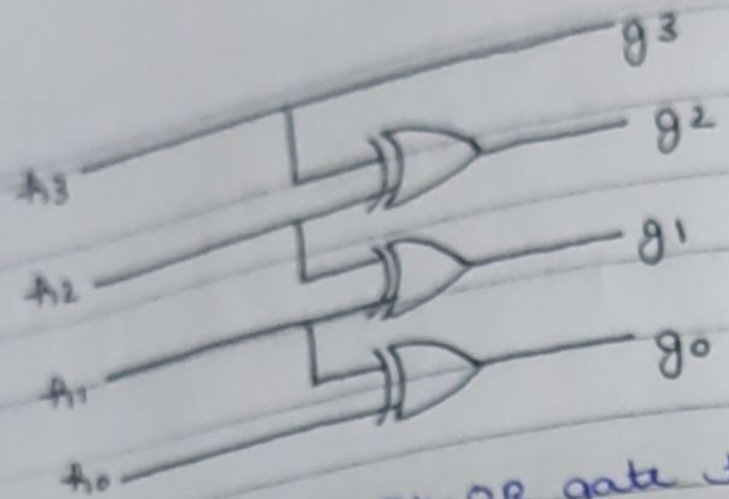
Objective : Implementing binary - to - gray, gray - to - binary code conversions

Theory :

In computers, we need to convert binary to gray to binary. The conversion of this can be done by using two rules namely binary to gray conversion In the first conversion, the MSB of the gray code is constantly equivalent to the MSB of the binary code Additional bits of the gray code's output can get using EX-OR logic gate concept to the binary codes at that present index as well as the earlier index Here MSB is nothing but the most significant bit In the first conversion, the MSB of the binary code is constantly equivalent to the MSB of the particular binary code. Additional bits of the binary code's output can get using EX-OR logic gate concept concept by verifying gray codes at that present code.

Binary to gray code converter

The conversion of binary to gray code can be done by using a logic circuit. The gray code is a none weighted code because there is no particular weight is assigned for the position of the bit. A n-bit code can be attained by reproducing a n-1 bit code on an axis subsequent to the rows 2^{n-1}, as well as placing the most significant bit of 0 over the axis with the most significant bit of 1 beneath the axis.



The method uses an EX-OR gate to perform among the binary bits.

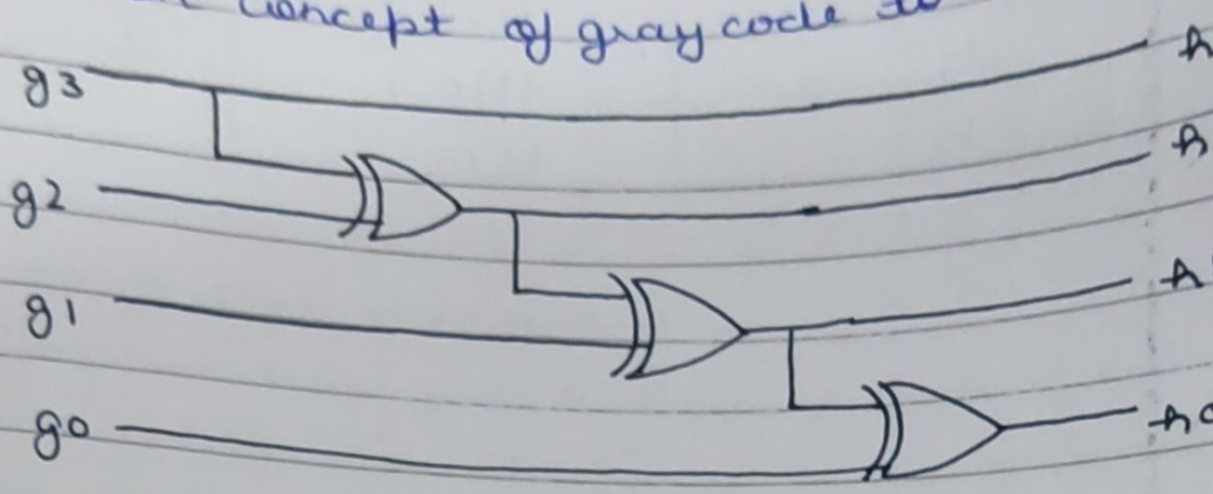
Binary to gray code conuerter table

Decimal number	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001

Gray to binary code conuerter

This gray to binary conuersion method also uses the working concept of EX-OR logic gate among the bits of gray as well as binary bits. The following example

with step by step procedure may help to know the conversion concept of gray code to binary code.



To change gray to binary code, take down the MSB digit of the gray code number, as the primary or the MSB of the gray code is similar to the binary digit. To get the next straight binary bit, it uses the XOR operation among the primary bit or MSB bit of binary to the next the next bit of the gray code.
Similarly, to get the third straight binary bit, it uses XOR operation among the second bit or MSB bit of binary to the third MSB bit of the gray code and so on.

Gray to binary code converter table

Decimal number	Gray code	Binary code
0	0001	0000
1	0001	0001
2	0010	0011
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000

9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100

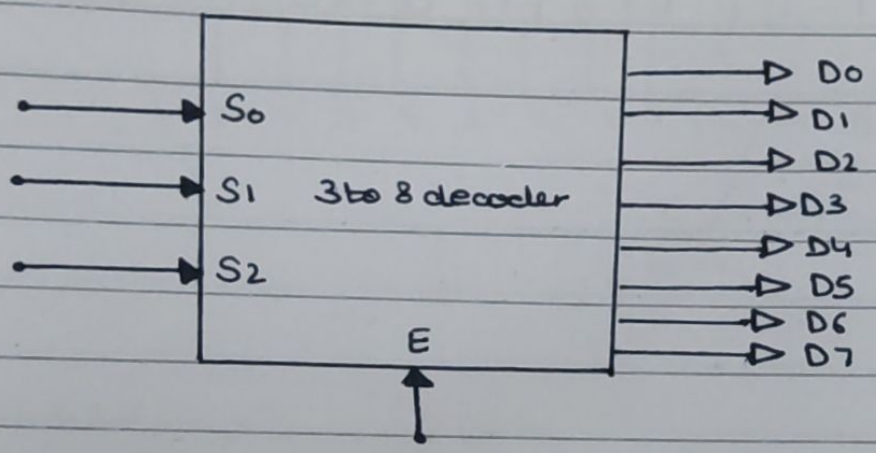
Experiment no. 3

Objective: Implement 3-8 line DECODER.

Theory:

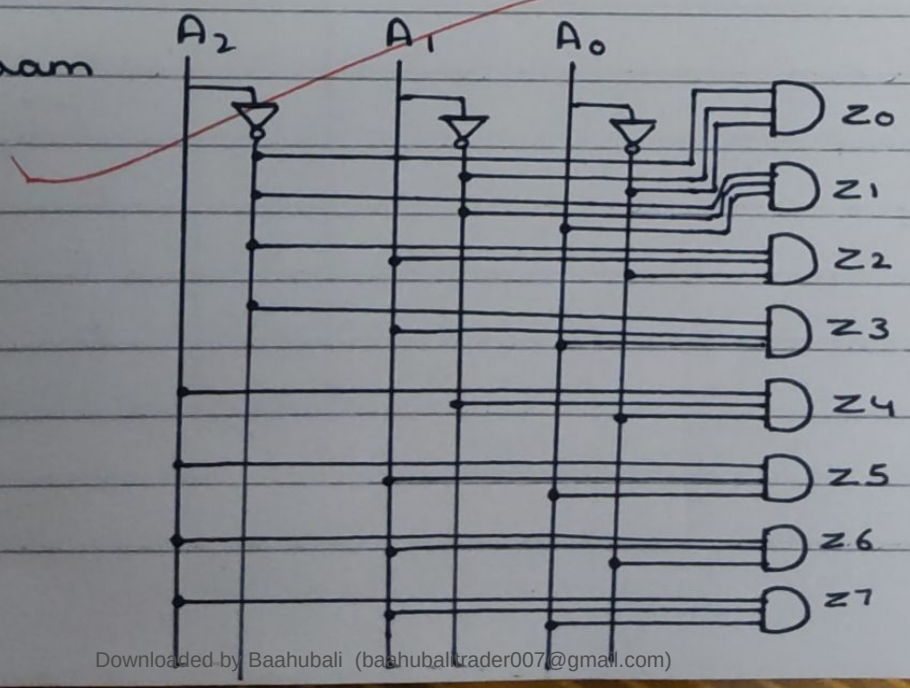
3 line to 8 line decoder

This decoder circuit gives 8 logic outputs for 3 input and has a enable pin. The circuit is designed with AND and NAND logic gates. It takes 3 binary inputs and activates one of the the eight outputs. 3 to 8 line decoder circuit is also called as binary to an octal decoder.



The decoder circuit works only when the enable pin (E) is high. S0, S1, S2 are three different inputs and D0, D1, D2, D3, D4, D5, D6, D7 are the eight outputs.

Circuit diagram



The below table gives the truth table of 3 to 8 decoder.

S0	S1	S2	E	D0	D1	D2	D3	D4	D5	D6	D7
X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Experiment no 4

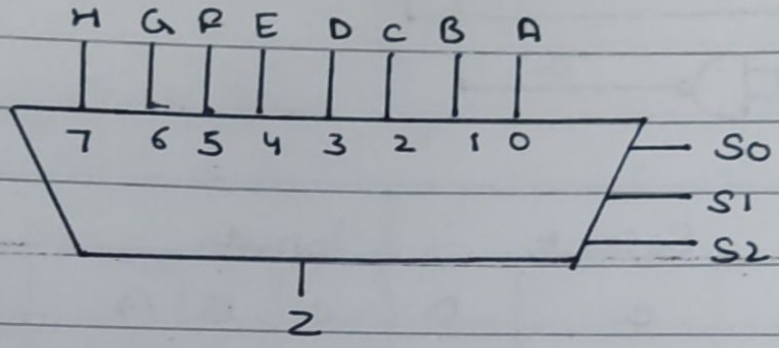
Objective : Implementing 4*1 and 8*1 multiplexer

Theory :

A multiplexer is a device that performs multiplexing i.e., it selects one of many analog or digital input signals and forwards the selected input into a single line. A multiplexer of 2^n inputs has n select lines, which are input line to be sent to the output

A boolean equation for 8x1 multiplexer is

$Z = A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC$



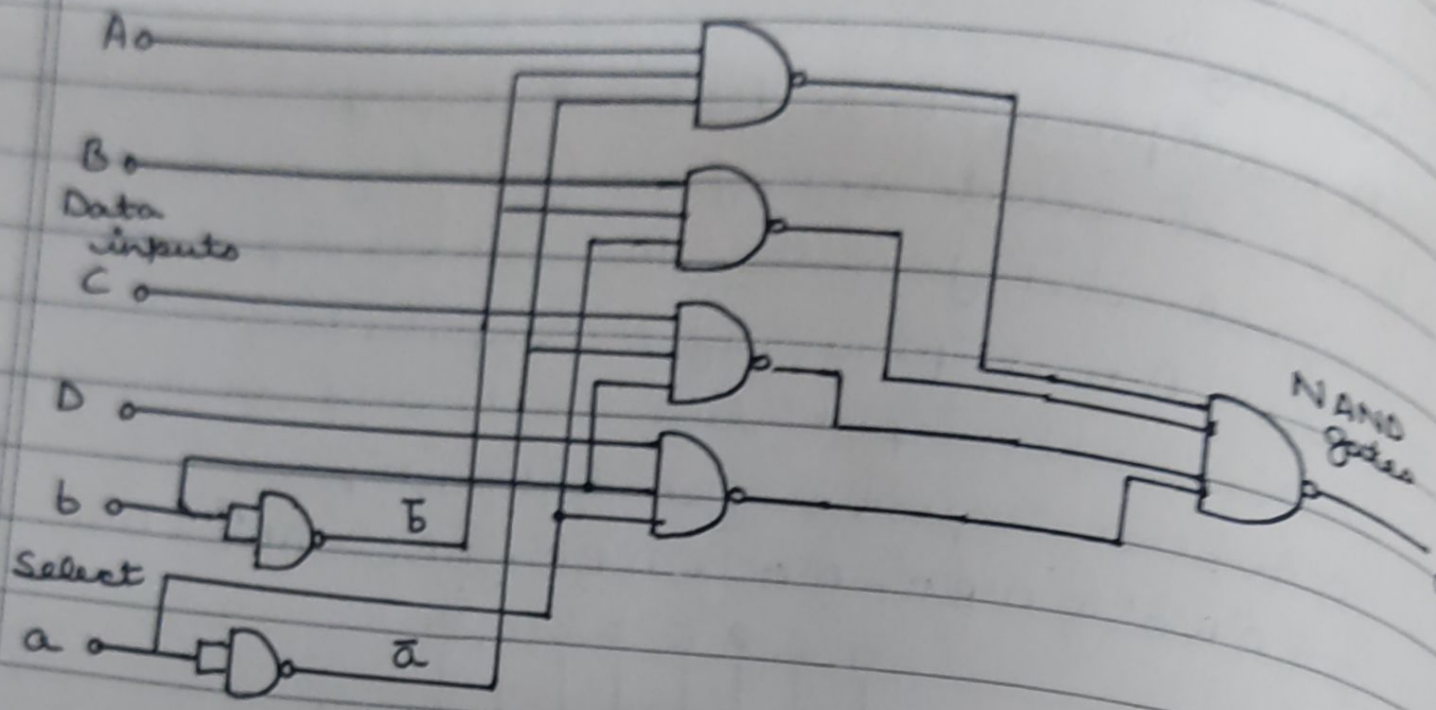
Truth table

S_0	S_1	S_2	Z
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	D
1	0	0	E
1	0	1	F
1	1	0	G
1	1	1	H

A boolean equation for 4×1 multiplexer is

$\phi = abA + abB + abC + abD$

Truth table



Select			Inputs			
b	a	D	C	B	A	ϕ
0	0	x	x	x	1	1
0	1	x	x	1	x	1
1	0	x	1	x	x	1
1	1	1	x	x	x	1

Experiment no 5

Objective: Verify the excitation tables of various FLIP FLOPS.

To realize and implement

1. Set-Reset (SR) latch using NOR gates (active High circuit)
2. SR, JK, D and T flip-flops using IC's and breadboard.

Components required:

- Mini digital training and digital Electronic sets.
- IC 7404, IC 7408, IC 7411, IC 7474, IC 7476.

Theory:

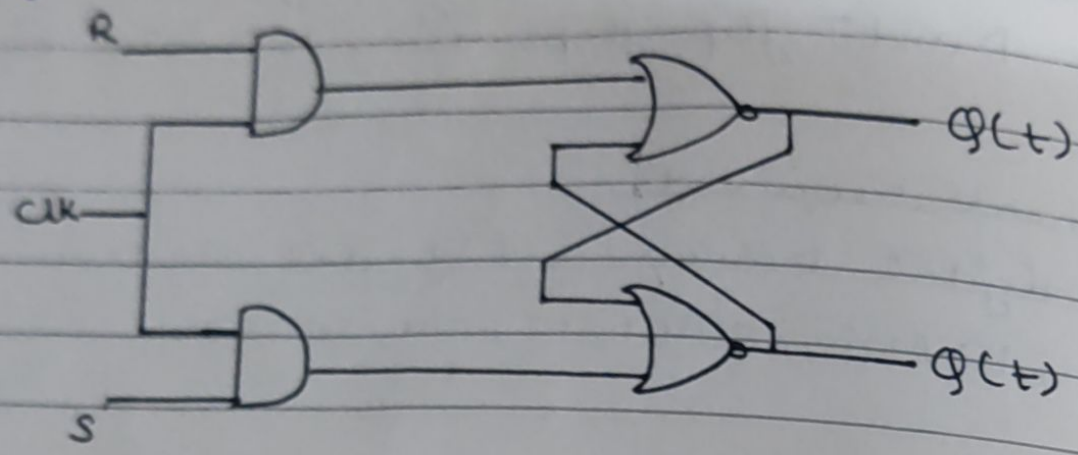
Logic circuits for digital systems are either combinational or sequential. The output of combinational circuits depends not only on the current inputs. In contrast, sequential circuits depends not only on the current value of the input but also upon the internal state of the circuit. Basic building blocks of a sequential circuit are the flip-flops (FFs). The FFs change their output state depending upon inputs at certain interval of time synchronized with some clock pulse applied to it.

We shall discuss most widely used latches that are listed below:

- SR flip flop
- D flip flop
- JK flip flop
- T flip flop

SR flip-flop

SR flip flop operates with only positive clock transitions or negative clock transitions. Whereas latch operates with enable signal. The circuit diagram of SR flip flop is shown in the following figure.



The circuit has two inputs S & R and 2 outputs Q(t) & Q(t)'. The operation of SR flip flop is similar to SR latch. But, this flip flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the state table of SR flip flop

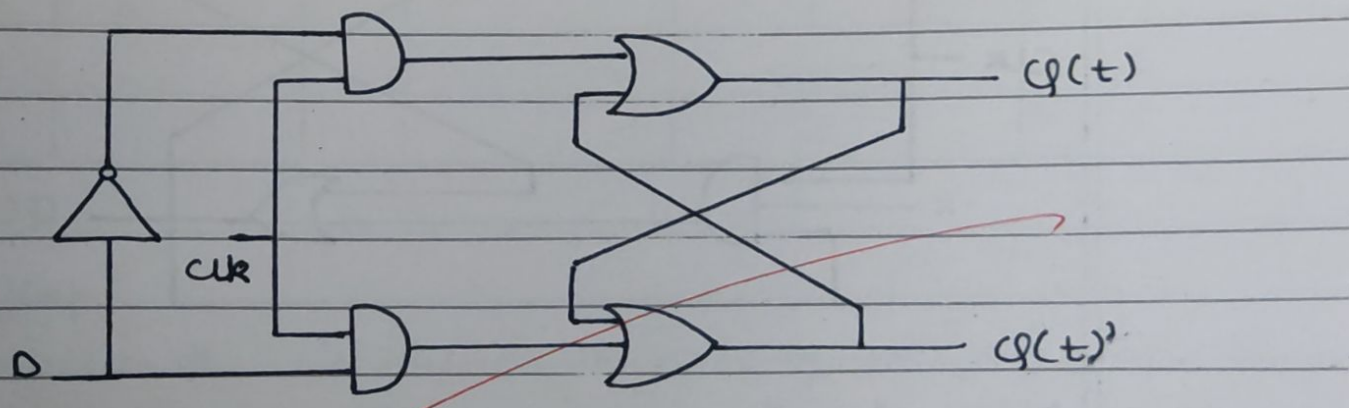
S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	-

Here Q(t) & Q(t+1) are present state & next state. The following table shows the characteristic table of SR flip flop.

Present inputs		Present state	Next state
S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

D flip-flop

D flip flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip flop is insensitive to the changes in the input, D except for active transition of the clock signal. The circuit diagram of D flip flop is shown in the following figure



The circuit has single input D and two outputs Q(t) and Q(t)'. The operation of D flip flop is similar to D latch. But, this flip flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the state table of D flip flop

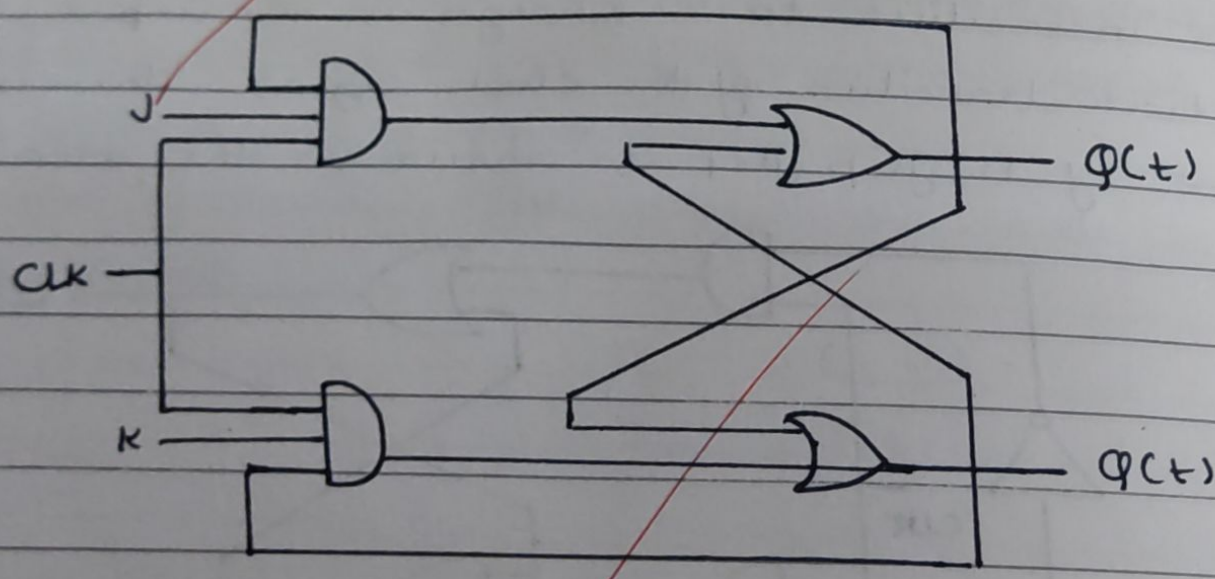
D	Q(t+1)
0	0
0	1

Therefore, D flip flop always hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation is.

$$Q(t+1) = D$$

JK flip flop

JK flip flop is the modified version of SR flip flop. It operates only positive clock transitions or negative clock transitions. The circuit diagram of JK flip flop is shown in the following figure.



The circuit has two inputs J & K and two outputs Q(t) & Q(t)'. The operation of JK flip flop is similar to SR flip flop.

The following table shows the state table of JK flip flop.

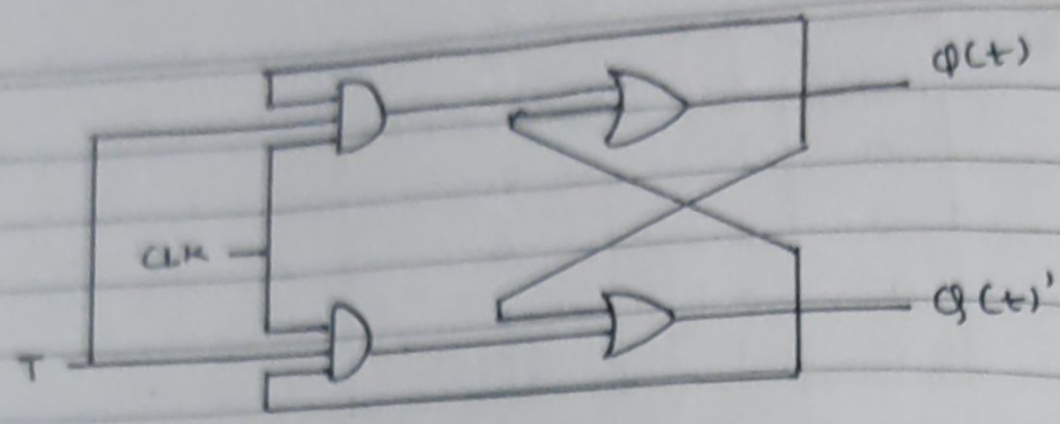
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

Here, Q(t) & Q(t+1) are present state and next state respectively.
The following table shows the characteristic table of JK flip flop

Present inputs		Present state	Next state
J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

T flip flop

T flip flop is the simplified version of JK flip flop. It is obtained by connecting the same input 'T' to both inputs of JK flip flop. It operates with only positive clock transitions or negative clock transitions. The circuit diagram of T flip flop is shown in the following figure.



The circuit has single input T and two outputs Q(t) & Q(t)' The operation of T flip flop is same as that of JK flip flop. Here we considered the inputs of JK flip flop as J = T and K = T in order to utilize the modified JK flip flop for 2 combinations of inputs.

The following table shows the state table of T flip flop

D	Q(t+1)
0	Q(t)
1	Q(t)'

The following table shows the characteristic table of T flip flops.

Inputs	Present state	Next state
T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Output

We implemented various flip flops by providing the cross coupling between NOR gates. Similarly, you can implement these flip flops by using NAND gates.

Experiment 6

Objective : Design of an 8 bit input/output system with four 8 bit internal registers

Apparatus required → Experimental trainer board
→ Bread board
→ ICs 7495/74594/74161
→ Connecting leads

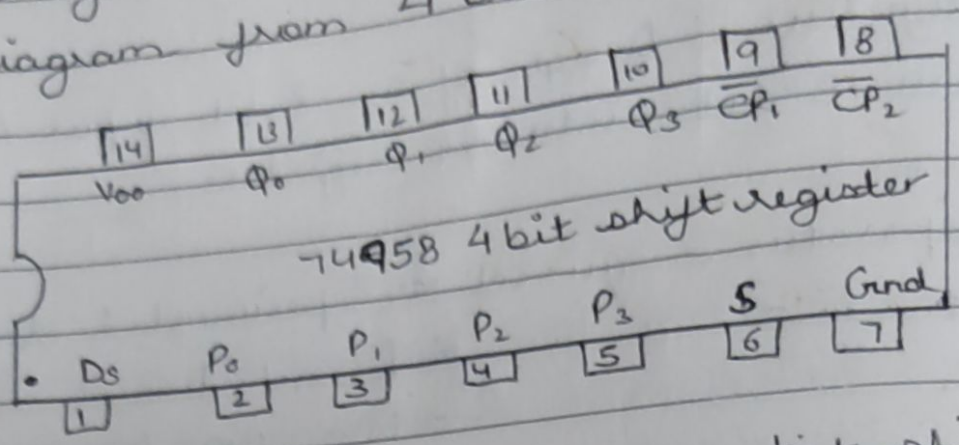
Brief theory : Serial-in-parallel-out (SIPO)
This configuration allows conversion from serial to parallel format. Data input is serial, as described in the SISO section above. Once the data has been clocked in, it may be read off at each output simultaneously or it can be shifted out.
In this configuration, each flip flop is edge triggered. All flip flops operate the given clock frequency. Each input bit makes its way down to the nth output after N clock cycles, leading to parallel output.
In general, the practical application of the serial-in parallel-out shift register is to count data from serial format on a single wire to parallel format on multiple wires.

State table for 4 bit SIPO

Clock	Serial I/P	Q_0	Q_1	Q_2	Q_3
1	0	0	x	x	x
2	1	1	0	x	x
3	1	1	1	0	x
4	1	1	1	1	0

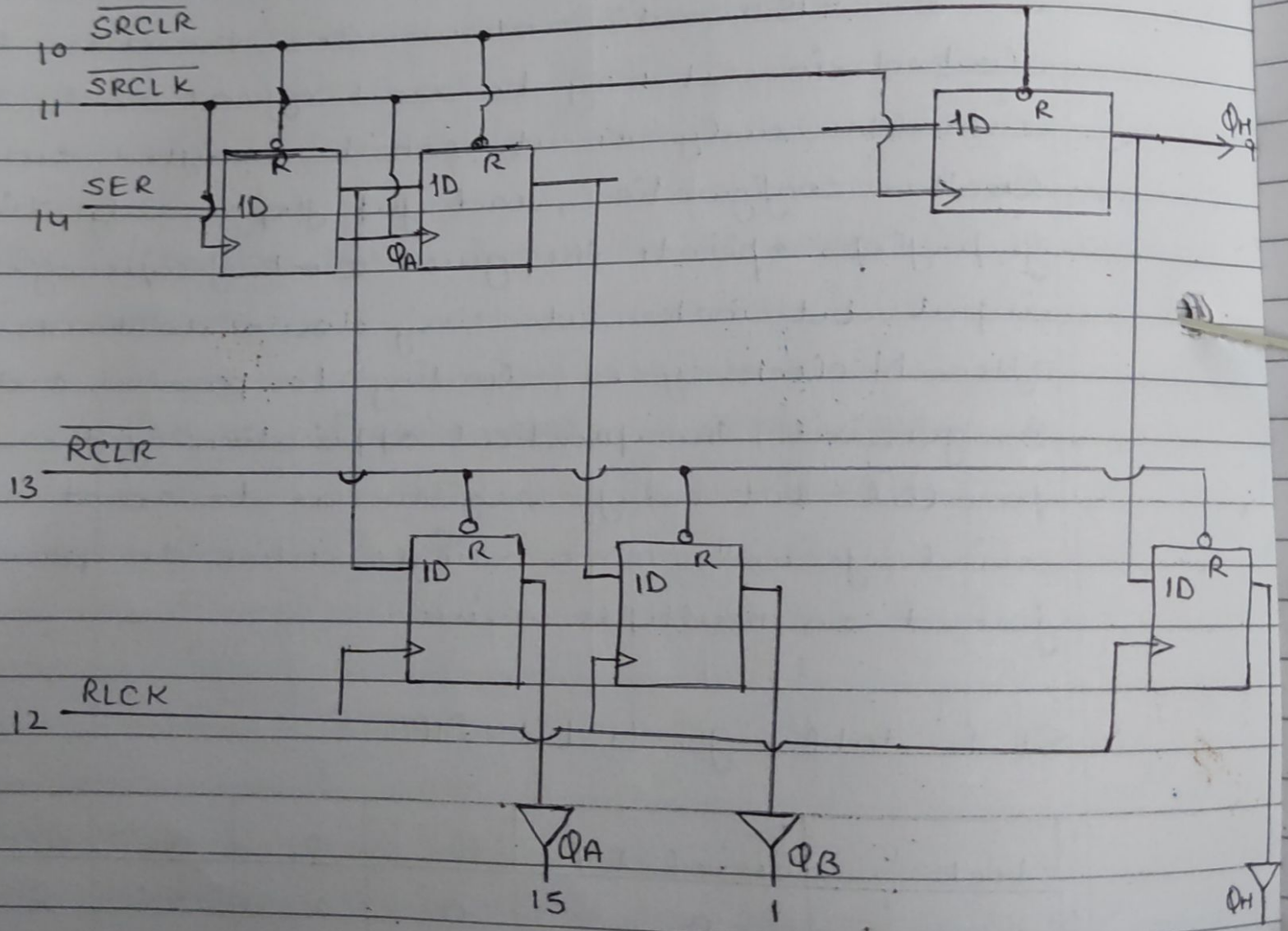
Circuit diagram

Pining diagram from 4 bit SIPO:



S is the mode input. If it is low, bit shifting occurs at Q

Circuit diagram of 8 bit SIPO shift register



Procedure

- Connections are made as per circuit diagram
- Apply the data at serial input
- Apply one clock pulse at clock 2 (right shift), observe the data on Q_0 will shift at Q_1 & the new data applied will appear at Q_1.
- Apply one clock pulse at clock 1 (right shift), observe this data at Q_0.
- Apply the next data at serial input.
- Repeat the step 2 and 3 till the 4 bits data are entered one by one into shift register.

Result

Shift register using IC7495 in SIPO mode is verified.

Precautions

- All the ICs should be checked before use of apparatus
- All the LEDs should be checked.
- All connections shoulde be tight.
- Always connect ground first and then Vcc.

Experiment 7

Objective: Design the 8 bit arithmetic logic unit

Theory

The designing 8 bit ALU using verilog programming language. It includes writing compiling and simulating verilog code in modelsim on a windows program. Modelsim is an easy to use, versatile VHDL/ system verilog / verilog / system / C simulator by mentor graphics. It supports behavioural, register transfer level & gate level modelling

First, install Modelsim on a windows PC.

- Start Modelsim from desktop; you will see Modelsim 10.4 dialouge window.
- Create a project by clicking jumpstart on the welcome screen.
- A create project windows pops up. Select a suitable name for your project.
- An add items to the project window pops up. On this window, select create a new file option.
- Select an appropriate file name for the file you want to add; choose a verilog as add file as type and top level as folder.
- On the workspace section of the main window, double click on the file you have just created.
- Type in your verilog code for an 8 bit ALU in the new window. Save your code from file menu.
- Now add relevant files as per the architecture, which includes arithmetic, logic, shift and MUX units.

Compiling / debugging project files

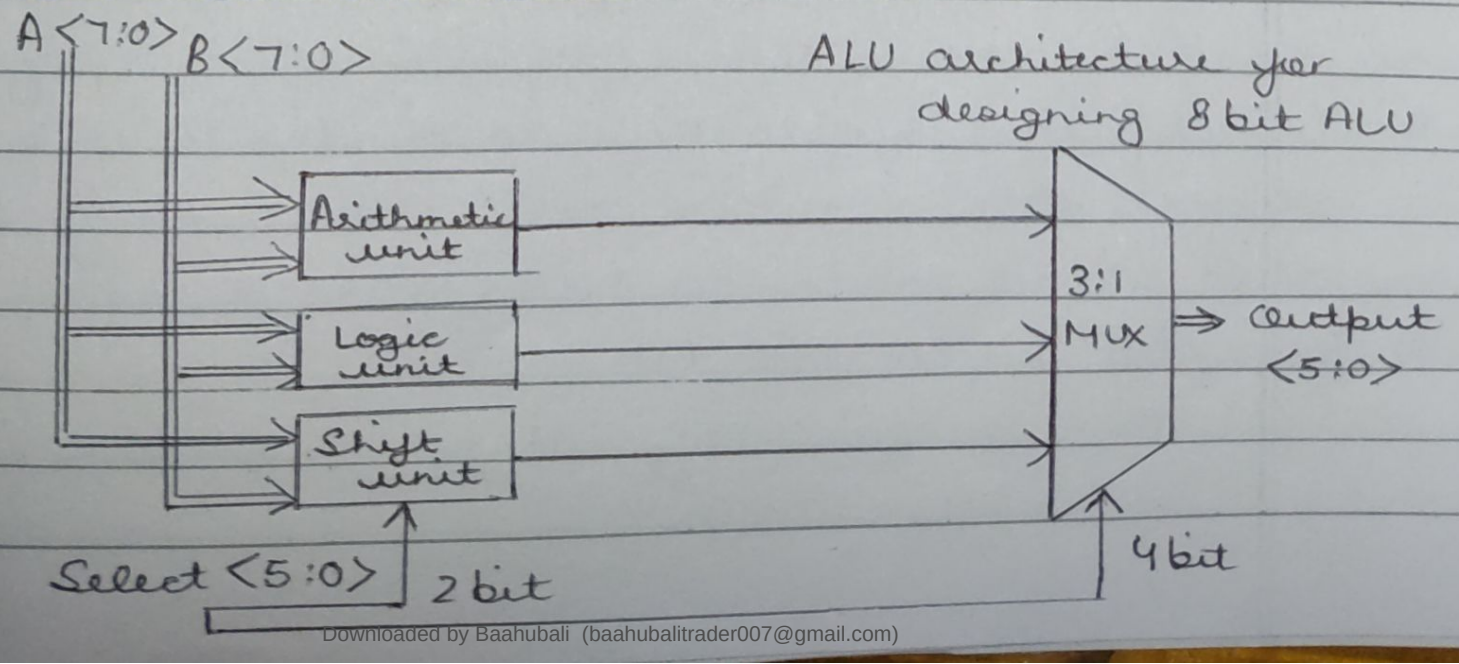
- Select compile → compile all options
- The compilation result is shown on the main window. A green tick is shown against each file name, which means there are no errors in the project.

Simulating the ALU design

- Click on the library menu from the main window and then click on plus (+) sign next to work library. You should see the name TOP_ALU code that we have just compiled.

Double click on the ALU to load the file. This should open a third tab sim in the main window
Go to add → to wave → all the items in region options
Select the signals that you want to monitor for simulation purposes.
Provide values manually to monitor the simulation of the 8 bit ALU design.

Right click on the selected signals, we are now ready to stimulate our design by clicking run in the simulation window.



Result : The ALU design is verified from this output waveform.

Experiment 8

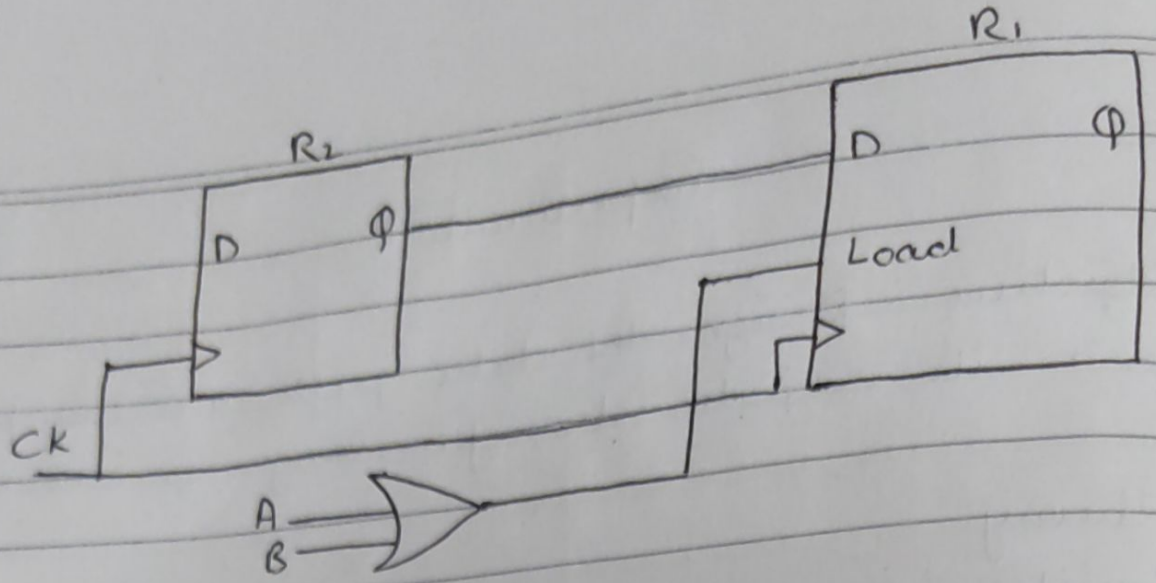
Objective : Design the data path of a computer from its register transfer language description

Theory

Register transfer language, RTL (sometimes called register transfer notation) is a powerful high level method of describing the architecture of a circuit. VHDL code and schematics are often created from RTL. RTL describes the transfer of data from register to register known as micro instructions or micro operation Transfers may be conditional. Each micro instructions completes in one clock cycle. A typical RTL statement will look like the following:

$$AVB \rightarrow R_1 \leftarrow R_2$$

This is read as "If signal A or signal B is true then register R2 is transferred to register R1". The first part AVB is a logical expression that must be true for the transfer to take place. The → symbol separates the logical expression from the micro instruction It is the if then part of the statement. If there isn't a logical expression → isn't in the statement and the micro instruction will always take place. To the right of → is the micro instruction. It describes a transfer of data and operations on the data from register to register The above RTL statement is equivalent to the following schematic.



For RTL we will use the following symbols

<- -	Register transfer
[ ]	Word index
< >	Bit index
n ... m	Index range
- ->	if - then
: =	Definition
#	Concatenation
:	Parallel separator
;	Sequential separator
@	Replication
{ }	Operation modifier
( )	Operation or value grouping
= ! = < < = > > =	Comparison operators
+ - * /	Arithmetic operators
^ V ' xor	logical operators

Experiment 7

Objective : Design the control unit of a computer using either hardwiring or microprogramming based on its register transfer language description.

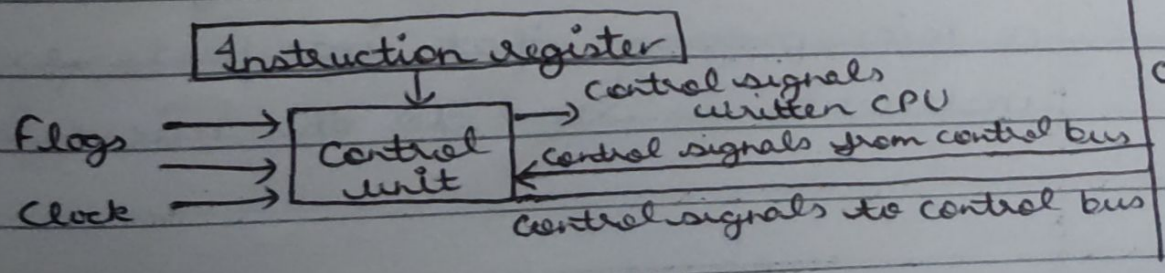
Theory :

Control unit is the part of the computer's central processing unit (CPU) directs the operation of the processor. It was included as part of the Von Neumann architecture by John von Neumann. It is the responsibility of the control unit to tell computer's memory, arithmetic / logic unit and input and output devices how to respond to the instructions that have been sent to the professor. It fetches internal instructions of the program from main memory to the processor instruction register and based on this register contents.

A control unit works by recieving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform examples of devices that require a CU are :

Central processing units (CPU)

Graphics processing units (GrPUs)



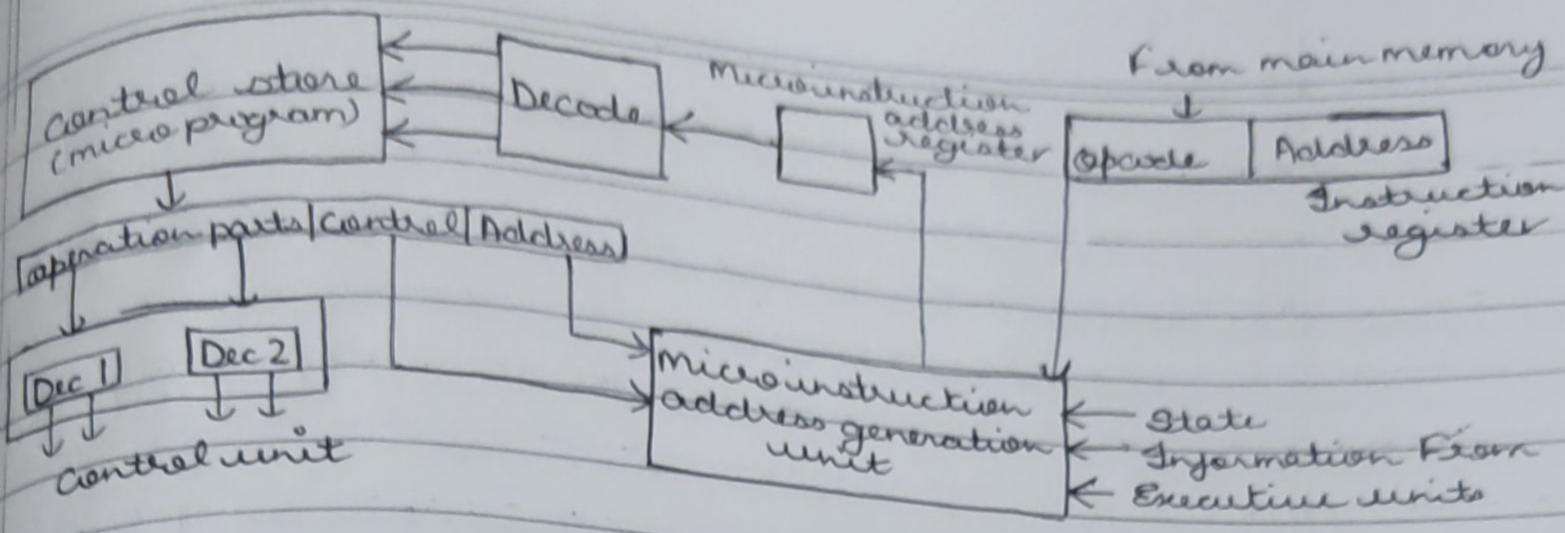
Block diagram of the control unit

Types of control unit
There are two types of control units: Hardwired control unit and microprogram control unit.

1) Hardwired Control unit
In the hardwired control unit, the control signals th are important for instruction execution control are generated by specially designed hardware logic circuit in which we can not modify the signal generation method without physical change of the circuit structure.

2) Micro programmable control unit
The fundamentals differences between these unit struct and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution. However, the operation code each instruction is not directly decoded to enable immediate control signal generation but it comprise the initial address of a microprogram contained in the control store.

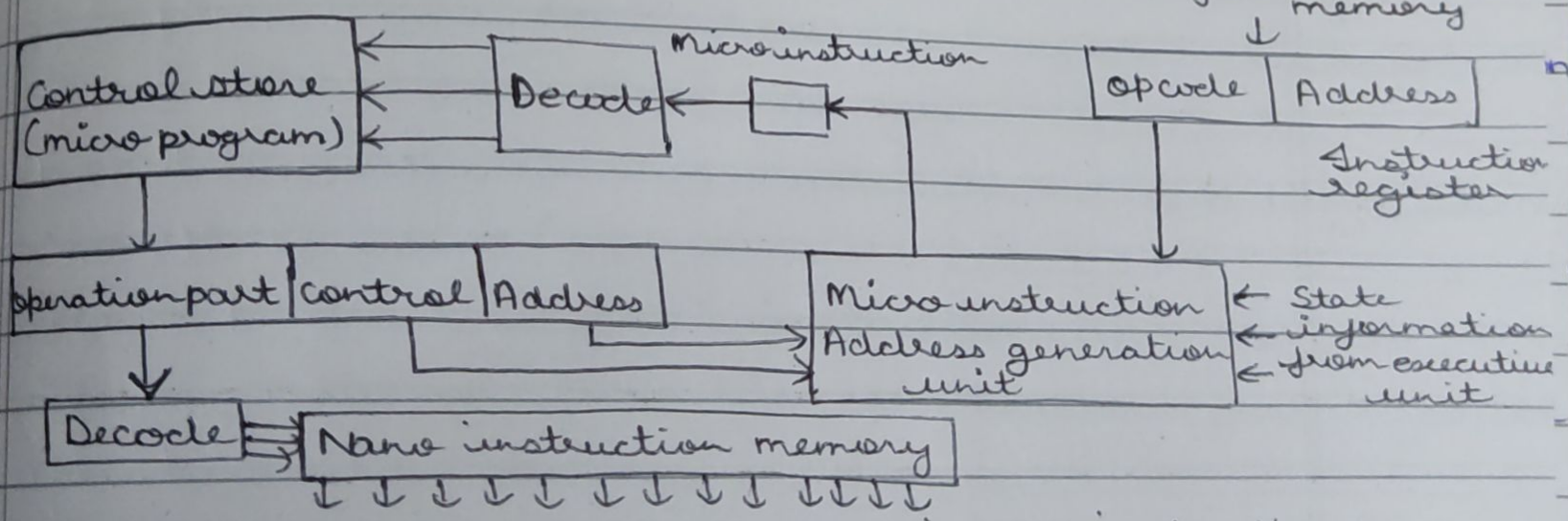
- With a single-level control store
In this, the instruction opcode from the instruction register is sent to the control store address registe Based on this address, the first micro instruction o a microprogram that interprets execution of this instruction is read to the micro instruction register.



Micro programmed control unit with a single level control store

. With a two-level control store

In this, in a control unit with a two-level control store, besides the control memory for micro instruction, a nano-instruction memory is included. In such control unit, micro instruction do not contain encoded control signals.



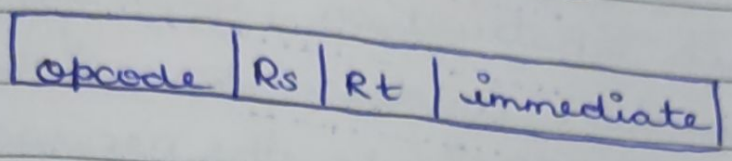
Microprogrammed control units a two level control store

Result: The control unit design in verified from this output signals.

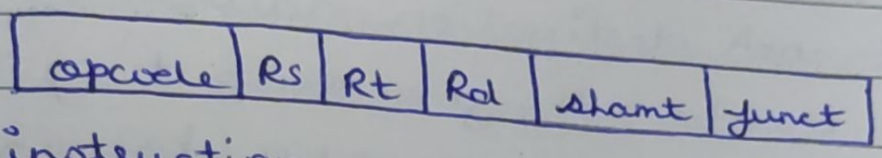
Experiment 10

Objective: Implement a simple instruction set computer with a control unit and a data path.

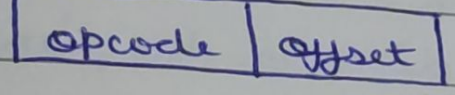
Theory: I type instruction



R type instruction



J type instruction

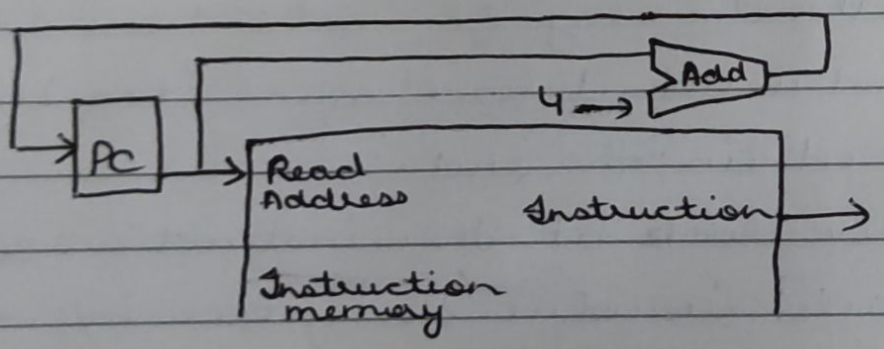


We will examine the MIPS implementation for a simple subset that shows most aspects of implementation
The instructions considered are:
The memory reference instructions load word (lw) and store word (Sw).
The arithmetic logical instructions add, sub, and, or
The instruction branch equal (beq) and jump (j) to be considered in the end.
When we look at the instruction cycle of any processor it should involve the following operations

- Fetch instruction from memory
- Decode the instruction
- Execute the instruction
- Write the result

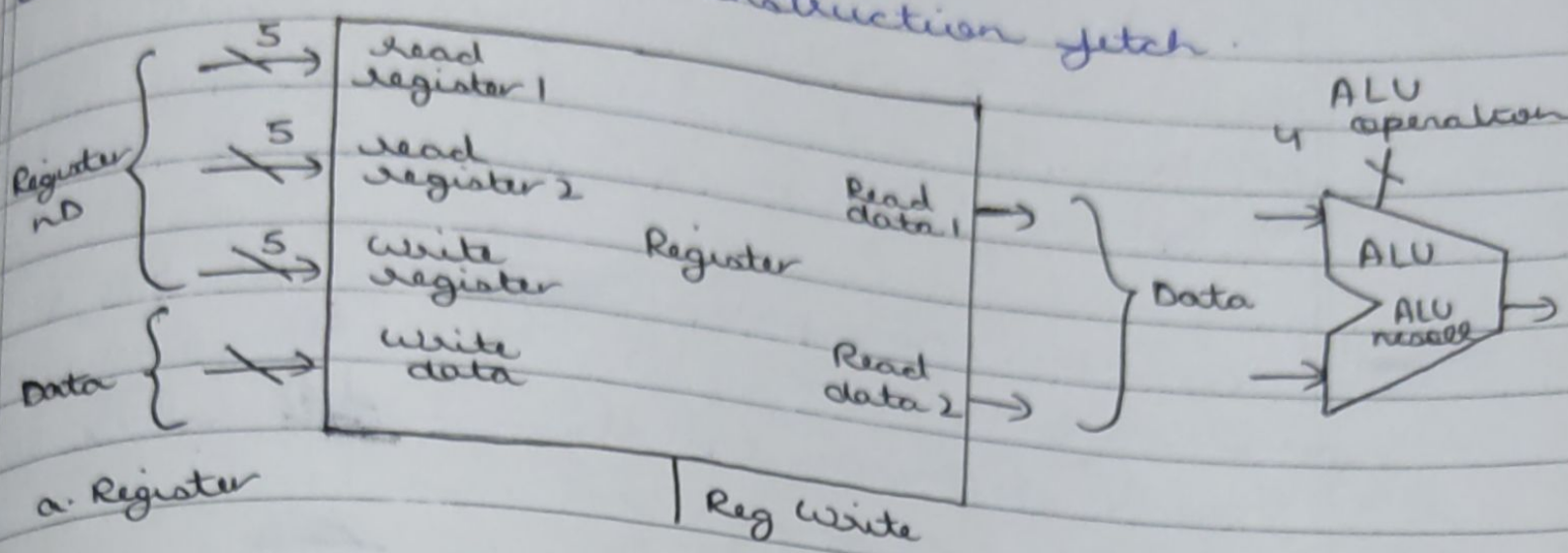
Send the program counter (PC) to the program memory that contains the code and fetch the instruction read

one or two registers using the register specifier fiel
the instruction.
For a load instruction, a memory read has to be
peyormed yor a store instruction, a memory write h
to be peyormed. A load instruction also has to wr
the data yetched yrom memory to a register. Lastly
yor a branch instruction, we may need to change the
next instruction address based on the comparison
We also need to include the necessary control signal
The third multiplexer, which determines whether PC
or the branch destination address is written into the
PC set based on the zero output of ALU, which is used
to peryorm comparison of a branch on equal
instruction.
Now we shall discuss the implementation of the dat
path. The data path comprises of the element that
process data and address in the CPU register ALU's
MUX's memories etc.
The portion of CPU that carries out the instruction
yor operation.



As mentioned earlier, The PC is used to the instruction
memory do yetch the instruction. At the same time, the PC
value is also yed to the adder unit and added with 4
so that PC+4, which is the address of the next
instruction in MIPS written into the PC, thus making it

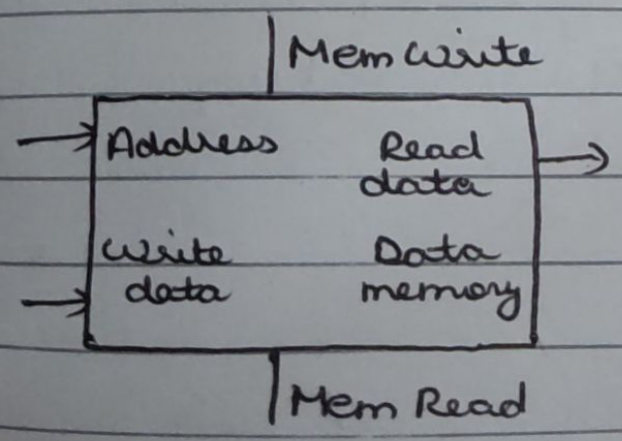
ready for the next instruction fetch.



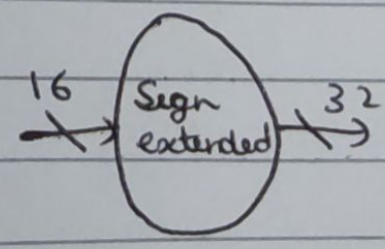
a. Register

b. ALU

The R-format instruction have three register operands and we will need to read two data words from the register file for each instruction. To write a data word, we will need 2 inputs - one to specify the register number to be written one to supply the data to be written into the register. The 5 bit register specifiers indicate one of the 32 registers to be used.
The same arithmetic logical operations with an immediate operand and a register operand, uses the I-types of operand. These two will have to be fed to the ALU. Before that, the 16 bit immediate operand is sign extended to form a 32 bit operand. The sign extension is done by the sign extension unit.



a. Data memory unit



b. Sign extension unit

Now, that we have examined the datapath components needed for the individual instruction classes, we can combine them into a single datapath and add the control to complete the implementation.

The simplest datapath might attempt to execute all instructions in one clock cycle. This means that no datapath resources can be used more than once per instruction, so any element needed more than once per be duplicated. We therefore need a memory for instruction seperate from one for data. While adding multiplexers, we should not that though the operations of arithmetic / logical (R type) instructions and the memory related instructions datapath are quite similar there are certain key differences.
The R type instructions use two register operands coming from the register file. The memory instructions also use the ALU to do the address calculation, but the second input is the sign extended 16 bit offset field from the instruction.
The value stored into a destination register comes from the ALU for an R type instruction, whereas the data comes from memory for a load.

Result : Implementation of simple instruction set is verified from this output working.