The Gate Level Implementation of Computer Architecture

Name of Student- Keshav Maheshwari (8102163)
- Bhuvan Gupta (8102323)

Name of Supervisor- Mr Saurabh Chaturvedi

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY UNIVERSITY, NOIDA

TABLE OF CONTENTS

CERTIFICATE

This is to certify that the work titled “**The Gate Level Implementation of Computer Architecture**” submitted by “**Keshav Maheshwari and Bhuvan Gupta**” in minor project of degree of Bachelor of Technology of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor…………………………………….

Name of Supervisor: **Mr. Saurabh Chaturvedi**

Designation: Lecturer

Department of Electronics and Communications Engineering

Jaypee Institute of Information Technology, Noida

ACKNOWLEDGEMENT

First and foremost, we would like to thank God for blessing us with the strength, intelligence, and patience to complete this project. We would like to express my sincere thanks to **Mr. Saurabh Chaturvedi** and **Mr. Sandeep Joshi** who have been helping us by giving their valuable suggestions and guiding us right way throughout the project.

We am extremely thankful to the Dean, **Prof. S. L. Maskara** and the HOD of Electronics and Communication Engineering Department, **Prof. R. C. Jain** for providing the infrastructural facilities to work in, without which this work would have not been possible.

Signature of Students…………………………………………

Name of Student: Keshav Maheshwari , Bhuvan Gupta

Date: …………………..

Abstract

We believe that learning in Electronics Engineering should reflect the current state of field as well as introduce the principles that are shaping computing.
As in engineering course we start with microcontroller and microprocessor applications but we don’t get chance how do they work, how do they instruct and how do they perform , thus the audience of this project , will get a great help in understanding how computer works, how it is designed and why it perform as according to given instruction.

The first chapter gives an introduction to the project stating the meaning of processor and its evolution. Second chapter discusses about the Logisim software which we have used to implement our project work. Third and fourth chapter is about ALU and Memory giving detail about their specification and diagram. Chapter five shows projected Opcodes we have used . Chapter six concludes the body of project that is Datapath and its working. Chapter seven is giving details about Control Unit.

CHAPTER 1

Introduction

What is a processor?

The five classic components of a computer are input, output, memory, data path and control, with the last two sometimes combined and called the **processor**.

Instruction set architecture:

The instruction set architecture includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on.

Basically processor are divided into two core category which are:

1) Von newmann architecture.

2) Harvard architecture.

Von newmann architecture:

In this architecture instruction fetching and data reading, writing cannot be done simultaneously, as there is single data path for transmission of data.

Harvard architecture:

In this architecture instruction fetching and data reading, writing can be done simultaneously, as it contains two different memory for code and data.

MIPS

MIPS (originally an acronym for Microprocessor without interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems (now MIPS Technologies).

It is come under von newmann architecture. It has it’s own assembly language.

MIPS assembly language

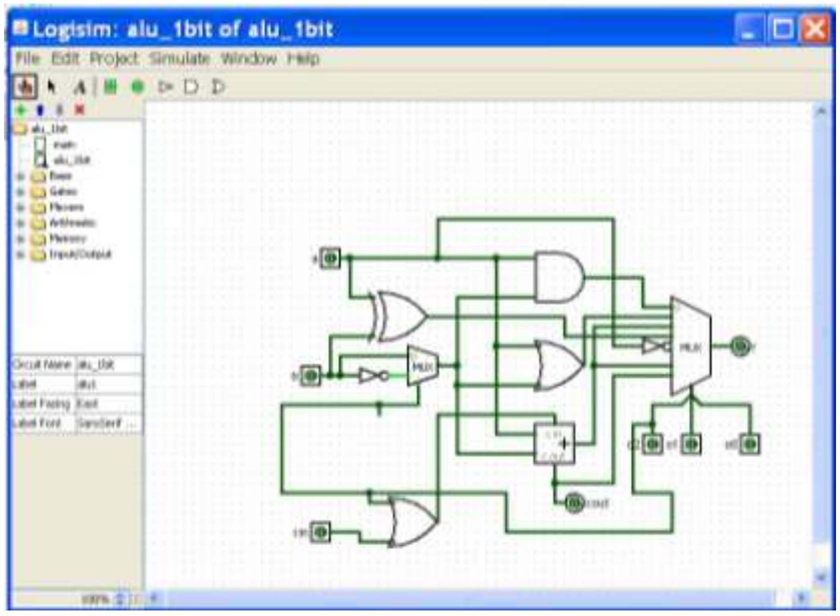
These is assembly language instructions that have direct

hardware implementation.

CHAPTER 2

Tool Used (Logisim)

ABOUT THE SOFTWARE


Logisim: alu_1bit of alu_1bit
File Edit Project Simulate Window Help
MUX

LOGISIM

Logisim is an educational tool for designing and simulating digital logic circuits. With its simple toolbar interface and simulation of circuits as they are built, it is simple enough to facilitate learning the most basic concepts related to logic circuits. With the capacity to build larger circuits from smaller subcircuits, and to draw bundles of wires with a single mouse drag, Logisim can be used (and is used) to design and simulate entire CPUs for educational purposes.

Students at colleges and universities around the world use Logisim for a variety of purposes, including:

- A module in general-education computer science surveys
- A unit in sophomore-level computer organization courses
- Over a full semester in upper-division computer architecture courses

TIME DELAY CONSIDERATION:

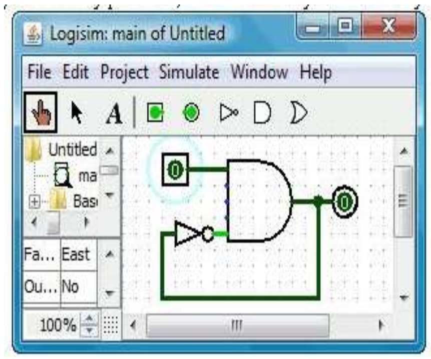
Every component has a delay associated with it. More sophisticated components built into Logisim tend to have larger delays, but these delays are somewhat arbitrary and may not reflect reality.

From a technical point of view, it is relatively easy to deal with this level of sophistication in a single circuit. Dealing with gate delays well across subcircuits, though, is a bit more complex; Logisim does attempt to address this correctly by placing all primitive component's propagation values into a single schedule regardless of the subcircuit in which the component lies.

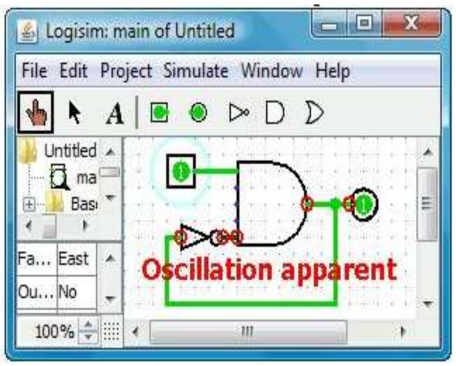
(Via the Project Options window's Simulation tab, you can configure Logisim to add a random, occasional delay to a component's propagation. This is intended to simulate the unevenness of real circuits. In particular, an R-S latch built using two NOR gates will oscillate without this randomness, as both gates will process their inputs in lockstep. This randomness is disabled by default.)

OSCILLATION ERRORS:

The propagation algorithm, which normally works silently without any problems, will become very visible when you create a circuit that oscillates.


Logisim: main of Untitled
File Edit Project Simulate Window Help
Untitled
ma
Bas
Fa... East
Ou... No
100%
0
0

This circuit is currently in a stable condition. But if you change the input to 1, the circuit will effectively enter an infinite loop. After a while, Logisim will simply give up and show an "Oscillation apparent" message telling you that it believes that the circuit is oscillating.


Logisim: main of Untitled
File Edit Project Simulate Window Help
Untitled
ma
Basi
Fa... East
Ou... No
100%
Oscillation apparent

It will display the values it has at the time it gives up. These values will look wrong - in this screen shot, the AND gate is emitting 1 although one of its inputs is 0, but it could be that the NOT gate has a 1 input and a 1 output.

Logisim helpfully circles in red each location that seems to be involved in the oscillation. If an involved point lies within a subcircuit, Logisim will draw that subcircuit's outline in red.

When Logisim detects oscillation, it shuts down all further simulation. You can re-enable simulation using the Simulate menu's Simulation Enabled option.

Logisim detects oscillation using a fairly simple technique: If the circuit simulation seems to many iterations, then it will simply give up and report oscillation. (The points it identifies as being involved are those that were touched in the last 25% of the iterations.) Thus, it could erroneously report oscillation, particularly if you are working with an exceptionally large circuit; but it would be one that is larger than any I have built using Logisim. In any case, if you are confident that the reporting is in error, you can configure the number of iterations completed before oscillation occurs via the Project Options window's Simulation tab.

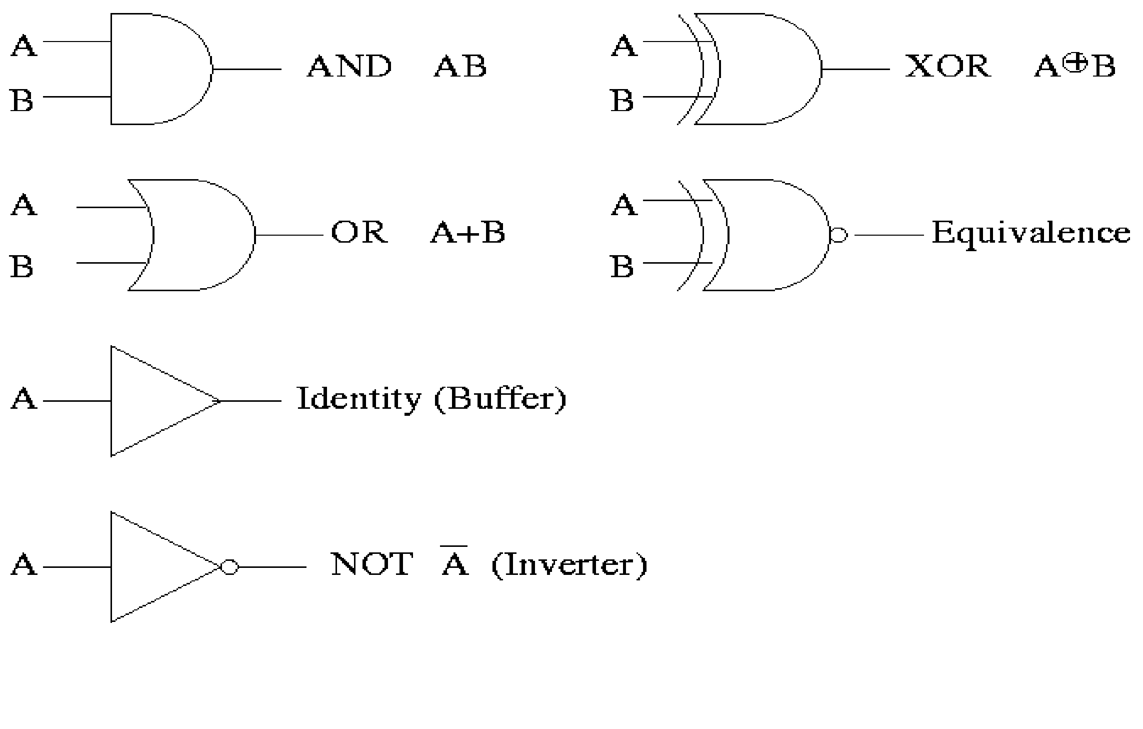
CHAPTER 3

A.L.U

In computing an **arithmetic logic unit** (**ALU**) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessor contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs

GATES

Gates implement basic logic functions: AND OR NOT XOR
Equivalence



PROJECT’S ALU

In our project we have designed 6 bit ALU. 6 bits have been chosen because our memory is of 10 bit, out of which 6 bits are operand. So we will do mathematical operation on them.

INPUT:

- Two 6 bit numbers   A & B
- Carry in (cin).
- Clock.
- Carry write
- opcodes

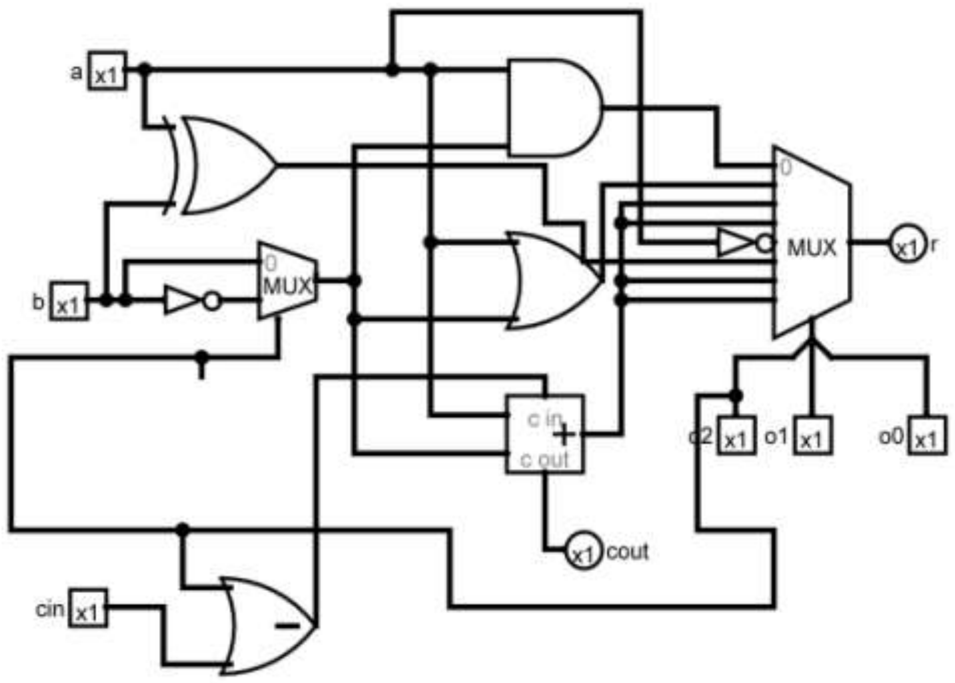
OUTPUT:

One 6 bit result and a zero flag which get 1 if 6 bit result is zero. A.L.U has been design by combining 6 one bit A.L.U in parallel.

OPERATIONS:

A.L.U can perform AND, OR, ADD, SUB, NOT, XOR, ADD with carry, SUB along with carry. A.L.U opcodes and operation are defined below. A.L.U is also able to store previous carry.

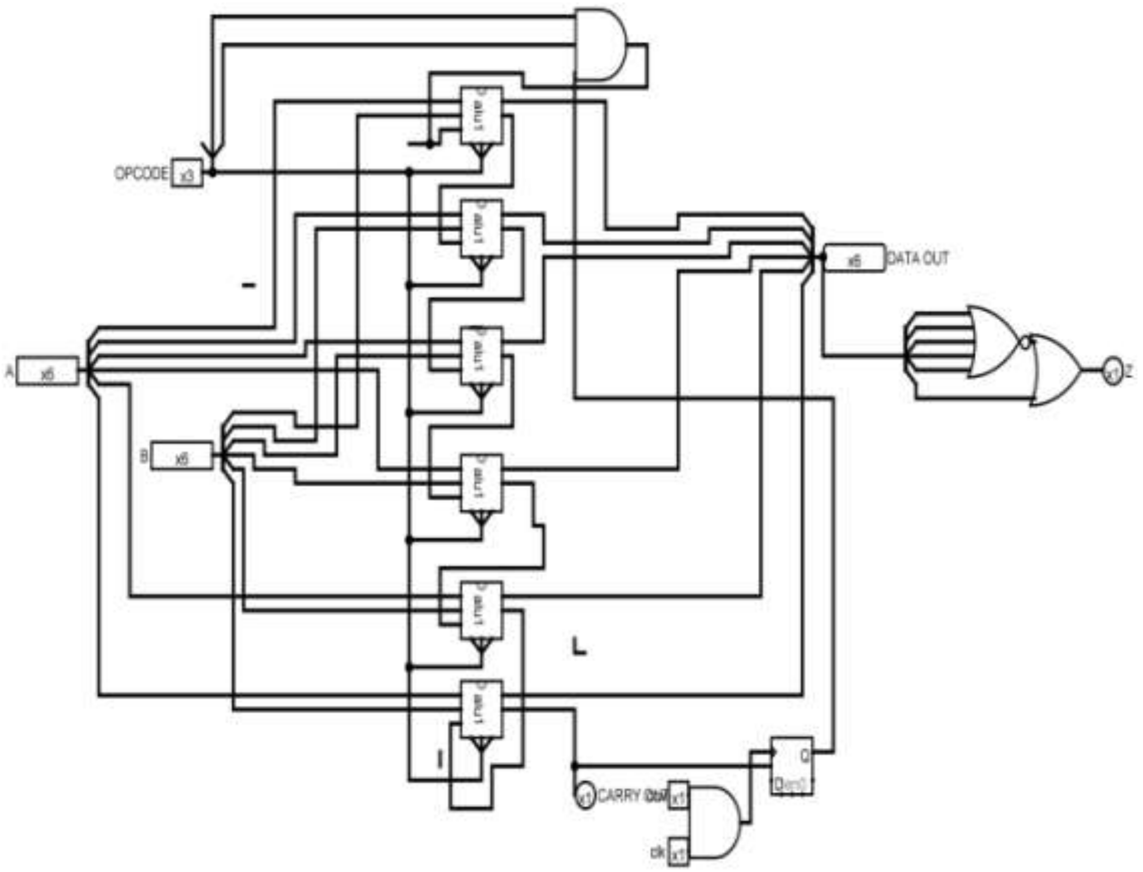
A.L.U OPERATION	A.L.U OPCODES
AND	000
OR	001
ADD	010
ADD ALONG WITH CARRY FLAG(ADC)	011
NOT (INPUT A ONLY)	100
XOR	101
SUBTRACT	110
SUBTRACT ALONG WITH CARRY	111



One bit A.L.U :

- A.L.U perform all the 8 operation on the given input and opcode select the required result using 3X8 multiplexer.
- Subtraction is done by using formula:

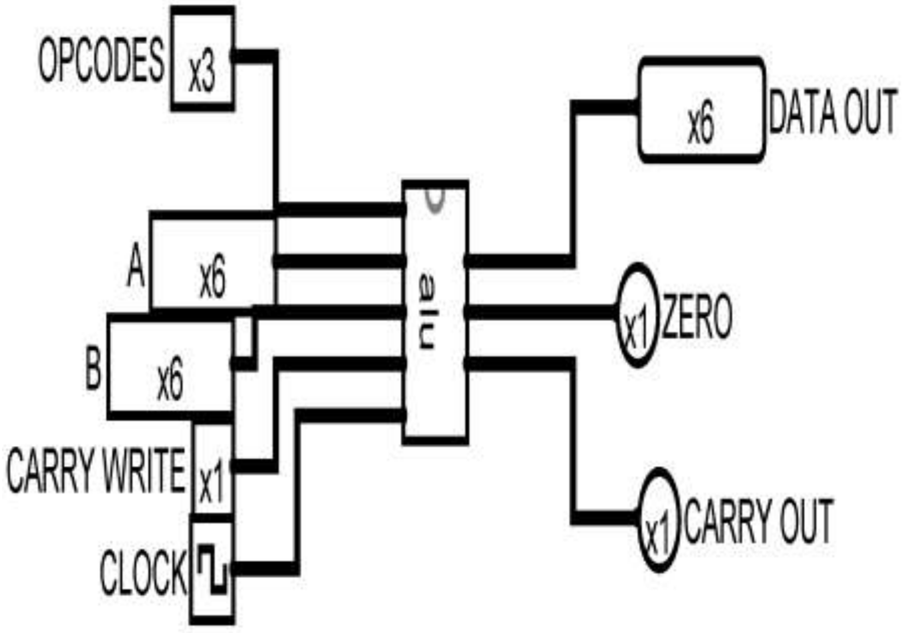
  A—B = A + B` + 1
- NOT operation is performed only on input A.



6 bit A.L.U:

- Each alu1 block is equivalent to 1 bit A.L.U designed above.
- Connecting 6 one bit a.l.u in parallel, a 6 bit A.L.U has been designed.
- Z get 1 if data out is zero else remain 0.
- Carry out is the write signal for storing carry out in D flip-flop.
- For opcode for ADC, SBC the carry flag also get add up.

Final Form of ALU



6 bit A.L.U I.C

INPUT:
1) 3 bit opcodes.
2) 6 bit Input A.
3) 6 bit Input B.
4) Carry write signal.
5) Clock.

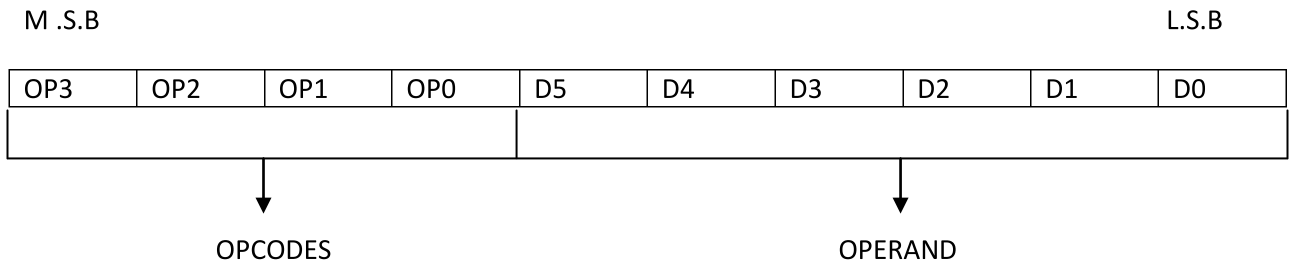
OUTPUT:
1) 6 bit data out.
2) Zero flag.
3) Carry out.

CHAPTER 4

MEMORY:

AIM: To Designing a memory.

- We are designing a 64x10 memory i.e. it will have 64 register each having capacity of storing 10 bits.
- 10 bit register is chosen as 4 bit are opcodes and 6 bit operand.

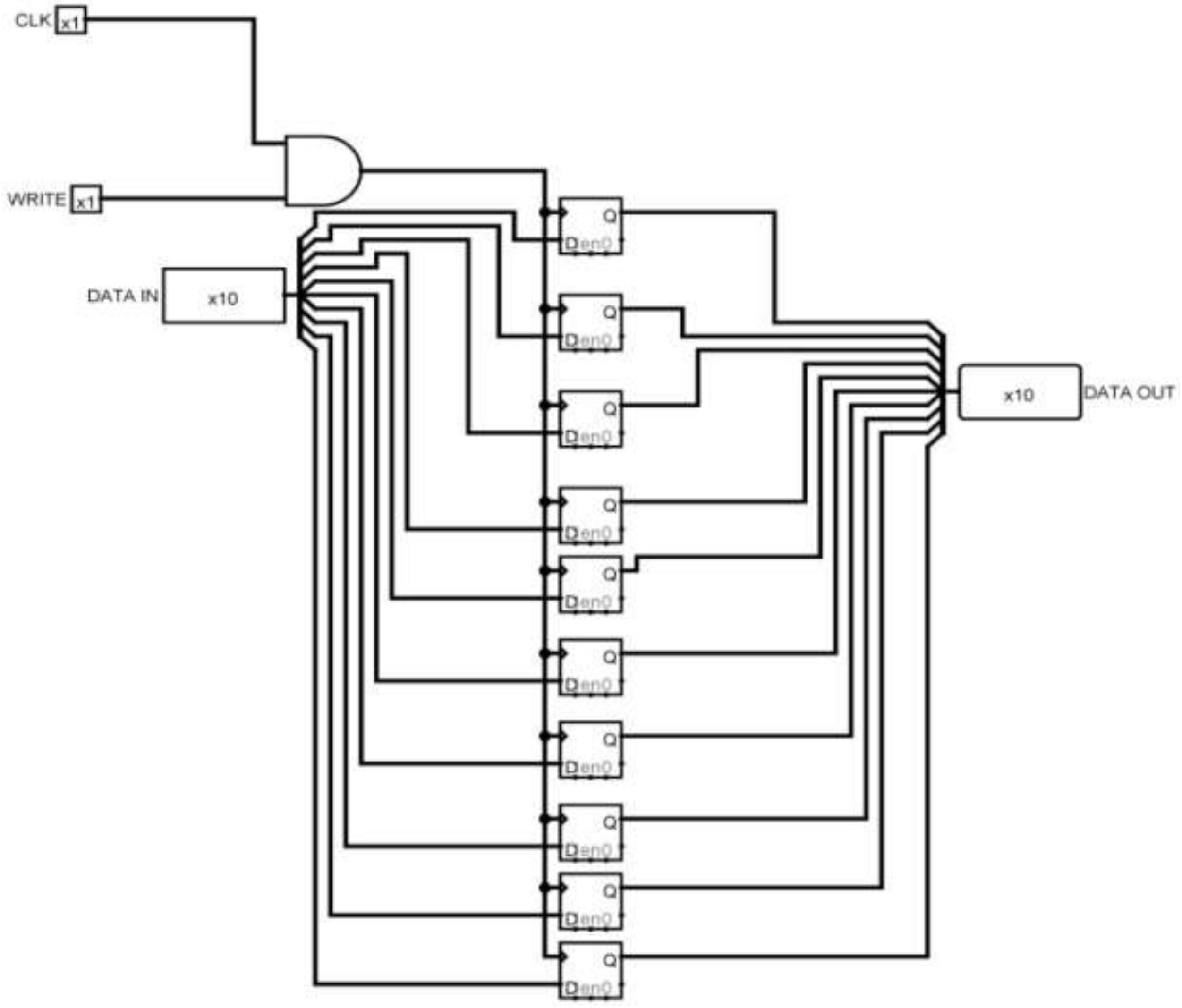


INPUT:

1) 6 bit input address.
2) Data input.
3) Data write signal.
4) Clock.

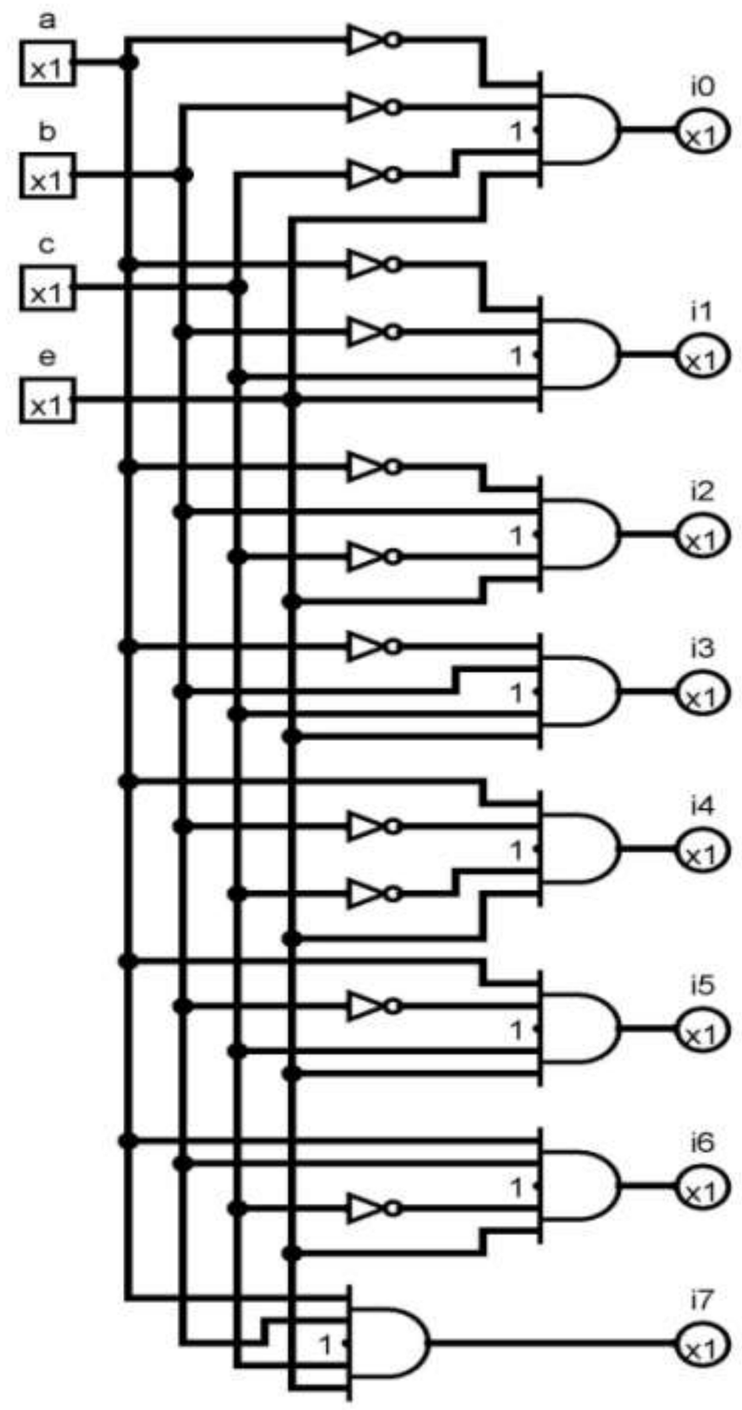
OUTPUT:

1) Data out.



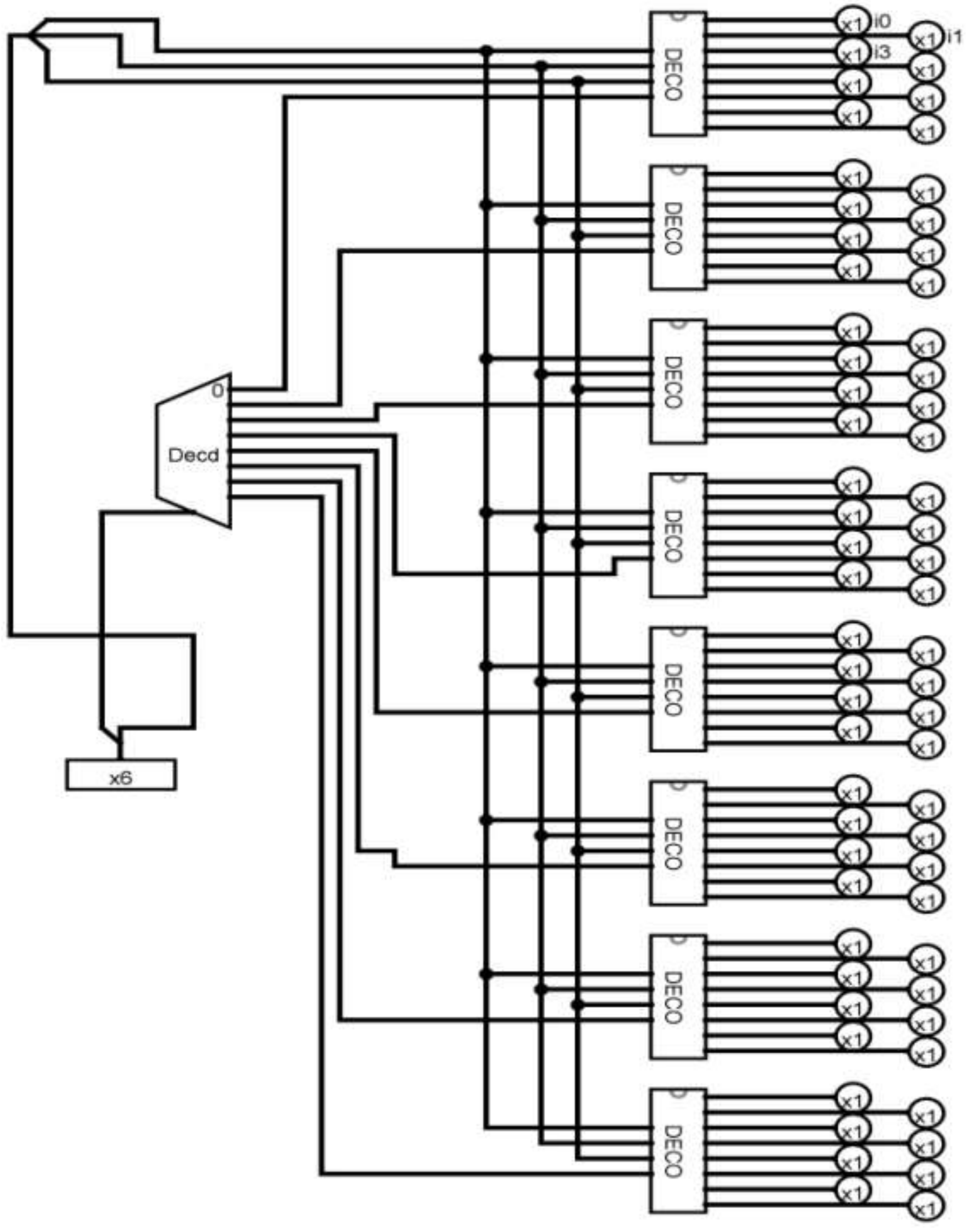
10 BIT REGISTER

1) Each flip-flop is falling edge.
2) Clock input is continuous vary pluses.
3) Whenever write signal get 1, flip-flop can see the clock and store data accordingly.



COMBINATIONAL CIRCUIT OF 3X8 DECODER WITH ENABLE INPUT

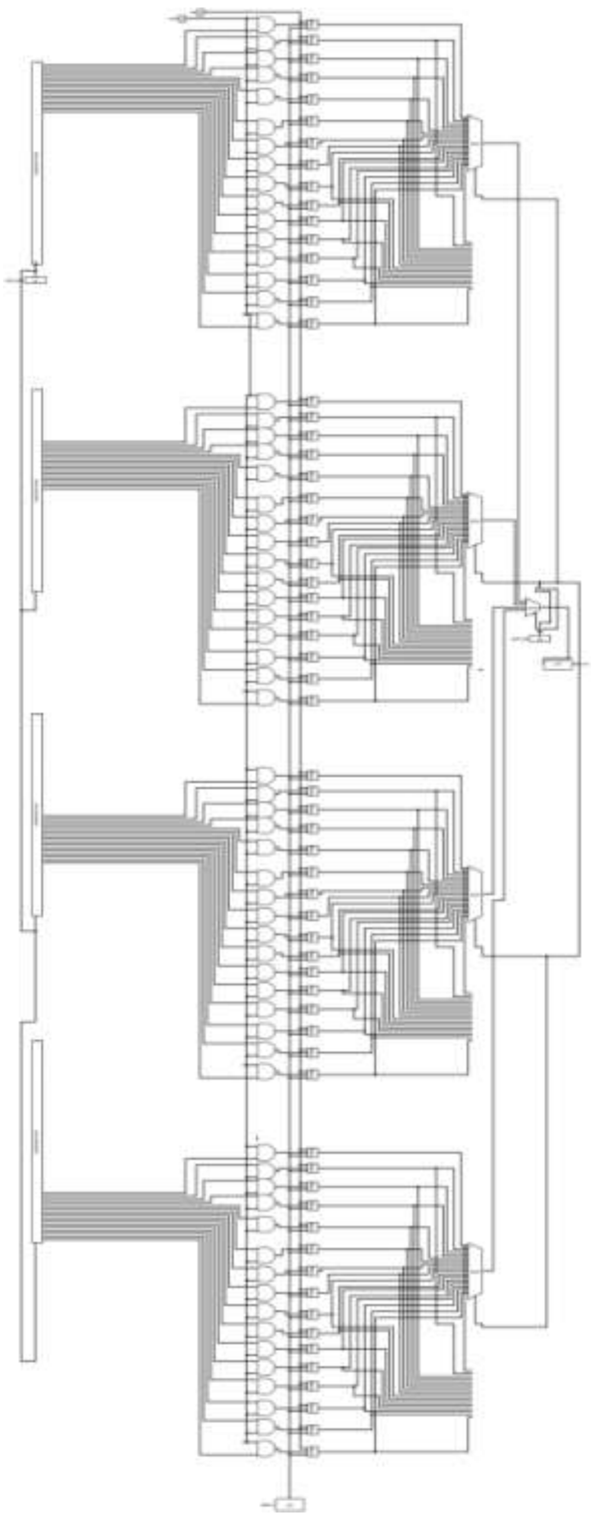
Due to none availability of 3x8 decoder along with enable input, a decoder has been designed using combination analysis. Where providing input, output and a logical relation, LOGISIM itself generate a circuit.



COMBINATIONAL CIRCUIT OF 6X64 BIT DECODER

Using above designed 3x8 bit decoder, a 6X64 bit decoder has been designed in which upper 3 bit select the decoder and lower 3 bit select the bit.

This 6 bit will work as address in the memory.



64X10 Memory

The above circuit is 64X10 memory which continuously reflect the output according to input address.

INPUT:

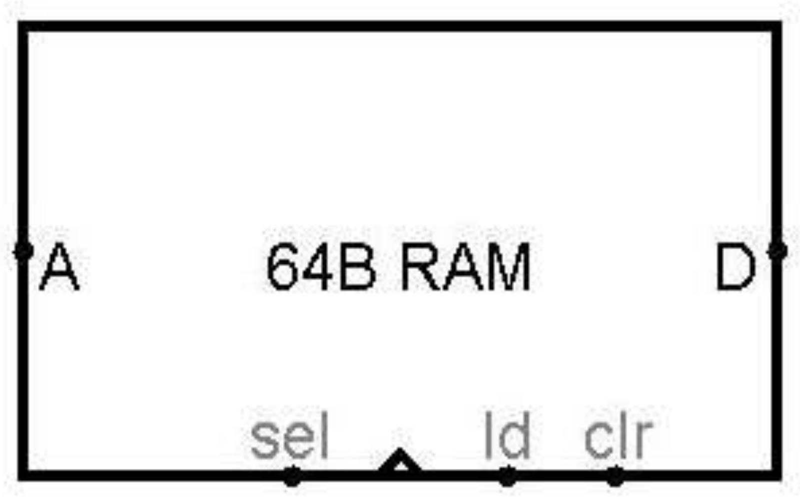
1) 6 bit address.
2) Write signal.
3) Data in(data to be written).
4) Clock.

OUTPUT:

1) Data out.

***

But we have used memory provided by software, As while writing code to memory software provide simple file through which we can write burn our code in it.



SOFTWARE PROVIDED RAM

CHAPTER 5

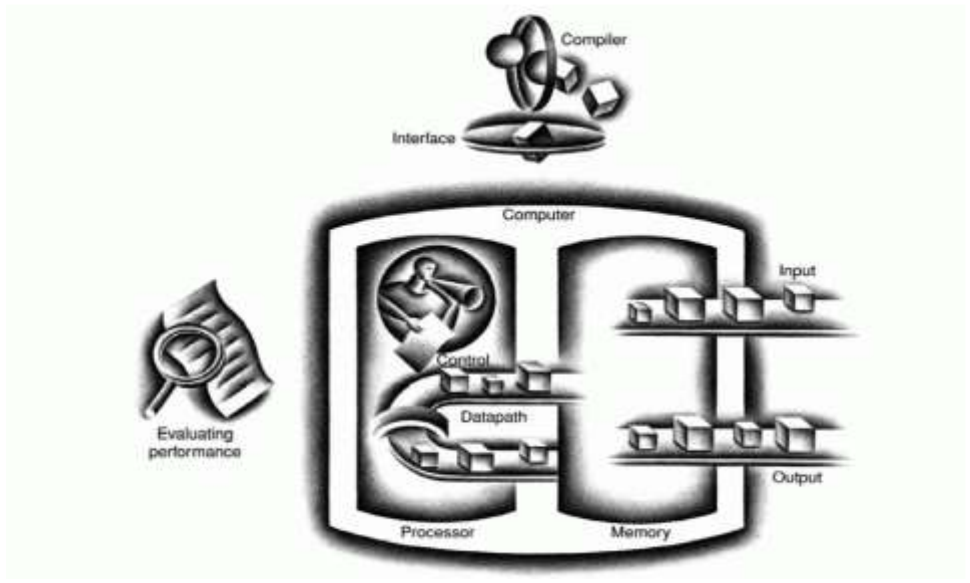
Opcodes

Projected Opcode and their mnemonics

0000	ADD	Add the operand to the accumulator and stores result on accumulator
0001	ADC	Adds operand and carry bit to the accumulator and stores result in Accumulator.
0010	SUB	Subtracts the operand from accumulator and stores result in accumulator
0011	SBC	Subtract operand and carry bit from accumulator
0100	AND	Bitwise logical AND
0101	OR	Bitwise logical OR
0110	XOR	Bitwise XOR
0111	NOT	Logical inverse of accumulator
1000	LDI	Load the operand from the instruction word into accumulator
1001	LDM	Load accumulator with value stored at memory addressed by 6 bit operand stored in instruction register.
1010	SPM	Store the accumulator at specific location
1011	JMP	Transfer program control to the instruction in target address
1100	JPI	Transfer program to address stored in operand of instruction
1101	JPZ	Transfer program counter if accumulator is zero
1111	JPC	Transfer program control to instruction in the target add if carrybit is 1

CHAPTER 6

Data Path



A datapath is a collection of functional units, such as arithmetic logic units or multipliers, that perform data processing operations. This functional unit is used to operate on or hold data within processor. The DATA PATH is the body of architecture of computer.

Just as in human, body is the physical presence in which veins carries blood same is the case with architecture of computer. Here Data Path is the main body and data and address busses acts like its transport carrier and its 5 main components acts as its body organs.

In our project Data Path contains 5 main components

- Program Counter
- Memory
- Instruction Register
- Accumulator
- ALU

6.1 Program Counter

Program Counter is a register which contains the address of next instruction to be executed. Depending on the details of the particular computer, the PC holds either the address of the instruction being executed, or the address of the next instruction to be executed.

Input to the Program Counter : In our project program counter can have three input

1. Clock
2. Write signal
3. Data input

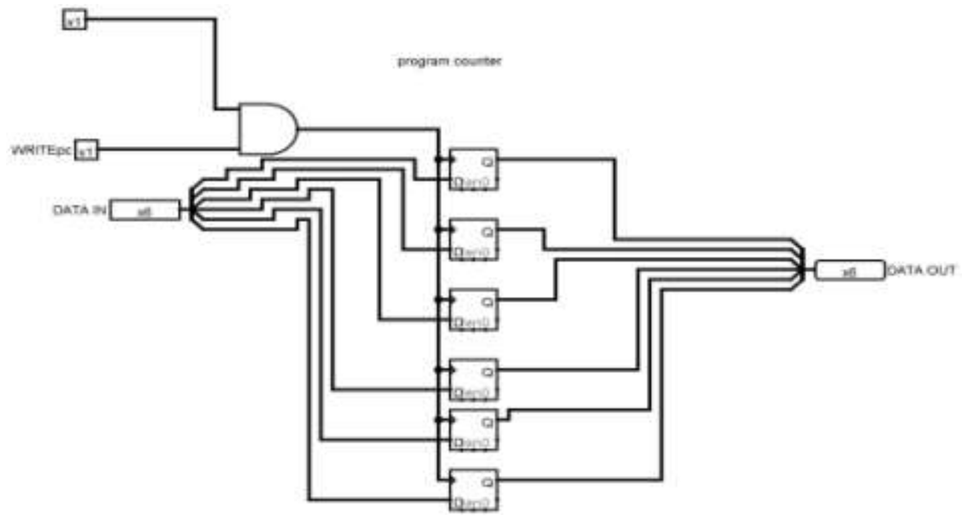
   Data can be injected in P.C in three ways.

- Input from instruction register,
- Input from Data Register
- Input which comes from ALU, to increment the address by 1 bit.

Output from the Program Counter : There will be two output from P.C

- Output to ALU which will increment the address by 1 bit.
- Output to Memory, which will point to the memory location given by P.C

Specifications : P.C cosist of 6 D flipflops. These 6 flipflop serves as register , so continuously reflecting data will get stored in register.



6.2 ACCUMULATOR

In a computer's central processing unit (CPU), an accumulator is a register in which intermediate arithmetic and logic results are stored. Without a register like an accumulator, it would be necessary to write the result of each calculation (addition, multiplication, shift, etc.) to main memory, perhaps only to be read right back again for use in the next operation. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower than that used for a register.

In our project accumulator is 6 bit register where result of every instruction is stored. ACC stores data on falling edge.

INPUT:

1. Clock
2. Write signal
3. Data input

Data can be stored in 3ways

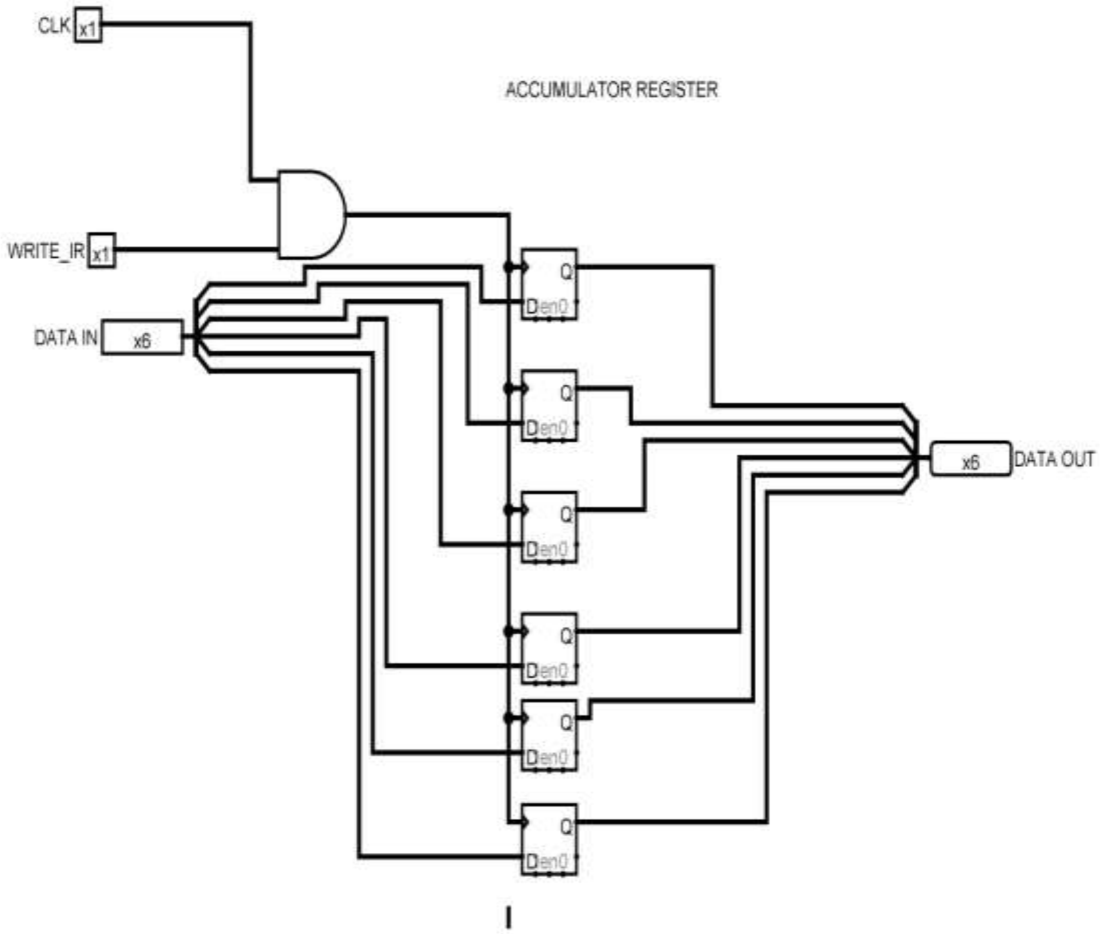
- From ALU to accumulator which is the result of any mathematical operation.
- Data from memory

OUTPUT:

One output goes to memory
One will go to ALU

Specification:

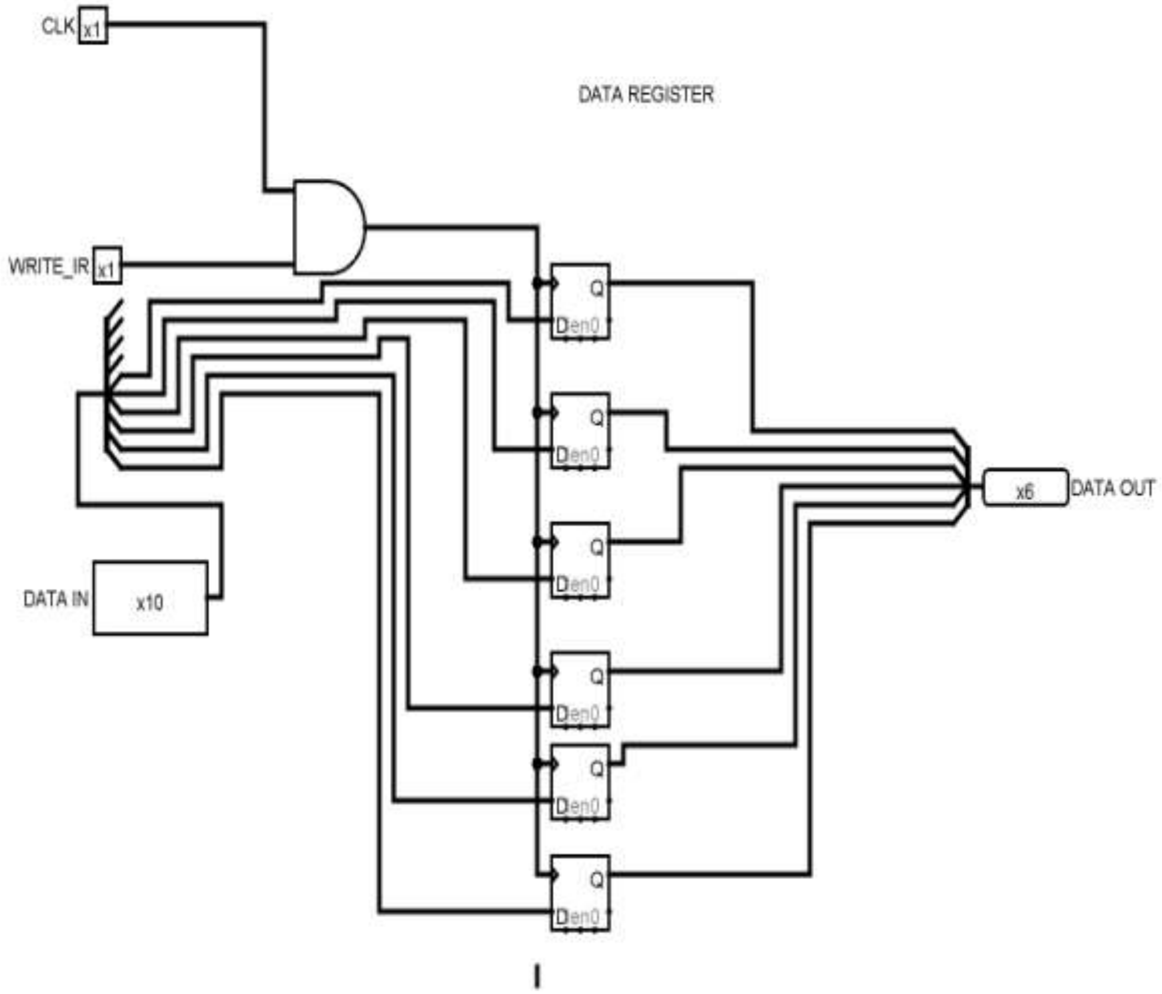
Accumulator will consist of 6D flip flops, so continuously reflecting data will get stored in register.



6 BIT ACCUMULATOR

6.3 DATA REGISTER

DATA REGISTER is playing same role as of accumulator and have same specification. The only difference is of data input and………



6 BIT DATA REGISTER

6 bit data register have been designed using 6 D flip-flop (falling edge).

6.4 INSTRUCTION REGISTER

In computing, an instruction register is the part of a CPU's control unit that stores the instruction currently being executed or decoded. In simple processors each instruction to be executed is loaded into the instruction register which holds it while it is decoded, prepared and ultimately executed, which can take several steps.

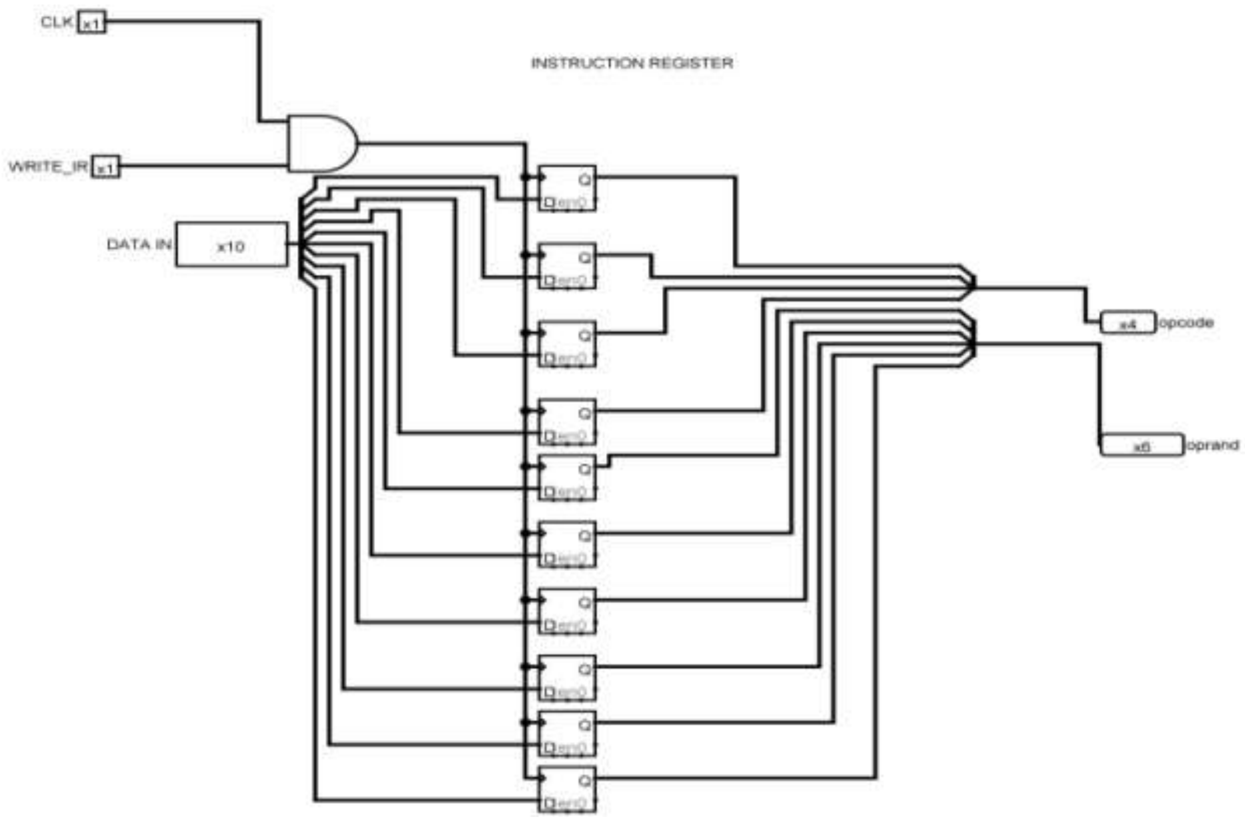
A 10 bit instruction register has been designed which holds current instruction, which includes opcode and an operand.

INPUT:

1) Clock
2) Data in
3) Write signal

OUTPUT:

1) Continuously reflecting opcode of current instruction.
2) Contiuously reflecting operand of current instruction.



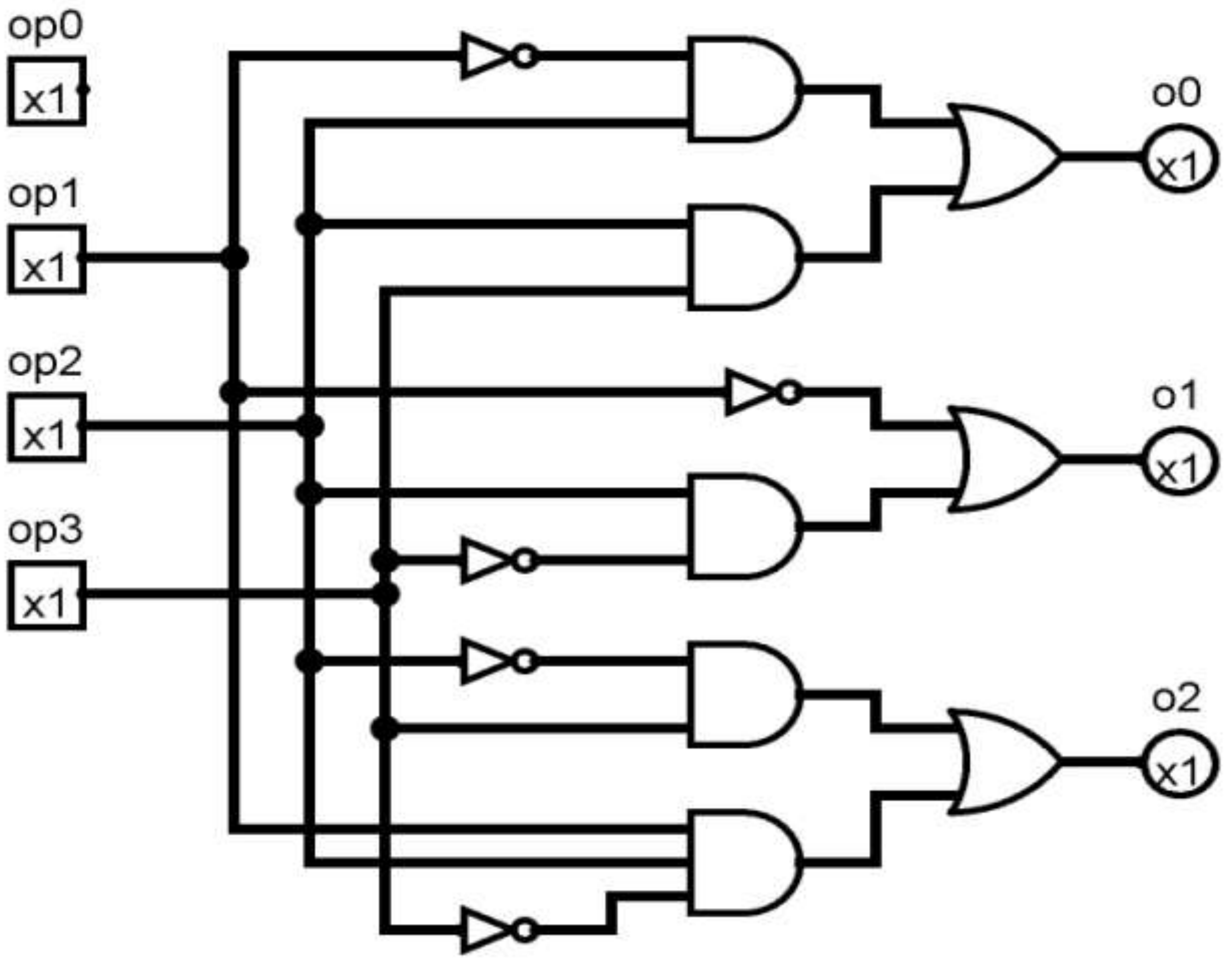
10 BIT INSTRUCTION REGISTER

6.5 A.L.U CONTROL UNIT:

We have to map our instruction set to A.L.U operation codes.

op0	op1	op2	op3	o0	o1	o2
0	0	0	0	0	1	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	1	0	0

This is done using combinational analysis option provided in LOGISIM where providing input, output and truth table, circuit can be drawn.



Combinational circuit drawn by LOGISIM having opcode as input and A.L.U operation as output using truth table defined above.

6.6 Working of Data Path

Till now we have specified about the 5 main organs of datapath that is P.C ,Memory, IR, ALU and Accumulator but the main thing is how these five main organs are assembled to make work of datapath.

The working of datapath starts with 000000 in P.C .since our project is multicycle, every instruction takes 4cycles to execute completely, thus after every 4 cycles P.C is is incremented by 1 through ALU. P.C is connected to memory, so whatever be the value of P.C will act as 6 Bit address to memory. So whatever is the data input be given will get stored in memory location at the given address. Memory is connected to IR, made up of D flip flop. So the 10 bit input data will get stored IR whose starting 4 bits acts as opcode and next 6 bits acts as operand. Now IR is connected to accumulator, we know that accumulator is used to store operand for mathematical operation………………………

First PC will be in 000000 state and after every instruction it will increase by 1 as one of its output goes through ALU. Now for 15 different instruction we are describing separate methods for executing them.

ADD,ADC,SUB,SBC : In all these 4 instruction data will be fetched from memory to data register and from data register output will go to ALU for mathematical operation with contents of Accumulator and mathematical output will get stored in Accumulator.

AND,OR,NOT,XOR : All these 4 instruction will work in same manner as of ADD , ADC, SUB , SBC but the only difference is that of ALU operation will change from former.

LDI (Load accumulator with immediate value):
In this instruction whatever be the operand given by user in instruction register, will get stored in accumulator .For eg. LDI 101010 – means with opcode 1000 and operand 101010 , instruction register will get stored by value 1000101010 and accumulator will get 101010.

LDM (Load accumulator with value stored at memory addressed by 6 bit operand stored in instruction register):

In this instruction whatever be the operand in IR will act as address for memory, so whatever is written at that memory location will get stored in Accumulator. For example LDM 101010 – means with opcode 1001 and operand 101010, instruction register will get stored by value 1000101010 and accumulator will get 101010.

SPM (Store the accumulator at specific location) :

In this instruction whatever data is stored in accumulator will get stored in memory at the user given address. For example SPM 101010 – means content of accumulator will get stored in memory at the address 101010.

JMP (Transfer program counter to the instruction in target address):

In this instruction program counter will go to the user given address. For example JMP 1010101 – means program counter will point memory location whose address is 101010.

CHAPTER 7

CONTROL UNIT

DIFFERENT CONTROL SIGNALS

There are total of 14 control signals abbreviated as follow:

PCML	Program counter multiplexor low
PCMH	Program counter multiplexor high
PCW	Program counter write
MASM	Memory address source multiplexor
MW	Memory write
IRW	Instruction Register write
AML	Accumulator multiplexor high
AMH	Accumulator multiplexor low
AW	Accumulator write
ALUAM	ALU A multiplexor
ALUBM	ALU B multiplexor
ALUOM	ALU operation multiplexor
DRW	Data Register write
CRW	Carry register write

STATES ACTION:

Since we are designing multicycle micro controller, controller can be in following states:

0 – Instruction fetch

1—Instruction Interpretaion

2—Data Fetch

3—Arithmatic Instruction

4—Logic instruction

5—Load Accumulator Immediate

6—Load accumulator from memory

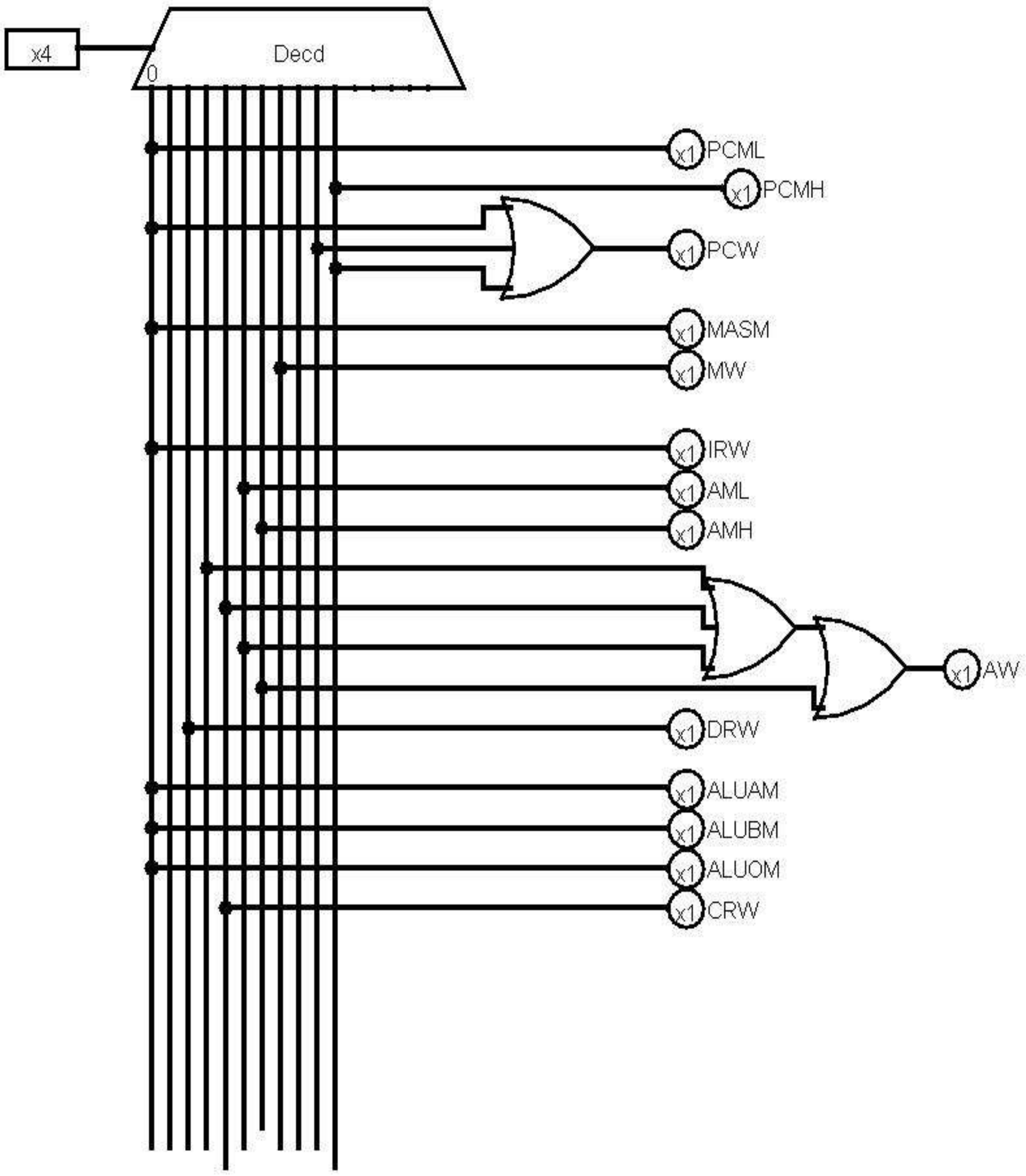
7—Store accumulator to memory

9—Jump immediate

10—Jump indirect

Below is a truth table showing what happens to control signal during each states

STATES	0	1	2	3	4	5	6	7	9	10
PCML	1	0	0	0	0	0	0	0	0	0
PCMH	0	0	0	0	0	0	0	0	0	1
PCW	1	0	0	0	0	0	0	0	1	1
MASM	1	0	0	0	0	0	0	0	0	0
MW	0	0	0	0	0	0	0	1	0	0
IRW	1	0	0	0	0	0	0	0	0	0
AML	0	0	0	0	0	1	0	0	0	0
AMH	0	0	0	0	0	0	1	0	0	0
AW	0	0	0	1	1	1	1	0	0	0
ALUAM	1	0	0	0	0	0	0	0	0	0
ALUBM	1	0	0	0	0	0	0	0	0	0
ALUOM	1	0	0	0	0	0	0	0	0	0
DRW	0	0	1	0	0	0	0	0	0	0
CRW	0	0	0	1	0	0	0	0	0	0



State decoder built using 4x16 decoder

Current State	Condition (Operation)	Next State
0	X	1
1	ADD,ADC,SUB,SBC,AND,OR,XOR	2
1	NOT	4
1	LDI	5
1	SPM	7
1	JMP	9
1	JPI	10
1	JPZ/JPC	9
1	JPZ/JPC	10
2	ADD,ADC,SUB,SBC	3
2	AND,OR,XOR,NOT	4
1	LDM	6

Now we will design saperate circuit for saperate pin of next state .

Least significant pin will be 1 when state is either 1,3,5,7 or 9.

Current state	Opcode (instruction)	Zero	Carry
0000	X	X	X
0001	1010	X	X
0001	1000	X	X
0001	1101	1	0
0001	1111	0	1
0010	0000	X	X
0010	0001	X	X
0010	0010	X	X
0010	0011	x	X

When pin 2 is high state can be 2,3,6,7,10

Current state	Opcode (instruction)	Zero	Carry
0001	0000	X	X
0001	0001	X	X
0001	0010	X	X
0001	0011	X	X
0001	0100	X	X
0001	0101	X	X
0001	0110	X	X
0010	0000	X	X
0010	0001	X	X
0010	0010	X	X
0010	0011	X	X
0001	1001	X	X
0001	1010	x	X

Pin 3 will be high when following state will occur 4,5,6,7

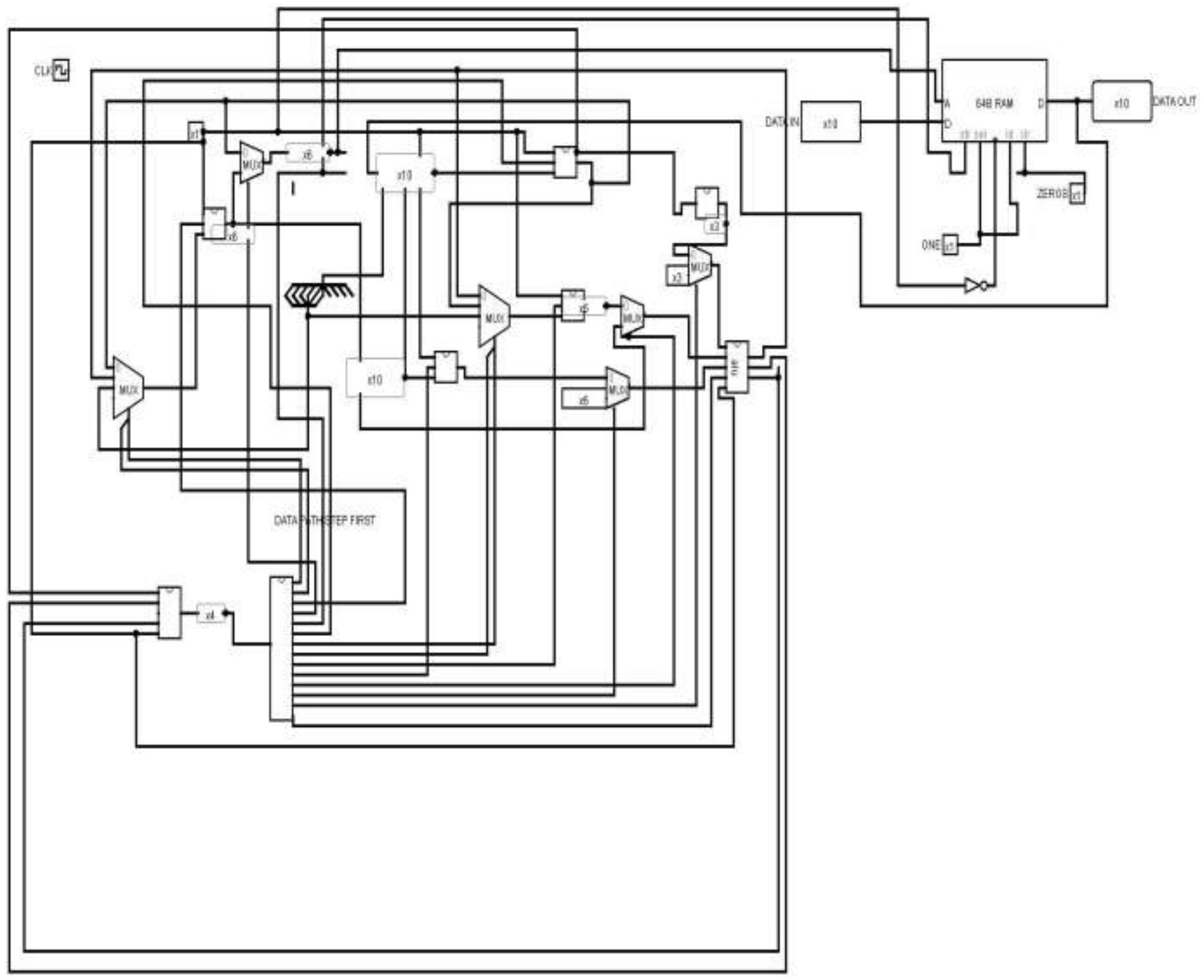
0001	0111	X	X
0010	0100	X	X
0010	0101	X	X
0010	0111	X	X
0010	0110	X	X
0001	0111	X	X
0001	1001	X	X
0001	1010	X	X

Pin 4 will be high when next pin is either 9 or 10

0001	1011	X	X
0001	1101	1	X
0001	1111	X	1
0001	1100	X	X

CHAPTER 8

Main Circuit Diagram



References

1) David A Patterson And John l.Hennessy, Computer organization & design,3rd ed

Morgan Koffman Publishers,2005

2) . Morris Mano, Digital Design, Third Edition, Prentice Hall, 2002

3) Barry.B.Brey, Intel. Microprocessors 8086-8088,4th ed,Prentice Hall Inc 1997

4) Intel Manual “ http://www.**intel**.com/design/pentiumii/**manual**s/243191.htm”