Experiment No. 4(B)

Group 6:

Alamanda Nikhil Teja(130101005)

Midhul Varma(130101047)

N. Rahul(130101050)

K. Sai Krishna(130101039)

Ankit Kalavagunta(130101029)

Objective: To design and fabricate a 4-bit CPU.

Instruction Set Design and format:

The processor has been designed to recognize a simple instruction set which includes operations to load, store and perform some arithmetic and logic operations on the data. In addition to these instructions, a load and special branch instruction are also included to manipulate program flow.

In general the instructions perform operations on data stored in memory (referenced by the input address given along with the instruction) and the single Accumulator (ACC) register which is present in the processor. Direct access to any of the other registers is not given, and hence the need for register addressing is eliminated

The implemented instructions are described below:

- START - Indicates the start of the program
- JUMP - Jumps to a specified 8-bit address in the RAM (i.e sets PC)
- LOAD - Loads the value stored in a particular memory address in the RAM into the Accumulator (ACC)
- STORE - Stores the value of the Accumulator (ACC) in a particular memory address
- BRANCH - Checks if the value stored in a particular memory location is equal to the value stored in the accumulator. If true, it jumps to a particular memory location (ie. PC value changes)
- ADD/SUB/AND/OR - These operations all take the value stored in the Accumulator and the value stored in a given memory address, perform the respective operation on these two and store the result back into the accumulator
- STOP - indicates the stop of the program

Every instruction has a unique Opcode and is followed by any required operands in successive memory locations

All the instructions except BRANCH, simply take a single 8 bit address input. (i.e 2 4-bit blocks or 2 words in memory). BRANCH takes 4 4-bit blocks or words. (i.e 2 8-bit addresses)

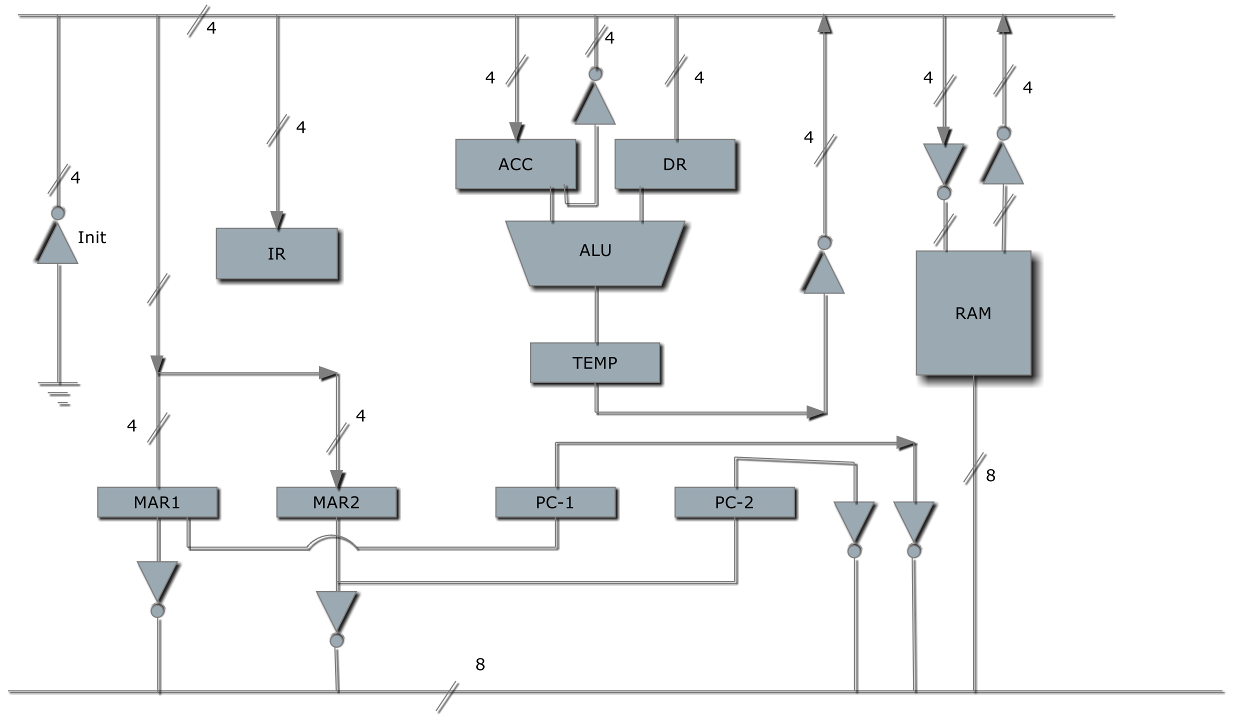
The structure of each of these instructions is given in the table below:

Opcode	Operation	Structure
0000	START	[opcode]
0001	JUMP	[opcode] [MSB of address][LSB of address]
0010	LOAD	[opcode] [MSB of address][LSB of address]
0011	STORE	[opcode] [MSB of address][LSB of address]
0100	ADD	[opcode] [MSB of address][LSB of address]
0101	SUB	[opcode] [MSB of address][LSB of address]
0110	AND	[opcode] [MSB of address][LSB of address]
0111	OR	[opcode] [MSB of address][LSB of address]
1000	BRANCH	[opcode] [MSB of cmp addr][LSB of cmp addr] [MSB of branching addr][LSB of branching addr]
1001	STOP	[opcode]

(Every group mentioned in square brackets consists of 4 bits.)

Processor Structure (Data Path):

The data path unit of the processor contains 4 registers, 1 RAM module, 2 4-bit Memory address registers and 2 4-bit Program counters. The component's inputs and outputs are connected to the data/address bus as shown in the figure. Input to the various components is controlled using the control signals from the control unit described in the following section. The output connections to the buses from the components are controlled with the help of tri-state buffers to selectively control the data being written to the bus.



To control the flow of information in the data path unit, several connections to various components are declared as control signals and are given by the control circuit in the required sequence.

Control signals in general:

- Each register's Load enable (Mode control)
- Counter's parallel load enable and Count enable (i.e to increment counter)
- RAM's read/write enable
- Every tri-state buffer's enable

Specific Control Signals with their use:

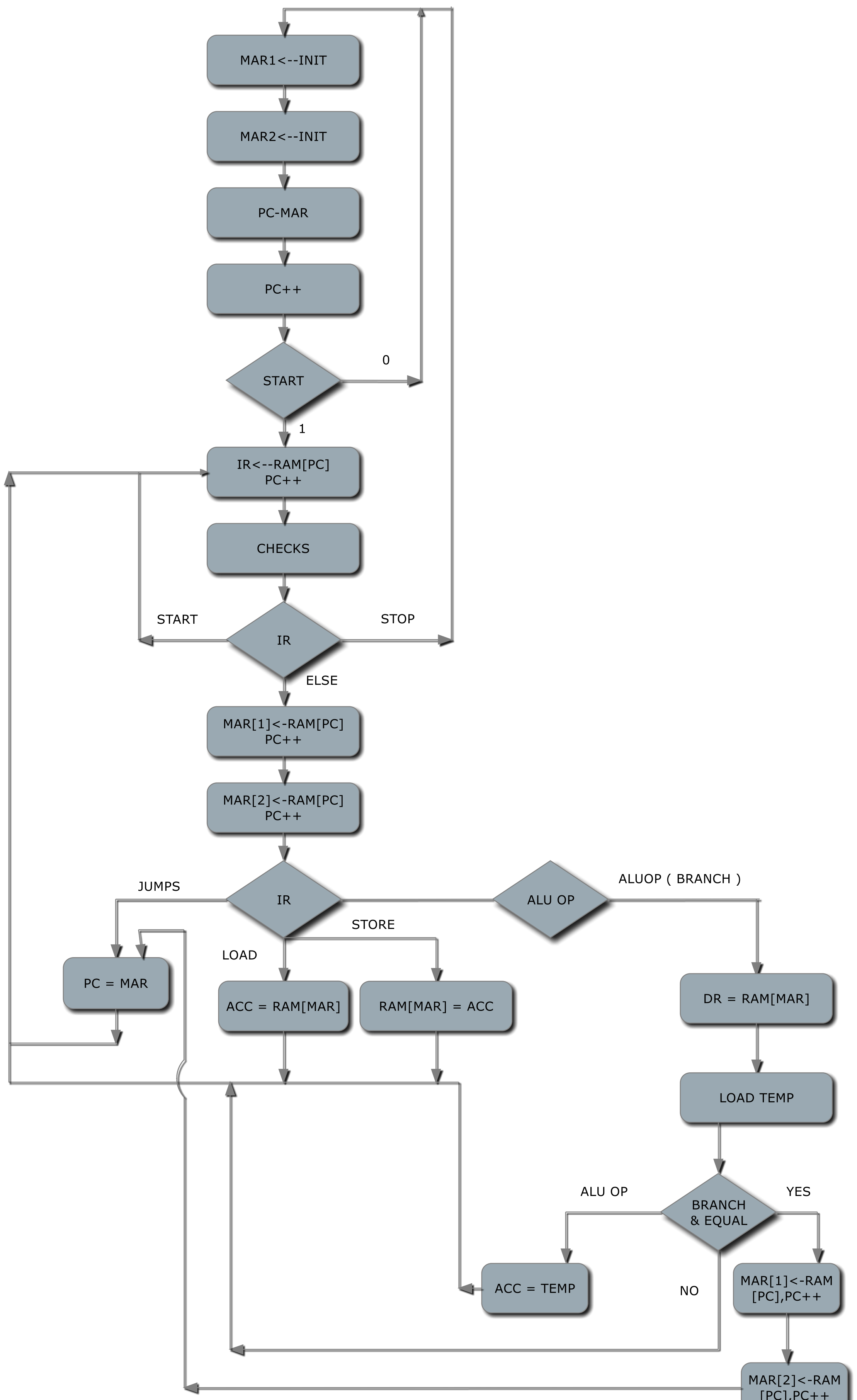
Load DR	Loads register DR
Load ACC	Loads register ACC
Load PC	Loads register PC
Load MAR1	Loads register MAR1
Load MAR2	Loads register MAR2
Load IR	Loads register IR
RAM-W	Enables Write Mode for RAM
Con ACC	Connects ACC to the Data BUS
Con TEMP	Connects TEMP to the Data BUS
Con PC	Connects PC to the Address BUS
Con MAR	Connects MAR to the Address BUS
Con Init	Initializing inputs to the Data BUS
RAM-R	Enables Read Mode for RAM
Incr PC	Increments PC

These 14 control signals are sufficient to completely control the components in the data path unit. The control circuit is designed to give the correct signals to these outputs (LOW/HIGH) depending on the state of signal during the sequence.

The sequence of the processor consists of 16 states. The ASM chart describing the logical sequence of states is described below.

Control Steps for instructions:

The control unit is implemented as a typical sequential circuit which moves from one state to another depending on inputs and present state. The state diagram (ASM chart) is shown below.


MAR1<--INIT
MAR2<--INIT
PC-MAR
PC++
START
0
1
IR<--RAM[PC]
PC++
CHECKS
START
IR
STOP
ELSE
MAR[1]<-RAM[PC]
PC++
MAR[2]<-RAM[PC]
PC++
JUMPS
IR
ALU OP
ALUOP (BRANCH)
STORE
LOAD
PC = MAR
ACC = RAM[MAR]
RAM[MAR] = ACC
DR = RAM[MAR]
LOAD TEMP
ALU OP
BRANCH
& EQUAL
YES
ACC = TEMP
NO
MAR[1]<-RAM
[PC],PC++
MAR[2]<-RAM
[PC],PC++

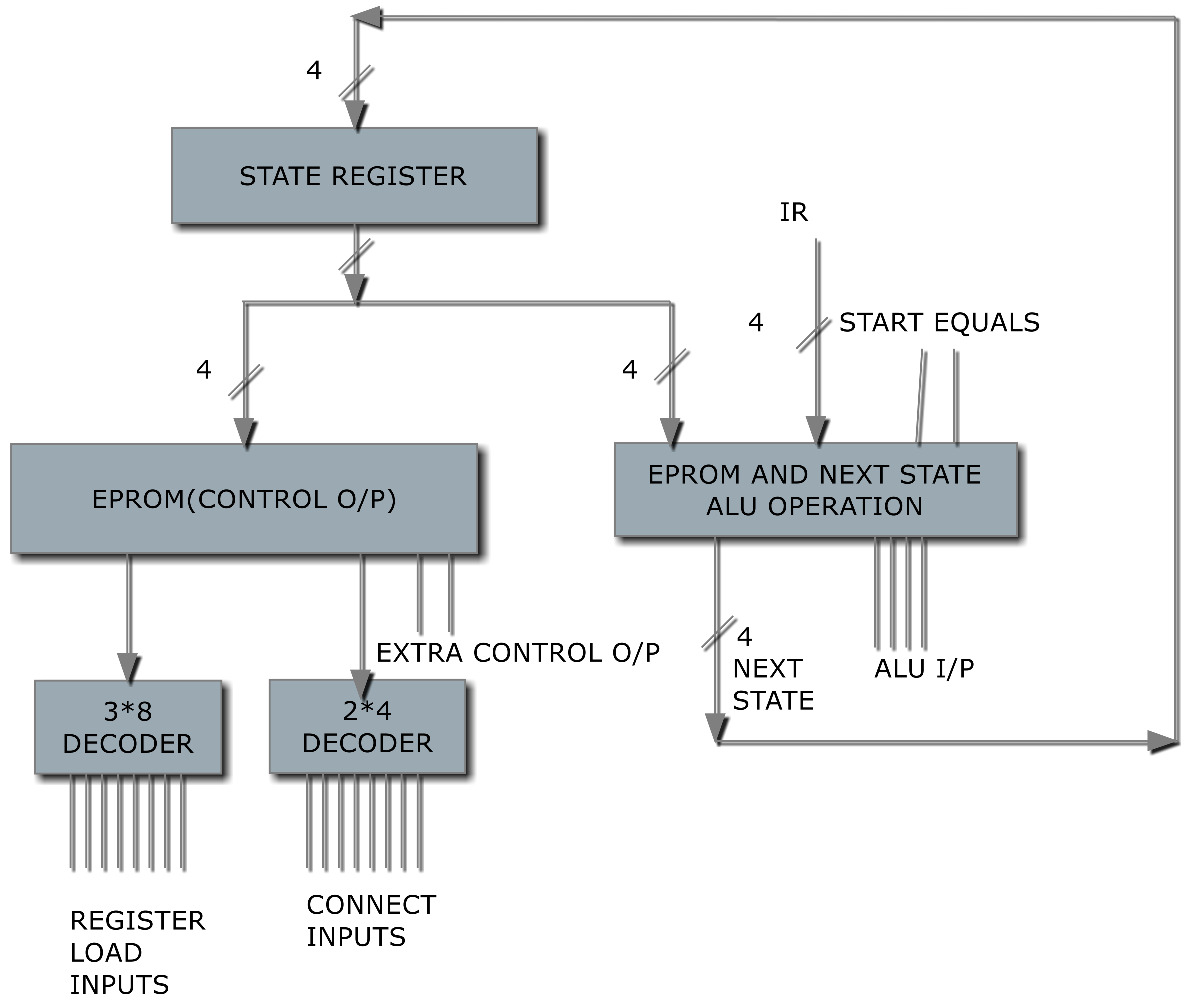
The corresponding values of control signals at each of the states is shown in the table below:

SIGNAL/STATES	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
Load DR	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Load Acc	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
Load PC	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
LOad MAR1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
Load MAR2	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0
Load IR	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM-W	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
ConAcc	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
ConTEMP	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ConPC	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0
ConMAR	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
ConInit	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
RAM-R	0	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0
IncrPC	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0

It can be clearly be seen that quite a lot of the control signals are mutually exclusive. Based on this observation an optimization is made to reduce the effective number of control outputs. The signals are grouped into two groups and each group is decoded using two line decoders of different sizes. The exact grouping and decoding of the signals is described in the section below

<u>Structure of Control Unit:</u>

The control unit is built using a 4-bit state register and 2 EPROM chips. The EPROM chips produce the combinatorial logic required to decide the control signals and next state decision based on the present state. One of the EPROM’s is responsible for generating the control signals for a given state, this is done with the help of two extra line decoders as mentioned before. The other EPROM decides the next state and gives ALU operation signals depending on the present state, values present in the IR and the Start and Equals control inputs. The schematic is shown below:



It is observed that the control outputs depend only on the present state of the circuit and hence they are handled completely by one EPROM. This EPROM chip gives outputs to 2 decoders which decode the correct control signals and also gives 3 extra control outputs which cannot be decoded.

The general organization of control signals is as follows:

- The 3x8 decoder mainly handles Load signals
- The 2x4 decoder mainly handles Connect signals

The inputs are labelled as follows:

- Inputs of the 3x8 decoder are L1,L2,L3
- Inputs of the 2x4 decoder are C1,C2

The final reduced table of EPROM outputs (i.e effective control outputs before decoding) is given below:

State	L1	L2	L3	C1	C2	~ConACC	~RAM-R	IncrPC	HexCode
Q0	0	0	0	0	0	1	1	0	06
Q1	0	1	1	0	1	1	0	1	6D
Q2	0	0	0	0	1	1	0	1	0D
Q3	0	0	1	0	1	1	0	1	2D
Q4	0	1	0	1	0	1	1	0	56
Q5	1	0	1	1	0	1	0	0	B4
Q6	1	1	0	1	0	0	1	0	D2
Q7	1	0	0	1	0	1	0	0	94
Q8	1	1	1	0	0	1	1	0	E6
Q9	1	0	1	1	1	1	1	0	BE
Q10	0	0	0	0	1	1	0	1	0D
Q11	0	0	1	0	1	1	0	1	2D
Q12	0	1	0	1	0	1	1	0	56
Q13	1	1	1	0	0	1	1	1	E7
Q14	0	1	0	0	0	1	1	0	46
Q15	1	1	1	0	0	1	1	0	E6

The other EEPROM is responsible for determining the next state and the ALU operation deciding signal. This decision is made taking the IR outputs, present state and Start and Eq signals as inputs. This makes a total of 10 inputs for this EPROM. So 2^10 (1024) addresses need to be given corresponding values. Doing this manually is very tedious and errors may easily crawl in. To avoid this, A python program which could simulate the ASM chart described above was written. This program automatically created the binary file which contains the outputs for each of the 1024 possible input combinations

Processor I/O:

Input to the RAM of the processor is given from the host computer. (i.e program and constant's are loaded into the memory before execution). This is done with the help of an *Arduino board* (A programmable microcontroller board). The board is programmed to accept serial input from the host computer 4-bits at a time and write it to consecutive locations in the on board RAM. This gives an interactive interface for programmers to input programs and data into the processor's memory

All the above components when put together and connected on the bread-board with the proper buses make up a working 4-bit processor which can process program written with the above described simple instruction set.

Design Issues of the Processor:

- The processor to be designed has to have a 4-bit data bus. This means each memory location in the RAM will consist of only 4-bits. It is not possible to fit a whole instruction into 4 bits and hence instructions will have to span multiple address locations in the RAM.
- The number of elements in the data path need to be minimized as the number of control signals increase with the number of elements
- The combinatorial logic involved in the control unit cannot be hardwired using logic gates as it is very complex, hence we use EPROM chips to implement the required combinatorial logic.
- The number of control outputs turns out to be much larger than the number of outputs available on the EPROM chip. To overcome this, we reduce the effective number of outputs by using two line decoders.
- Tri state buffers need to used to control which component of the data path will force it's output to the data/address bus

- Since the RAM module uses the same pins for both input and output for reading and writing purposes, this flow of data has to be carefully controlled using tri-state buffers

<u>Example programs which run on processor:</u>

- Simple addition program:

  <u>Assembly language code:</u>

  START

  LOAD $TWO

  ADD $THREE

  JUMP (LOAD)

  STOP

  <u>Machine Code:</u>

  (Note: addresses start from 0 in RAM)

  (Note: processor takes address 0 as garbage)

  0000

  //program section

  0000

  0010

  0000

```
1100
0100
0000
1101
0001
0000
0010
1001
//data segment
0010
0011
```

- More complex program with BRANCH involved:

  <u>Assembly Language code:</u>

```
START
LOAD $ZERO
ADD $FIVE
SUB $THREE
BRANCH $THREE (LOAD)
OR $THREE
```

AND $FIVE

STOP

<u>Machine code:</u>

```
0000
//program section
0000
0010
0001
0111
0100
0001
1000
0101
0001
1001
1000
0001
1001
0000
```

```
0010
0111
0001
1000
1001
//data section
0000
0101
0011
```