IITB-CPU

Final Project Report for EE224: Digital Systems

TEAM ID: 2

Anshu Arora

(22B1207)

Chinmay Moorjani

(22B1212)

Sachi Deshmukh

(22B1213)

Sravan K Suresh

(22B3936)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering,

Indian Institute of Technology, Bombay

November, 2023

December 4, 2023

Contents

1 Introduction

The IITB-CPU implements 14 instructions and utilises 8 programmer registers, among which the register 7 ($R7$) is used as a program counter or an instruction register. Internally, 17 states are used to implement these instructions and are described in the following section.

2 Work Distribution

Tasks performed	**Team member**
Ideation and Initial Design of Partial Datapaths for each Instruction	Chinmay Moorjani
Drawing Partial datapaths for each Instruction	Sravan K Suresh
Listing the set of states for all instructions along with control signals for each state	Sachi Deshmukh
Elaboration of instruction flow for each of the individual states	Sachi Deshmukh
State reduction and pen-paper design of minimised FSM	Chinmay Moorjani
Construction of final FSM using draw.io	Sravan K Suresh
Complete Datapath by merging the partial datapaths (Pen-paper design)	Chinmay Moorjani
Design and documentation of final Datapath using draw.io	Anshu Arora
MUX tables that decide control flow	Anshu Arora
Output process/Control Signal table for every state	Anshu Arora
Documentation of merging the entire CPU	Sachi Deshmukh
VHDL description of Multiplexers, Demultiplexers and Registers	Chinmay Moorjani and Sravan K Suresh
VHDL description of Register File	Sravan K Suresh and Anshu Arora
VHDL description of ALU	Sachi Deshmukh and Chinmay Moorjani
VHDL description of Memory	Chinmay Moorjani
VHDL realisation of CPU involving port mapping of components for Datapath	Chinmay Moorjani
MUX processes and coding of FSM via clock, state-transition and o/p processes	Chinmay Moorjani
Building Testbenches and TRACEFILES for each of the VHDL entities	Sravan K Suresh
Cleaning, indentations and comments on all the VHDL codes	Sravan K Suresh
Testing of Final CPU by hard-coded instructions in memory	Chinmay Moorjani and Sravan K Suresh
Debugging and verification of Final CPU	Chinmay Moorjani and Sravan K Suresh
Report drafting and document organisation	Sravan K Suresh

Table 1: Work Distribution Table

Link to the GitHub repository of this project: Project Repository

3 States

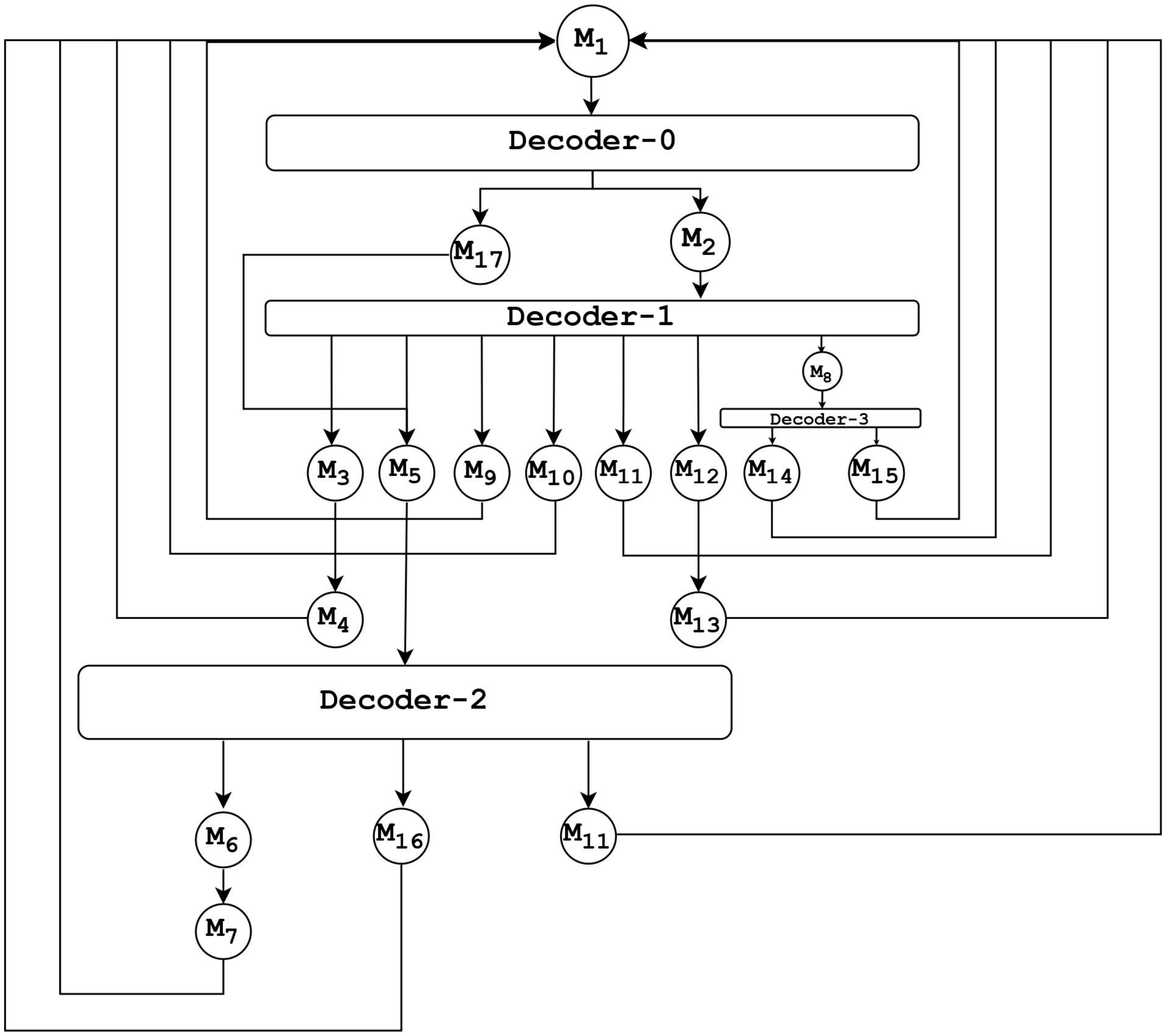
Description

The merged states and a short description of each is given below:

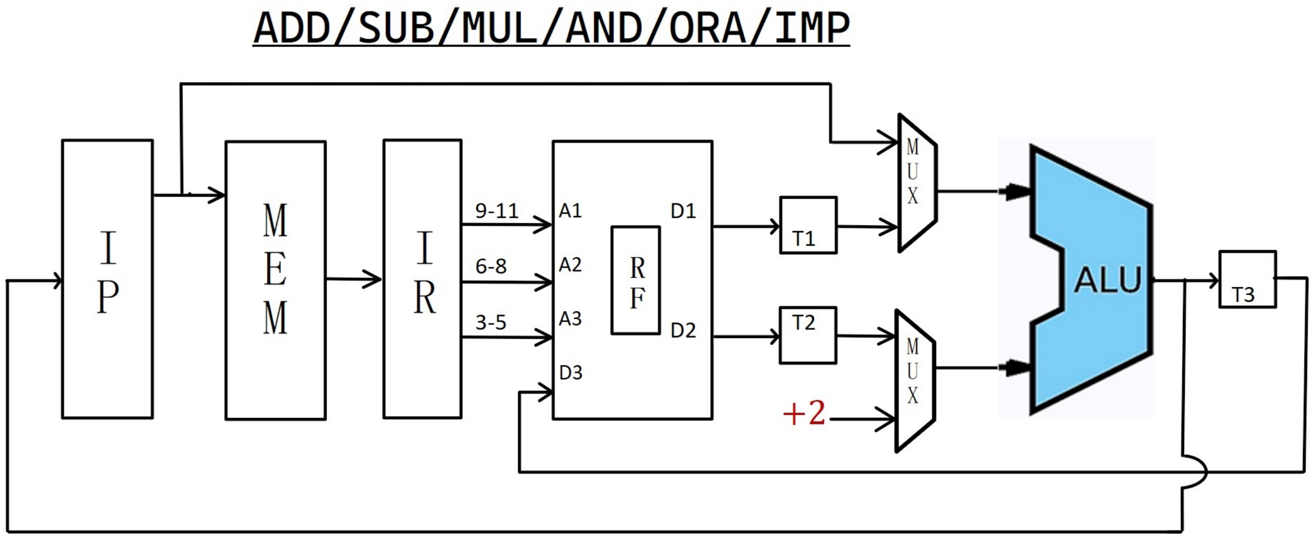
Merged States	Function executed during the merged state
M1	Fetch information and update instruction pointer (S1, S5, S9, S13, S17, S21, S25, S29, S32, S35, S40, S44, S48, S52)
M2	Understand and fetch operand (S2, S6, S10, S14, S18, S22, S26, S30, S33, S45, S49, S53)
M3	Execute operation (S3, S7, S11, S19, S23, S27)
M4	Update result (S4, S8, S12, S20, S24, S28)
M5	Execute addition operation (S15, S37, S42)
M6	Read memory (S38)
M7	Update register (S39)
M8	Store current instruction pointer (S50, S54)
M9	Store required value in specified location (S31)
M10	Store required value in specified location (S34)
M11	Read memory (S43)
M12	Compute Z (S46)
M13	If $z == 1$, then ip $=$ ip$+2+$(imm6)$\times 2$ (S47)
M14	Compute and update IP (S51)
M15	Update IP (S55)
M16	Update result (S16)
M17	Understand and fetch operand (S36, S41)

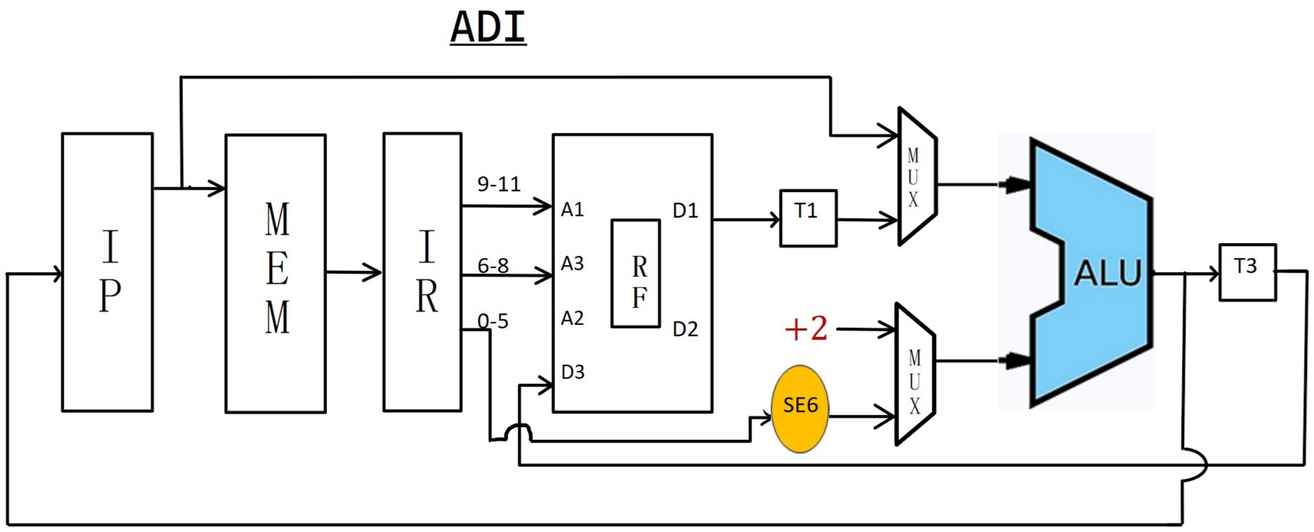
Table 2: Description Table

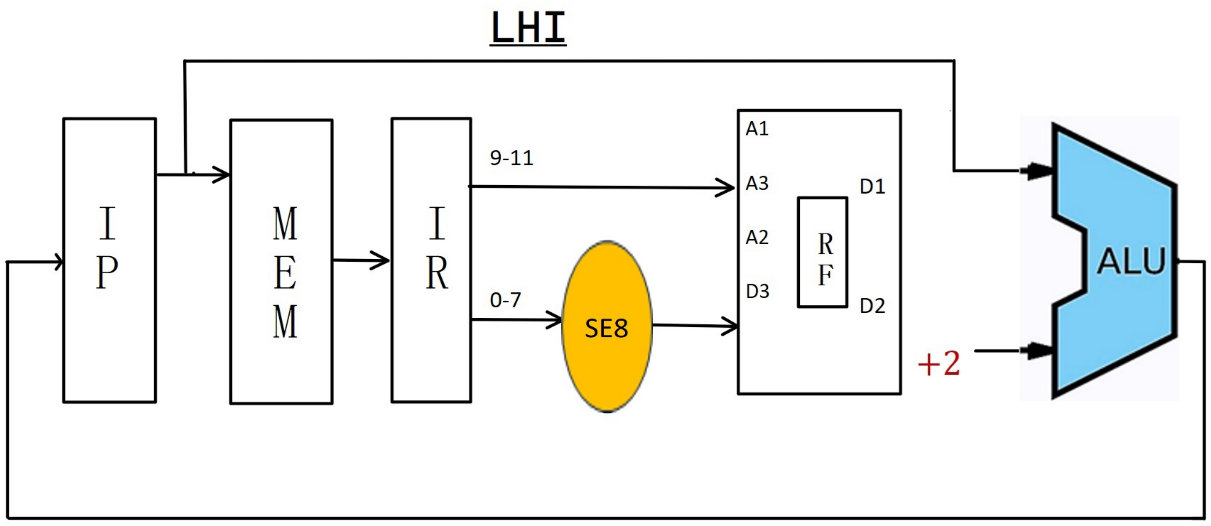
3.1 State Diagram



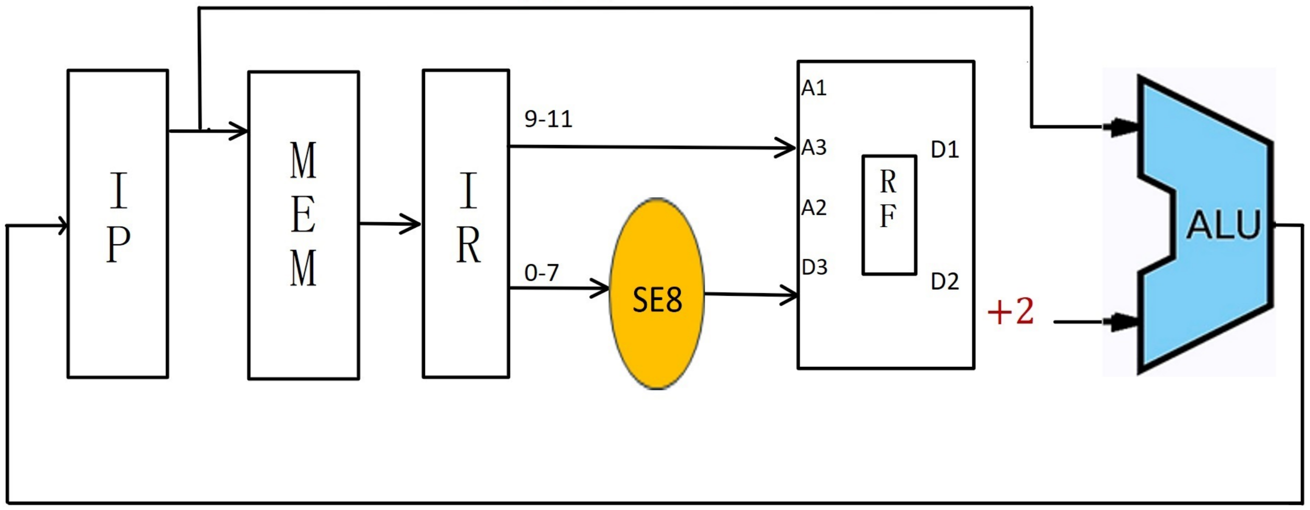
3.2 Components of IITB-CPU Design





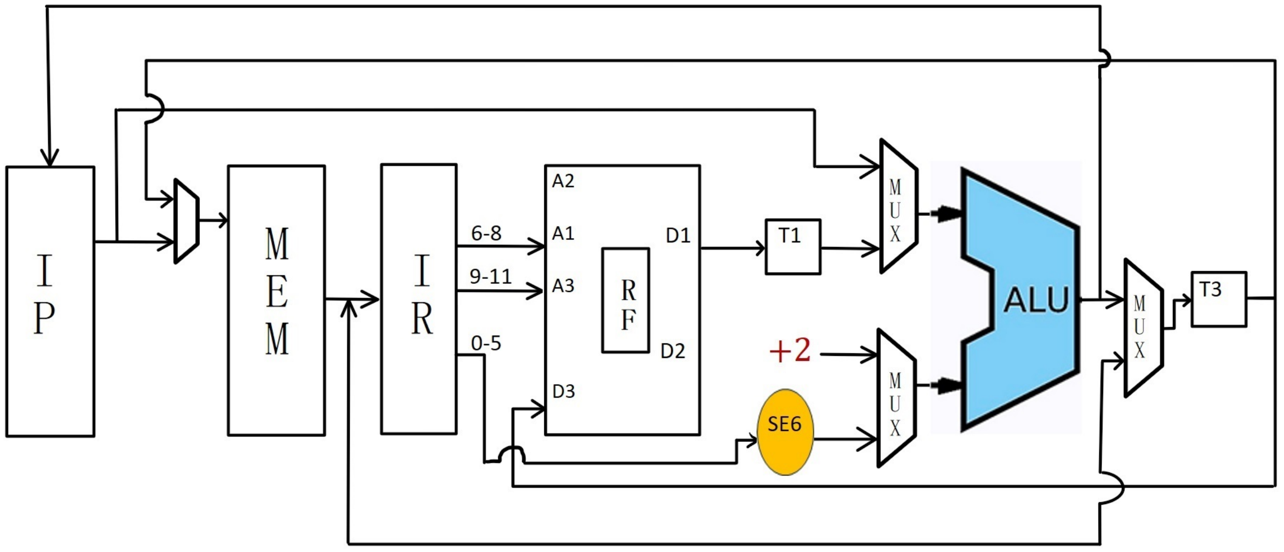


LLI



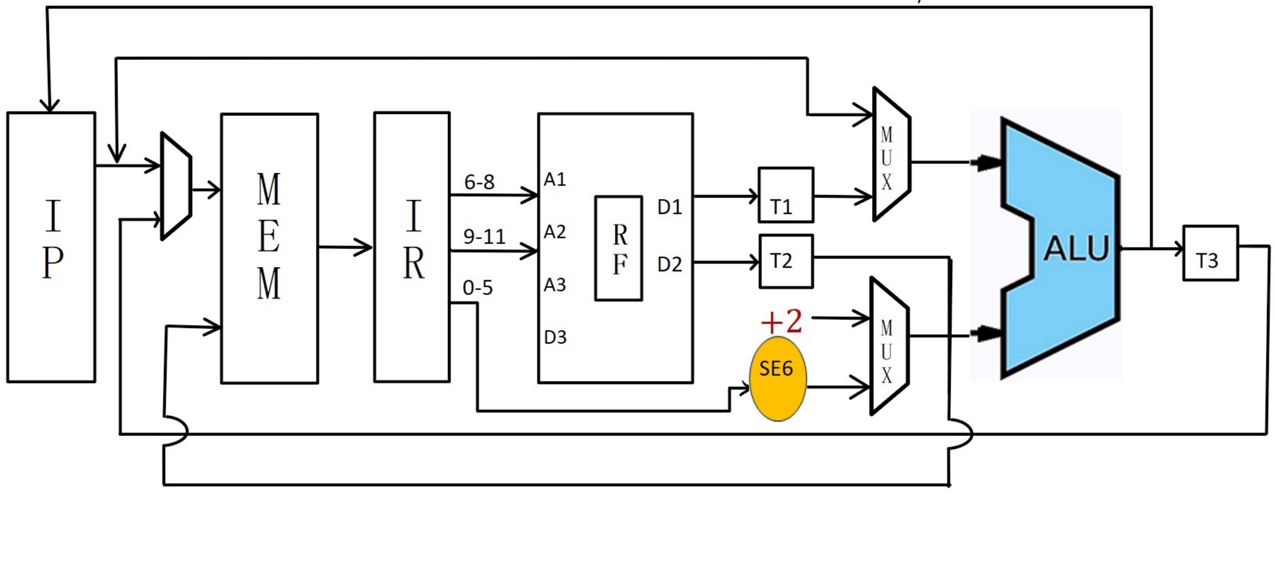
IP
MEM
IR
9-11
0-7
SE8
A1
A3
A2
D3
RF
D1
D2
+2
ALU

LW



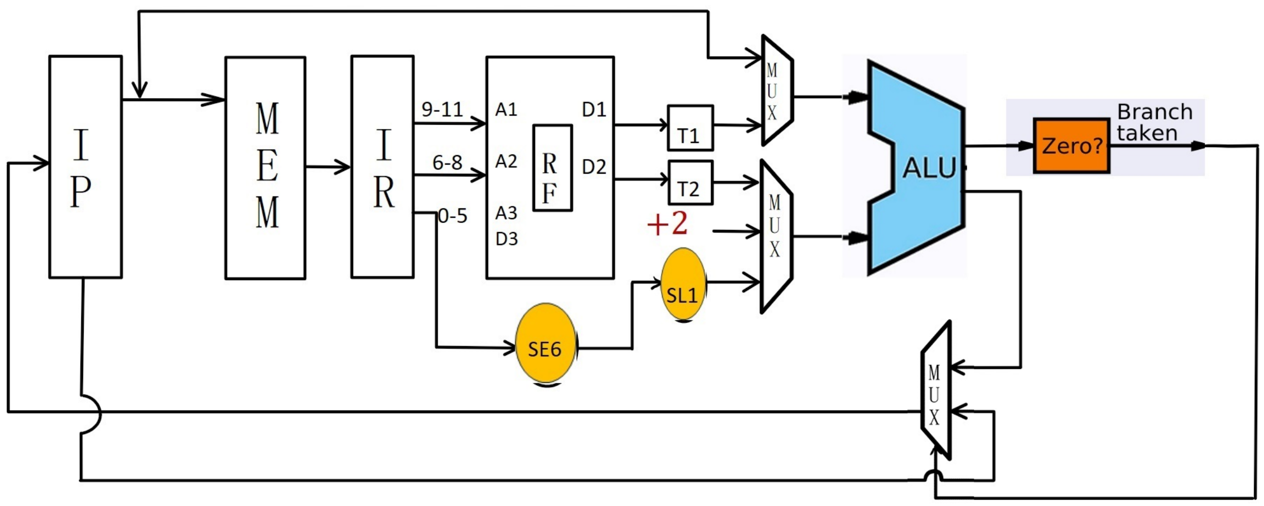
IP
MEM
IR
6-8
9-11
0-5
A2
A1
A3
D3
RF
D1
D2
T1
+2
SE6
MUX
MUX
ALU
MUX
T3

SW



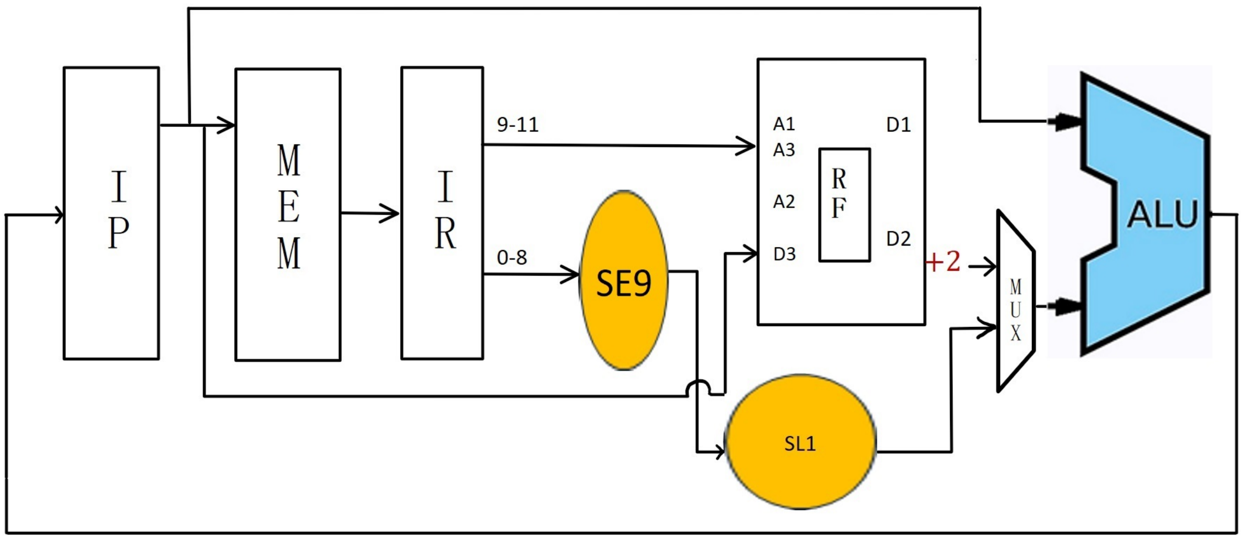
IP
MEM
IR
6-8
9-11
0-5
A1
A2
A3
D3
RF
D1
D2
T1
T2
+2
SE6
MUX
MUX
ALU
T3

BEQ



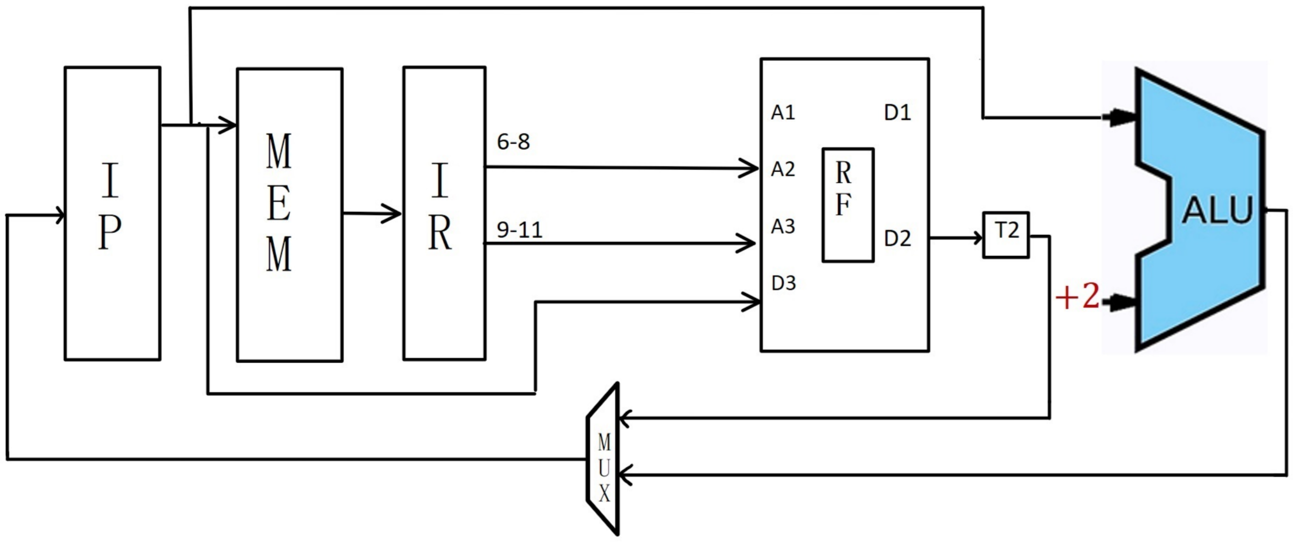
I P
M E M
I R
9-11
6-8
0-5
A1
A2
A3
D3
R F
D1
D2
T1
T2
+2
SE6
SL1
MUX
MUX
ALU
Zero?
Branch taken
MUX

JAL



I P
M E M
I R
9-11
0-8
SE9
A1
A3
A2
D3
R F
D1
D2
+2
MUX
ALU
SL1

JLR



I P
M E M
I R
6-8
9-11
A1
A2
A3
D3
R F
D1
D2
T2
+2
ALU
MUX

4 MUX Tables and Final Datapath of CPU

B1	**B0**	**OUTPUT SELECTED**
0	**0**	Do Nothing
0	**1**	Addition
1	**0**	Subtraction
1	**1**	IR 12_15

Table 3: Table of **MUX-1**

B3	**B2**	**OUTPUT SELECTED**
0	**0**	Do Nothing
0	**1**	IR 3_5
1	**0**	IR 6_8
1	**1**	IR 9_11

Table 4: Table of **MUX-2**

B6	**B5**	**B4**	OUTPUT SELECTED
0	**0**	**0**	Do Nothing
0	**0**	**1**	T3
0	**1**	**0**	IR 0_7 $\rightarrow$ SE8 Right
0	**1**	**1**	IR 0_7 $\rightarrow$ SE8 Left
1	**0**	**0**	IP
1	**0**	**1**	M8

Table 5: Table of **MUX-3**

B8	B7	OUTPUT SELECTED
0	0	Do Nothing
0	1	IP
1	0	T1
0	1	1

Table 6: Table of **MUX-4**

B11	B10	B9	OUTPUT SELECTED
0	0	0	Do Nothing
0	0	1	+2
0	1	0	T2
0	1	1	SE6
1	0	0	SE6 $\rightarrow$ LS1
1	0	1	SE9

Table 7: Table of **MUX-5**

B13	B12	OUTPUT SELECTED
0	0	Do Nothing
0	1	IP
1	0	T3

Table 8: Table of **MUX-6**

B15	B14	OUTPUT SELECTED
0	0	Do Nothing
0	1	ALU_C
1	0	Memory Data

Table 9: Table of **MUX-7**

B17	B16	OUTPUT SELECTED
0	0	Do Nothing
0	1	ALU_C
1	0	BEQ
1	1	T2

Table 10: Table of **MUX-8**

B18	**OUTPUT SELECTED**
0	IR 9_11
1	IR 6_8

Table 11: Table of **MUX-9**

B19	**OUTPUT SELECTED**
0	IR 6_8
1	IR 9_11

Table 12: Table of **MUX-10**

5 Control Signal table

X to	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
M1	0	0	0	1	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	1
M2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	1
M4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
M5	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0
M6	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
M8	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0
M9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0
M10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
M11	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
M12	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
M13	0	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0	0	1
M14	0	0	0	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	1
M15	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
M16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
M17	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13: Output process/Control Signal table for every state

In the implementation of a Finite State Machine (FSM), multiple Multiplexers (MUX) are employed to regulate signal flow and manage the execution of operations. The system comprises 17 MUX units, collectively featuring 20 select lines. Altering the control signal proves instrumental in facilitating state transitions.

6 Testing

6.1 Condensed code of CPU.vhdl

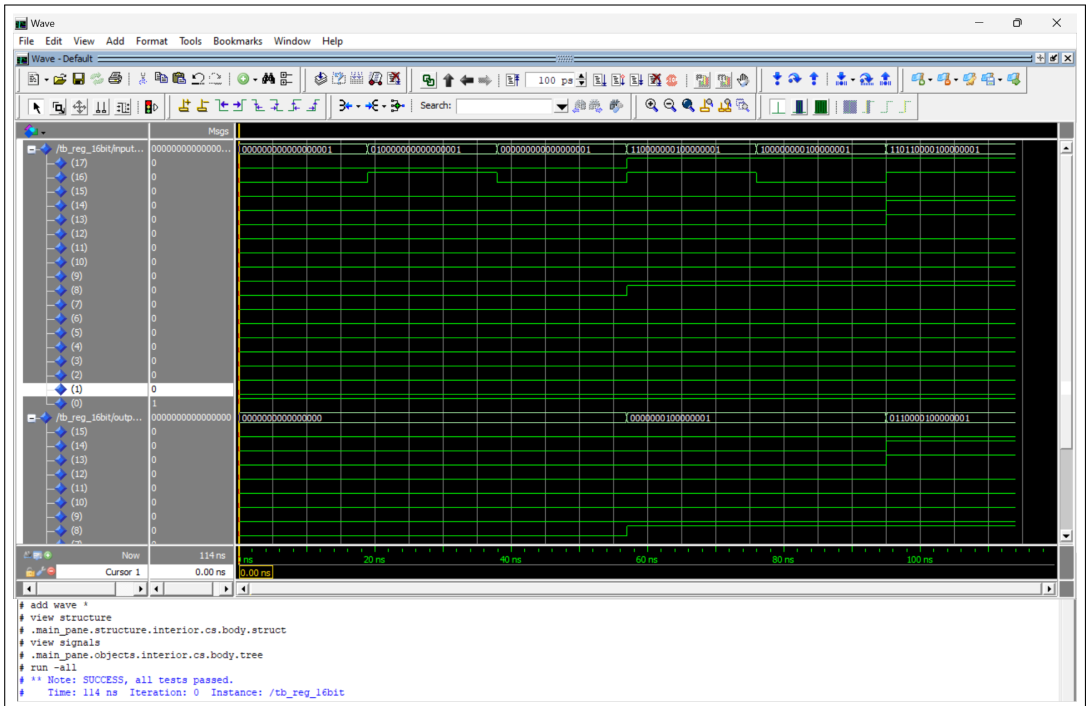
The code below is a condensed version of our VHDL description of CPU with only `clk` and `reset` as input signals.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity CPU is
    port (
        Clk, Reset: in std_logic
    );
end entity CPU;

architecture struct of CPU is

    type state is (rst, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17);
    -- Declaration of components Reg_File, MUX_8, MUX_4, MUX_2, Memory, ALU
    component Reg_File is
    {...}
    end component;
    component MUX_8 is
    {...}
    end component;
    component MUX_4 is
    {...}
    end component;
    component MUX_2 is
    {...}
    end component;
    component Memory is
    {...}
    end component;
    component ALU is
    {...}
    end component;

    signal M2, M9, M10: std_logic_vector(2 downto 0);
    signal M1: std_logic_vector(3 downto 0);
    signal Mem_W, Mem_R, Z_flag, T1_W, T2_W, IP_store: std_logic;
    signal B: std_logic_vector(19 downto 0);
    signal state_present, state_next: state := rst;

begin

    Program_Counter: Reg_16BIT port map (Clk => Clk,
                              Reset => Reset,
                              data_in => M8,
                              data_out => IP);

    MyMemory: Memory port map (Address => M6,
                    data_write => T2_data,
                    data_out => Mem_data,
                    clock => clk,
                    MeM_W => Mem_W,
                    MeM_R => Mem_R);

    Instruction_Register: Reg_16BIT port map (Reset => Reset,
                                Clk => clk,
                                data_in => Mem_data,
                                data_out => IR);

    Reg_File1 : Reg_File port map (Clk   => Clk,
                        Reset => Reset,
                  Address_Read1 => M9,
                  Address_Read2 => M10,
                  Address_Write => M2,
                    data_Write => M3,
                    data_Read1 => DataA,
                    data_Read2 => DataB
                      );

    Temporary_Register1: Reg_16BIT port map (Clk => Clk,
                                Reset => Reset,
                                data_in => DataA,
                                data_out => T1_data);

    Temporary_Register2: Reg_16BIT port map (Clk => Clk,
                                Reset => Reset,
                                data_in => DataB,
                                data_out => T2_data);

    Arithmetic: ALU port map (A => M4,
                    B => M5,
                  Oper => M1,
                    Z => Z_Flag,
                    C => ALU_data);

    Temporary_Register3: Reg_16BIT port map (Clk => Clk,
                                Reset => Reset,
                              data_in  => M7,
                              data_out => T3_data);

    BEQ1: for j in 0 to 15 generate
        MUXA: MUX_2 port map (S => Z_Flag,
                  I(1) => ALU_Data(j),
                  I(0) => IP(j),
                    Y => BEQ(j));
    end generate BEQ1;

    -- The following processes help integrate the FSM and Datapath for our CPU!
    clock_proc: process(clk, reset)
    {...}
    end process state_transition_proc;

    state_transition_proc: process(state_present, IR)
    {...}
    end process state_transition_proc;

    output_proc: process(state_present, Mem_W, Mem_R)
    {...}
    end process output_proc;

    MUX1: process (B, M1, IR)
    {...}
    end process MUX1;

    MUX2: process (B, M2, IR, IP_store)
    {...}
    end process MUX2;

    MUX3: process (B, M3, M8, IR, T3_data, IP)
    {...}
    end process MUX3;

    MUX4: process (B, M4, IP, T1_data)
    {...}
    end process MUX4;

    MUX5: process (B, M5, T2_data, IR)
    {...}
    end process MUX5;

    MUX6: process (B, M6, IP, T3_data)
    {...}
    end process MUX6;

    MUX7: process (B, M7, ALU_data, Mem_data)
    {...}
    end process MUX7;

    MUX8: process (B, M8, ALU_data, BEQ, T2_data)
    {...}
    end process MUX8;

    MUX9: process (B, M9, IR, T1_W)
    {...}
    end process MUX9;

    MUX10: process (B, M10, IR, T2_W)
    {...}
    end process MUX10;
```

The testbench for the CPU `tb_CPU` generates a clock signal (clk) and a reset signal (reset) with a period of 20ns. To facilitate testing with this particular testbench, predefined instructions have been embedded into the Memory storage and Register Files.

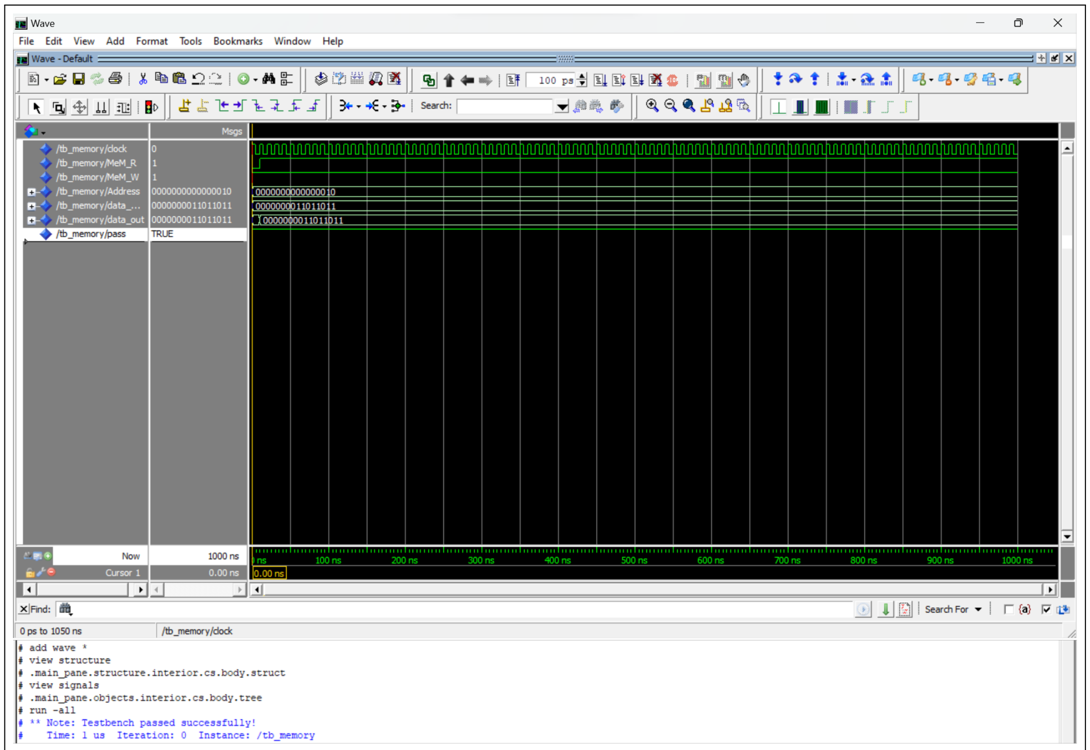
6.2 Testing and Simulation Results

The following simulations were obtained when the Memory and Programmer registers were hard-coded with different instructions to test the functioning of our CPU by loading the corresponding Testbenches which were built separately. The following are the simulation results:


```
/tb_reg_16bit/input...   000000000000000...
  (17)  0
  (16)  0
  (15)  0
  (14)  0
  (13)  0
  (12)  0
  (11)  0
  (10)  0
  (9)   0
  (8)   0
  (7)   0
  (6)   0
  (5)   0
  (4)   0
  (3)   0
  (2)   0
  (1)   0
  (0)   1
/tb_reg_16bit/outp...    0000000000000000
  (15)  0
  (14)  0
  (13)  0
  (12)  0
  (11)  0
  (10)  0
  (9)   0
  (8)   0

000000000000000001 | 010000000000000001 | 000000000000000001 | 110000000100000001 | 100000000100000001 | 110110000100000001
0000000000000000 | 0000000100000001 | 0110000100000001

Now      114 ns
Cursor 1 0.00 ns
```

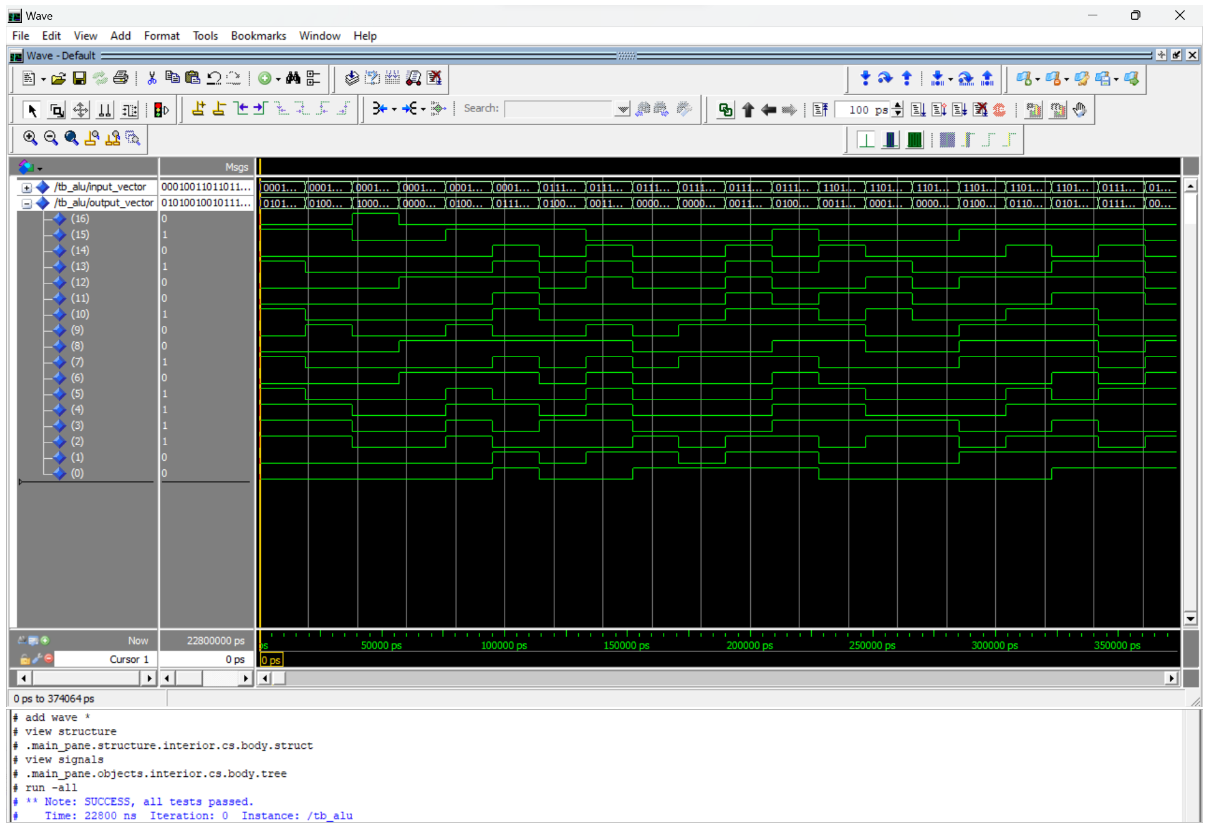
```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 114 ns  Iteration: 0  Instance: /tb_reg_16bit
```

Figure 1: Simulation showing the perfect functioning of a 16-bit Register


```
/tb_memory/clock      0
/tb_memory/MeM_R      1
/tb_memory/MeM_W      1
/tb_memory/Address    0000000000000010
/tb_memory/data_...   0000000011011011
/tb_memory/data_out   0000000011011011
/tb_memory/pass       TRUE

Now      1000 ns
Cursor 1 0.00 ns
0 ps to 1050 ns    /tb_memory/clock
```

```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: Testbench passed successfully!
#    Time: 1 us  Iteration: 0  Instance: /tb_memory
```

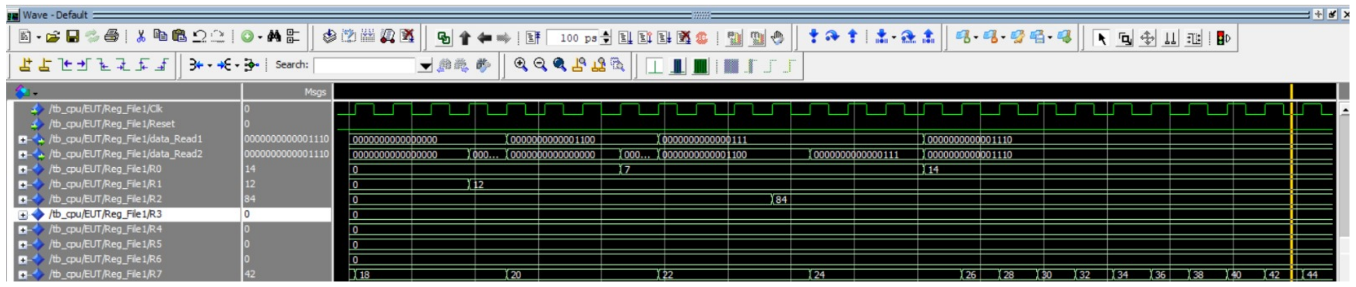
Figure 2: Simulation showing the perfect functioning of Memory



```
/tb_alu/input_vector    00010011011011...
/tb_alu/output_vector   01010010010111...
  (16)  0
  (15)  1
  (14)  0
  (13)  1
  (12)  0
  (11)  0
  (10)  1
  (9)   0
  (8)   0
  (7)   1
  (6)   0
  (5)   1
  (4)   1
  (3)   1
  (2)   1
  (1)   0
  (0)   0
Now       22800000 ps
Cursor 1  0 ps
0 ps to 374064 ps
```

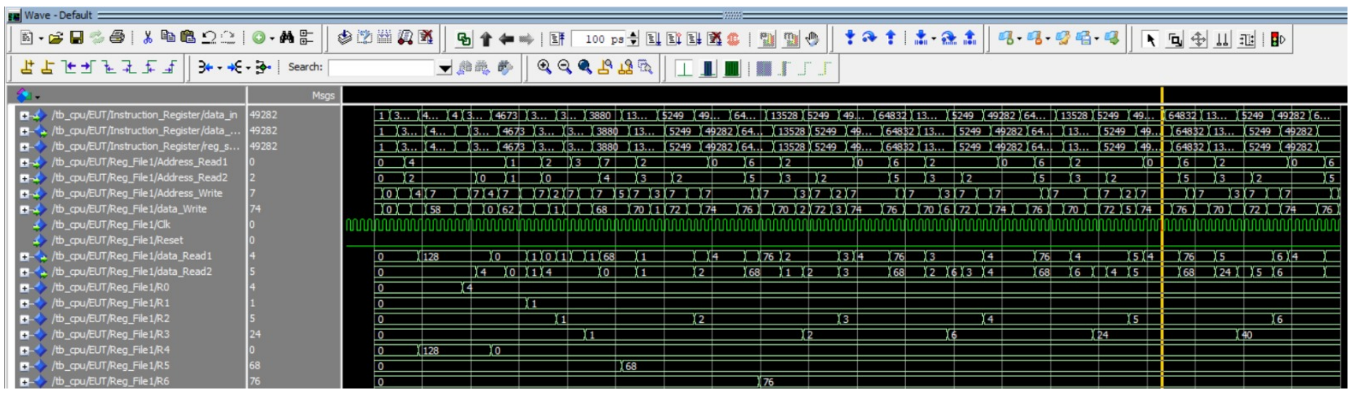
```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 22800 ns  Iteration: 0  Instance: /tb_alu
```

Figure 3: Simulation showing the perfect functioning of ALU that passed its generic Testbench



```
/tb_cpu/EUT/Reg_File1/Clk          0
/tb_cpu/EUT/Reg_File1/Reset        0
/tb_cpu/EUT/Reg_File1/data_Read1   0000000000001110
/tb_cpu/EUT/Reg_File1/data_Read2   0000000000001110
/tb_cpu/EUT/Reg_File1/R0           14
/tb_cpu/EUT/Reg_File1/R1           12
/tb_cpu/EUT/Reg_File1/R2           84
/tb_cpu/EUT/Reg_File1/R3           0
/tb_cpu/EUT/Reg_File1/R4           0
/tb_cpu/EUT/Reg_File1/R5           0
/tb_cpu/EUT/Reg_File1/R6           0
/tb_cpu/EUT/Reg_File1/R7           42
```

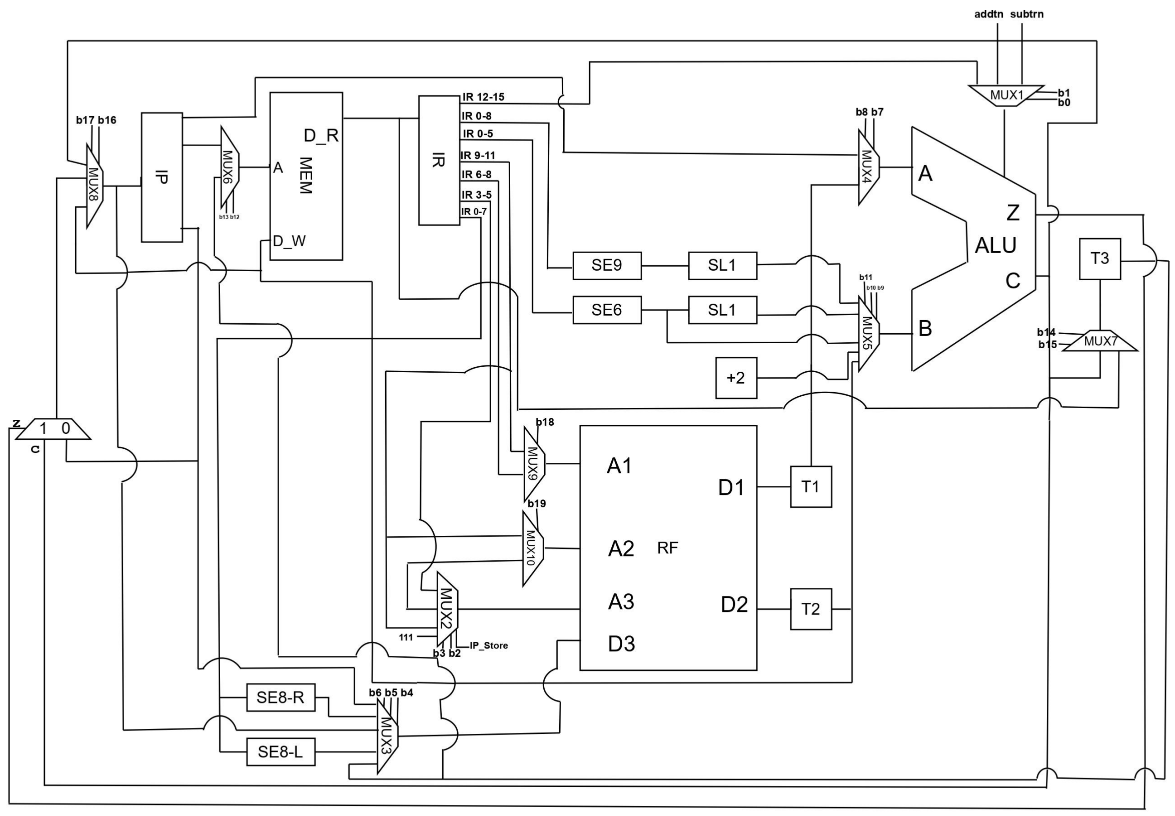
Figure 4: Test Case 1: `ADI-ADI-MUL`



	Msgs
/tb_cpu/EUT/Instruction_Register/data_in	49282
/tb_cpu/EUT/Instruction_Register/data_...	49282
/tb_cpu/EUT/Instruction_Register/reg_s...	49282
/tb_cpu/EUT/Reg_File1/Address_Read1	0
/tb_cpu/EUT/Reg_File1/Address_Read2	2
/tb_cpu/EUT/Reg_File1/Address_Write	7
/tb_cpu/EUT/Reg_File1/data_Write	74
/tb_cpu/EUT/Reg_File1/Clk	0
/tb_cpu/EUT/Reg_File1/Reset	0
/tb_cpu/EUT/Reg_File1/data_Read1	4
/tb_cpu/EUT/Reg_File1/data_Read2	5
/tb_cpu/EUT/Reg_File1/R0	4
/tb_cpu/EUT/Reg_File1/R1	1
/tb_cpu/EUT/Reg_File1/R2	5
/tb_cpu/EUT/Reg_File1/R3	24
/tb_cpu/EUT/Reg_File1/R4	0
/tb_cpu/EUT/Reg_File1/R5	68
/tb_cpu/EUT/Reg_File1/R6	76

Figure 5: Test Case 2: `LW-LLI-ADI-ADD-MUL-BEQ-JLR-SW`

7 Datapath



8 Conclusion

1. We have synthesized a General Fabric called CPU, which consists of a processing unit and a Memory. It has been created by orchestrating an Datapath which is controlled by a Controller. The Input to the CPU is a program and data (Depicted as a sequence of instructions hard coded into our memory).

2. The CPU in use interfaces with a randomly designated memory location, providing the flexibility to select any memory address for the input and output processes. This design allows for the writing of instructions and the observation of corresponding data outputs.

3. Our Datapath architecture consists of various components such as Registers, a Register File, Memory, Arithmetic Logic Unit (ALU) and Multiplexers. Our Control Logic maps out a Finite State Machine (FSM) which changes state every clock pulse based on the Instruction that has to be performed (Decoder Implementation). It is a **MULTI-CYCLE IMPLEMENTATION** to reduce the amount of hardware used.

4. Our Instruction Set Architecture (ISA) consists of a set of 14 instructions which the processor/computer architecture supports. This is a smaller set of simple instructions and hence is called **Reduced Instruction Set Computing (RISC)**. The implementation of this ISA has been completely tested and verified via RTL simulations and signal analysis.

5. This Instruction set comprised of:
   a) Arithmetic and Logical Instructions (`ADD,SUB,MUL,ADI,AND,ORA,IMP`)
   b) Data Transfer Instructions (`LHI,LLI,LW,SW`)
   c) Control Transfer Instructions (`BEQ,JAL,JLR`)

6. The specific details of the logic inside the processes (`state_transition_proc`, `output_proc`, and multiplexer processes) determiness the exact behavior and functionality of the CPU.