EE 224 Course Project

IITB-CPU

TEAM MEMBERS

Kartik UC

Natasha Ramineni

Monika Khond

Soham Nivargi

30th November, 2022

OVERVIEW

The purpose of this experiment was to design a 16-Bit CPU capable of performing a specified set of tasks based on given instructions.

This report consists of an overview of the different components that were used to construct the CPU as well as its working.

WORKING PRINCIPLE AND COMPONENTS

The CPU works on the principle of the fetch execute cycle. Here different parts of the fetch execute cycle are grouped into states which are then carried out by creating an FSM using those states.Using an FSM model helps in reducing the number of computations required hence increasing speed and also reducing chance of bugs and errors.

The components of the CPU are given below:

1) **Register File**
    - This component stores 8 16-bit registers and can read data from registers and write values into them. It is used for storage of data when the CPU carries out a process.
2) **4 16-Bit registers**
    - These single 16- bit registers are used to temporarily store values and send them to one or more different components at a time.
3) **Memory Bank**
    - This component stores the 16-bit instruction values in 2^{16} addresses which are used to carry out different tasks in the CPU.
4) **16-Bit Instruction Register**
    - This single 16-bit register is used for taking the data from the memory and distributing them to specific components according to the instruction to be performed which is determined by the data.

5) **ALU**
   - This component is used to carry out ADD and NAND operations, with carry and zero flags as outputs. The carry flag is only enabled when the enable input is ‘1’.

6) **1-Bit Signals**
   - These are used to store the carry flag and zero flag given by the ALU output as well as the ‘loopcondn’ output and ‘eq’ output from the Priority Encoder and Comparator respectively.

7) **10-Bit Sign Extender**
   - This component is used to convert 6-bit values from the instruction register to 16 bit values by adding 0’s or 1’s on the left end based on the sign so that they can be used in other components.

8) **7-Bit Sign Extender**
   - This component is used to convert 9-bit values from the instruction register to 16 bit values by adding 0’s or 1’s on the left end based on sign so that they can be used in other components.

9) **7-Bit Shifter**
   - This component shifts the values of a given input to the left by 7 bits and adds 0’s to the 9 least significant bits.

10) **7-Bit Sequence Extender**
    - This component simply adds seven 0’s to the left end of the given input.

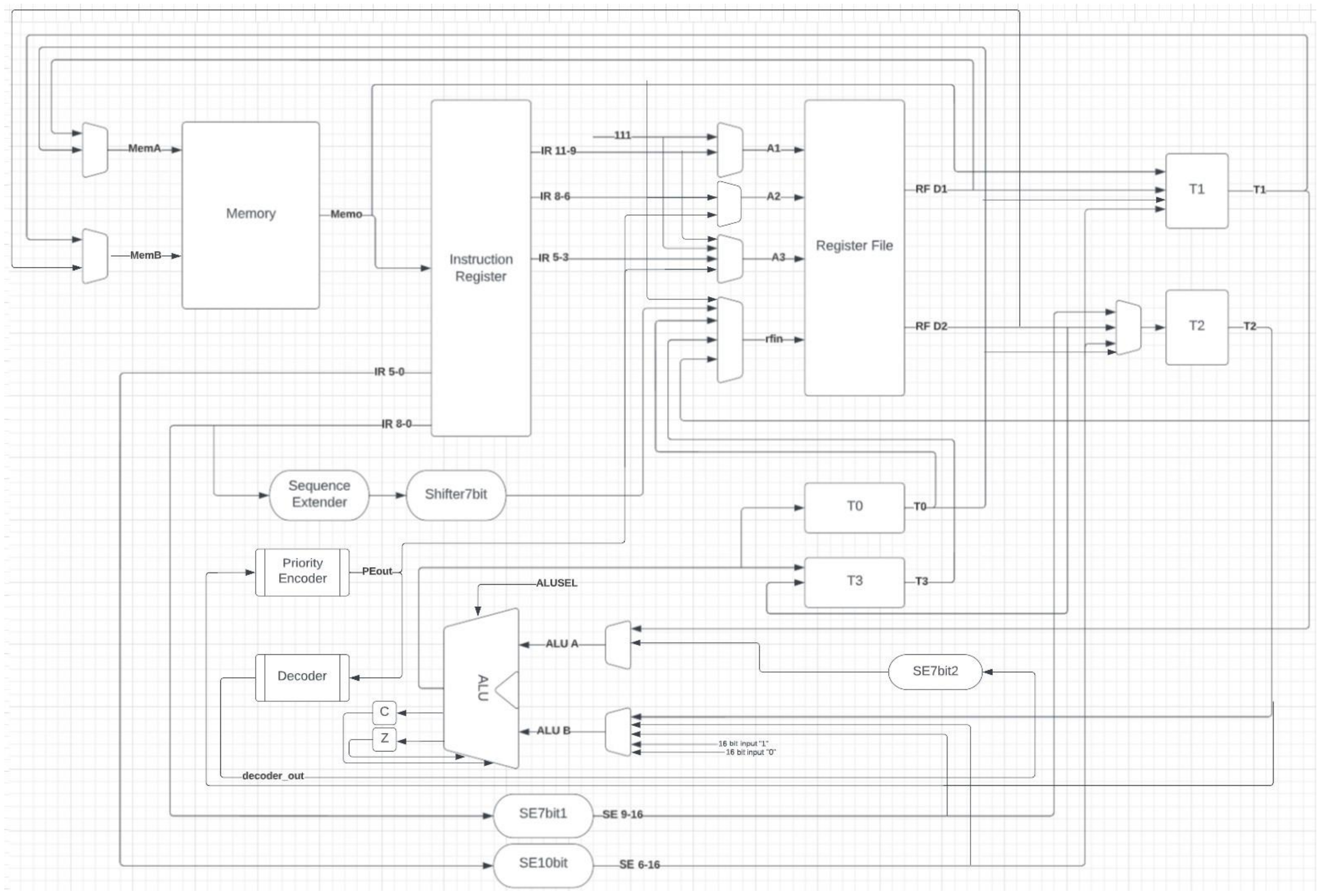
11) **8 to 3 Priority Encoder**
    - This component is used to encode 8-bit input into 3-bit output according to the highest significant bit that is set as ‘1’. Another output of the encoder is the loopcondn output which is 0 if the input of the encoder is “00000000”, else it is 1.

12) **3 to 9 Decoder**
    - This component is used to convert the 3-bit input to 8-bits and the most significant bit is set as 0 to give the final 9-bit output.

13) **Comparator**
    - This component takes 2 inputs and outputs 1 if they are equal, else it outputs 0.

DATAPATH

Following is a diagram which shows how different components of the CPU are connected :



FINAL STATES USED

S0

Initial State = NULL

S1

A1 → 111
RFD1 → alu_A, in_memA, R7_curr
mem_out →IR
+1 → alu_B
alu_out → T3

S2

$IR_{11\text{-}9}$ → A1
RFD1 → T1
$IR_{8\text{-}6}$ → A2
RFD2 → T2

S3

T1 → alu_A
T2 → alu_B
alu_out → T0

S4

$IR_{5\text{-}3}$ → A3
T0 → RFin

S5

T3 → RFin
111 → A3

S6

$IR_{11\text{-}9}$ → A1
RFD1 → T1
$IR_{5\text{-}0}$ → SE(6-16) → T2

S7

$IR_{8\text{-}6}$ → A3
T0 → RFin

S8

$IR_{8\text{-}0}$ → seq.ex → 7 bit shifter → RFin
$IR_{11\text{-}9}$ → A3

S9

R7curr → RFin
$IR_{11\text{-}9}$ → A3
$IR_{8\text{-}6}$ → A2
RFD2 → T3

S10

$IR_{8\text{-}6}$ → A2
RFD2 → T2
$IR_{5\text{-}0}$ → $SE_{6\text{-}16}$ → T1

S11

T0 → in_memA
mem_out → T1

S12

T1 → RFin, alu_A
0 → alu_B
$IR_{11\text{-}9}$ → A3

S13

T1 → in_memD
T0 → in_memA

S14

$IR_{8\text{-}0}$ → $SE_{9\text{-}16}$ → T2
$IR_{11\text{-}9}$ → A1
RFD1 → T1

S15

$T2_{0\text{-}7}$ → PE(7-0)
PE_out → A3
T1 → in_memA, alu_A
+1 → alu_B
alu_out → T0
mem_0 → RFin

S16

T0 → T1
PE_out → deocoder_in
decoder_out → SE$_{(9-16)}$ → alu_A
T2 → alu_B
alu_out → T0

S17

T0 → T2

S18

R7curr → alu_A
IR$_{5-0}$ → SE$_{(6-16)}$ → alu_B
alu_out → T3

S19

R7curr → alu_A, RFin
IR$_{11-9}$ → A3
IR$_{8-0}$ → SE$_{(9-16)}$ → alu_B
alu_out → T3

S20

T2$_{0-7}$ → PE$_{(7-0)}$
PE_out → A2
RFD2 → in_memD
T1 → in_memA, alu_A
+1 → alu_B
alu_out → T0

S21

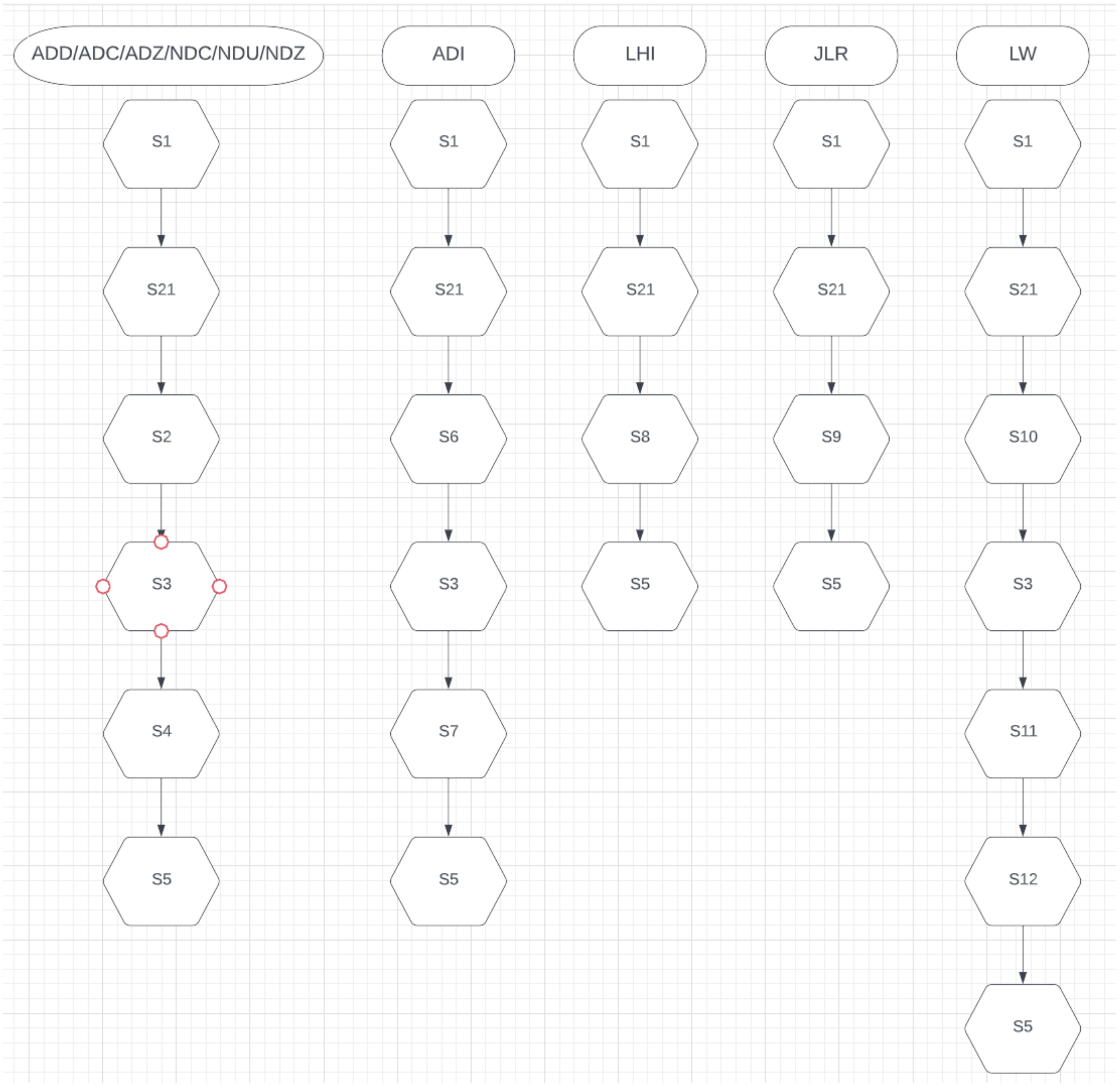
Dummy state

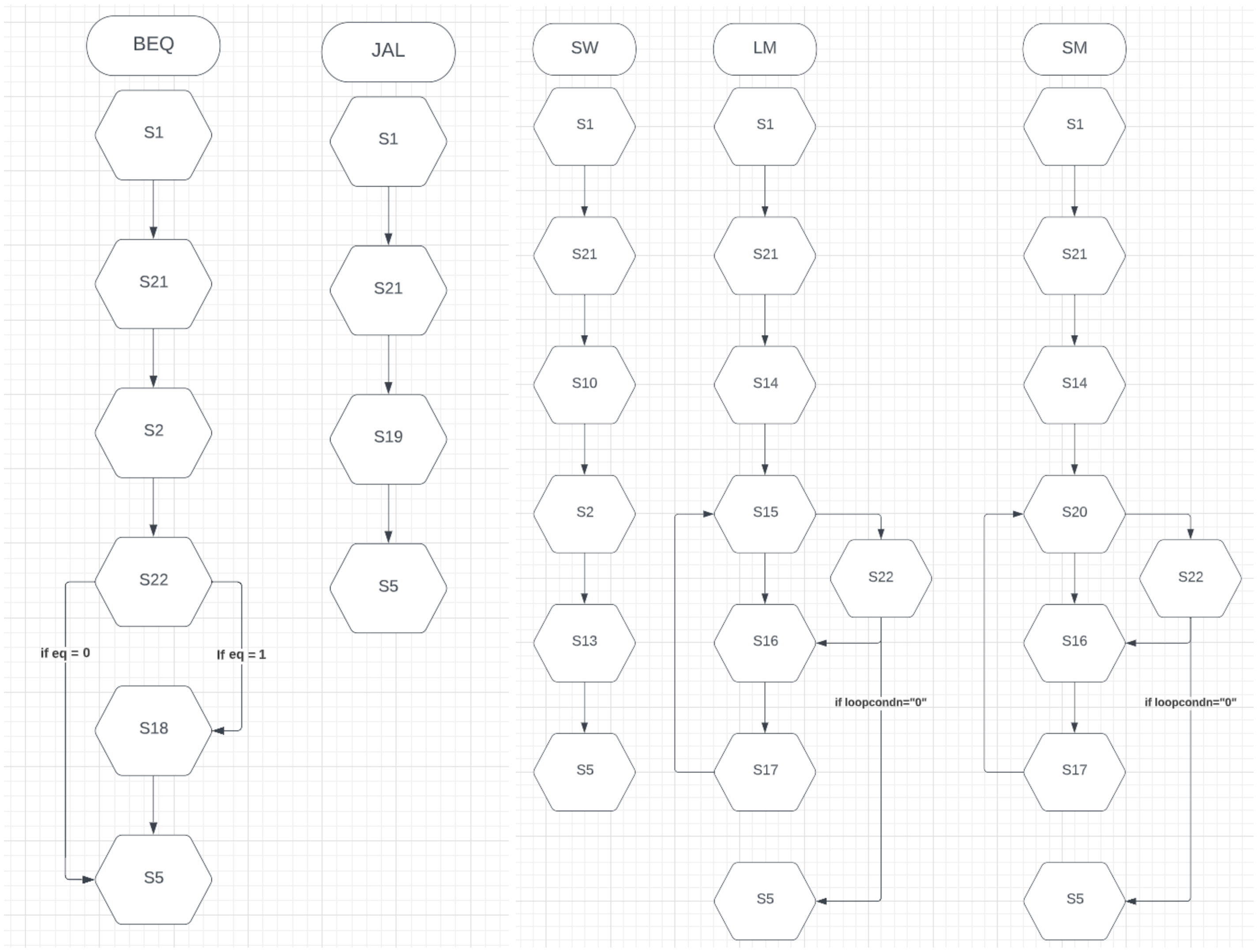
S22

Dummy state

STATE DIAGRAMS

The following state diagrams describe how different state transitions are carried out to perform different instructions:





CODE-WRITING PROCESS

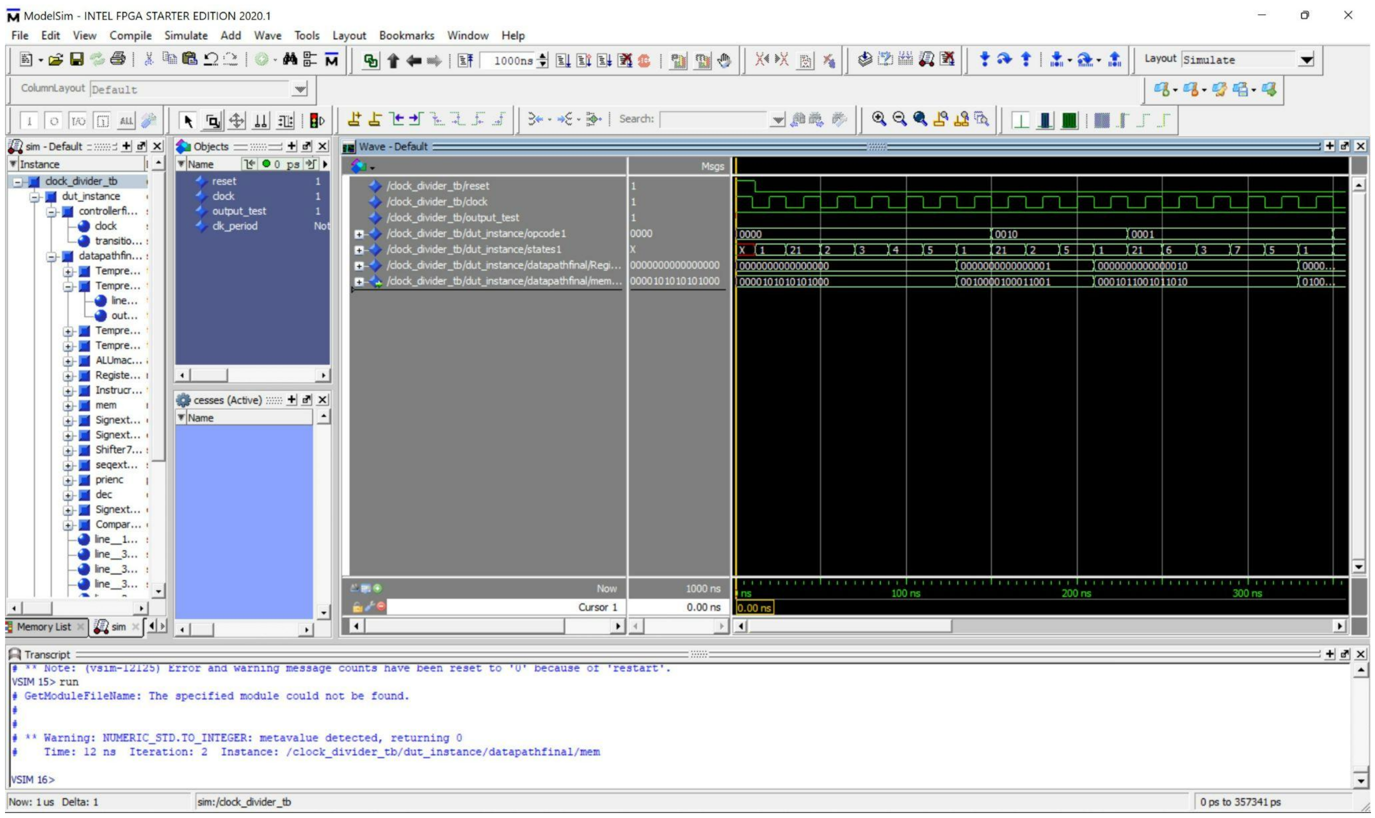
The states written above were written in structure.vhdl and the state diagrams were referred to for statetransition.vhdl

The two dummy states (s21 and s22) ensure that the opcode, loopcondn and eq signals don't use their previous inputs to determine the next state.

In each state, the write enable and read enable signals were set as per whichever component was being used.

We made use of a testbench that had been provided to us in our EE-214 lab problems to generate the clock input. So no tracefile or DUT was required for going ahead with the RTL simulation.

CONCLUSION


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Msgs
/clock_divider_tb/reset 1
/clock_divider_tb/clock 1
/clock_divider_tb/output_test 1
/clock_divider_tb/dut_instance/opcode1 0000
/clock_divider_tb/dut_instance/states1 X
/clock_divider_tb/dut_instance/datapathfinal/Regi... 0000000000000000
/clock_divider_tb/dut_instance/datapathfinal/mem... 0000101010101000
Now 1000 ns
Cursor 1 0.00 ns
100 ns
200 ns
300 ns
Transcript

```
# ** Note: (vsim-12125) Error and warning message counts have been reset to '0' because of 'restart'.
VSIM 15> run
# GetModuleFileName: The specified module could not be found.
#
#
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 12 ns  Iteration: 2  Instance: /clock_divider_tb/dut_instance/datapathfinal/mem

VSIM 16>
```

```
Now: 1 us  Delta: 1    sim:/clock_divider_tb    0 ps to 357341 ps
```

The RTL simulation shows all the signal waveforms, so we can check whether the states are going as per the instructions. In our memory.vhdl file, we added a few instructions to check whether the cpu was working properly, and it was. We can add other instructions in the memory addresses as well and the cpu will work accordingly.

And that's about it. This was a fun learning process to go through once we figured out the logistics of how to go about the project. We got an application-based experience and understanding of how basic hardware programming works and what are the possible problems one could face while doing the same as well as how to overcome them. Overall this project was a great way to learn something new and hopefully we can make use of this CPU for more advanced projects later on.