DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING

UNIVERSITY OF MORATUWA

EN3021 - Digital System Design

Non-pipelined Single Stage CPU Design

Individual Project

RATHNAYAKE R.N.P. 200537F

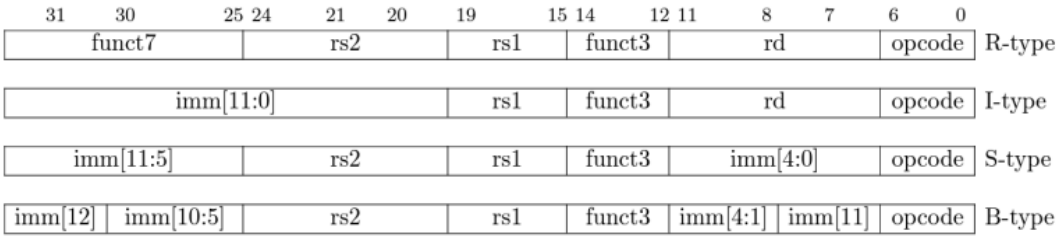
OCTOBER 16, 2023

1. Introduction

This project aims to design and implement a 32-bit non-pipelined RISC-V processor based on the RV32I architecture using microprogramming and a 3-bus architecture. This processor executes instructions from multiple classes, specifically R and I, Load and Store, and SB. The implementation follows a systematic process, which includes the stages of instruction fetch, decode, and execute.

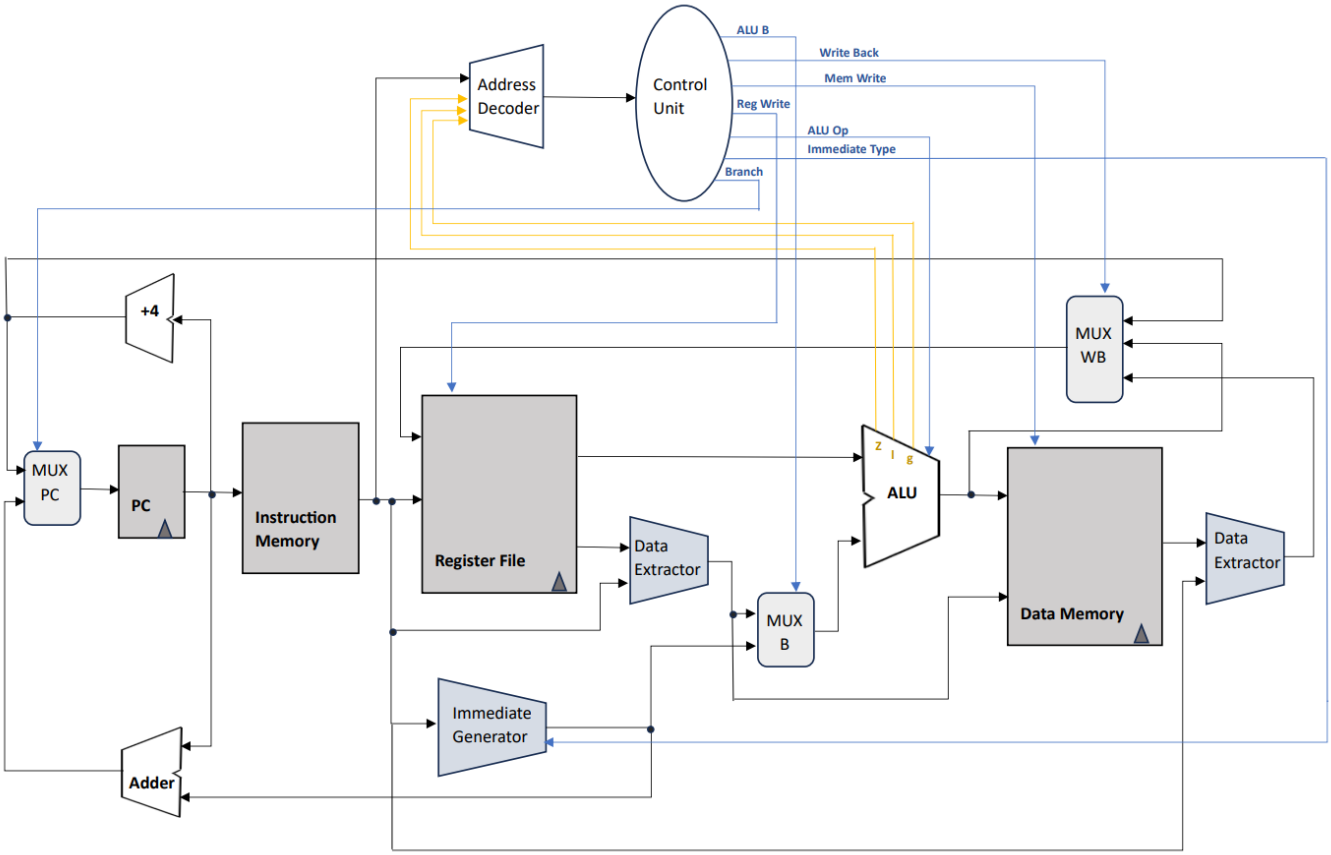
2. Instruction Set Architecture

This design supports RISC-V instructions under the above-mentioned classes. Each instruction has a length of 32 bits. The following diagrams show the instruction formats that were considered to implement the instructions and all the instructions.



Type	Instruction	Description
R-Type	add	rd = rs1 + rs2
	sub	rd = rs1 - rs2
	xor	rd = rs1 ^ rs2
	or	rd = rs1 | rs2
	and	rd = rs1 & rs2
	sll	rd = rs1 << rs2
	srl	rd = rs1 >> rs2
	sra	rd = rs1 >> rs2
	slt	rd = (signed(rs1) < signed(rs2)) ? 1 : 0
	sltu	rd = (signed(rs1) < unsigned(rs2)) ? 1 : 0
I-Type	addi	rd = rs1 + signed(imm)
	slti	rd = (signed(rs1) < signed(imm)) ? 1 : 0
	stliu	rd = (unsigned(rs1) < unsigned(rs2)) ? 1 : 0
	xori	rd = rs1 ^ signed(imm)
	ori	rd = rs1 | signed(imm)
	andi	rd = rs1 & signed(imm)
	slli	rd = rs1 << unsigned(imm)
	srli	rd = rs1>> usigned(imm)
	srai	rd = signed(rs1) >> unsigned(rs2)
	lb	rd = signed(Mem_byte[rs1 + imm])
	lh	rd = signed(Mem_hw[rs1 + imm])
	lw	rd = Mem[rs1 + imm]
	lbu	rd = unsigned(Mem_byte[rs1 + imm])
	lhu	rd = unsigned(Mem_hw[rs1 + imm])
	jalr	rd = pc + 4, pc = rs1 + imm
S-Type	sb	Mem_byte[rs1 + imm] = rs2
	sh	Mem_hw[rs1 + imm] = rs2
	sw	Mem[rs1 + imm] = rs2
SB-Type	beq	pc = pc + (rs1 == rs2) ? Imm : 4
	bne	pc = pc + (rs1 != rs2) ? Imm : 4
	blt	pc = pc + (rs1 < rs2) ? Imm : 4
	bge	pc = pc + (rs1 >= rs2) ? Imm : 4
	bltu	pc = pc + (rs1 < rs2) ? Imm : 4
	bgeu	pc = pc + (rs1 >= rs2) ? Imm : 4

3. Data Path



4. Modules

a. Program Counter

Program counter points to the address of the next instruction of the instruction memory to be executed. Here the instruction memory is implemented as word addressable. Therefore, after execution program counter will increment by one. If there is a branch the input to the program counter comes from an adder. These two different inputs are connected through a mux to the program counter.

b. Instruction Memory

A word-addressable instruction memory is implemented. The instruction memory contains 256 addresses which can store 256 32-bit length instructions. Each instruction has a 7-bit length address. Instruction memory outputs the instruction for a given address.

c. Register File

The register file consists of 32 32-bit registers. When the instruction is given to the register file it decodes the instruction and gets the relevant read and write register addresses. Data can be written to a given register at the positive edge of the clock and when the register write is enabled. If reset all register values will be flushed and be zero.

d. Data Memory

Data Memory has an 8192-bit space. In other words, it has 256 memory locations each 32 bits length. Therefore, a memory address is 8 bits, and it is word addressable. Data can be written to a given location at the positive edge of the clock and when the memory write is enabled.

e. ALU

The arithmetic and logic unit of the processor can do 12 operations. The ALU outputs three flags that are helpful in determining whether there is a branch. They are zero flag, branch less than flag and branch greater than or equal flag.

1	Add	0000
2	Subtraction	0001
3	Shift left	0010
4	Shift right	0011
5	Shift right arithmetic	0100
6	Bitwise AND	0101
7	Bitwise OR	0110
8	Bitwise XOR	0111
9	Signed Branch	1000
10	Unsigned Branch	1001
11	Set less than	1010
12	Set less than unsigned	1011

f. Data Extractor

In the RV321 implementation, all the data bus widths are 32-bit. Therefore, when there is a load and store instruction which requires to get a byte or a halfword from a word the data should be manipulated to fill the whole 32-bit length. For that purpose, a data extractor is used. It will fill the rest 24 or 16 bits according to the instructions considering the sign extensions. In this implementation, 2 data extractors were used. One is implemented for Load instructions. In there the data coming from the memory is first fed to the data extractor. Another one is for the Store instructions, and it gets the data coming from the register.

g. Immediate Generator

In I type, S type and SB type the immediate values are encoded in different places in the instruction as parts. To connect these parts and make the whole immediate value according to the instruction an immediate generator is used. An immediate select control signal is used to decide the different instruction format which includes immediate values differently and combine them.

h. Adder

There are 2 adders in the circuit. One is for incrementing the program counter. Another one is to use when there is a branch instruction.

i. Multiplexer

One 2-1 multiplexer is used to choose the program counter input whether the next address comes from a branch or the normal way. Another one is to select the second ALU input directly from the register or an immediate value. A 3-1 multiplexer is used to select from where the data that should be written to the register comes from: from memory, directly from alu output or the next address.

j. **Address Decoder**
The address decoder takes the instruction, and alu flags as inputs and maps them to relevant the control memory address which stores the control signals for that instruction.

k. **Control Memory**
The control memory takes the memory address from the address decoder and outputs the stored control signals in that memory address.

5. Control Logic

The control memory outputs the following control signals according to the instruction.

AluB determines whether the source of the ALU input B is from read register 2 or immediate data.

Write Back signal specifies whether the data written to the write register is from memory, from alu output or the next address value.

Memory Write Enable indicates the data should be written to the data memory.

Register Write Enable indicates the data should be written to the register.

ALU Operation is a 4-bit signal that determines the ALU operation to do.

Immediate Select is a 3-bit signal that determines how the immediate value is combined from different places in the instruction.

PC Select indicates whether the next address comes from branch operation or by incrementing the current address by one.

The Derivation of the Control Logic

R- Type

In R-type the alu operation is decided from the function 3 and function 7. The alu output is written to the register and ALU B input comes from the register. PC is incremented by one.

I-Type(Immediate)

In I-type the ALU B input comes from the immediate generator. The immediate selection should be done accordingly. The ALU output is written back to the register. PC is incremented by one.

I-Type (Load)

In load instructions, ALU operation is always addition. The ALU B comes from the immediate generator hence immediate select control signals are needed. The data to be written to the register comes from the memory. PC is incremented by one.

I-type (jalr)

This writes the “PC+1” value to the register. The next PC value comes from the adder.

S-Type

In store instructions, ALU operation is always addition. The ALU B is an immediate value. No data is written to the registers. Data is written to the memory. PC is incremented normally.

SB-Type

Here no writeback is happening. The ALU B comes from the read register. Immediate selection must be done to get the next address. The next pc value is determined after checking whether there is a branch or not using the ALU flags.

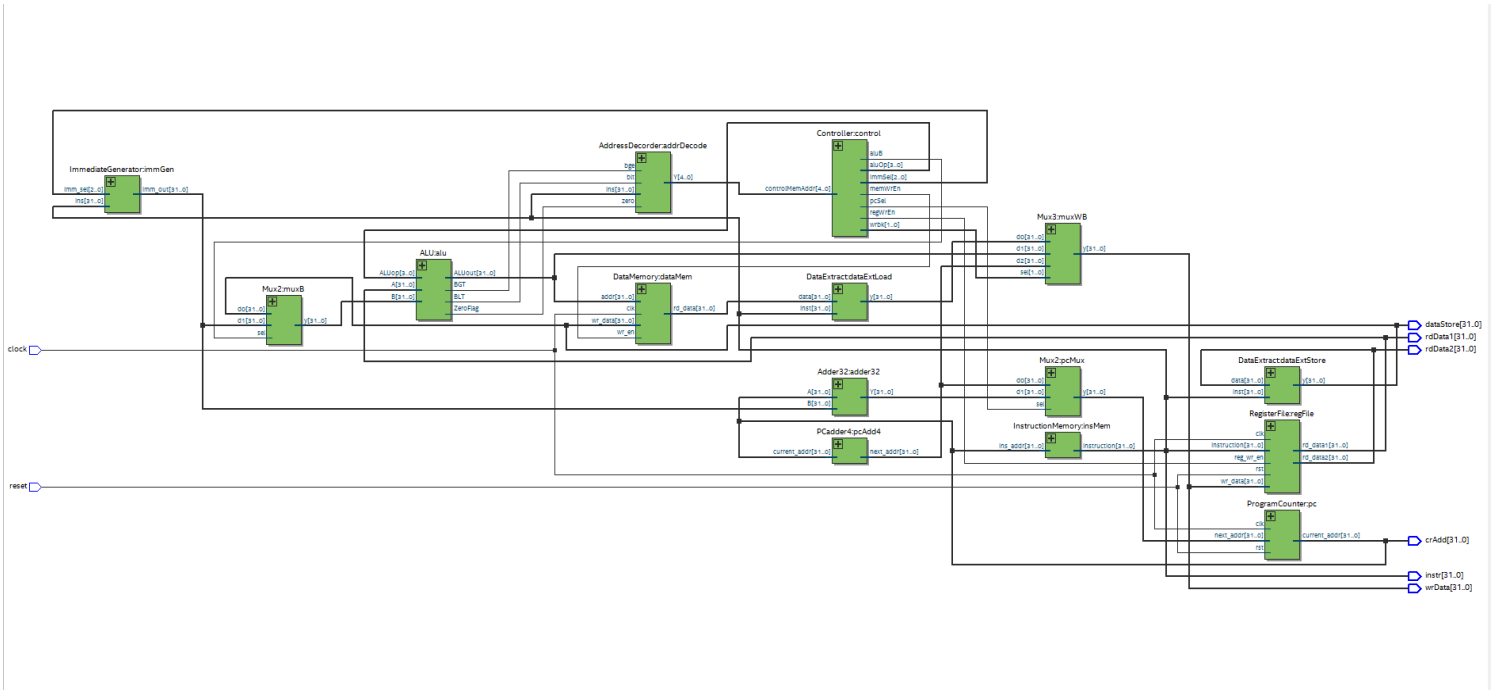
The control signals for each instruction are as follows.

Instruction	Opcode[6:0]	funct3[14:12]	funct7[31:25]	BLT	BGT	Zero	aluB	WriteBack	MemWrEn	RegWrEn	AluOP	ImmSel	PCsel
add	0110011	000	0000000	x	x	x	0	01	0	1	0000	xxx	0
sub	0110011	000	0100000	x	x	x	0	01	0	1	0001	xxx	0
slt	0110011	010	0000000	x	x	x	0	01	0	1	1010	xxx	0
sltu	0110011	011	0000000	x	x	x	0	01	0	1	1011	xxx	0
xor	0110011	100	0000000	x	x	x	0	01	0	1	0111	xxx	0
or	0110011	110	0000000	x	x	x	0	01	0	1	0110	xxx	0
and	0110011	111	0000000	x	x	x	0	01	0	1	0101	xxx	0
sll	0110011	001	0000000	x	x	x	0	01	0	1	0010	xxx	0
srl	0110011	101	0000000	x	x	x	0	01	0	1	0011	xxx	0
sra	0110011	101	0100000	x	x	x	0	01	0	1	0100	xxx	0
addi	0010011	000	xxxxxxx	x	x	x	1	01	0	1	0000	000	0
slti	0010011	010	xxxxxxx	x	x	x	1	01	0	1	1010	000	0
stliu	0010011	011	xxxxxxx	x	x	x	1	01	0	1	1011	010	0
xori	0010011	100	xxxxxxx	x	x	x	1	01	0	1	0111	000	0
ori	0010011	110	xxxxxxx	x	x	x	1	01	0	1	0110	000	0
andi	0010011	111	xxxxxxx	x	x	x	1	01	0	1	0101	000	0
slli	0010011	001	0000000	x	x	x	1	01	0	1	0010	001	0
srli	0010011	101	0000000	x	x	x	1	01	0	1	0011	001	0
srai	0010011	101	0100000	x	x	x	1	01	0	1	0100	001	0
lb	0000011	000	xxxxxxx	x	x	x	1	00	0	1	0000	000	0
lh	0000011	001	xxxxxxx	x	x	x	1	00	0	1	0000	000	0
lw	0000011	010	xxxxxxx	x	x	x	1	00	0	1	0000	000	0
lbu	0000011	100	xxxxxxx	x	x	x	1	00	0	1	0000	000	0
lhu	0000011	101	xxxxxxx	x	x	x	1	00	0	1	0000	000	0
sb	0100011	000	xxxxxxx	x	x	x	1	01	1	0	0000	011	0
sh	0100011	001	xxxxxxx	x	x	x	1	01	1	0	0000	011	0
sw	0100011	010	xxxxxxx	x	x	x	1	01	1	0	0000	011	0
beq	1100011	000	xxxxxxx	x	x	0	0	xx	0	0	1000	100	0
	1100011	000	xxxxxxx	x	x	1	0	xx	0	0	1000	100	1
bne	1100011	001	xxxxxxx	x	x	0	0	xx	0	0	1000	100	1
	1100011	001	xxxxxxx	x	x	1	0	xx	0	0	1000	100	0
blt	1100011	100	xxxxxxx	0	x	x	0	xx	0	0	1000	100	0
	1100011	100	xxxxxxx	1	x	x	0	xx	0	0	1000	100	1
bge	1100011	101	xxxxxxx	x	0	x	0	xx	0	0	1000	100	0
	1100011	101	xxxxxxx	x	1	x	0	xx	0	0	1000	100	1
bltu	1100011	110	xxxxxxx	0	x	x	0	xx	0	0	1001	100	0
	1100011	110	xxxxxxx	1	x	x	0	xx	0	0	1001	100	1
bgeu	1100011	111	xxxxxxx	x	0	x	0	xx	0	0	1001	100	0
	1100011	111	xxxxxxx	x	1	x	0	xx	0	0	1001	100	1
jalr	1100111	000	xxxxxxx	x	x	x	1	10	0	1	0000	000	1

6. System Verilog Implementation

The processor is developed in Quartus Prime Lite Edition using System Verilog as the hardware description language. Each module was implemented separately and then compiled and tested using test benches. Then each module is connected to the Datapath. The data paths were analyzed and validated through the Netlist Viewer (RTL Viewer) option.

Netlist Viewer of the Processor



The Flow Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Sun Oct 15 03:15:21 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Processor
Top-level Entity Name	DataPath
Family	Cyclone IV E
Total logic elements	13,031 / 15,408 (85 %)
Total registers	9248
Total pins	194 / 344 (56 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	6 / 112 (5 %)
Total PLLs	0 / 4 (0 %)
Device	EP4CE15F23C6
Timing Models	Final

The Resource Usage

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	17,160
2		
3	Total combinational functions	8008
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	7186
2	-- 3 input functions	673
3	-- <=2 input functions	149
5		
6	Logic elements by mode	
1	-- normal mode	7796
2	-- arithmetic mode	212
7		
8	Total registers	9248
1	-- Dedicated logic registers	9248
2	-- I/O registers	0
9		
10	I/O pins	194
11		
12	Embedded Multiplier 9-bit elements	6
13		
14	Maximum fan-out node	clock~input
15	Maximum fan-out	9248
16	Total fan-out	59352
17	Average fan-out	3.36

Resource Utilization Summary

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins
1	|DataPath	8008 (0)	9248 (0)	0	6	0	3	194	0
1	|ALU:alu|	565 (537)	0 (0)	0	6	0	3	0	0
1	|lpm_mult:Mult0|	28 (0)	0 (0)	0	6	0	3	0	0
1	|mult_7dt:auto_generated|	28 (28)	0 (0)	0	6	0	3	0	0
2	|AddressDecorder:addrDecode|	44 (44)	0 (0)	0	0	0	0	0	0
3	|Controller:control|	28 (28)	0 (0)	0	0	0	0	0	0
4	|DataExtract:dataExtLoad|	1 (1)	0 (0)	0	0	0	0	0	0
5	|DataExtract:dataExtStore|	41 (41)	0 (0)	0	0	0	0	0	0
6	|DataMemory:dataMem|	5740 (5740)	8192 (8192)	0	0	0	0	0	0
7	|ImmediateGenerator:immGen|	17 (17)	0 (0)	0	0	0	0	0	0
8	|InstructionMemory:insMem|	52 (52)	0 (0)	0	0	0	0	0	0
9	|Mux2:muxB|	56 (56)	0 (0)	0	0	0	0	0	0
10	|Mux3:muxWB|	334 (334)	0 (0)	0	0	0	0	0	0
11	|PCadder4:pcAdd4|	32 (32)	0 (0)	0	0	0	0	0	0
12	|ProgramCounter:pc|	32 (32)	32 (32)	0	0	0	0	0	0
13	|RegisterFile:regFile|	1066 (1066)	1024 (1024)	0	0	0	0	0	0

7. Testing and Simulation

A test bench is written for each module and their operation is verified using the integrated ModelSim simulation within Quartus Prime.

ALU

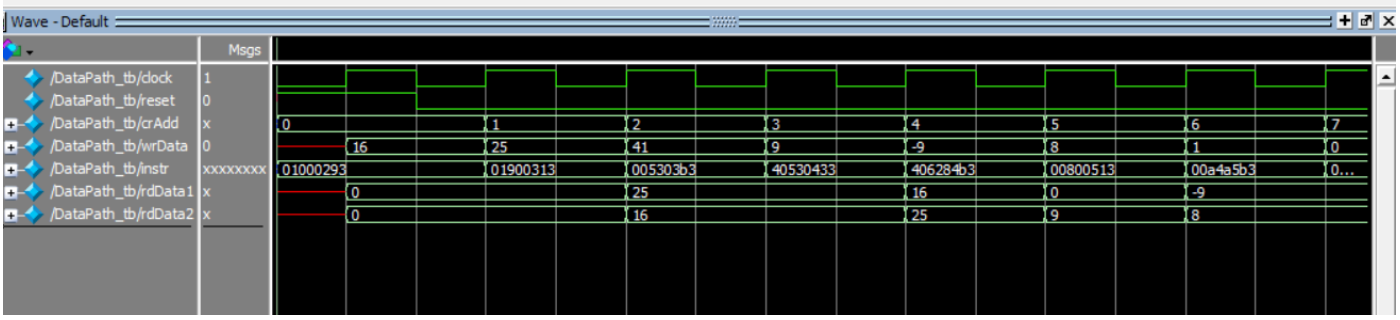
Each one of the ALU operations were tested.

Data Path

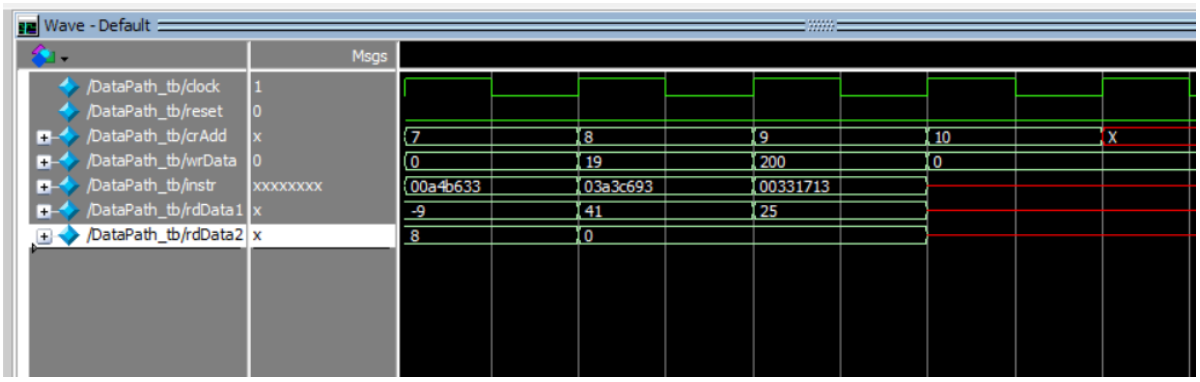
After implementing the Datapath each type of instruction was tested. By giving instructions to the instruction memory representing all types.

```
ins_mem[0] = 32'h01000293; //addi x5, x0, 16
ins_mem[1] = 32'h01900313; //addi x6, x0, 25
ins_mem[2] = 32'h005303b3; //add x7, x6, x5
ins_mem[3] = 32'h0062c463; //blt x5, x6, 9
ins_mem[4] = 32'h40530433; //sub x8, x6, x5
ins_mem[5] = 32'h406284b3; //sub x9, x5, x6
ins_mem[6] = 32'h00800513; //addi x10, x0, 8
ins_mem[7] = 32'h00a4a5b3; //slt x11, x9, x10
ins_mem[8] = 32'h00a4b633; //sltu x12, x9, x10
ins_mem[9] = 32'h03a3c693; //xori x13, x7, 58
ins_mem[10] = 32'h00331713;   //slli x14, x6, 3
ins_mem[11] = 32'h06e6aea3;   //sw x14, x13, 125
ins_mem[12] = 32'h5dc70793;   //addi x15, x14, 1500
ins_mem[13] = 32'h00f2a323;   //sw x15, x5, 6
ins_mem[14] = 32'h0062a803;   //lw x16, x5, 6
ins_mem[15] = 32'h00629883;   //lh x17, x5, 6
ins_mem[16] = 32'h00628883;   //lb x17, x5, 6
```

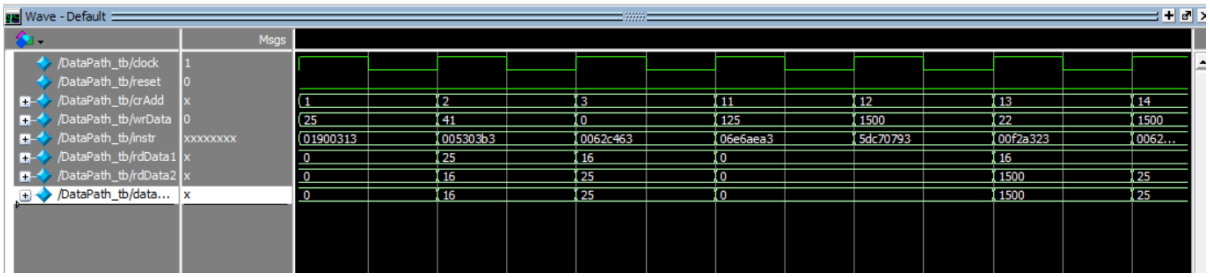
R Type



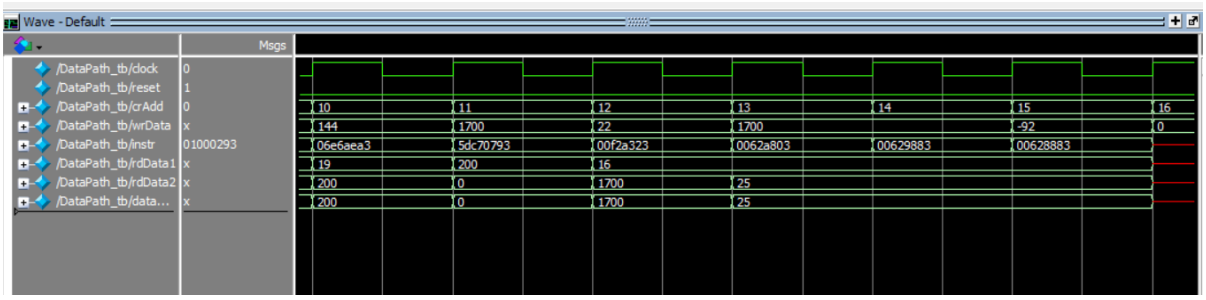
I Type

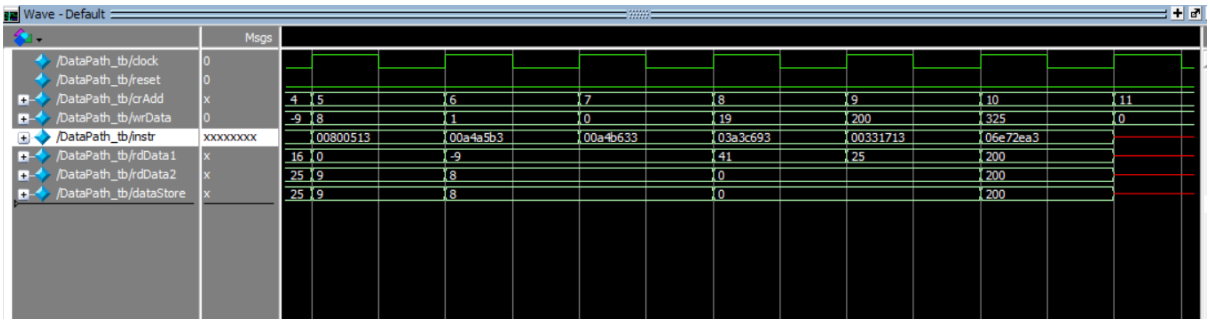


Branch



Load and Store





8. Implementing Additional Instructions

a. MUL

This instruction was implemented similarly to the R-type instructions. For this an additional operation is implemented in the ALU that outputs the multiplication of the given two inputs.

The instruction format is the same as R type.
Opcode – 0110011
Function3 – 000
Function7 – 0010000

This multiplies the values in Read register 1 and Read register 2 and writes the output to the write register.

Rd = Rs1 * Rs2

ALU Operation:

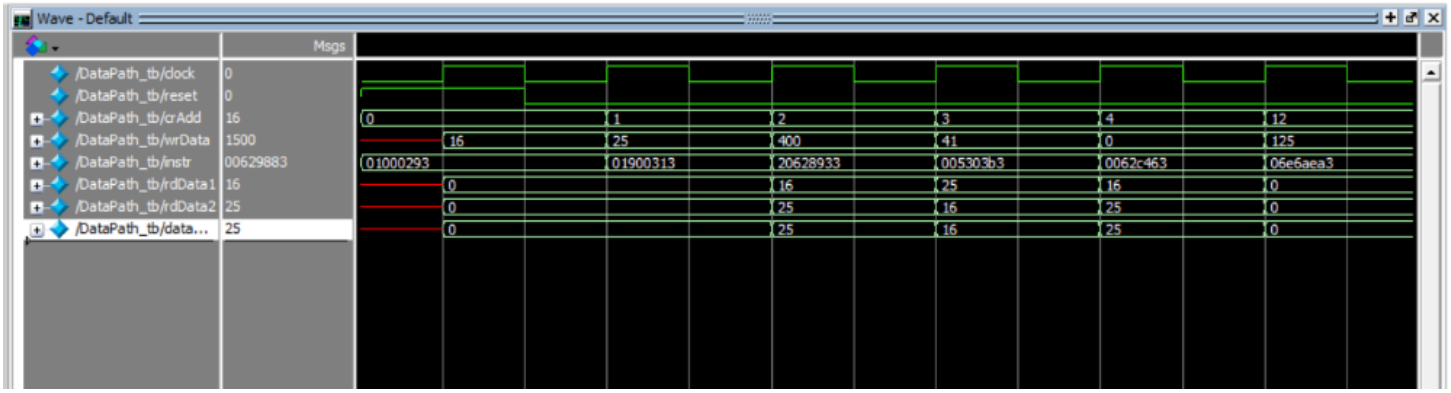
1100	Multiplication

The control signals for the MUL instruction is as follows.

aluB	WriteBack	MemWrEn	RegWrEn	AluOP	ImmSel	Pcsel
0	01	0	1	1100	xxx	0

The MUL instruction was tested as follows.

```
ins_mem[0] = 32'h01000293; //addi x5, x0, 16
ins_mem[1] = 32'h01900313; //addi x6, x0, 25
ins_mem[2] = 32'h20628933; //mul x18, x5, x6
```



Since a 32-bit number is multiplied by 32-bit number the result can outgrow 32 bits. Therefore, there is a limitation for the operands in this instruction. If the multiplication of the two operands result a more than 32 bit value there will be an overflow and the result will be incorrect.

b. MEMCOPY

This instruction copies an array of size N from one memory location to another memory location.
In the instruction, the start location of the array, the start location of the destination and the number N are encoded as follows.

31:25	24:20	19:15	14:7	opcode
imm[14:7]	rs2	rs1	imm [7:0]	0010010

Here the combined immediate value which represents N is 15 bits. Therefore, the maximum N that supports MEMCOPY is:

$2^{15} = 32768$