Computer Systems Engineering

ENCS4370

COMPUTER ARCHITECTURE

Project 2
Multicycle Processor

Student's Name: Abbas Nassar | **Partner's Name:** Mohammad Ataya

Student's ID: 1210482 **Partner's ID:** 1211555

Partner's Name: Ghassan Qandeel

Partner's ID: 1212397

Date: 6/6/2024

❖ Abstract

The goal of this project is to design a RISC Multi-Cycle processor using Verilog. The design process consists of building functional units, control units with truth tables and logic equations for all the signals needed to support 4 different instruction formats, and 21 different instructions. Then, the process requires to build full data-path, and state diagram. In addition to that, it requires building modules using Verilog to simulate the designed processor. At last, It’s obligatory to create testbenches to verify the design.

Abbas Nassar: Worked on report and theoretical designs (units, datapath, etc..) and testbench.

Ghassan Qandeel: Worked on Code and testing it, also helped with RTL design of instructions and datapath.

Mohammad Ataya: Worked on code and helped with Truth tables and logic equation.

❖ Table of Contents

❖ List of Figures

❖ List of Tables

❖ Theory

First: Functional Units:

- **PC Register:**

The PC register is a 16-Bit register that holds the address of the next instruction. It’s synchronized on a clock.

PC Register
Clock

Figure 1: PC Register Block Diagram.

- **Instruction Memory:**

The Instruction memory is a memory block with 2^{16} 16-Bit Registers, each register can hold the value of a different instruction. It takes 16-Bit address and clock signal as inputs. Outputs Instruction[address].

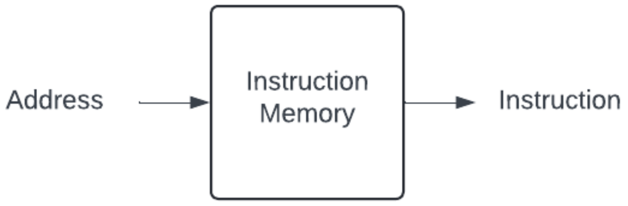


Figure 2: Instruction Memory Block Diagram.

- ## Register File:

The Register file is a registers block with eight 16-Bit General Purpose Registers, each register can hold an integer value. It takes three 3-Bit Addresses, 16-Bit write bus, 1- Bit write signal and clock signal as inputs. Outputs Two 16-Bit data busses, two flags: zero and sign.

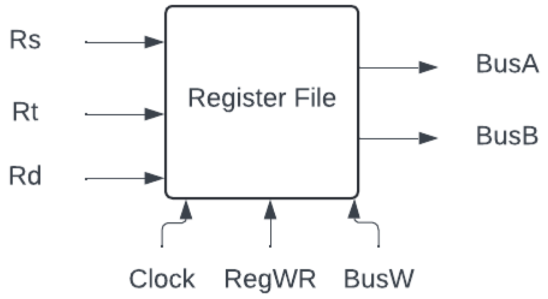


Figure 3: Register File Block Diagram.

- ## Data Memory:

The Data memory file is a memory file with 2^{16} 16-Bit registers that holds integer value. It takes 16-Bit address, 16-Bit data bus, clock signal, memory read signal, memory write signal, as input. Outputs 16-Bit data bus.

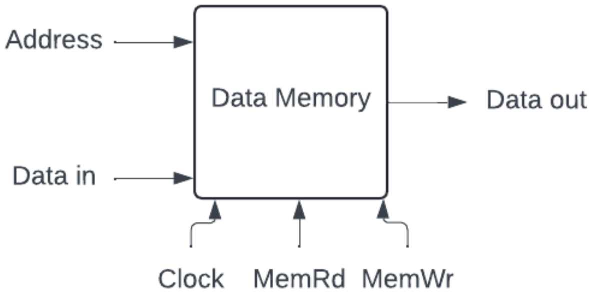


Figure 4: Data Memory Block Diagram.

- **Mux:**

The Mux is used to select one input to output from multiple inputs.

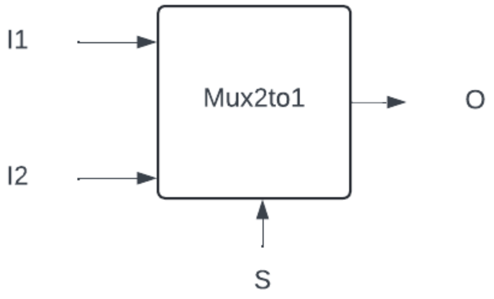


Figure 5: Mux2to1 Block Diagram.

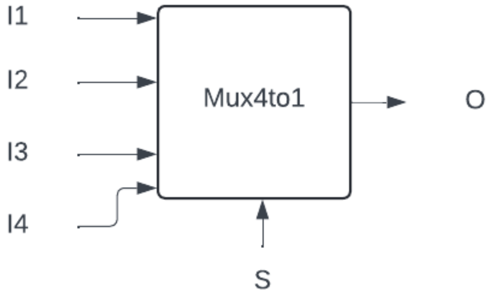


Figure 6: Mux4to1 Block Diagram.

- **Extender:**

The Extender is used to extend (signed or not signed) an input bus.

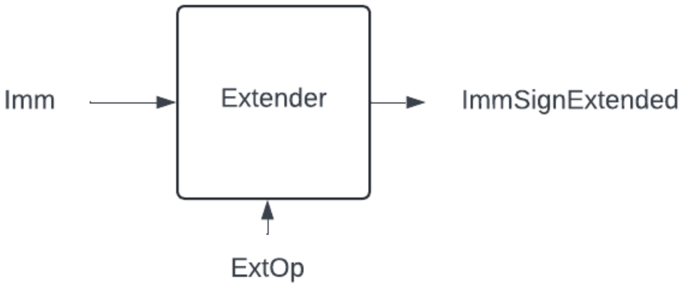


Figure 7: Sign Extender Block Diagram

- **Adder:**

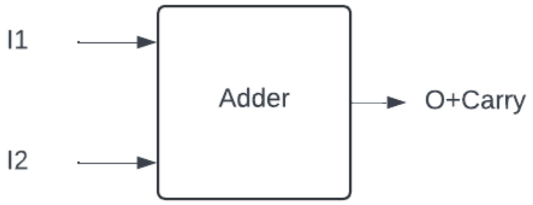


Figure 8: Adder Block Diagram.

- **ALU Unit:**

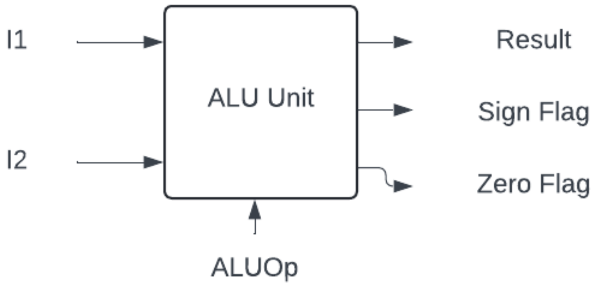


Figure 9: ALU Unit Block Diagram

Second: Control Units:

- ### Pc Control Unit:

The PC control unit takes OpCode, mode bit, and a set of flags as inputs to generate **OpCode source**, The opcode source is used as a selection line for a mux that generates the input of PC register. OpCode is used to differentiate between jump instructions. OpCode and mode bit are used to differentiate between branch instructions. Sign and zero flags are used to check whether the condition of the branch is satisfied or not. More of the specifications will be discussed later on.

OpCode Source	PC Register
0	PC + 2.
1	Jump Address.
2	Branch Address.
3	Content Of R_7.

Table 1: PC Values.

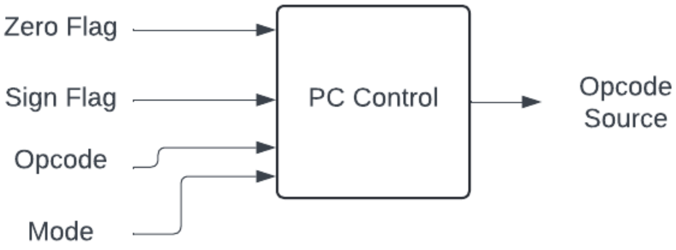


Figure 10: PC Control Unit.

- **Main Control Unit:**

The main control unit takes OpCode and mode bit as inputs and generates the main signal for the data path.

Signal	Effect when '0'	**Effect when '1'**
RBDst	RB = Rs2	RB = Rd
RegWr	No register is written	Destination register (Rt or Rd) is written with the data on BUSW
ExtOp	16-bit immediate is zero-extended	16-bit immediate is sign-extended
ALUSrc	Second ALU operand is the value of register Rt that appears on BUSB	Second ALU operand is the value of the extended 16-bit immediate
MemRd	Data memory is NOT read	Data memory is read Data Out ← Memory[address]
MemWr	Data Memory is NOT written	Data memory is written Memory[address] ← Data In
WBData	BUSW = ALU result	BUSW= Data Out from Memory
ByteExt	Byte read from memory is not sign extended.	Byte read from memory is sign extended.
CallSgn	Register write address = Rd	Register write address = R7.
RETSgn	RA = Rs1 Or R0.	RA = R7 Or R0.
SvSgn	Data In Memory = BUSB.	Data In Memory = $(\text{Imm})_{\text{SignExtend}}$.
BRSgn	Branch when Rd = Rs.	Branch when Rd = R0.
DataStrd	BUSW = Data.	BUSW = PC+2.

Table 2: Main Signals.

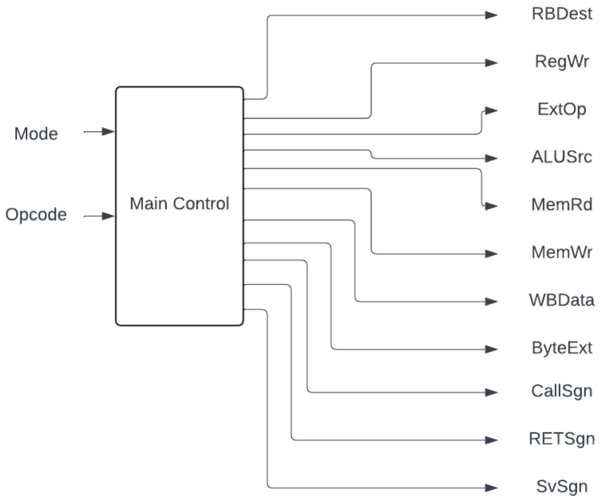


Figure 11: Main Control Unit

- ### ALU Control Unit:

The ALU control unit takes the OpCode as an input and generates ALUOp which determines the ALU operation to execute. In this project we have 3 unique operations expressed in the following table.

ALUOp	Operation To Execute
00	Bitwise AND.
01	Addition.
10	Subtraction.

Table 3: ALU Operations.

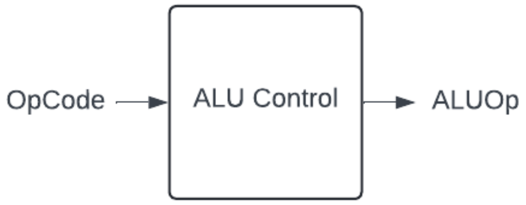


Figure 12: ALU Control Unit.

- **Truth Tables:**

- PC Control Truth Table:

Operation	Zero Flag	Sign Flag	PC Source
R-Type	X	X	0 = Increment PC
JMP	X	X	1 = Jump Target Address
BGT & BGTZ	0	0	2 = Branch Target Address
BGT & BGTZ	0	1	0 = Increment PC
BGT & BGTZ	1	X	0 = Increment PC
BLT & BLTZ	0	0	0 = Increment PC
BLT & BLTZ	0	1	2 = Branch Target Address
BLT & BLTZ	1	X	0 = Increment PC
BEQ & BEQZ	0	X	0 = Increment PC
BEQ & BEQZ	1	X	2 = Branch Target Address
BNE & BNEZ	0	X	2 = Branch Target Address
BNE & BNEZ	1	X	0 = Increment PC
CALL	X	X	1 = Jump Target Address
RET	X	X	3 = Content Of R_7

Table 4: PC Control Truth Table.

Logic Equations For PC Control Signal:

- **PCSrc0**=(R-Type) + (BGT·~ZF·SF) + (BLT·~ZF·~SF) + (BLT·ZF) + (BEQ·ZF) + (BNE·ZF)
- **PCSrc1**=(JMP)+(CALL)
- **PCSrc2**=(BGT·~ZF·~SF) +(BLT·~ZF·SF) + (BEQ·ZF) + (BNE·~ZF)
- **PCSrc3**=(RET)

- ALU Control Truth Table:

Operation	ALU OpCode	2-bit Coding
AND	AND	00 = Bitwise AND
ADD	ADD	01 = Addition
SUB	SUB	10 = Subtraction
ADDI	ADD	01 = Addition
ANDI	AND	00 = Bitwise AND
LW	ADD	01 = Addition
LBu	ADD	01 = Addition
LBs	ADD	01 = Addition
SW	ADD	01 = Addition
BGT	SUB	10 = Subtraction
BGTZ	SUB	10 = Subtraction
BLT	SUB	10 = Subtraction
BLTZ	SUB	10 = Subtraction
BEQ	SUB	10 = Subtraction
BEQZ	SUB	10 = Subtraction
BNE	SUB	10 = Subtraction
BNEZ	SUB	10 = Subtraction
JMP	X	X
CALL	X	X
RET	X	X
Sv	X	X

Table 5: ALU Control Truth Table.

Logic Equations For ALU Control Signal:

- **Bitwise AND** = AND + ANDI
- **Addition** = ADD + ADDI + LW+ LBu +LBs + SW
- **Subtraction**=SUB+BGT+BGTZ+BLT+BLTZ+BEQ+BEQZ+BNE+BNEZ

- Main Control Truth Table:

OP	RBDst	Reg Wr	ExtOp	ALUSrc	Mem Rd	Mem Wr	WBData	Byte Ext	Call Sgn	RET Sgn	Sv Sgn	BRSgn	DataStrd
R-Type	0 = Rs2	1	X	0 = BUSB	0	0	0 = Result	X	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
ADDI	X	1	1	1 = Imm_{ext}	0	0	0 = Result	X	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
ANDI	X	1	0	1 = Imm_{ext}	0	0	0= Result	X	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
LW	X	1	1	1 = Imm_{ext}	1	0	1 = Data Out	0	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
LBu	X	1	1	1 = Imm_{ext}	1	0	1 = Data Out	0	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
LBs	X	1	1	1 = Imm_{ext}	1	0	1 = Data Out	1	0 = Rd	0 = Rs1	X	0 = Rs1	0 = Data
SW	1= Rd	0	1	1 = Imm_{ext}	0	1	X	X	X	0 = Rs1	0 = BUSB	0 = Rs1	X
BGT	1= Rd	0	1	0 = BUSB	0	0	X	X	X	0 = Rs1	X	0 = Rs1	X
BGTZ	1= Rd	0	1	0 = BUSB	0	0	X	X	X	X	X	1 = R0	X
BLT	1= Rd	0	1	0 = BUSB	0	0	X	X	X	0 = Rs1	X	0 = Rs1	X
BLTZ	1= Rd	0	1	0 = BUSB	0	0	X	X	X	X	X	1 = R0	X
BEQ	1= Rd	0	1	0 = BUSB	0	0	X	X	X	0 = Rs1	X	0 = Rs1	X
BEQZ	1= Rd	0	1	0 = BUSB	0	0	X	X	X	X	X	1 = R0	X
BNE	1= Rd	0	1	0 = BUSB	0	0	X	X	X	0 = Rs1	X	0 = Rs1	X
BNEZ	1= Rd	0	1	0 = BUSB	0	0	X	X	X	X	X	1 = R0	X
JMP	X	0	1	X	0	0	X	X	X	X	X	X	X
CALL	X	1	1	X	0	0	X	X	1 = R7	X	X	X	1 = PC + 4
RET	X	0	1	X	0	0	X	X	X	1 = R7	X	0 = R7	X
Sv	X	0	1	X	0	1	X	X	X	0 = Rs1	1 = Imm_{ext}	0 = Rs1	X

Table 6: Main Control Truth Table.

Logic Equations For Main Control Signals:

- **RBDst** = R_Type
- **RegWr** = ADDI + ANDI + LW + LBu + LBs + CALL
- **ExtOp** = 1
- **ALUSrc** = ADDI + ANDI + LW + LBu + LBs + SW
- **MemRd** = LW + LBu + LBs
- **MemWr** = SW + Sv
- **WBData** = LW + LBu + LBs
- **ByteExt** = LBs
- **CallSgn** = CALL
- **RETSgn** = RET
- **SvSgn** = Sv
- **BRSgn** = BGTZ + BLTZ + BEQZ + BNEZ
- **DataStrd** = Sv

Third: Data-Path

Since the Multi-Cycle Datapath is very similar to the Multi-Cycle Datapath in slides, we took this design and made the necessary changes to it in order to match our design needs.

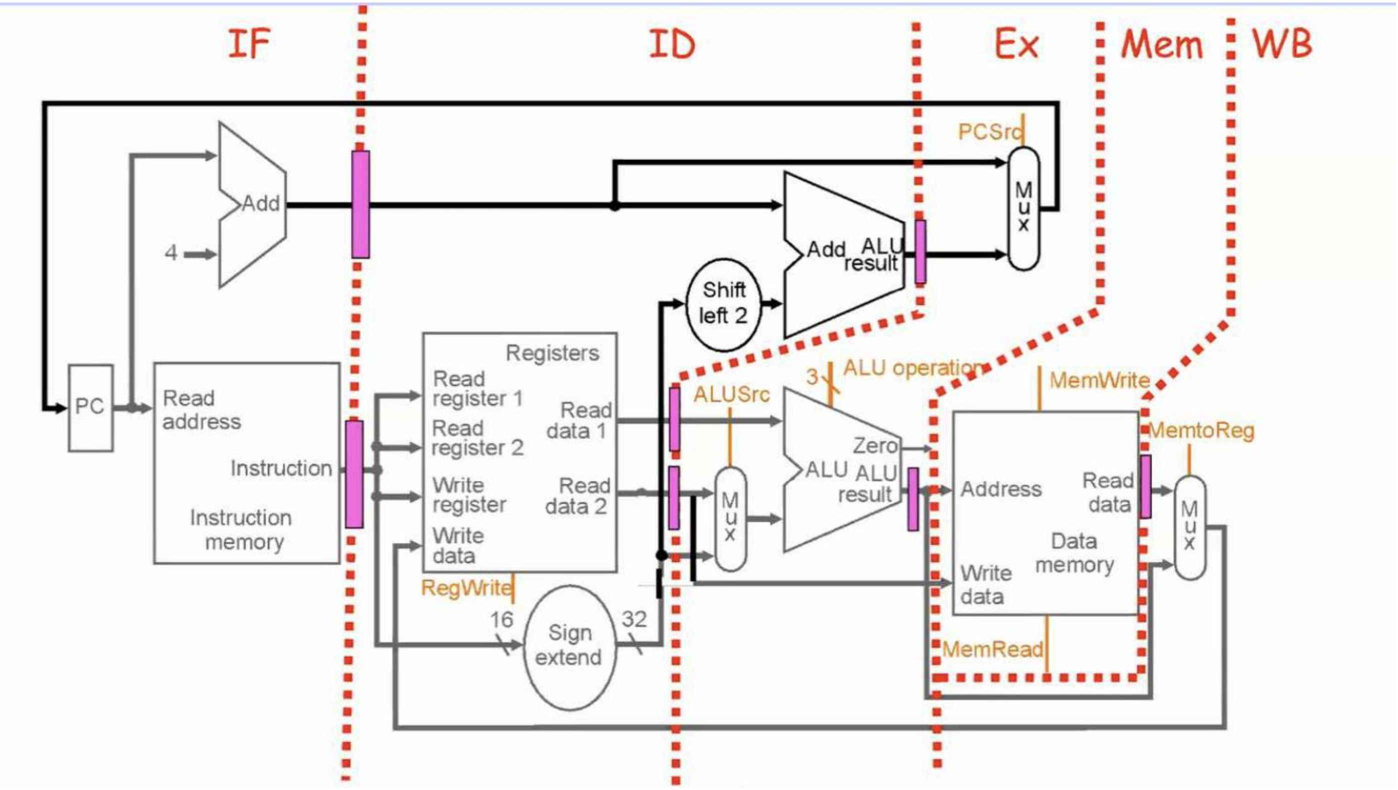


Figure 13: Multi-Cycle Datapath + Control.[1]

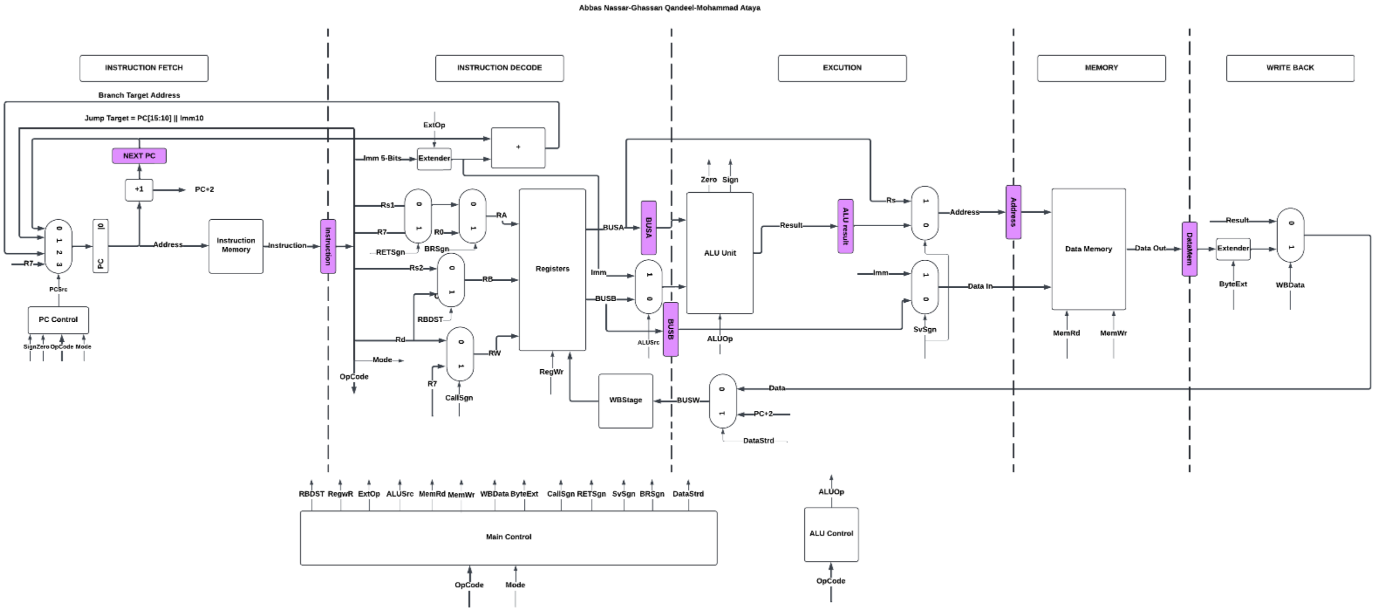


Figure 14: Multi-Cycle Datapath and Control.

Fourth: State Diagram:

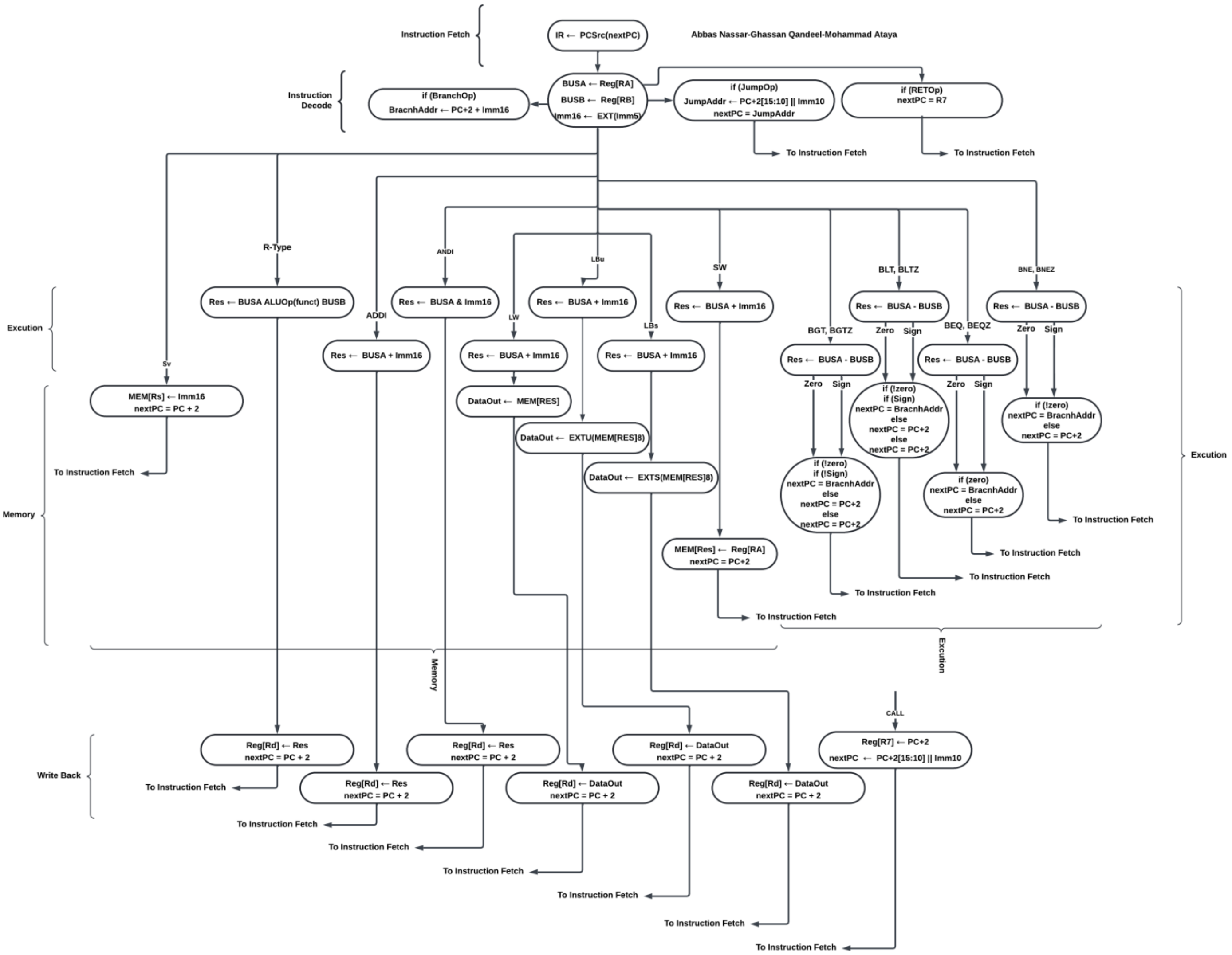


Figure 15: Multi-Cycle State Diagram.

❖ Procedure

Modules Code in a separate file will be attached with this report.

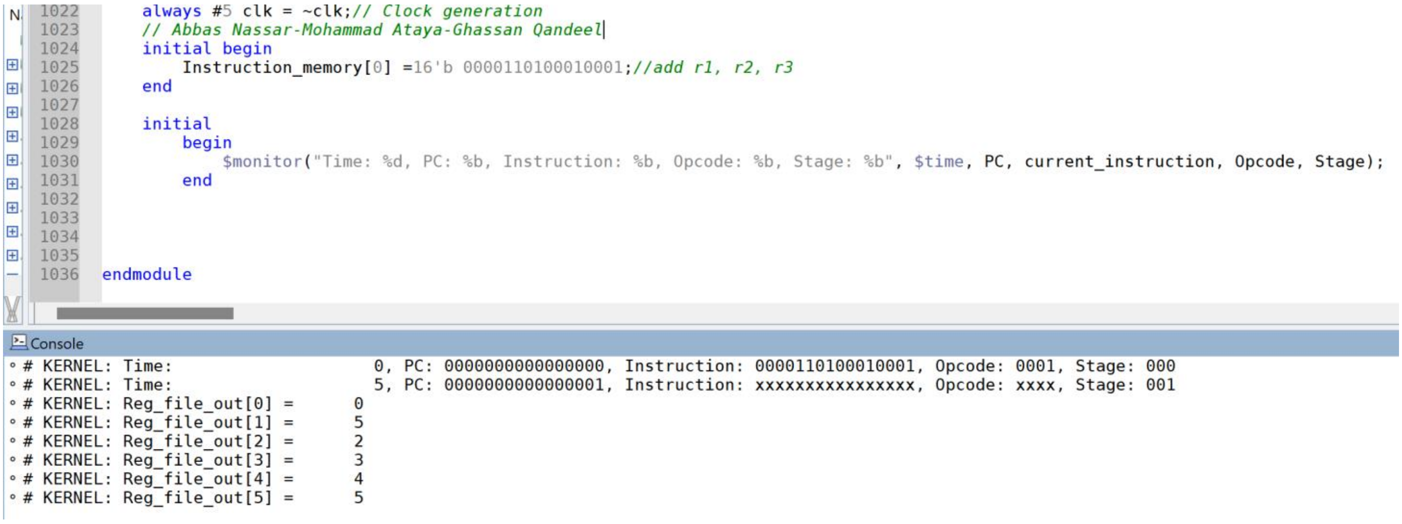
RTL Design of Instructions:

Instruction Type	RTL Description
R-Type	Fetch instruction: Instruction ← IMEM[PC] Fetch operands: data1 ← Reg [Rs1], data2 ←Reg [Rs2] Execute operation: ALU_result ← func(data1, data2) Write ALU result: Reg [Rd] ← ALU_result Next PC address: PC ← PC + 2
I-Type (ADDI & ANDI)	Fetch instruction: Instruction ← IMEM[PC] Fetch operands: data1 ← Reg [Rs1], data2 ← Extend(imm16) Execute operation: ALU_result ← func(data1, data2) Write ALU result: Reg [Rd] ← ALU_result Next PC address: PC ← PC + 2
I-Type (LW & LBU & LBS)	Fetch instruction: Instruction ← IMEM[PC] operands: data1 ← Reg [Rs1], data2 ← Extend(imm16) Execute operation: ALU_result ← func(data1, data2) Write ALU result: Reg [Rd] ← Mem (ALU_result) Next PC address: PC ← PC + 2
I-Type (SW)	Fetch instruction: Instruction ← IMEM[PC] Fetch operands: data1 ← Reg [Rs1], data2 ← Extend(imm16) Execute operation: ALU_result ← func (data1, data2) Write ALU result: Mem (ALU_result) ← Reg [Rd] Next PC address: PC ← PC + 2
I-Type (Branch)	Fetch instruction: Instruction ← IMEM[PC] Fetch operands: data1 ← Reg [Rs1], data2 ← Reg (Rs2) Branch target address = PC+2 + Extend (Imm5) Execute operation: ALU_result ← func(data1, data2) Next PC address: if (PCsrc == 2) PC ← Branch target address else PC ← PC + 2
J-Type	Fetch instruction: Instruction ← IMEM[PC] Target PC address: target ← {PC+2[15:10], Immediate10} Next PC address: PC ← target
S-Type	Fetch instruction: Instruction ← IMEM[PC] Fetch registers: data1 ← Rs1, data2←Simmd Write memory: Mem [data 1] = data2 Next PC address: PC ← PC + 2

Table 7: RTL Design Instruction.

Testbench For Each Instruction:

ADD:



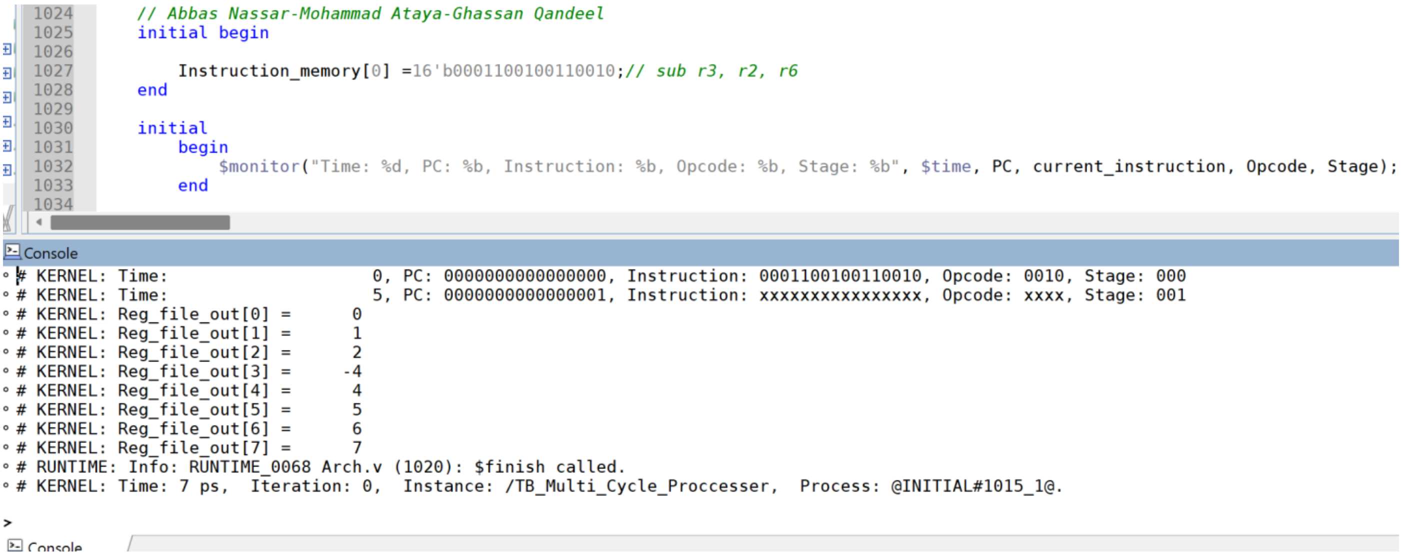
```
    always #5 clk = ~clk;// Clock generation
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin
        Instruction_memory[0] =16'b 0000110100010001;//add r1, r2, r3
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
        end




endmodule
```

```
# KERNEL: Time:                         0, PC: 0000000000000000, Instruction: 0000110100010001, Opcode: 0001, Stage: 000
# KERNEL: Time:                         5, PC: 0000000000000001, Instruction: xxxxxxxxxxxxxxxx, Opcode: xxxx, Stage: 001
# KERNEL: Reg_file_out[0] =      0
# KERNEL: Reg_file_out[1] =      5
# KERNEL: Reg_file_out[2] =      2
# KERNEL: Reg_file_out[3] =      3
# KERNEL: Reg_file_out[4] =      4
# KERNEL: Reg_file_out[5] =      5
```

Figure 16: ADD Instruction Test.

- Reg_file_out[1] = Reg_file_out[2] + Reg_file_out[3].

SUB:



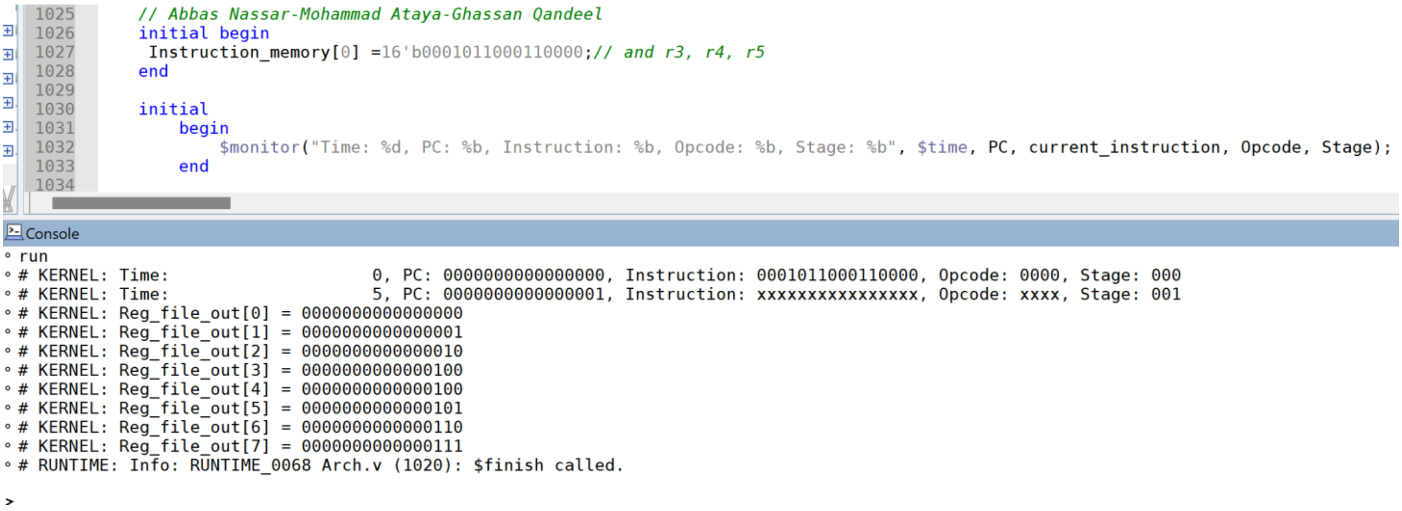
```
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin

        Instruction_memory[0] =16'b0001100100110010;// sub r3, r2, r6
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
        end

```

```
# KERNEL: Time:                         0, PC: 0000000000000000, Instruction: 0001100100110010, Opcode: 0010, Stage: 000
# KERNEL: Time:                         5, PC: 0000000000000001, Instruction: xxxxxxxxxxxxxxxx, Opcode: xxxx, Stage: 001
# KERNEL: Reg_file_out[0] =      0
# KERNEL: Reg_file_out[1] =      1
# KERNEL: Reg_file_out[2] =      2
# KERNEL: Reg_file_out[3] =     -4
# KERNEL: Reg_file_out[4] =      4
# KERNEL: Reg_file_out[5] =      5
# KERNEL: Reg_file_out[6] =      6
# KERNEL: Reg_file_out[7] =      7
# RUNTIME: Info: RUNTIME_0068 Arch.v (1020): $finish called.
# KERNEL: Time: 7 ps,  Iteration: 0,  Instance: /TB_Multi_Cycle_Proccesser,  Process: @INITIAL#1015_1@.

>
```

Figure 17: SUB Instruction Test.

- Reg_file_out[3] = Reg_file_out[2] + Reg_file_out[6].

AND:



```
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin
     Instruction_memory[0] =16'b0001011000110000;// and r3, r4, r5
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
        end

```

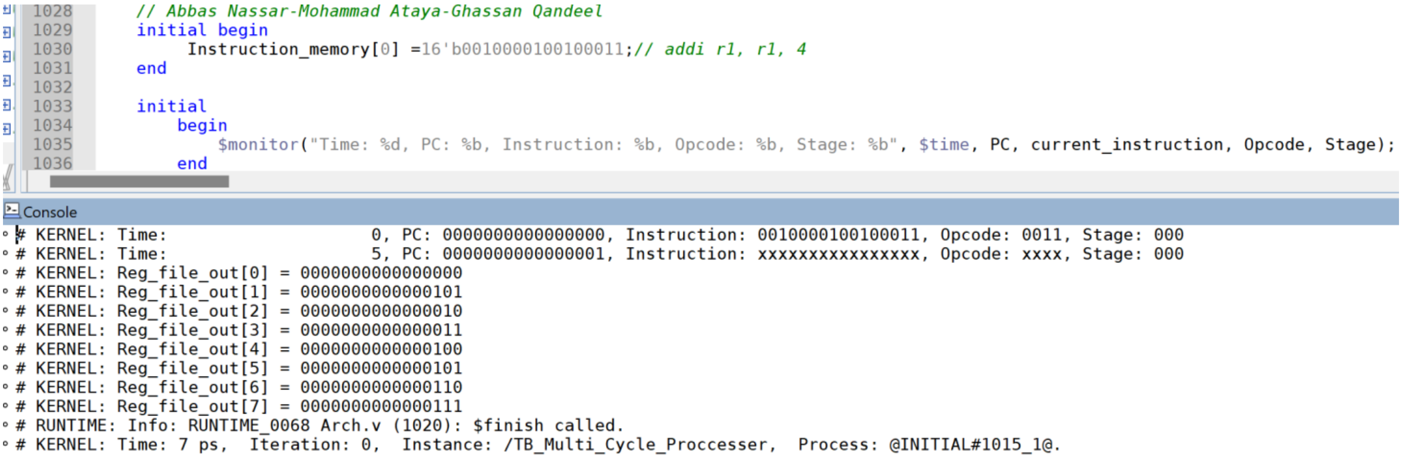
Console

```
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0001011000110000, Opcode: 0000, Stage: 000
# KERNEL: Time:                    5, PC: 0000000000000001, Instruction: xxxxxxxxxxxxxxxx, Opcode: xxxx, Stage: 001
# KERNEL: Reg_file_out[0] = 0000000000000000
# KERNEL: Reg_file_out[1] = 0000000000000001
# KERNEL: Reg_file_out[2] = 0000000000000010
# KERNEL: Reg_file_out[3] = 0000000000000100
# KERNEL: Reg_file_out[4] = 0000000000000100
# KERNEL: Reg_file_out[5] = 0000000000000101
# KERNEL: Reg_file_out[6] = 0000000000000110
# KERNEL: Reg_file_out[7] = 0000000000000111
# RUNTIME: Info: RUNTIME_0068 Arch.v (1020): $finish called.

>
```

Figure 18: AND Instruction Test.

- Reg_file_out[3] = Reg_file_out[4] & Reg_file_out[5].

ADDI:



```
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin
        Instruction_memory[0] =16'b0010000100100011;// addi r1, r1, 4
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
        end
```

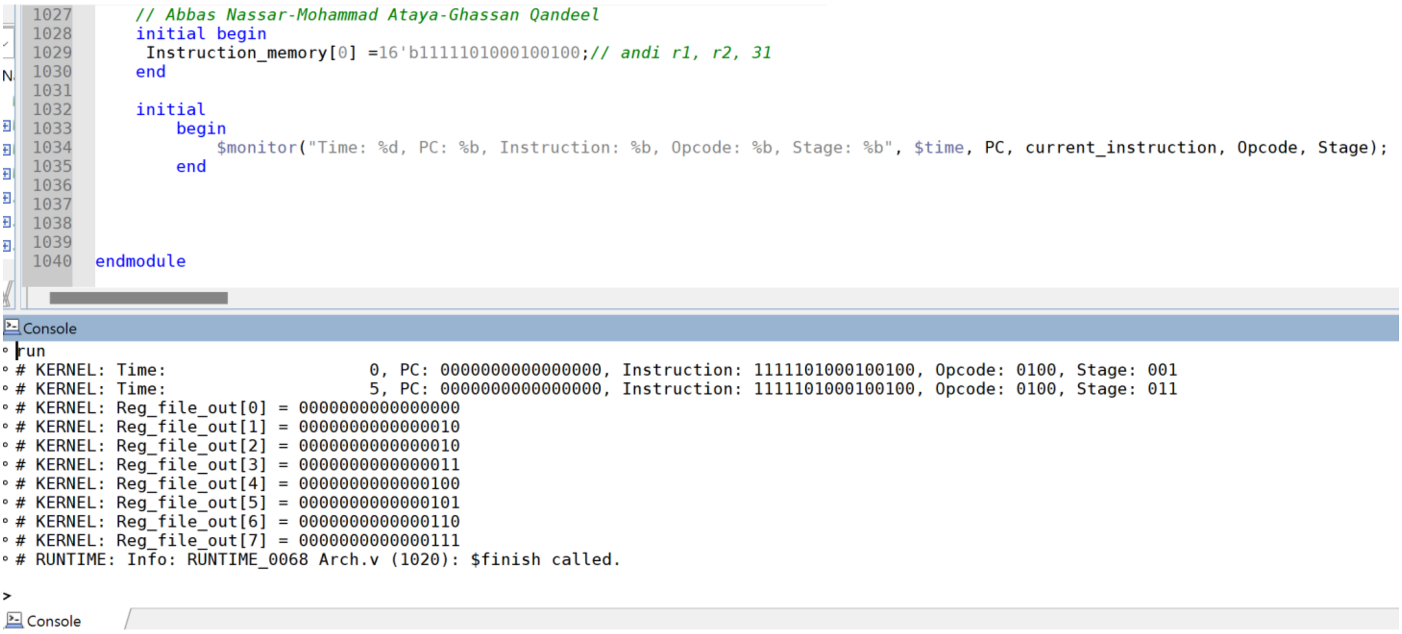
Console

```
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0010000100100011, Opcode: 0011, Stage: 000
# KERNEL: Time:                    5, PC: 0000000000000001, Instruction: xxxxxxxxxxxxxxxx, Opcode: xxxx, Stage: 000
# KERNEL: Reg_file_out[0] = 0000000000000000
# KERNEL: Reg_file_out[1] = 0000000000000101
# KERNEL: Reg_file_out[2] = 0000000000000010
# KERNEL: Reg_file_out[3] = 0000000000000011
# KERNEL: Reg_file_out[4] = 0000000000000100
# KERNEL: Reg_file_out[5] = 0000000000000101
# KERNEL: Reg_file_out[6] = 0000000000000110
# KERNEL: Reg_file_out[7] = 0000000000000111
# RUNTIME: Info: RUNTIME_0068 Arch.v (1020): $finish called.
# KERNEL: Time: 7 ps,  Iteration: 0,  Instance: /TB_Multi_Cycle_Proccesser,  Process: @INITIAL#1015_1@.
```

Figure 19: ADDI Instruction Test.

- Reg_file_out[1] = Reg_file_out[2] + 4.

ANDI:



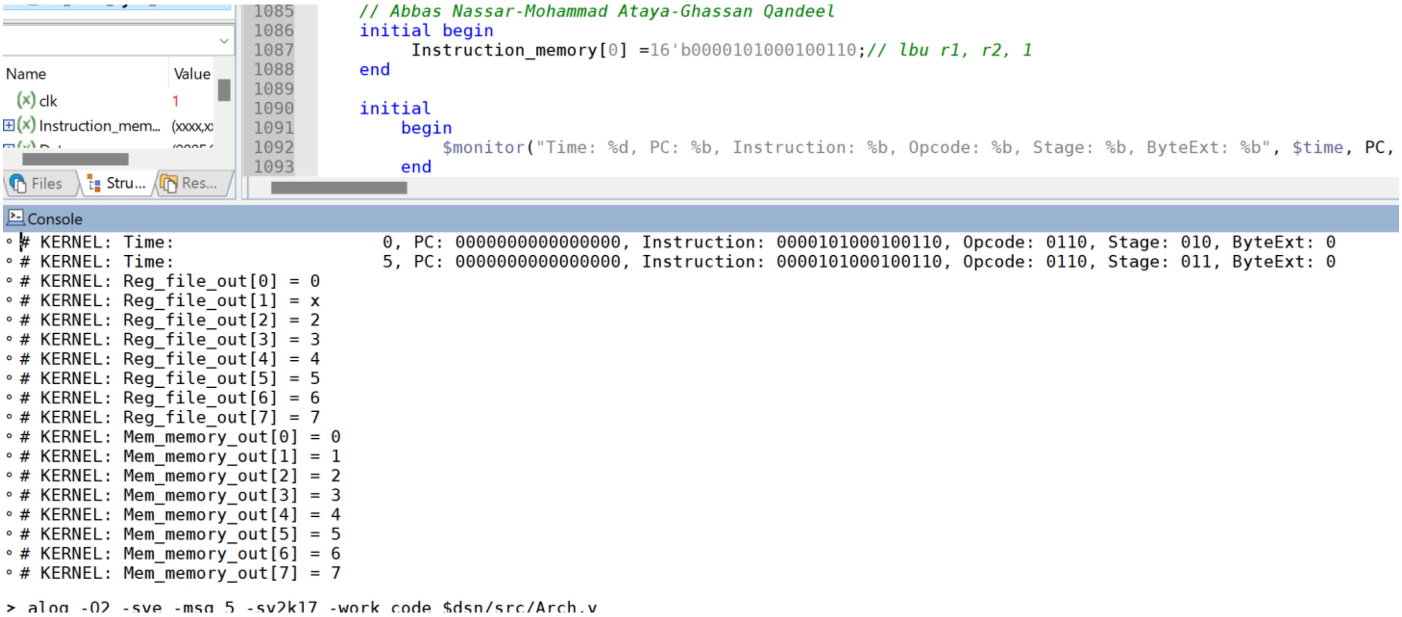
```
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin
     Instruction_memory[0] =16'b1111101000100100;// andi r1, r2, 31
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
        end




endmodule
```

```
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 1111101000100100, Opcode: 0100, Stage: 001
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 1111101000100100, Opcode: 0100, Stage: 011
# KERNEL: Reg_file_out[0] = 0000000000000000
# KERNEL: Reg_file_out[1] = 0000000000000010
# KERNEL: Reg_file_out[2] = 0000000000000010
# KERNEL: Reg_file_out[3] = 0000000000000011
# KERNEL: Reg_file_out[4] = 0000000000000100
# KERNEL: Reg_file_out[5] = 0000000000000101
# KERNEL: Reg_file_out[6] = 0000000000000110
# KERNEL: Reg_file_out[7] = 0000000000000111
# RUNTIME: Info: RUNTIME_0068 Arch.v (1020): $finish called.

>
```

Figure 20: ANDI Instruction Test.

- Reg_file_out[1] = Reg_file_out[2] & 16’b0000000000001111.

LBu:



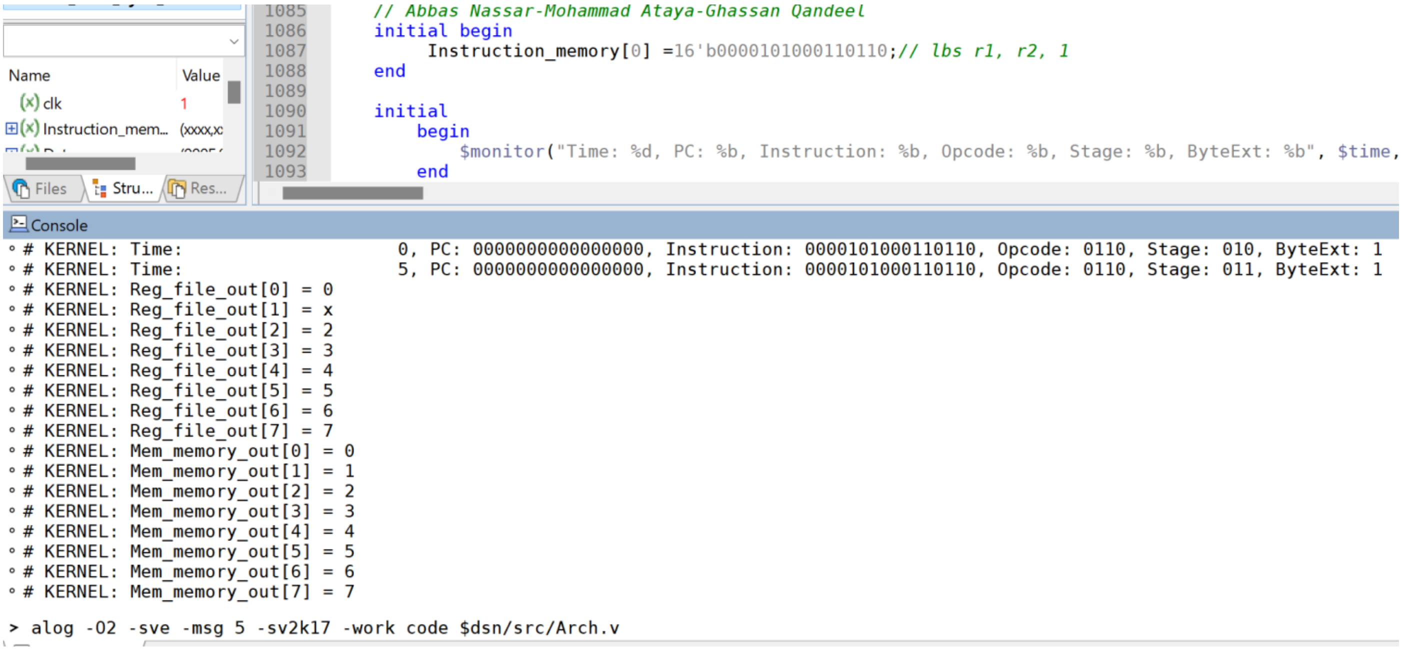
```
    // Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
    initial begin
        Instruction_memory[0] =16'b0000101000100110;// lbu r1, r2, 1
    end

    initial
        begin
            $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, ByteExt: %b", $time, PC,
        end
```

```
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0000101000100110, Opcode: 0110, Stage: 010, ByteExt: 0
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0000101000100110, Opcode: 0110, Stage: 011, ByteExt: 0
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = x
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
# KERNEL: Mem_memory_out[0] = 0
# KERNEL: Mem_memory_out[1] = 1
# KERNEL: Mem_memory_out[2] = 2
# KERNEL: Mem_memory_out[3] = 3
# KERNEL: Mem_memory_out[4] = 4
# KERNEL: Mem_memory_out[5] = 5
# KERNEL: Mem_memory_out[6] = 6
# KERNEL: Mem_memory_out[7] = 7

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 21: LBu Instruction Test.

- Reg_file_out[r1] = Unsigned Ext(byte(Mem_memory_out[Reg(r2) + 1]))

LBs:



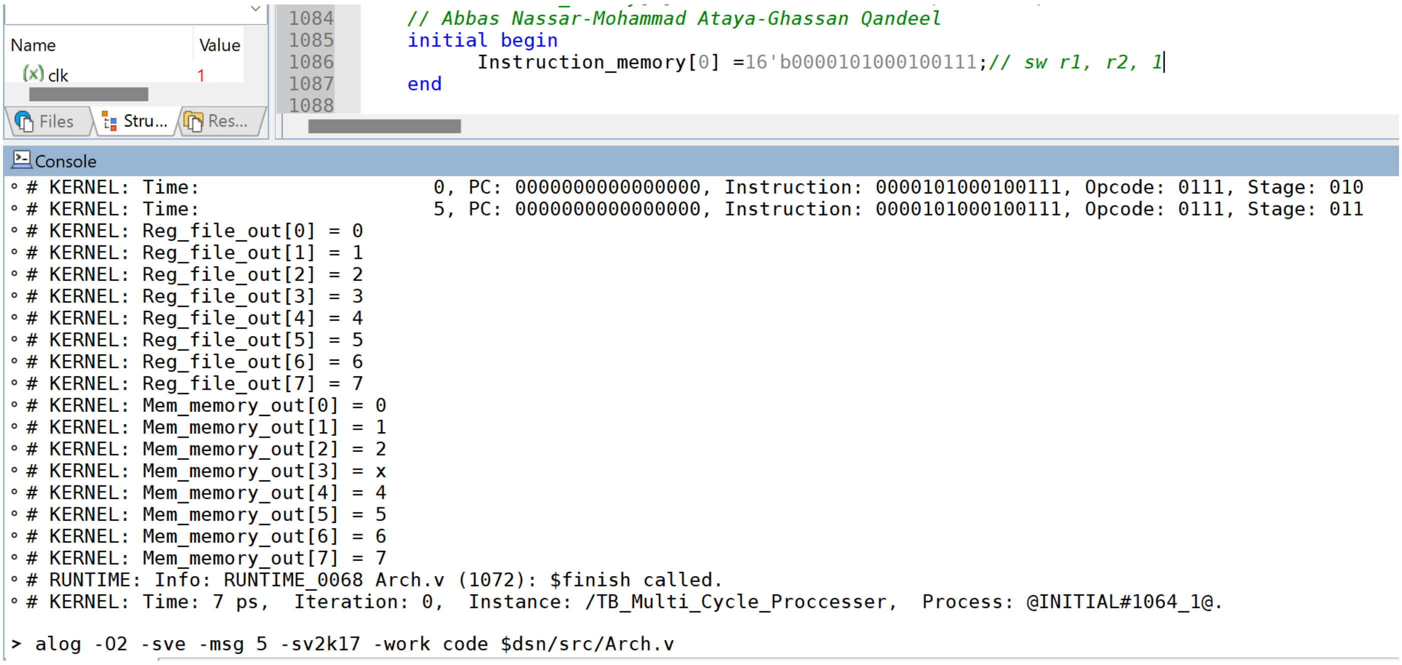
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000101000110110;// lbs r1, r2, 1
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, ByteExt: %b", $time,
    end
```

```
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0000101000110110, Opcode: 0110, Stage: 010, ByteExt: 1
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0000101000110110, Opcode: 0110, Stage: 011, ByteExt: 1
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = x
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
# KERNEL: Mem_memory_out[0] = 0
# KERNEL: Mem_memory_out[1] = 1
# KERNEL: Mem_memory_out[2] = 2
# KERNEL: Mem_memory_out[3] = 3
# KERNEL: Mem_memory_out[4] = 4
# KERNEL: Mem_memory_out[5] = 5
# KERNEL: Mem_memory_out[6] = 6
# KERNEL: Mem_memory_out[7] = 7

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 22: LBs Instruction Test.

- Reg_file_out[r1] = Signed Ext(byte(Mem_memory_out[Reg(r2) + 1]))

SW:



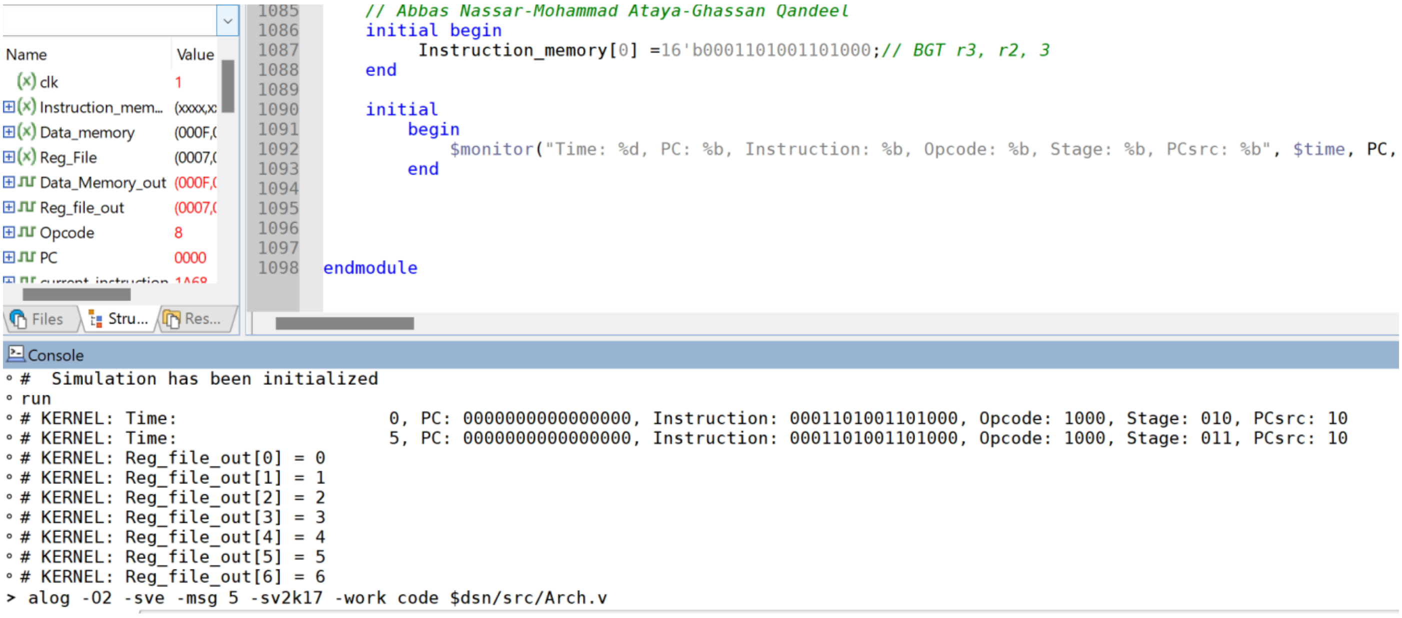
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000101000100111;// sw r1, r2, 1
end
```

```
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0000101000100111, Opcode: 0111, Stage: 010
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0000101000100111, Opcode: 0111, Stage: 011
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
# KERNEL: Mem_memory_out[0] = 0
# KERNEL: Mem_memory_out[1] = 1
# KERNEL: Mem_memory_out[2] = 2
# KERNEL: Mem_memory_out[3] = x
# KERNEL: Mem_memory_out[4] = 4
# KERNEL: Mem_memory_out[5] = 5
# KERNEL: Mem_memory_out[6] = 6
# KERNEL: Mem_memory_out[7] = 7
# RUNTIME: Info: RUNTIME_0068 Arch.v (1072): $finish called.
# KERNEL: Time: 7 ps,  Iteration: 0,  Instance: /TB_Multi_Cycle_Proccesser,  Process: @INITIAL#1064_1@.

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 23: SW Instruction Test.

- Mem_memory_out[Reg(r2) + 1] = Reg(r1).

BGT & BGTZ:



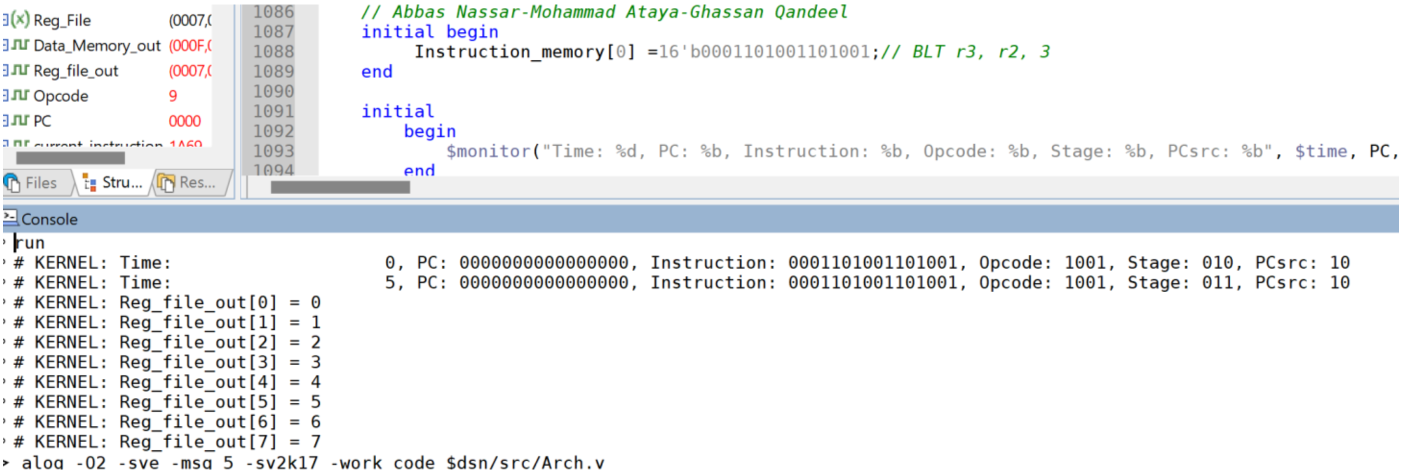
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0001101001101000;// BGT r3, r2, 3
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time, PC,
    end



endmodule
```

```
# Simulation has been initialized
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0001101001101000, Opcode: 1000, Stage: 010, PCsrc: 10
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0001101001101000, Opcode: 1000, Stage: 011, PCsrc: 10
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 24: BGT Instruction Test.

- If (r3-r2>0) PCSrc = 2 => nextPC = PC+2 + SignedExt(Imm5)
- Else PCSrc = 0 => nextPC = PC+2
- Error in updating the value of PC

BLT & BLTZ:



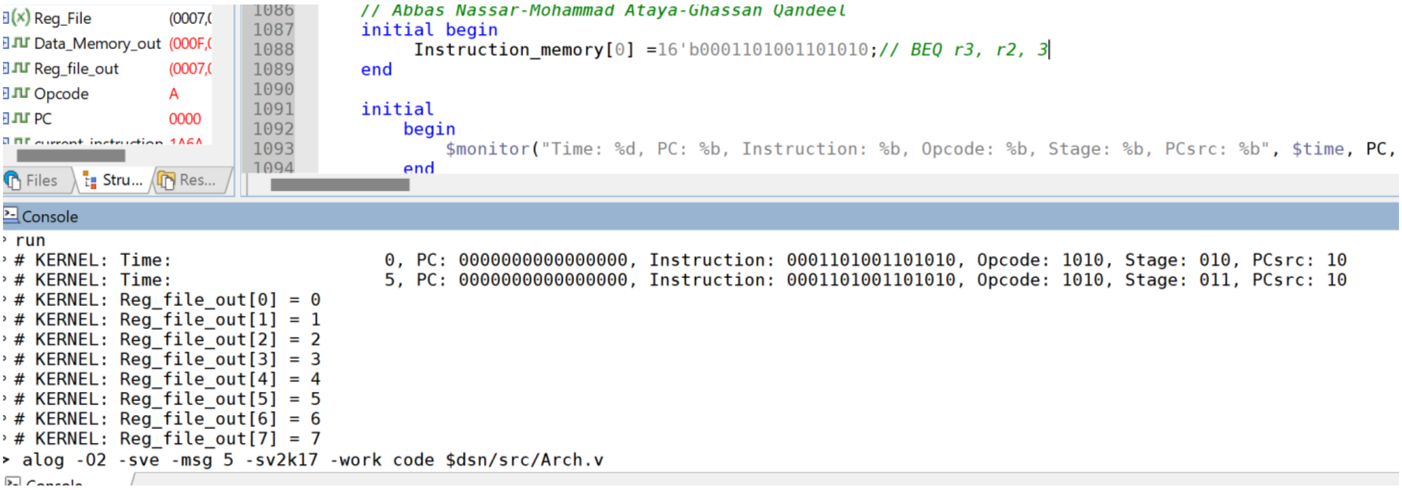
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0001101001101001;// BLT r3, r2, 3
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time, PC,
    end
```

```
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0001101001101001, Opcode: 1001, Stage: 010, PCsrc: 10
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0001101001101001, Opcode: 1001, Stage: 011, PCsrc: 10
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 25: BLT Instruction Test.

- If (r3-r2<0) PCSrc = 2 => nextPC = PC+2 + SignedExt(Imm5)
- Else PCSrc = 0 => nextPC = PC+2
- Error in PCSrc signal generation

BEQ & BEQZ:



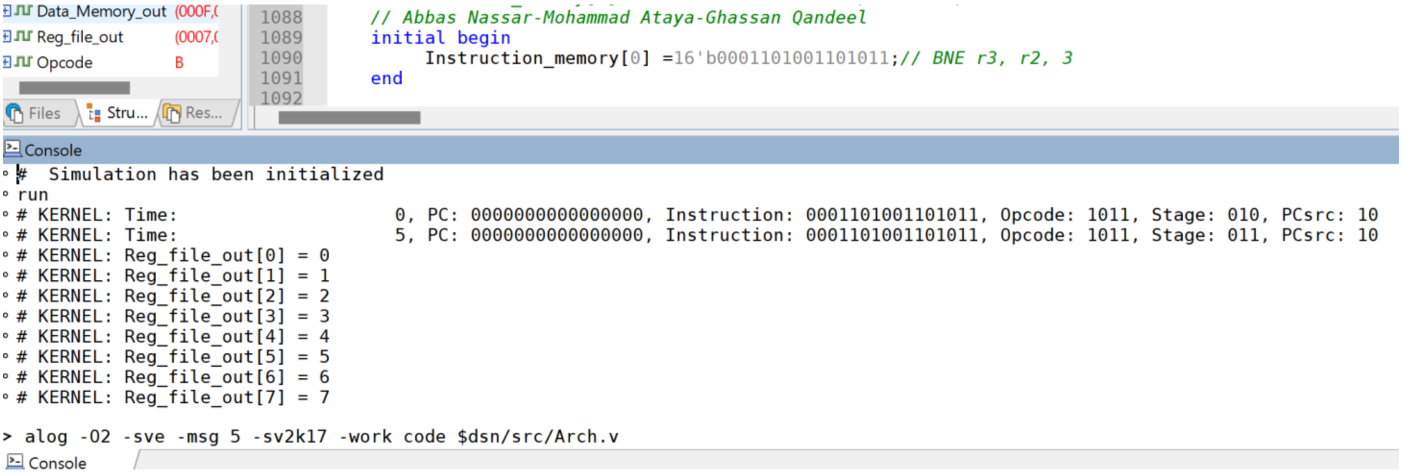
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0001101001101010;// BEQ r3, r2, 3
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time, PC,
    end
```

```
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0001101001101010, Opcode: 1010, Stage: 010, PCsrc: 10
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0001101001101010, Opcode: 1010, Stage: 011, PCsrc: 10
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 26: BEQ Instruction Test.

- If (r3==r2) PCSrc = 2 => nextPC = PC+2 + SignedExt(Imm5)
- Else PCSrc = 0 => nextPC = PC+2
- Error in PCSrc signal generation

BNE & BNEQ:



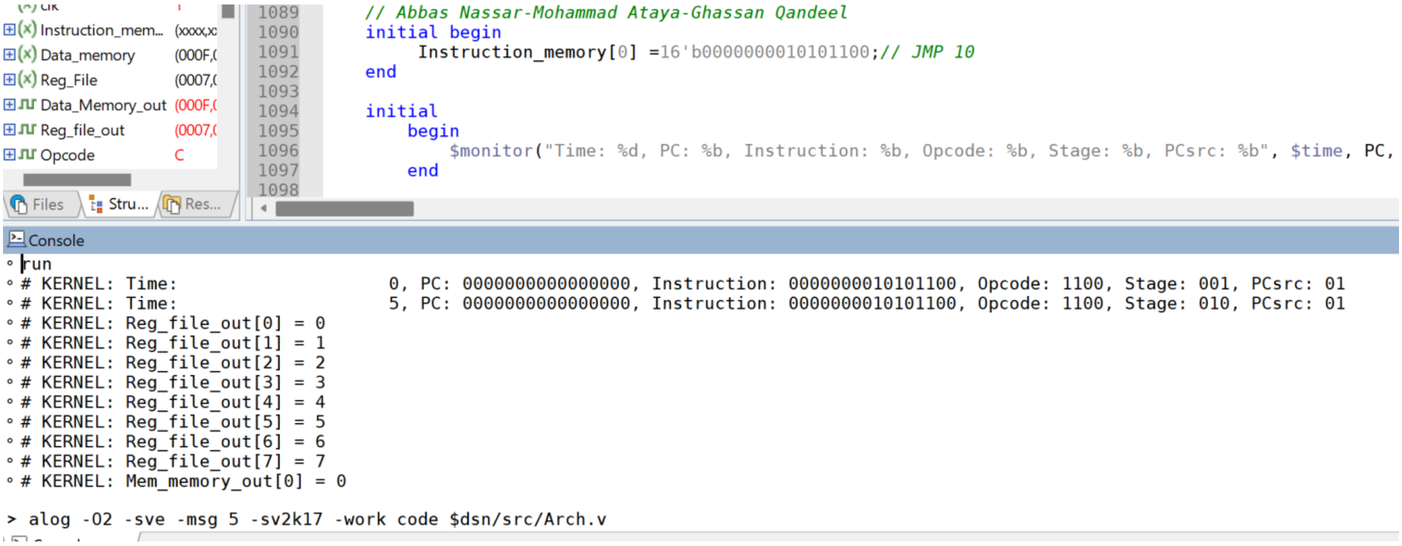
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0001101001101011;// BNE r3, r2, 3
end
```

```
#  Simulation has been initialized
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0001101001101011, Opcode: 1011, Stage: 010, PCsrc: 10
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0001101001101011, Opcode: 1011, Stage: 011, PCsrc: 10
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 27: BNE Instruction Test.

- If (r3!=r2) PCSrc = 2 => nextPC = PC+2 + SignedExt(Imm5)
- Else PCSrc = 0 => nextPC = PC+2
- Error in PCSrc signal generation
- Error in updating the value of PC

JMP:



```
Instruction_mem...   (xxxx,x
Data_memory          (000F,(
Reg_File             (0007,(
Data_Memory_out      (000F,(
Reg_file_out         (0007,(
Opcode               C
```

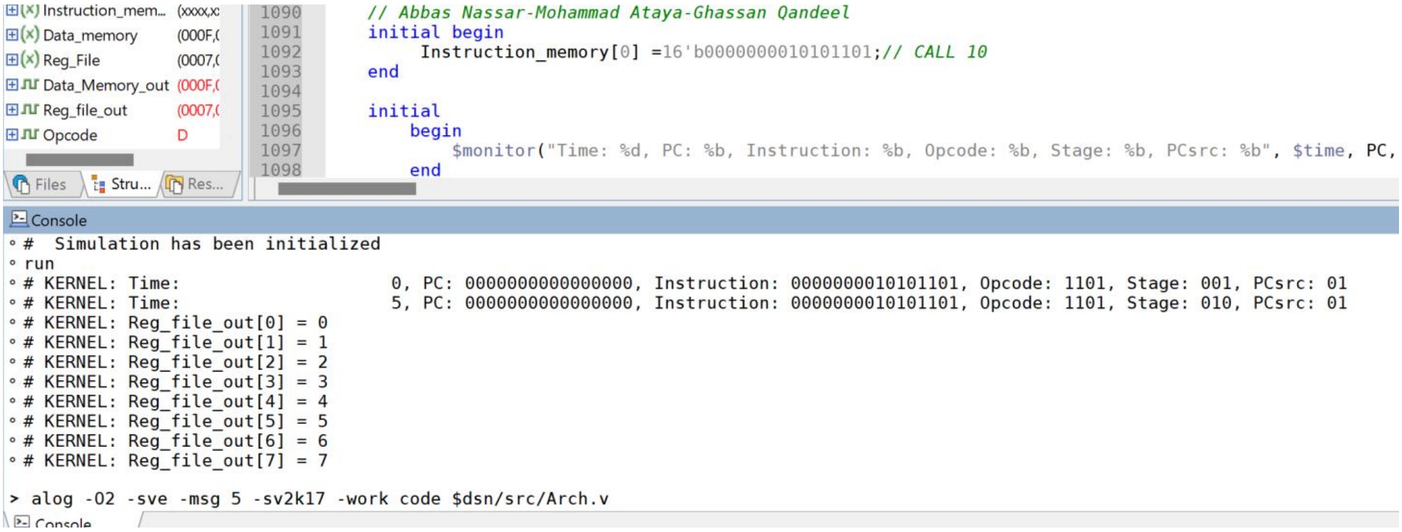
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000000010101100;// JMP 10
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time, PC,
    end
```

```
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0000000010101100, Opcode: 1100, Stage: 001, PCsrc: 01
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0000000010101100, Opcode: 1100, Stage: 010, PCsrc: 01
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
# KERNEL: Mem_memory_out[0] = 0

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 28: JMP Instruction Test.

- PCSrc = 1 => Next PC = {PC[15:10], Immediate}
- Error in updating the value of PC

CALL:



```
Instruction_mem...   (xxxx,x
Data_memory          (000F,(
Reg_File             (0007,(
Data_Memory_out      (000F,(
Reg_file_out         (0007,(
Opcode               D
```

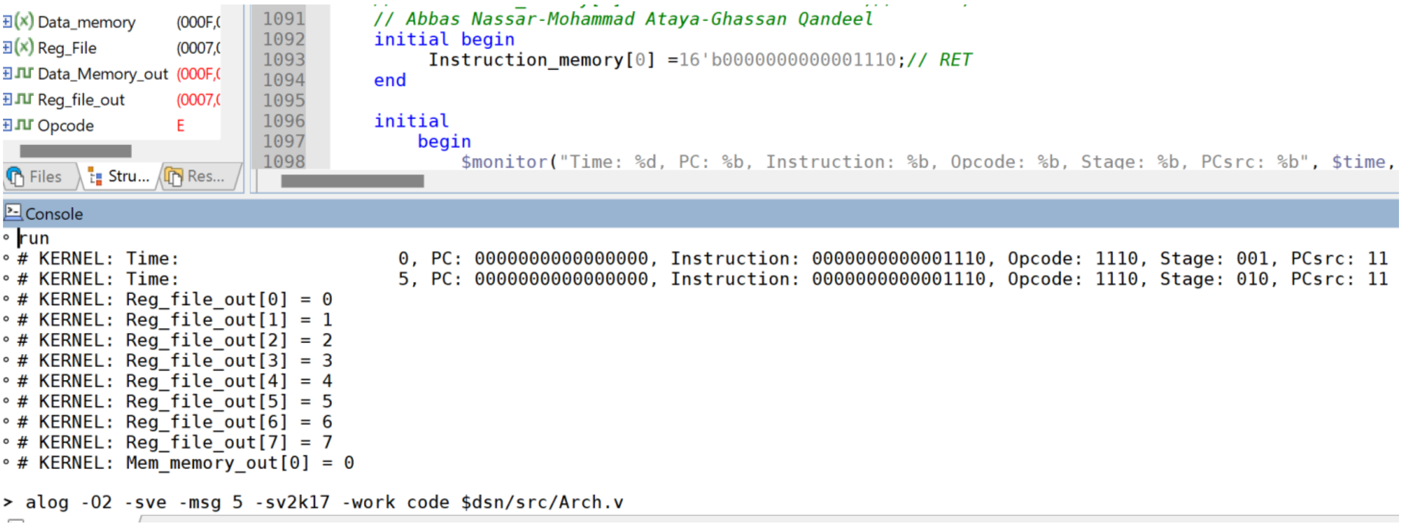
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000000010101101;// CALL 10
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time, PC,
    end
```

```
#  Simulation has been initialized
run
# KERNEL: Time:                    0, PC: 0000000000000000, Instruction: 0000000010101101, Opcode: 1101, Stage: 001, PCsrc: 01
# KERNEL: Time:                    5, PC: 0000000000000000, Instruction: 0000000010101101, Opcode: 1101, Stage: 010, PCsrc: 01
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 29: CALL Instruction Test.

- PCSrc = 1 => Next PC = {PC[15:10], Immediate}
- Reg_file_out[7] = PC + 2
- Error in updating the value of PC

RET:



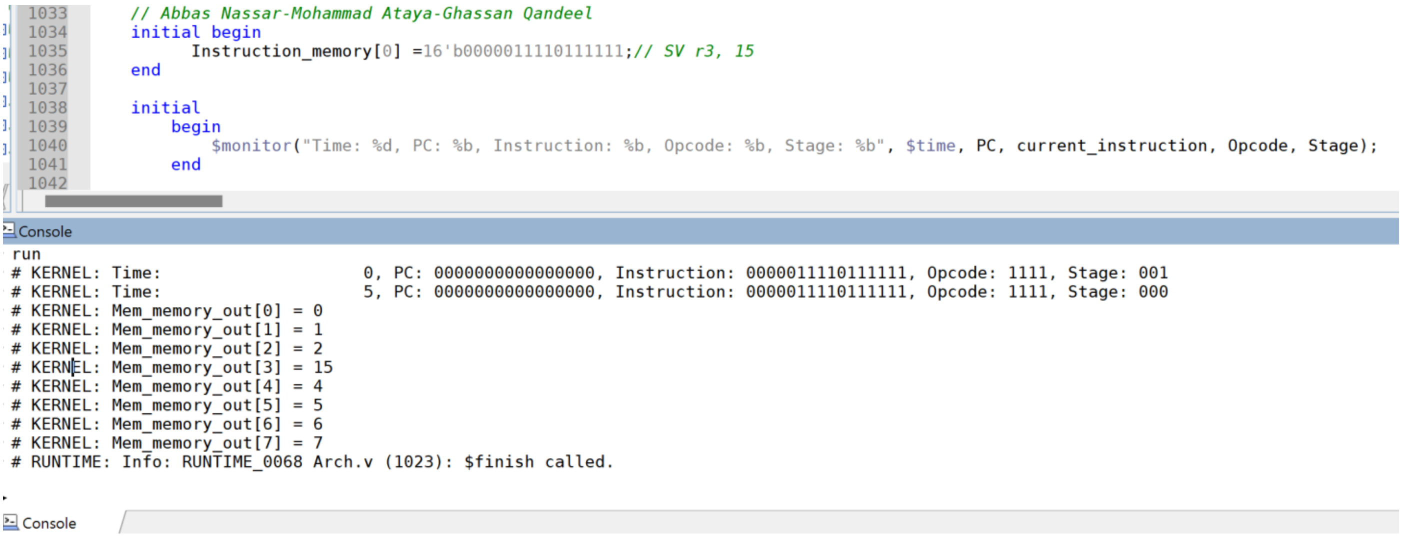
```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000000000001110;// RET
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b, PCsrc: %b", $time,
```

```
run
# KERNEL: Time:                     0, PC: 0000000000000000, Instruction: 0000000000001110, Opcode: 1110, Stage: 001, PCsrc: 11
# KERNEL: Time:                     5, PC: 0000000000000000, Instruction: 0000000000001110, Opcode: 1110, Stage: 010, PCsrc: 11
# KERNEL: Reg_file_out[0] = 0
# KERNEL: Reg_file_out[1] = 1
# KERNEL: Reg_file_out[2] = 2
# KERNEL: Reg_file_out[3] = 3
# KERNEL: Reg_file_out[4] = 4
# KERNEL: Reg_file_out[5] = 5
# KERNEL: Reg_file_out[6] = 6
# KERNEL: Reg_file_out[7] = 7
# KERNEL: Mem_memory_out[0] = 0

> alog -O2 -sve -msg 5 -sv2k17 -work code $dsn/src/Arch.v
```

Figure 30: RET Instruction Test.

- PCSrc = 3 => Next PC = Reg[r7]
- Error in updating the value of PC

Sv:



```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000011110111111;// SV r3, 15
end

initial
    begin
        $monitor("Time: %d, PC: %b, Instruction: %b, Opcode: %b, Stage: %b", $time, PC, current_instruction, Opcode, Stage);
    end
```

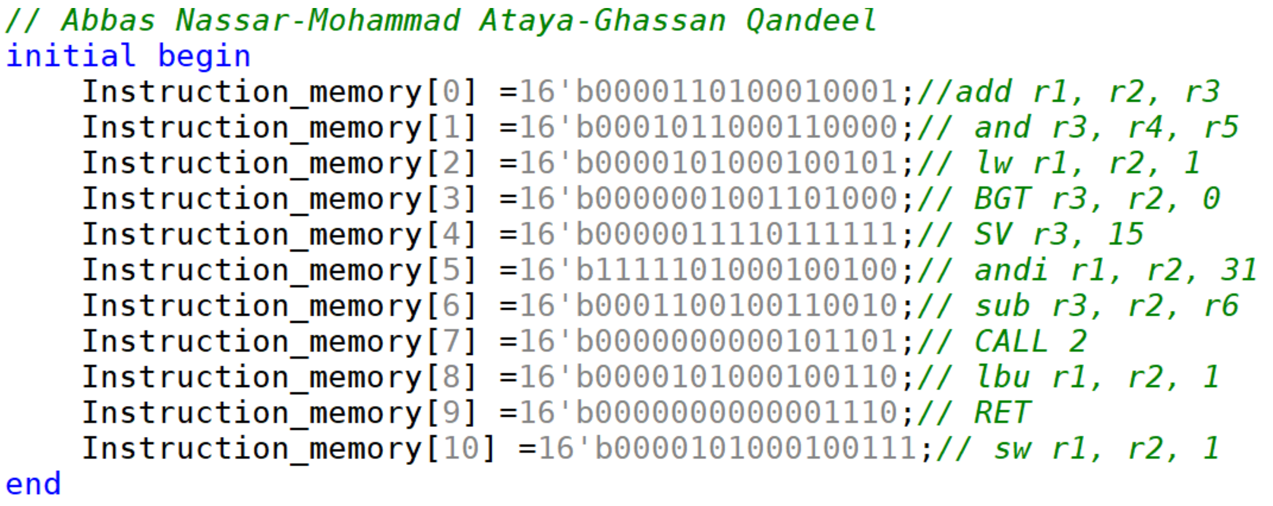
```
run
# KERNEL: Time:                     0, PC: 0000000000000000, Instruction: 0000011110111111, Opcode: 1111, Stage: 001
# KERNEL: Time:                     5, PC: 0000000000000000, Instruction: 0000011110111111, Opcode: 1111, Stage: 000
# KERNEL: Mem_memory_out[0] = 0
# KERNEL: Mem_memory_out[1] = 1
# KERNEL: Mem_memory_out[2] = 2
# KERNEL: Mem_memory_out[3] = 15
# KERNEL: Mem_memory_out[4] = 4
# KERNEL: Mem_memory_out[5] = 5
# KERNEL: Mem_memory_out[6] = 6
# KERNEL: Mem_memory_out[7] = 7
# RUNTIME: Info: RUNTIME_0068 Arch.v (1023): $finish called.
```

Figure 31: Sv Instruction Test.

- Mem_memory_out[3] = 15.

Code Like Scenario:

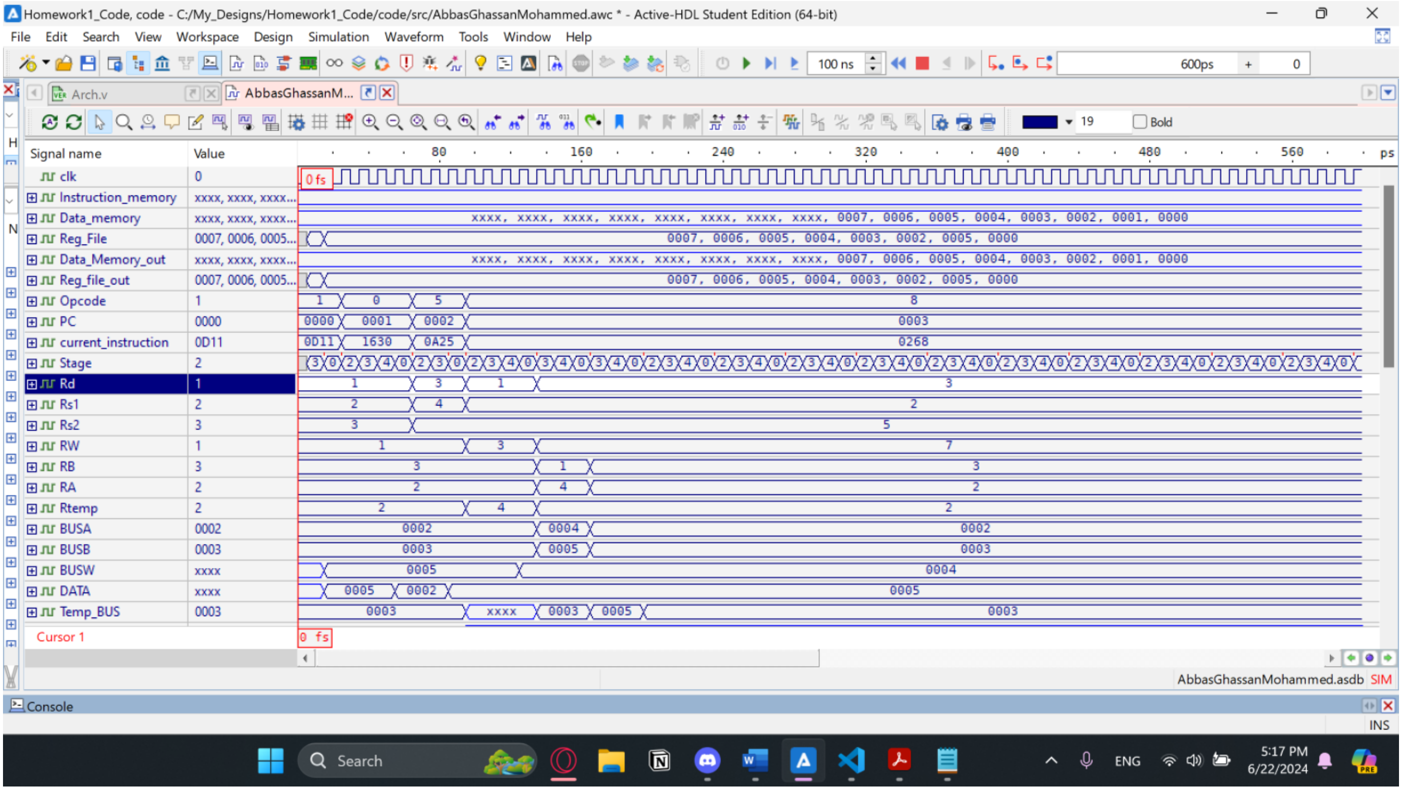
- We wrote a scenario of code:



```
// Abbas Nassar-Mohammad Ataya-Ghassan Qandeel
initial begin
    Instruction_memory[0] =16'b0000110100010001;//add r1, r2, r3
    Instruction_memory[1] =16'b0001011000110000;// and r3, r4, r5
    Instruction_memory[2] =16'b0000101000100101;// lw r1, r2, 1
    Instruction_memory[3] =16'b0000001001101000;// BGT r3, r2, 0
    Instruction_memory[4] =16'b0000011110111111;// SV r3, 15
    Instruction_memory[5] =16'b1111101000100100;// andi r1, r2, 31
    Instruction_memory[6] =16'b0001100100110010;// sub r3, r2, r6
    Instruction_memory[7] =16'b0000000000101101;// CALL 2
    Instruction_memory[8] =16'b0000101000100110;// lbu r1, r2, 1
    Instruction_memory[9] =16'b0000000000001110;// RET
    Instruction_memory[10] =16'b0000101000100111;// sw r1, r2, 1
end
```

Figure 32: Code Like Scenario.

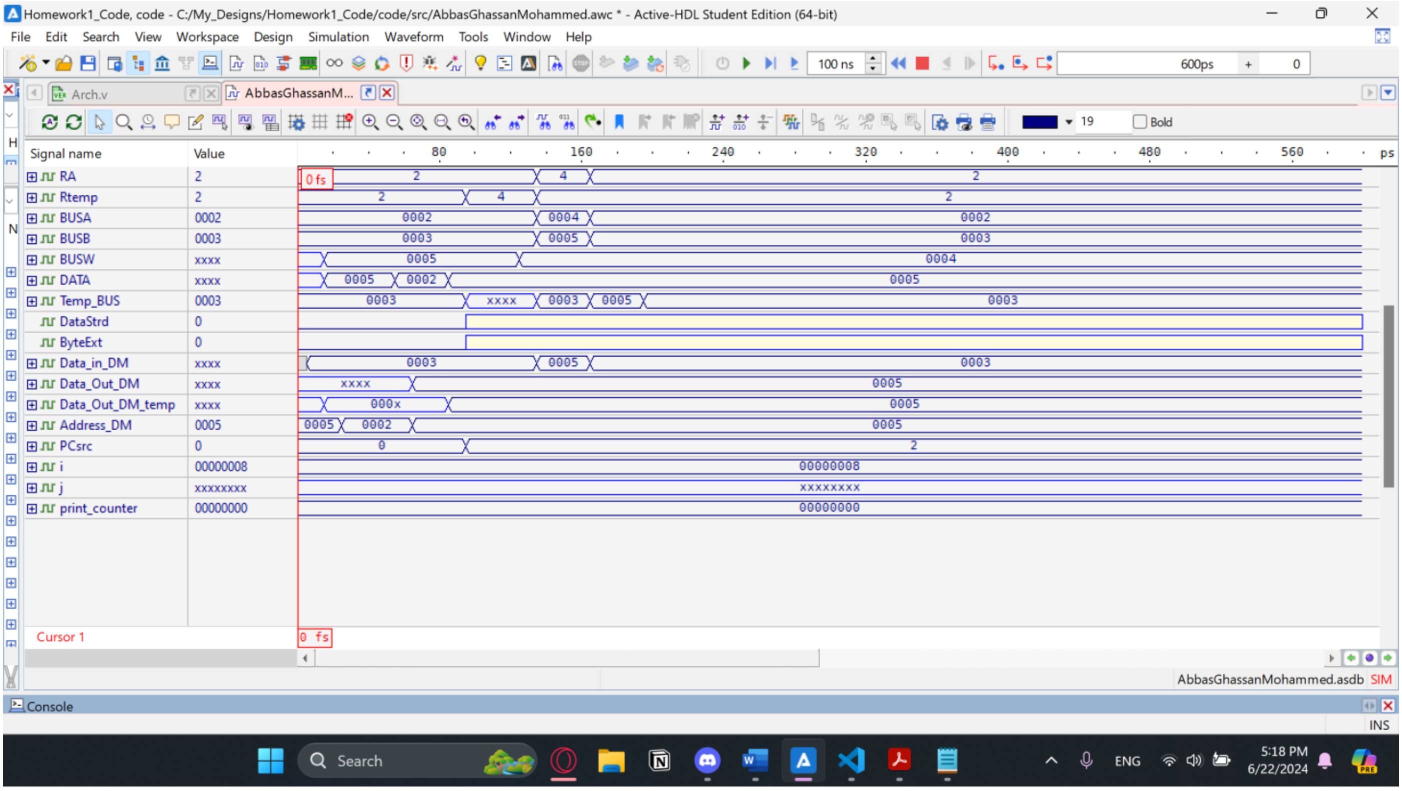
- Result:



Homework1_Code, code - C:/My_Designs/Homework1_Code/code/src/AbbasGhassanMohammed.awc * - Active-HDL Student Edition (64-bit)

Signal name	Value
clk	0
Instruction_memory	xxxx, xxxx, xxxx...
Data_memory	xxxx, xxxx, xxxx...
Reg_File	0007, 0006, 0005...
Data_Memory_out	xxxx, xxxx, xxxx...
Reg_file_out	0007, 0006, 0005...
Opcode	1
PC	0000
current_instruction	0D11
Stage	2
Rd	1
Rs1	2
Rs2	3
RW	1
RB	3
RA	2
Rtemp	2
BUSA	0002
BUSB	0003
BUSW	xxxx
DATA	xxxx
Temp_BUS	0003
Cursor 1	0 fs

Figure 33: Resulted Waveform 1.



Homework1_Code, code - C:/My_Designs/Homework1_Code/code/src/AbbasGhassanMohammed.awc * - Active-HDL Student Edition (64-bit)

Signal name	Value
RA	2
Rtemp	2
BUSA	0002
BUSB	0003
BUSW	xxxx
DATA	xxxx
Temp_BUS	0003
DataStrd	0
ByteExt	0
Data_in_DM	xxxx
Data_Out_DM	xxxx
Data_Out_DM_temp	xxxx
Address_DM	0005
PCsrc	0
i	00000008
j	xxxxxxxx
print_counter	00000000

Cursor 1 0 fs

Figure 34: Resulted waveform 2.

❖ Conclusion

In conclusion, this project successfully achieves the design and implementation of a RISC Multi-Cycle processor using Verilog. By building functional units and control units, the project supports 4 distinct instruction formats and 21 different instructions. The development of a full data-path and state diagram, along with the creation of Verilog modules, ensures the processor's functionality. The final verification through the constructed testbenches validates the design, confirming the efficacy and accuracy of the processor.

❖ References

[1]: 5-CPU Organization – Single Cycle Processor Design.pdf

Author: Dr. Aziz Qaroush.