SPECIFICATIONS

A simple single cycle implementation is used in this project where all instructions are fetched and executed in one clock cycle. Major specifications including the instructions supported are listed here.

- **Instruction Memory**

Instruction Size	: 16 bits
Address line (PC)	: 10 bits

- **Data Memory**

Data Width	: 8 bits
Address Width	: 8 bits

- **ALU :**
  8bit ALU Supporting Addition, Subtraction, OR, AND, XOR and Shift operation
- **Register File :**

R0-R7 Registers	: 8 bits each

- **Instructions Supported :**
  ADD,ADDI,SUB,AND,OR,XOR, Shift Left, Shift Right,
  LOAD,LOADI,STORE, JUMP, JUMPZ, RET and NOP
- **1 Interrupt Support**

Sub Modules:

- ALU
- Register File
- Data Memory: Distributed simple dual port RAM
- Instructions Memory: Distributed simple dual port RAM
- Control Unit and Instruction decoder

INSTRUCTION SET

Instruction set is designed considering that instruction decoding can be simple for all instruction. 16-bit instruction format is used where 5 bit is reserved as opcode for all instruction except ADDI instruction. Position of destination register, Source register1 and source Register2 is kept constant in all instructions so that decoding can be uniform between instructions.

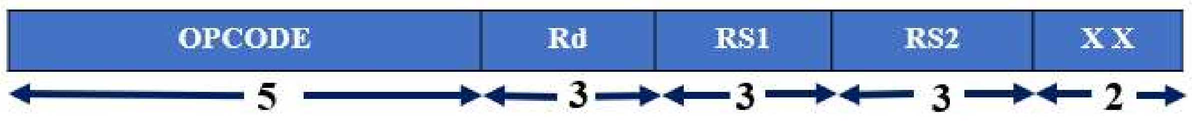
Instructions are mainly classified to 4 types.

1) R Type Instruction: - These instructions include arithmetic and logical operations on registers.
2) Memory Reference Instructions: - These are the instruction that require data memory access (read or right). Load and store instructions are included in this.
3) Branch Instructions: - Jump and Jumpz instructions are provided for unconditional and conditional jump.
4) Miscellaneous Instruction: - NOP and Return instructions are coming under this type of instructions.

Details of all these instructions with each field details are provided below.

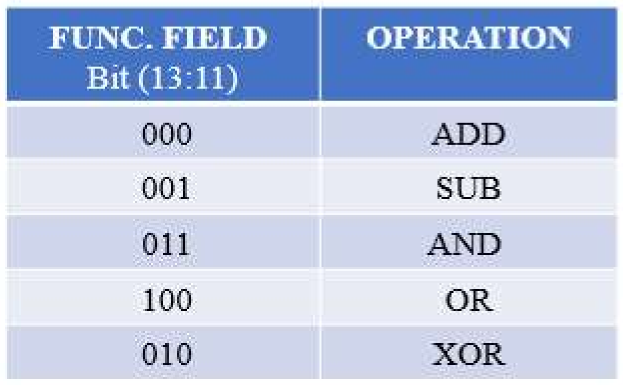
<u>R Type</u>

ADD, SUB, AND, OR, XOR



<u>Field Description:</u>

1. **Opcode :** 5 bits field:
   Bit(15:14) :10
   Bit (13:11) : Functional Field:- 3 bit field to describe the operation to be performed.
2. **Rd:** 3 bit field for destination register
3. **Rs2,Rs1 :** 3 bit field for two source register.

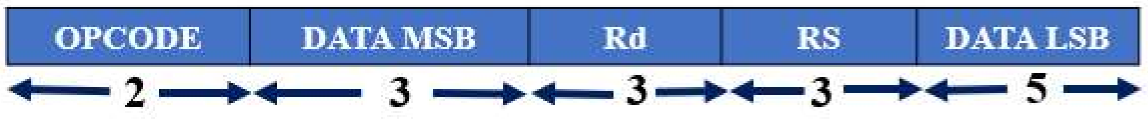


FUNC. FIELD Bit (13:11)	OPERATION
000	ADD
001	SUB
011	AND
100	OR
010	XOR

This instruction performs logical or arithmetic operations on register data given by RS1 and RS2 field of the instruction register. The ALU_Mux_Sel (IR(14)) signal will decide whether the second input of ALU will come from the register or immediate data from the instruction. For Arithmetic and Logical Instruction , ALU_MUX_Sel is 0. The ALU_OP_Sel control signal is set such that the function to be performed by ALU gets decided by functional field IR(13:11). The result of the operation gets stored in the register given by Rd field of the instruction set.

R Type

ADDI : Add Immediate Instruction



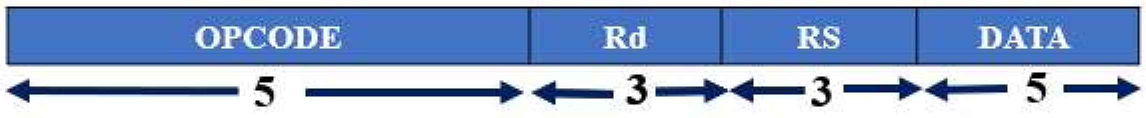
Field Description:

1. **Opcode** : 2 bits(11)
2. **Data MSB:** 3 bit field for immediate data MSB
3. **Rd:** 3 bit field for destination register
4. **RS** : 3 bit field for source register.
5. **Data LSB :** 5 bit field for immediate data LSB

This instruction performs add immediate on register data given by Rs of the instruction and immediate data IR(13:11 :: 4:0). The ALU_Mux_Sel (IR(14)) signal will decide whether the second input of ALU will come from the register or immediate data from the instruction. For immediate instruction , ALU_MUX_Sel is 1. The result of the operation gets stored in the register field given by Rd field of the instruction set.

R Type

SHIFTLEFT , SHIFTRIGHT



Field Description:

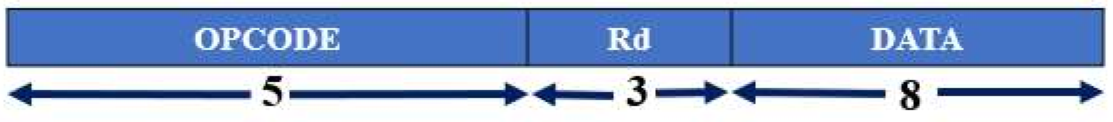
1. **Opcode** : 5 bits field:
   Bit(15:14) :**01**
   Bit (13:11) : **Functional Field**:- 3 bit field to describe the operation to be performed
2. **Rd:** 3 bit field for destination register
3. **Rs:** 3 bit field for two source register.
4. **Data :** bit field for data amount at which the shift should occur, last 3 bits is used

FUNC. FIELD Bit (13:11)	OPERATION
101	Shift Left
110	Shift Right

This instruction performs shifting operation on register data given by Rs of the instruction and shifted value IR(4:0). The IR(13:11) decides which shifted operation should be performed. The result of the operation gets stored in the register given by Rd field of the instruction.

Memory Reference Type :-

LOAD, LOADI



Field Description:

1. **Opcode** : 5 bits field:
   Bit(15:14) :**00**
   Bit (13:11) : **Functional Field**:- 3 bit field to describe the operation to be performed.
2. **Rd:** 3 bit field for destination register
3. **Data** : 8 bit field : For LOAD :- 8 bit address
   For LOADI :- 8 bit data

FUNC. FIELD Bit (13:11)	OPERATION
101	LOAD
100	LOADI

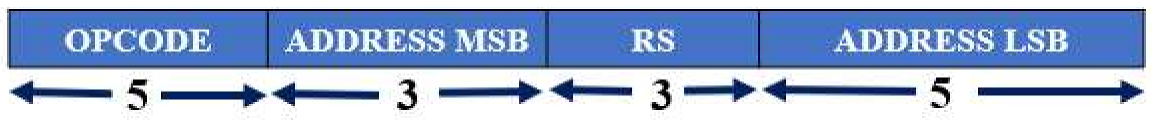
This instruction performs loading operation on registers. The IR(13:11) decided which loading operation should be performed.

For LOAD instruction, IR (7:0) will the data memory address and the content of that location will be loaded to the register given by Rd field of the instruction.

For LOADI instruction, IR (7:0) will the 8 bit data and that data will be loaded to the register given by Rd field of the instruction.

Memory Reference Type:-

STORE:- To Store the data from the register to the data memory.



Field Description:

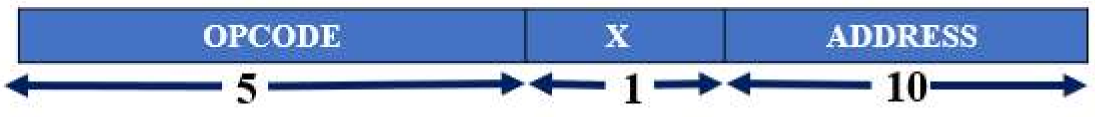
1. **Opcode** : 5 bits(00 011)

2. **Address MSB:**3 bit field for Address Location MSB
3. **RS:** 3 bit field for source register
4. **Address LSB :** 5 bit field for Address Location LSB

This instruction performs storing operation which stores the register given by Rs of the instruction to the data memory whose address is given in the IR (10:8 :: 4:0).

Branch Type :-

JUMP (Unconditional Jump)
JUMPZ (Conditional Jump if zero flag set in ALU)



Field Description:

1. **Opcode** : 5 bits field:
   Bit(15:14) :01
   Bit (13:11) : **Functional Field**:- 3 bit field to describe the operation to be performed.
2. **Address :** 10 bit field for Address Location

FUNC. FIELD Bit (13:11)	OPERATION
111	JUMP
110	JUMPZ

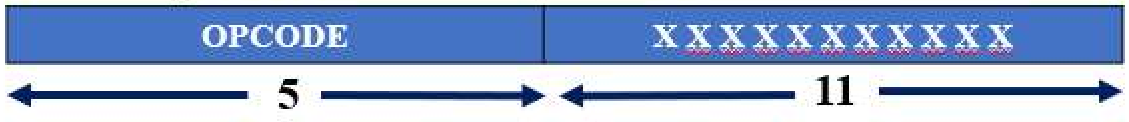
This instruction performs the Jump operation. IR(13:11) will decide which jump operation should be performed.

JUMP instruction is the unconditional jump and the address where PC have to jump is given by IR(9:0) bits.

JUMPZ instruction is the conditional jump and will happen if zero flag is set by ALU and the address where PC have to jump is given by IR(9:0) bits when the zero flag in the ALU will be set.

Miscellaneous Instruction:-

NOP:- No operation

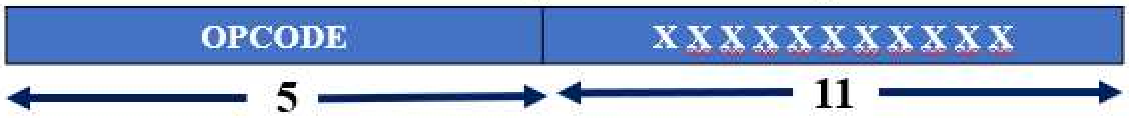


Field Description:

1. **Opcode :** 5 bits(00 000)

This instruction will perform No operation. Can be used to provide delays.

RET:- Return

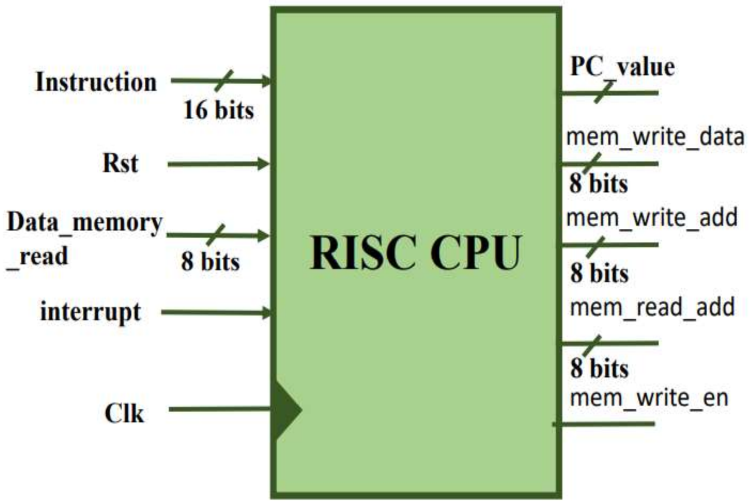


Field Description:

1. **Opcode** : 5 bits(00 001)

This instruction will be used to return from the interrupt. At the end of the ISR , once RET instruction fetched that will enable restoring of PC from the saved PC temporary register.

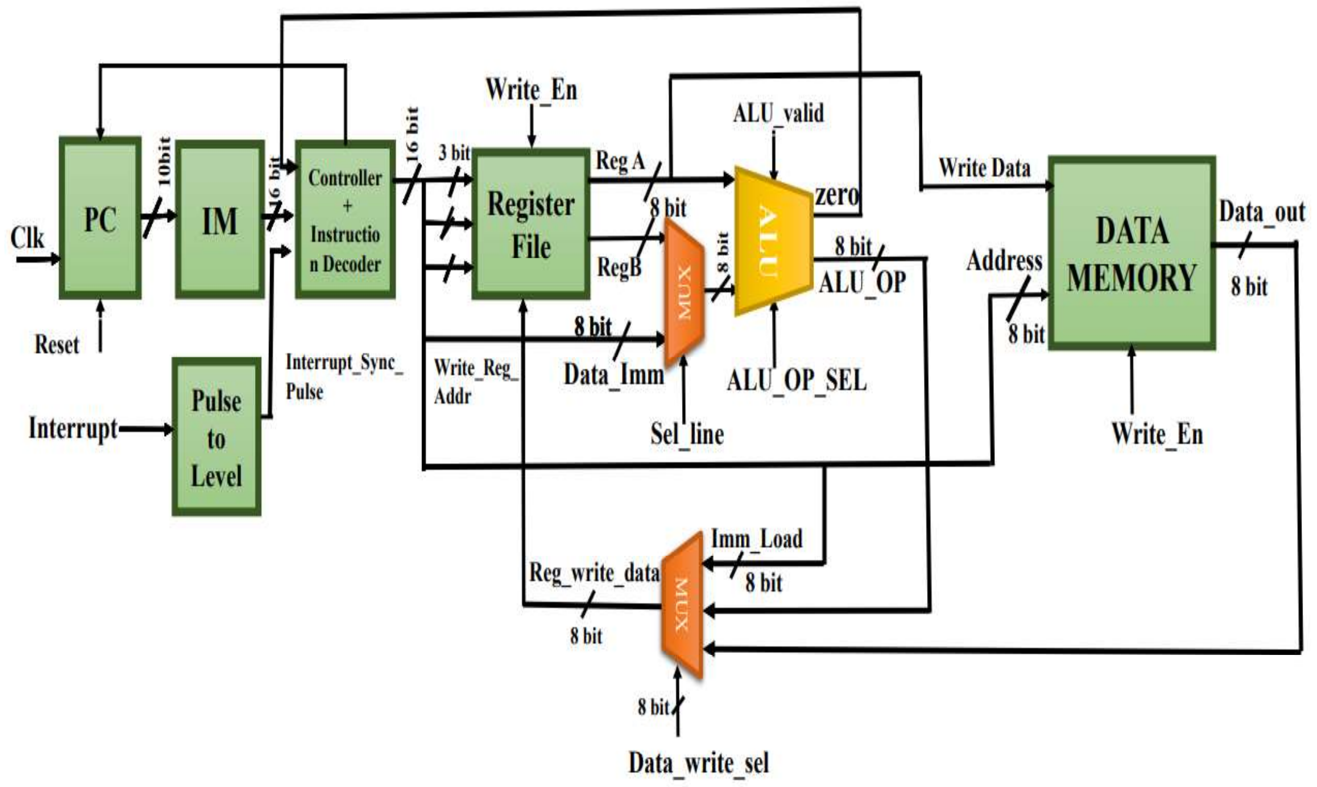
I/O PORTS



Port	Width	Mode	Description
Clk	1 bit	Input	Global clock for all the units
Rst	1 bit	Input	Global reset for all units
Data_Memory_read	8 bits	Input	8 bit out from data memory
Interrupt	1 bit	Input	External device interrupts processor through this port
Instruction	16 bits	Input	16 bit instruction read from instruction memory
PC_Value	10 bits	Output	Current Program counter to Instruction Memory
Mem_write_data	8 bits	Output	8 bit data to Data memory
Mem_write_add	8 bits	Output	Write address to data memory
Mem_read_add	8 bits	Output	Read address to data memory
Mem_write_en	8 bits	Output	Data memory write enable signal

Input port and output port details of single cycle RISC processor is tabulated ab

Single Cycle Data Path



A simplified data path of total design is given in the figure. Once reset is deasserted, Processor will start executing instruction form 0x00 address locations. PC will be updated in each clock cycle (Usually increment by one or change to the branch address if Jump operations are executed). PC is given as address to the instruction memory which is having an unregistered read operation. Once the instruction is read that information will be decoded by decoder and required control signals such as source register 1 address, source register 2 address, write register address, ALU operation selection bits etc will be decoded and given to data path elements. Datapath elements operate based on this control signal and execute the operation and write back to register if required(based on registers).

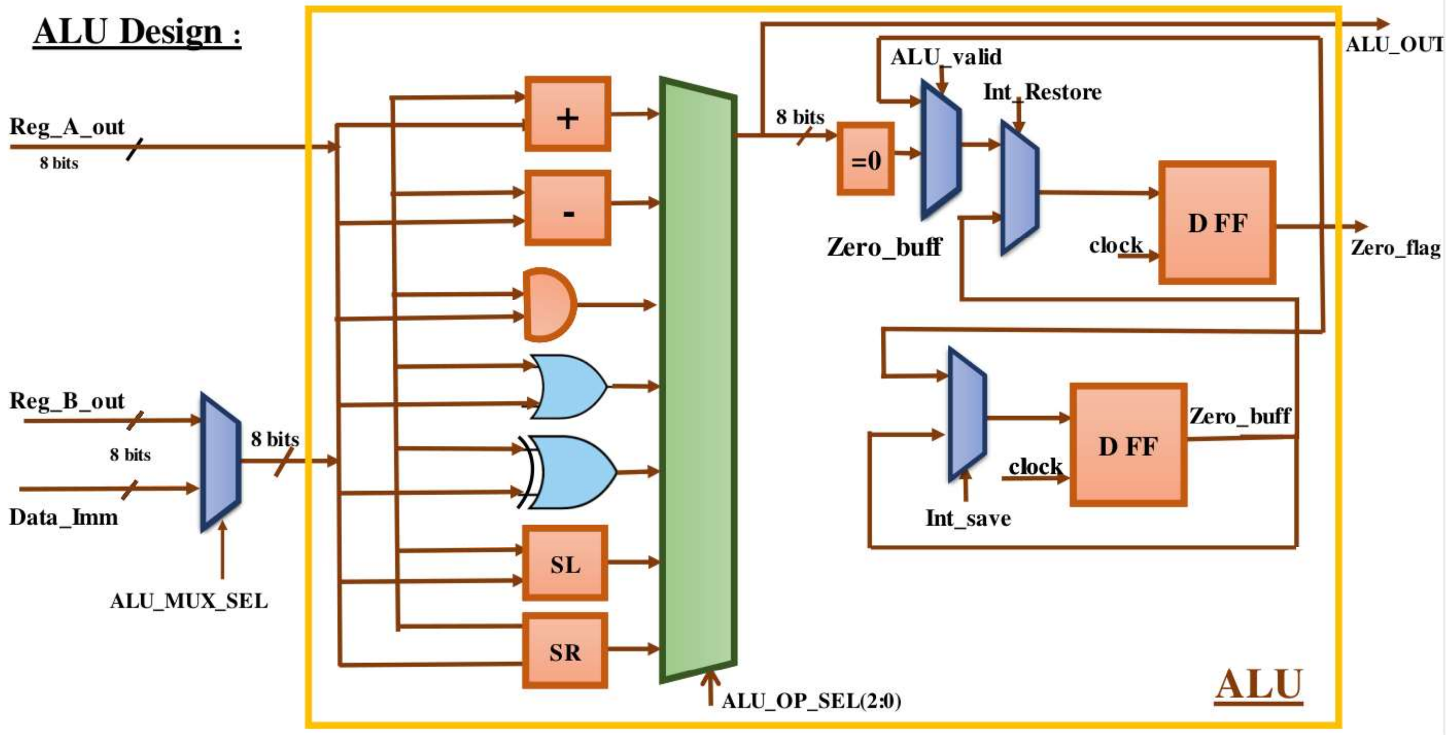
In case of an ADD instruction, RS1, RS2 and RD fields of instruction gives the address of the source registers and destination register to which result to be written. ALU operation will be indicated by the Instruction(13:11) field and ALU will compute the added result and Reg_Write Mux will be selected such that RD register will be written with ALU output in the clock edge. Similarly other instructions also will work

Functional Units

ALU:-

The ALU operation are ADD, ADDI, SUB, AND, OR, XOR, Shift Left , Shift Right and the operations is selected based on the ALU_OP_SEL signal . There are two input to ALU where the first one is always Reg_A_Out from the register file. Second input can be from register file or it can be an immediate data which is a part of instruction itself (ADDI instruction). Second input is selected using mux which is having ALU_MUX_SEL as the select line.

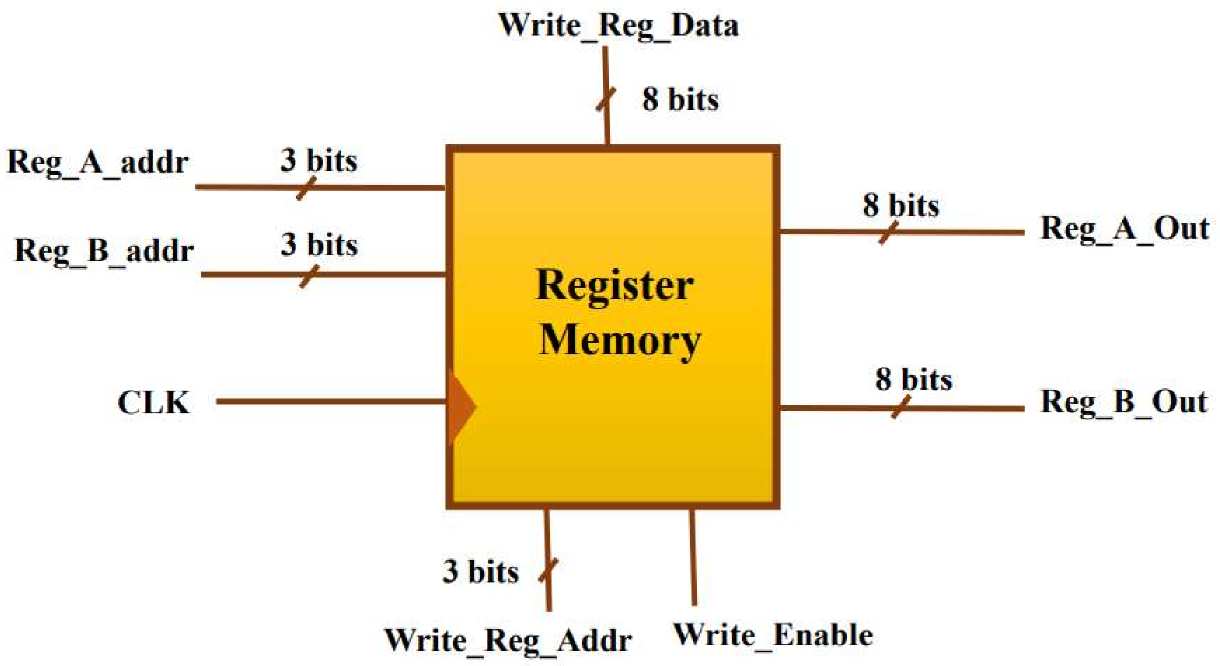
ALU is having a Zero flag as an additional output which indicates ALU output is 0x00. This flag is used to implement the JUMPZ instruction. JUMPZ instruction can be used following with any ALU operation. Zero flag is registered so that non valid ALU operation such as LOAD, LOADI etc can't change the zero-flag information (Controlled by ALU_Valid). Also, Zero flag will be saved to a temporary buffer when an interrupt happens and can be restored while returning from interrupt (Controlled by Int_Restore and Int_Save signals). Block diagram of ALU is given below.



Register File:-

Register File Contains 8 register of 8-bit width each. Write is synchronous to clk with Write_Enable signal controls the write to register and the register to which data to be written is provided by the Write_Reg_addr port and the data which is to be written is provided by Write_Reg_Data port.

Read is asynchronous and 2 read ports (Reg_A_Out and Reg_B_Out) are available to read. Any registers can be read parallelly by specifying the register address which the user want to read on Reg_A_addr and Reg_B_addr.



Reg Write Mux:-

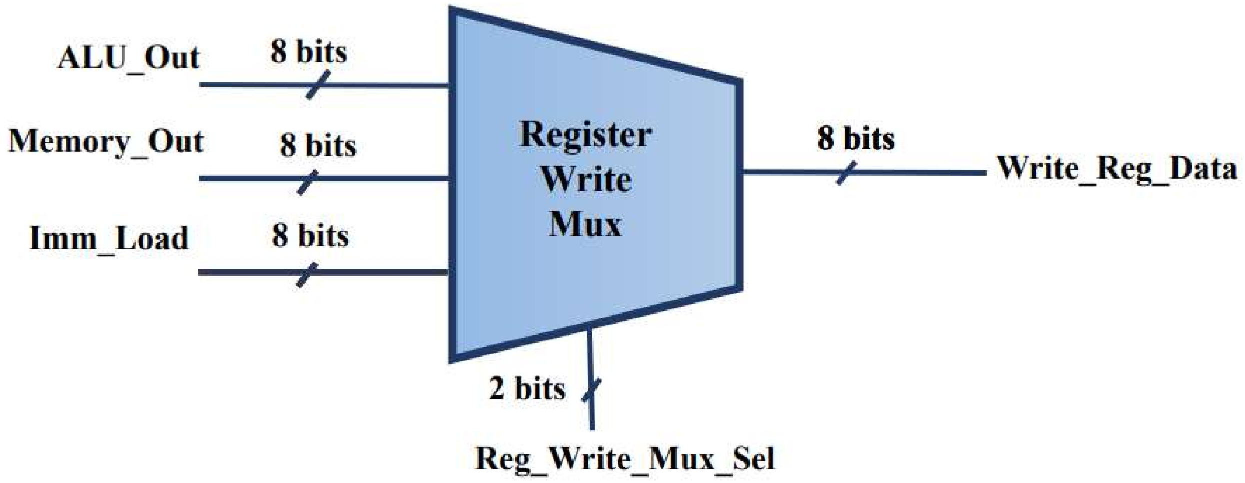
Reg Write Mux determines from where the data is coming to write to the register. There are 3 different ways by which register data can be updated.

ALU_Out:- This is the output of ALU and during Arithmetic and Logical Instruction, this data is to be written to the destination register.

Memory_Out:- This is the output of data memory and during load instruction ,this data is to be written to the destination register.

Imm_Load :- This is the immediate data directly from the instruction and during immediate load instruction (LOADI) this data is to be written to the destination register.

Reg_Write_MUX is used to select between these three input based on the Reg_write_Mux_Sel control signal. Control signal will be derived from the instruction by the decoder.

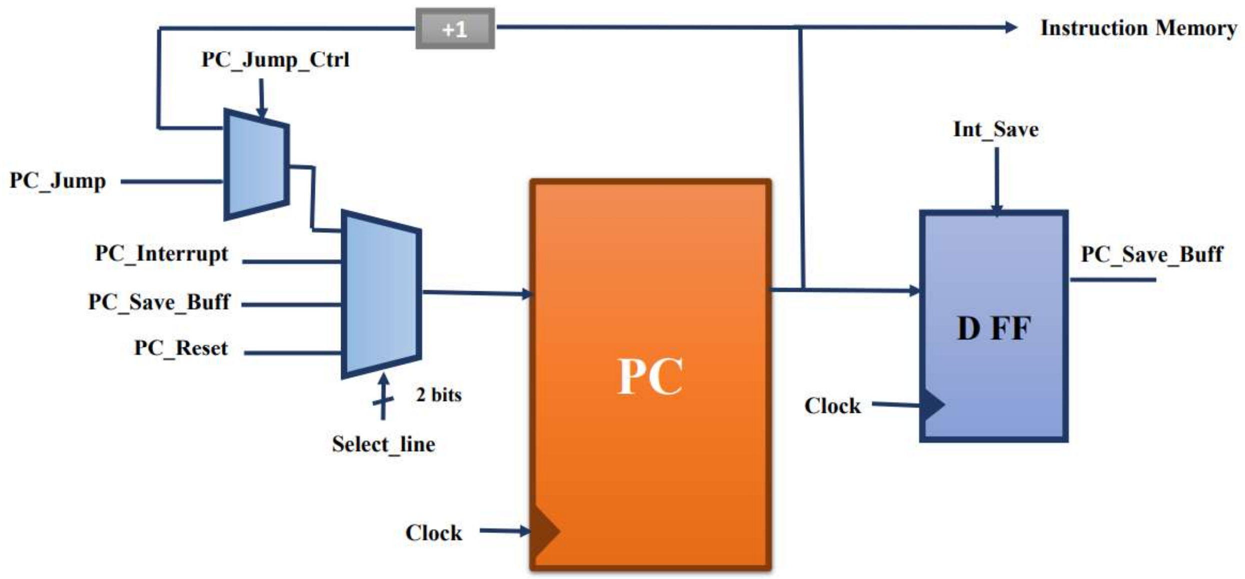


Reg_Write_Mux_Sel	Write_Reg_Data
00	ALU_Out
01	Memory_Out
10	Imm_Load
11	ALU_Out

Program Counter: -

Program counter is a 10-bit register which is having the address of the current instruction that is being executed. It is updated in each clock cycle. During normal operations it will be incremented by one or updated to branch address if an jump instruction is executed.

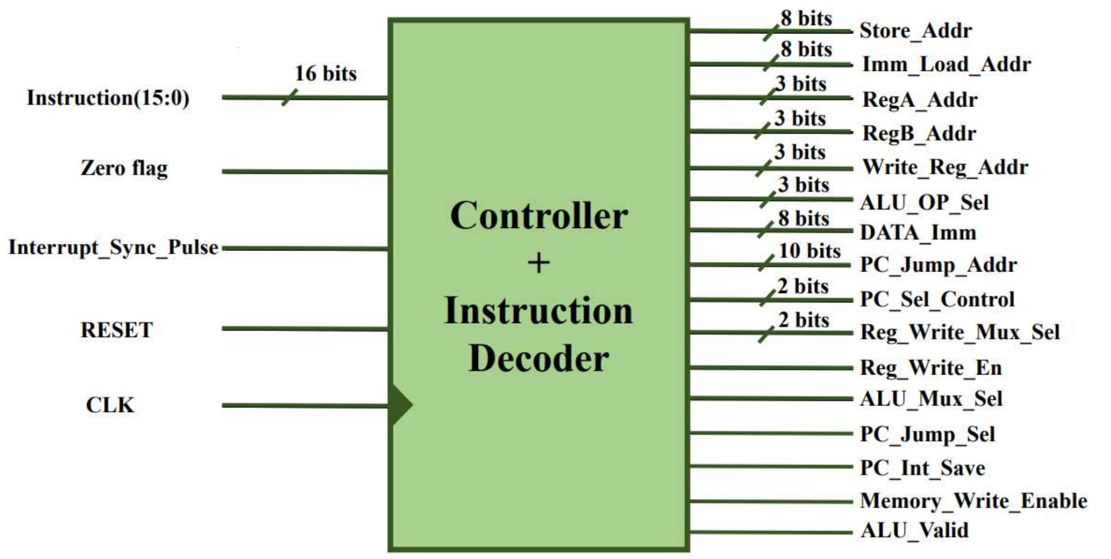
Once an interrupt signal is acknowledged by the system, PC will be updated to fixed ISR address 0x300 and current PC value will be stored to PC_Save_Buff. These are controlled by the PC_MUX_Sel and Int_Save signals. Similarly after executing the ISR, once return instruction is fetched then saved PC value will be restored so that main thread operation can be continued from where we left. If any other context to be saved during ISR, user must use LOAD and store instruction for context saving and restoring.



Control Unit

Control unit takes Instruction, Interrupt pulse status and zero flag status and generate the adequate control signals required for operation. Control signal details are tabulated below.

Controller + Instruction Decoder :-



Description of Control Signals:-

1-bit Control Signal :

Signal \ Value	0	1
Reg_Write_En	No Operation	Write data written to destination register in clock edge
ALU_Mux_Sel	2nd Input of ALU from Reg B	2nd Input of ALU from immediate data in instruction
PC_Jump_Sel	PC=PC+1	PC=Jump Address
ALU_Valid	Not a valid ALU operation, Zero flag wont get updated	Indicate Valid ALU operation
PC_Int_Save	PC_Save buffer retains the old value	PC_Save_Buff updates with current PC value
Memory_Write_Enable	No Operation	For Store instruction(When user wants to write in the data memory)

2-bits Control Signal :

Signal \ Value	00	01	10	11
PC_Sel_Control	Reset Mode	Normal Operation Mode	When Interrupt Comes, PC updates with 0x300	Returning from the Interrupt PC updates with saved PC value from buffer
Reg_Write_Mux_Sel	write_reg_data<= ALU_Out (R Instruction)	write_reg_data<= Memory_Out (Load operation)	write_reg_data<= Imm_Load (LoadI operation)	write_reg_data<= ALU_Out

3-bits Control Signal :

Signal \ Value		
RegA_Addr	Source Register 1 Address to Register File	Instruction(7:5)
RegB_Addr	Source Register 2 Address to Register File	Instruction(4:2)
Write_Reg_Addr	Destination Register Address to Register File	Instruction(4:2)
ALU_OP_Sel	000: Add Immediate and Add Instruction 001: Subtract Instruction 010: Xor Instruction 011: Or Instruction 100: AND Instruction 101: Shift Left 110: Shift Right 111: No operation	"111" during Interrupt_Save & restore state "000" if ADDI instruction Else Instruction(13:11)

8-bits Control Signal :

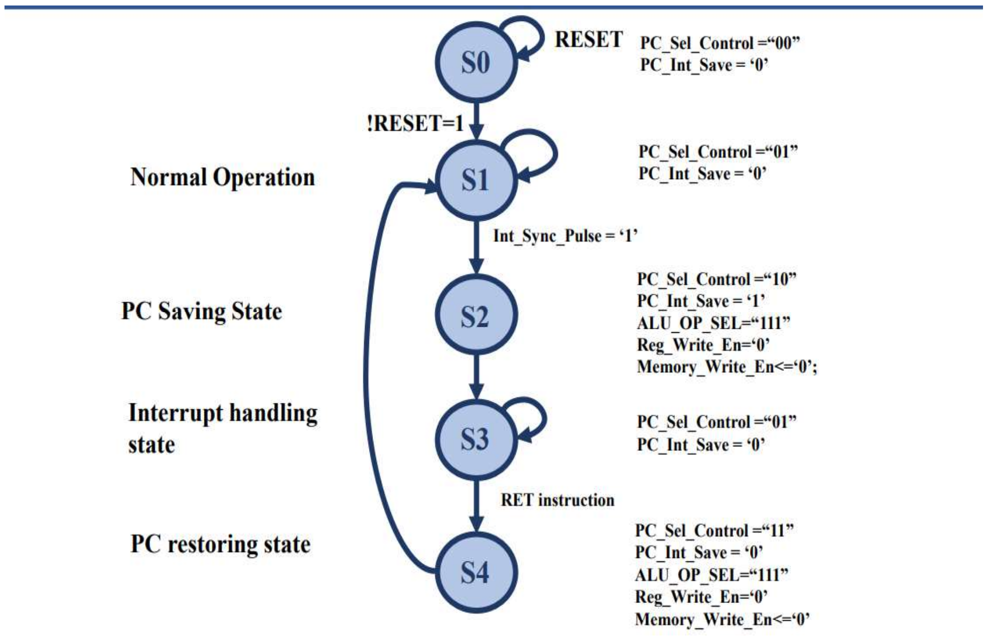
Value / Signal		
Store_Addr	Write Address to Data memory during store instruction	Instruction(10:8)&Instruction(4:0)
Imm_Load_Add	Immediate Data for LOADI instruction Data Memory Read Address for LOAD Instruction	Instruction(7:0)
DATA_Imm	Immediate Data for ADDI Instruction, input to the ALU Mux	Instruction(13:11)& Instruction(4:0)

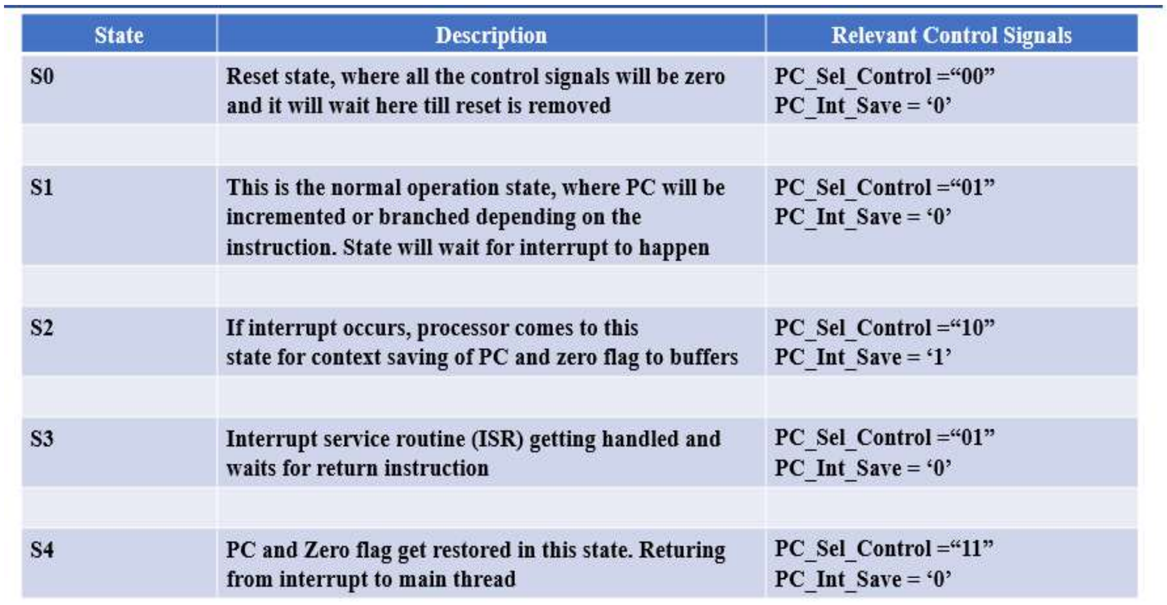
10-bits Control Signal :

Value / Signal		
PC_Jump_Addr	Address to branch during a JUMP or JUMPZ	Instruction(9:0)

FSM State Diagram:-

A simple State machine is used for controlling the context saving and restoring during interrupt. FSM diagram and states explanation is given below.

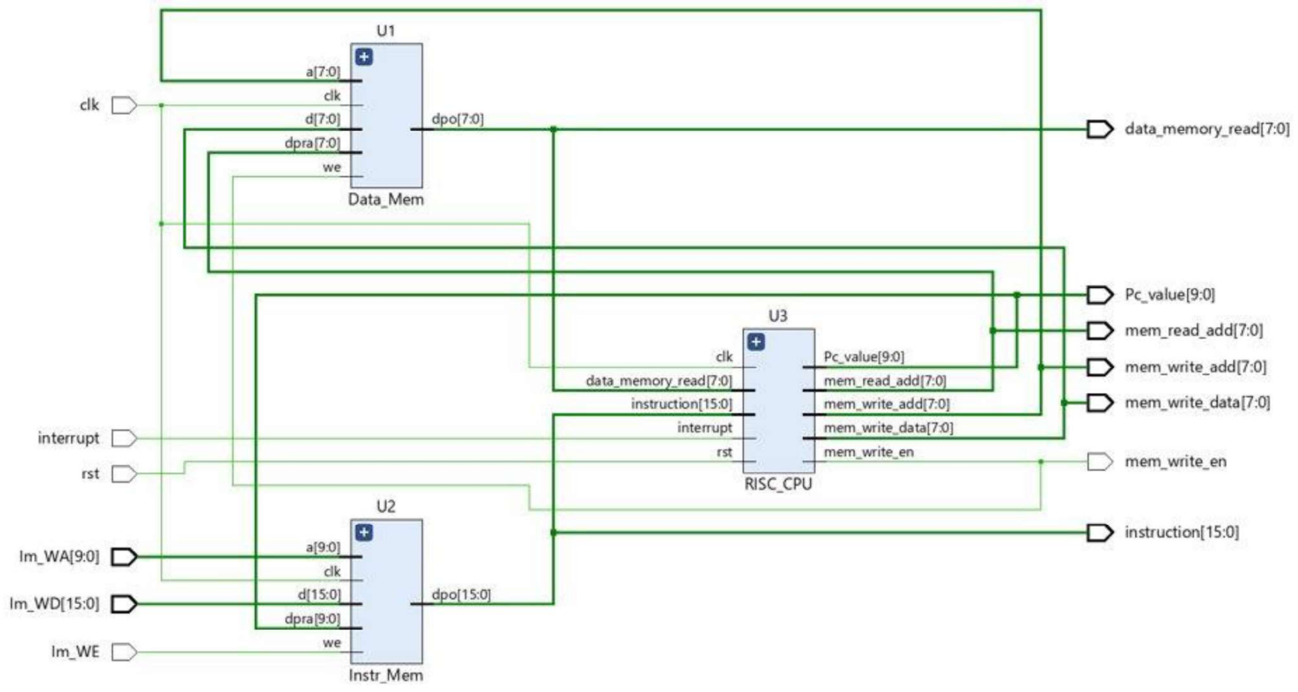




State	Description	Relevant Control Signals
S0	Reset state, where all the control signals will be zero and it will wait here till reset is removed	PC_Sel_Control ="00" PC_Int_Save = '0'
S1	This is the normal operation state, where PC will be incremented or branched depending on the instruction. State will wait for interrupt to happen	PC_Sel_Control ="01" PC_Int_Save = '0'
S2	If interrupt occurs, processor comes to this state for context saving of PC and zero flag to buffers	PC_Sel_Control ="10" PC_Int_Save = '1'
S3	Interrupt service routine (ISR) getting handled and waits for return instruction	PC_Sel_Control ="01" PC_Int_Save = '0'
S4	PC and Zero flag get restored in this state. Returing from interrupt to main thread	PC_Sel_Control ="11" PC_Int_Save = '0'

Functional Verification

In order to test the Processor Module, Data memory and instruction memory are required. Simple dual port RAM with separate read and write ports generated using distributed RAM block of Xilinx FPGA is used for this purpose. Instruction memory write port is used in testbench for writing to instruction memory from the text file with instruction. Once all instruction memory is written reset signal to the Processor is deasserted and processor module will start functioning. Schematic of top module instantiated with Instruction memory and Data memory is given below.



Testing Code:-

In order to test the module, A sample program for summing first 15 natural number was written in instruction memory using Jump and Jumpz instructions. Also in interrupt service routine two registers will be loaded with AA and E0 and bit wise OR operation will be done and the result will be saved to 0x03 address. Sum result will be stored in 0x00. The answers are verified using load instruction at the end of code in both these locations.

Sum of first 15 Natural Numbers:-

Main Code

```
0x0000: NOP
0x0001: LOADI R0,00H
0x0002: LOADI R1,00H
0x0003: LOADI R2,15H
0x0004: LOADI R3,01H
(LOOP)
0x0005: ADDI R0,01H
0x0006: ADD R1,R0,R1
0x0007: SUB R2,R2,R3
0x0008: JUMPZ FINISH
0x0009: JUMP LOOP
(FINISH)
0x000A: STORE R1,00H
0x000B: NOP
0x000C: LOAD R5,00H
0x000D: LOAD R7,03H
0x000E: NOP
0x000F: NOP
0x0010: JUMP #0EH
```

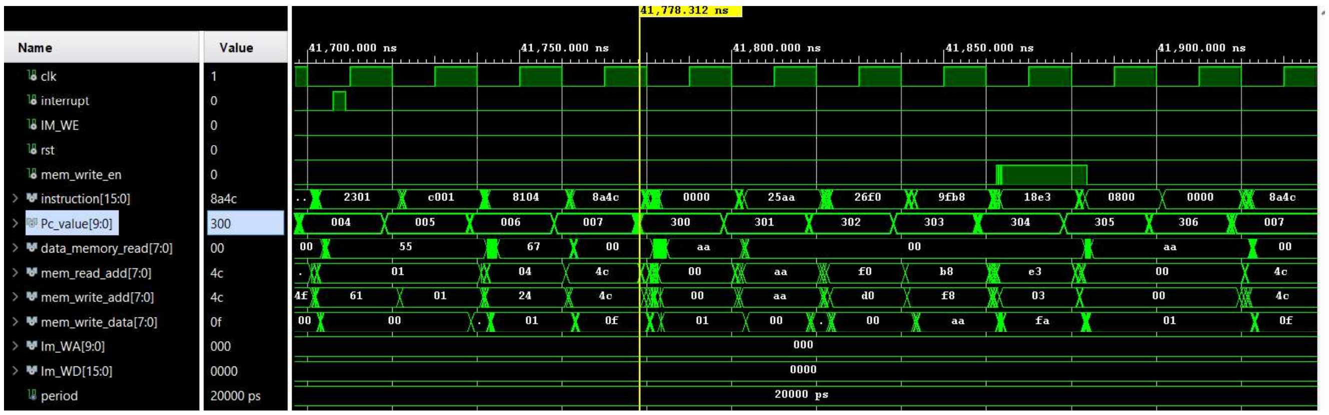
Interrupt

```
0x0300 : NOP
0x0301 : LOADI R5, AA
0x0302 : LOADI R6, E0
0x0303 : OR R7, R5, R6
0x0304 : STORE R7, #03
0x0305 : RET
```

OR, AA and E0 and store in data memory location 03

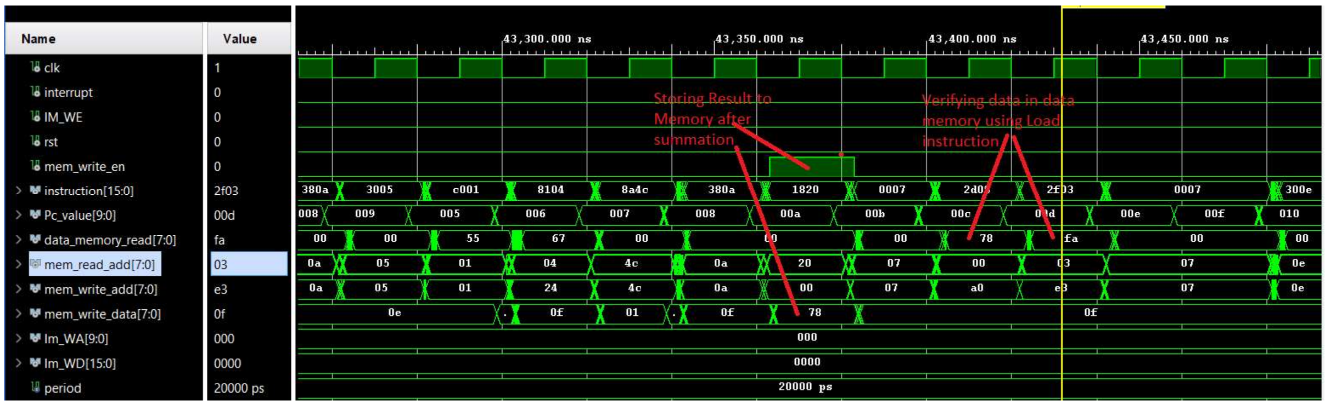
Timing Simulation Result:-

Interrupt servicing and Return



Name	Value
clk	1
interrupt	0
IM_WE	0
rst	0
mem_write_en	0
instruction[15:0]	8a4c
Pc_value[9:0]	300
data_memory_read[7:0]	00
mem_read_add[7:0]	4c
mem_write_add[7:0]	4c
mem_write_data[7:0]	0f
Im_WA[9:0]	000
Im_WD[15:0]	0000
period	20000 ps

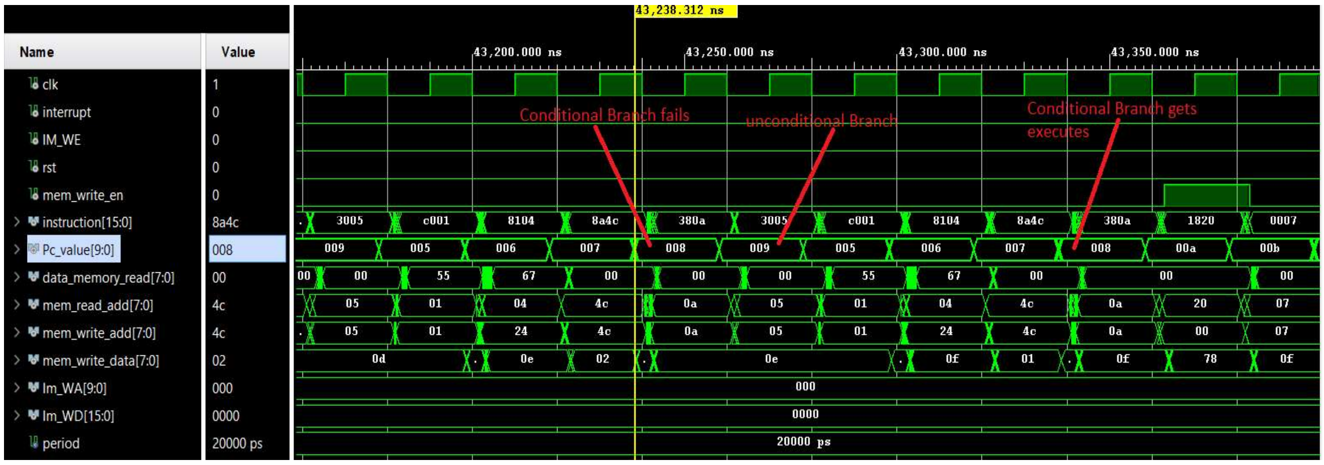
Store and Load Instruction verification



43,300.000 ns
43,350.000 ns
43,400.000 ns
43,450.000 ns
Storing Result to Memory after summation
Verifying data in data memory using Load instruction

Name	Value
clk	1
interrupt	0
IM_WE	0
rst	0
mem_write_en	0
instruction[15:0]	2f03
Pc_value[9:0]	00d
data_memory_read[7:0]	fa
mem_read_add[7:0]	03
mem_write_add[7:0]	e3
mem_write_data[7:0]	0f
Im_WA[9:0]	000
Im_WD[15:0]	0000
period	20000 ps

Conditional (JUMPZ) and unconditional JUMP instructions



Name	Value
clk	1
interrupt	0
IM_WE	0
rst	0
mem_write_en	0
instruction[15:0]	8a4c
Pc_value[9:0]	008
data_memory_read[7:0]	00
mem_read_add[7:0]	4c
mem_write_add[7:0]	4c
mem_write_data[7:0]	02
Im_WA[9:0]	000
Im_WD[15:0]	0000
period	20000 ps

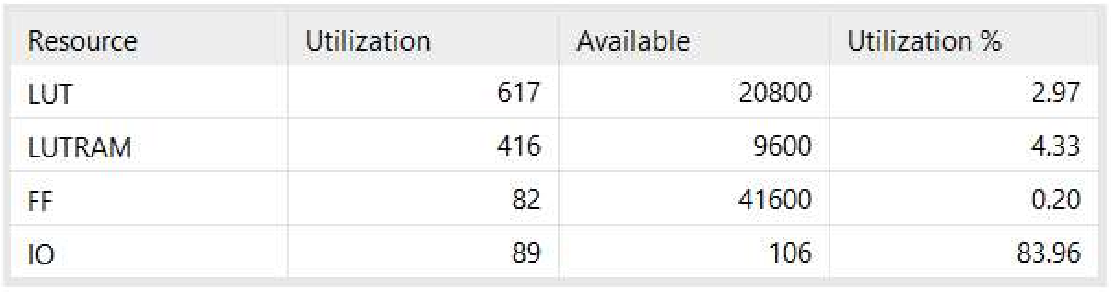
Results

<u>TARGET DEVICE:</u>

xc7a35tcpg236-1

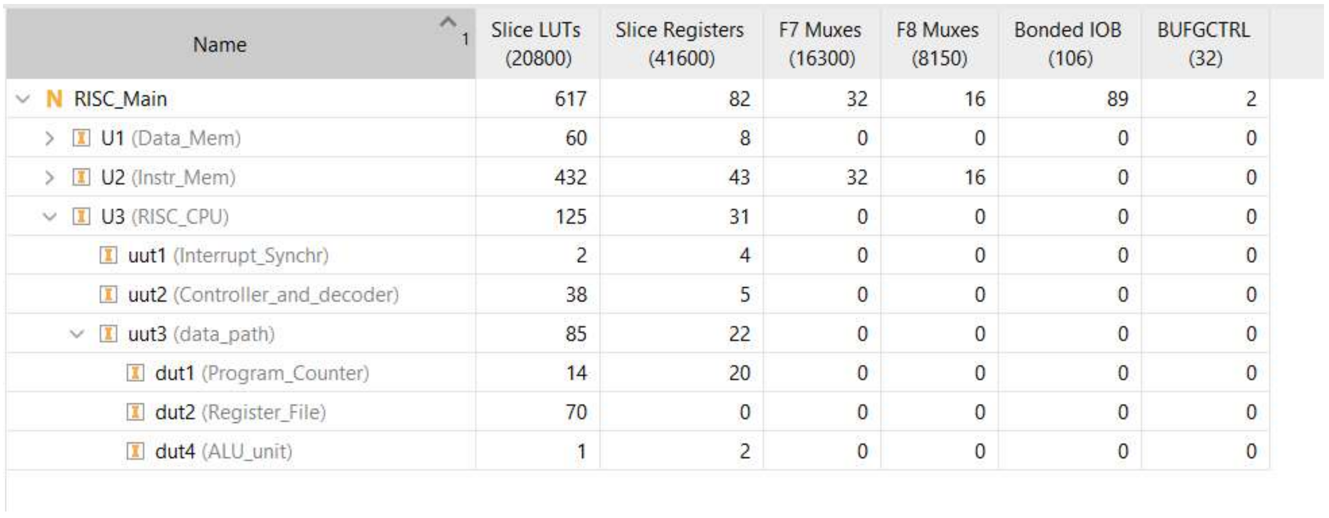
<u>Resource Utilisation:-</u>

Resource Utilization including Data Memory and Instruction Memory



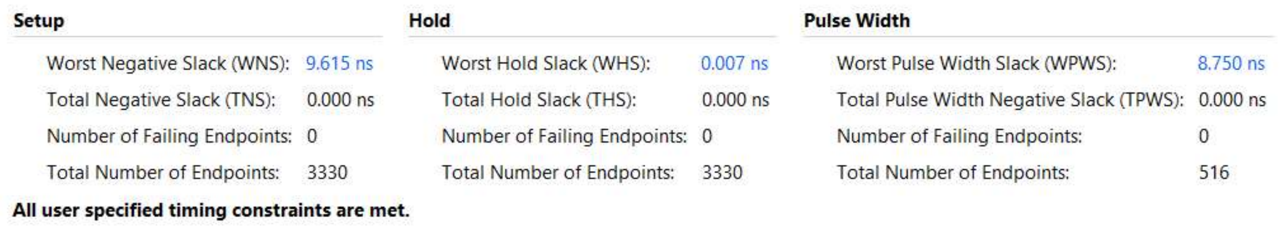
Resource	Utilization	Available	Utilization %
LUT	617	20800	2.97
LUTRAM	416	9600	4.33
FF	82	41600	0.20
IO	89	106	83.96

Individual Resource utilization



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (106)	BUFGCTRL (32)
RISC_Main	617	82	32	16	89	2
U1 (Data_Mem)	60	8	0	0	0	0
U2 (Instr_Mem)	432	43	32	16	0	0
U3 (RISC_CPU)	125	31	0	0	0	0
uut1 (Interrupt_Synchr)	2	4	0	0	0	0
uut2 (Controller_and_decoder)	38	5	0	0	0	0
uut3 (data_path)	85	22	0	0	0	0
dut1 (Program_Counter)	14	20	0	0	0	0
dut2 (Register_File)	70	0	0	0	0	0
dut4 (ALU_unit)	1	2	0	0	0	0

Timing Summary:-



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	9.615 ns	Worst Hold Slack (WHS):	0.007 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3330	Total Number of Endpoints:	3330	Total Number of Endpoints:	516

All user specified timing constraints are met.

Note:-

- Clock Constraint was given 50MHz
- Maximum clock frequency= 96.29MHz