

# **INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR**

## **COMPUTER ORGANIZATION LABORATORY**

### **ASSIGNMENT 7**

#### **RISC PROCESSOR DESIGN**

## **GROUP MEMBERS**

### **GROUP 54:**

- 1) Gaurav Madkaikar 19CS30018
- 2) Girish Kumar 19CS30019



# RISC PROCESSOR DESIGN

## Instruction Set Architecture

Class	Instruction	Usage	Meaning
Arithmetic	Add	add rs,rt	$rs \leftarrow (rs) + (rt)$
	Comp	comp rs,rt	$rs \leftarrow 2's \text{ Complement } (rs)$
	Add immediate	addi rs,imm	$rs \leftarrow (rs) + imm$
	Complement Immediate	compi rs,imm	$rs \leftarrow 2's \text{ Complement } (imm)$
Logic	AND	and rs,rt	$rs \leftarrow (rs) \wedge (rt)$
	XOR	xor rs,rt	$rs \leftarrow (rs) \oplus (rt)$
Shift	Shift left logical	shll rs, sh	$rs \leftarrow (rs)$ left-shifted by $sh$
	Shift right logical	shrl rs, sh	$rs \leftarrow (rs)$ right-shifted by $sh$
	Shift left logical variable	shllv rs, rt	$rs \leftarrow (rs)$ left-shifted by $(rt)$
	Shift right logical	shrl rs, rt	$rs \leftarrow (rs)$ right-shifted by $(rt)$
	Shift right arithmetic	shra rs, sh	$rs \leftarrow (rs)$ arithmetic right-shifted by $sh$
	Shift right arithmetic variable	shrav rs, rt	$rs \leftarrow (rs)$ right-shifted by $(rt)$
Memory	Load Word	lw rt,imm(rs)	$rt \leftarrow mem[(rs) + imm]$
	Store Word	sw rt,imm,(rs)	$mem[(rs) + imm] \leftarrow (rt)$
Branch	Unconditional branch	b L	goto L
	Branch Register	br rs	goto (rs)
	Branch on less than 0	bltz rs,L	if(rs) < 0 then goto L
	Branch on flag zero	bz rs,L	if (rs) = 0 then goto L
	Branch on flag not zero	bnz rs,L	if(rs) $\neq$ 0 then goto L
	Branch and link	bl L	goto L; 31 $\leftarrow$ (PC)+4
	Branch on Carry	bcy L	goto L if Carry = 1
	Branch on No Carry	bncy L	goto L if Carry = 0

## Instruction Format and Encoding

<u>Opcode</u>	<u>Binary Rep.</u>	<u>Functions</u>
0	0000	add, comp, AND, XOR, shll, shrl, shllv, shrlv, shra, shrav
1	0001	addi, compi
2	0010	lw, sw
3	0011	br, bltz, bz, bnz (With reg.)
4	0100	b, bl, bcy, bncy (Without reg.)

Number of operations (Opcode) that can be added =  $2^4 - 5 = 11$

## R-Format Instructions

### Opcode- 0000

Opcode	rs	rt	shamt	Don't Care	function
4 bits	5 bits	5 bits	5 bits	9 bits	4 bits

Number of functions that can be added =  $2^4 - 10 = 6$

### Functions

Function	Function Code	Binary Rep.
xor	0	000000
and	1	000001
add	2	000010
comp	3	000011
shll	4	000100
shrl	5	000101
shra	6	000110
shllv	7	000111
shrlv	8	001000
shrav	9	001001

## Immediate (I-Format) Instructions

**Opcode- 0001**

<b>Opcode</b>	<b>rs</b>	<b>immediate</b>	<b>function</b>
4 bits	5 bits	19 bits	4 bits

Number of functions that can be added =  $2^4 - 2 = 14$

### Functions

<b>Function</b>	<b>Function Code</b>	<b>Binary Rep.</b>
addi	0	0000
compi	1	0001

## Memory Instructions

Opcode - 0010

Opcode	rt	rs	immediate	function
4 bits	5 bits	5 bits	16 bits	2 bits

Number of functions that can be added =  $2^2 - 2 = 2$

### Functions

Function	Function Code	Binary Rep.
lw	0	00
sw	1	01

## **Branch with registers**

**Opcode - 0011**

<b>Opcode</b>	<b>rs</b>	<b>Label</b>	<b>function</b>
4 bits	5 bits	19 bits	4 bits

**Number of functions that can be added =  $2^4 - 4 = 12$**

### **Functions**

<b>Function</b>	<b>Function Code</b>	<b>Binary Rep.</b>
br	0	0000
bltz	1	0001
bz	2	0010
bnz	3	0011



## Branch without registers

**Opcode - 0100**

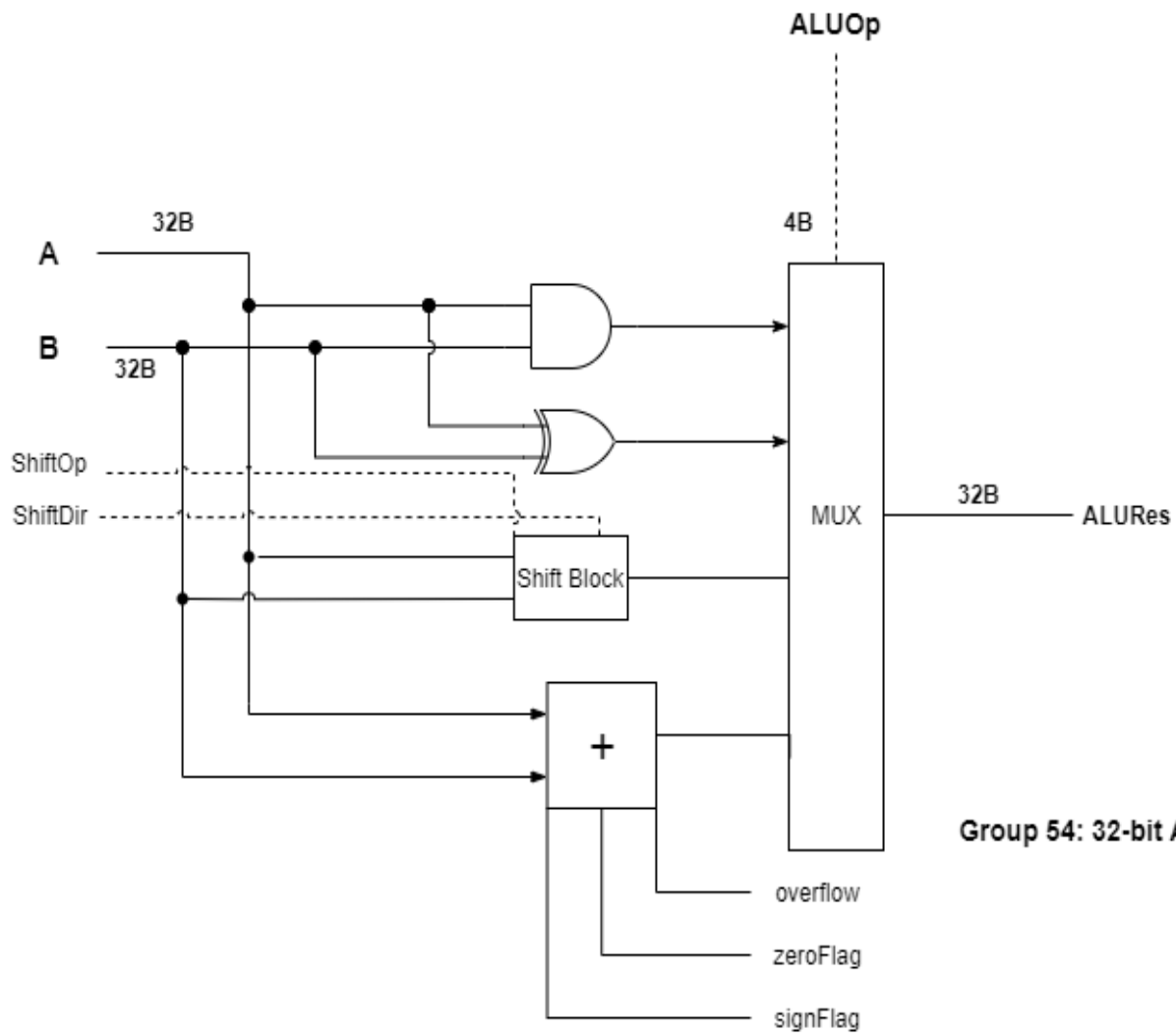
Opcode	Label	function
4 bits	24 bits	4 bits

Number of functions that can be added =  $2^4 - 4 = 12$

### Functions

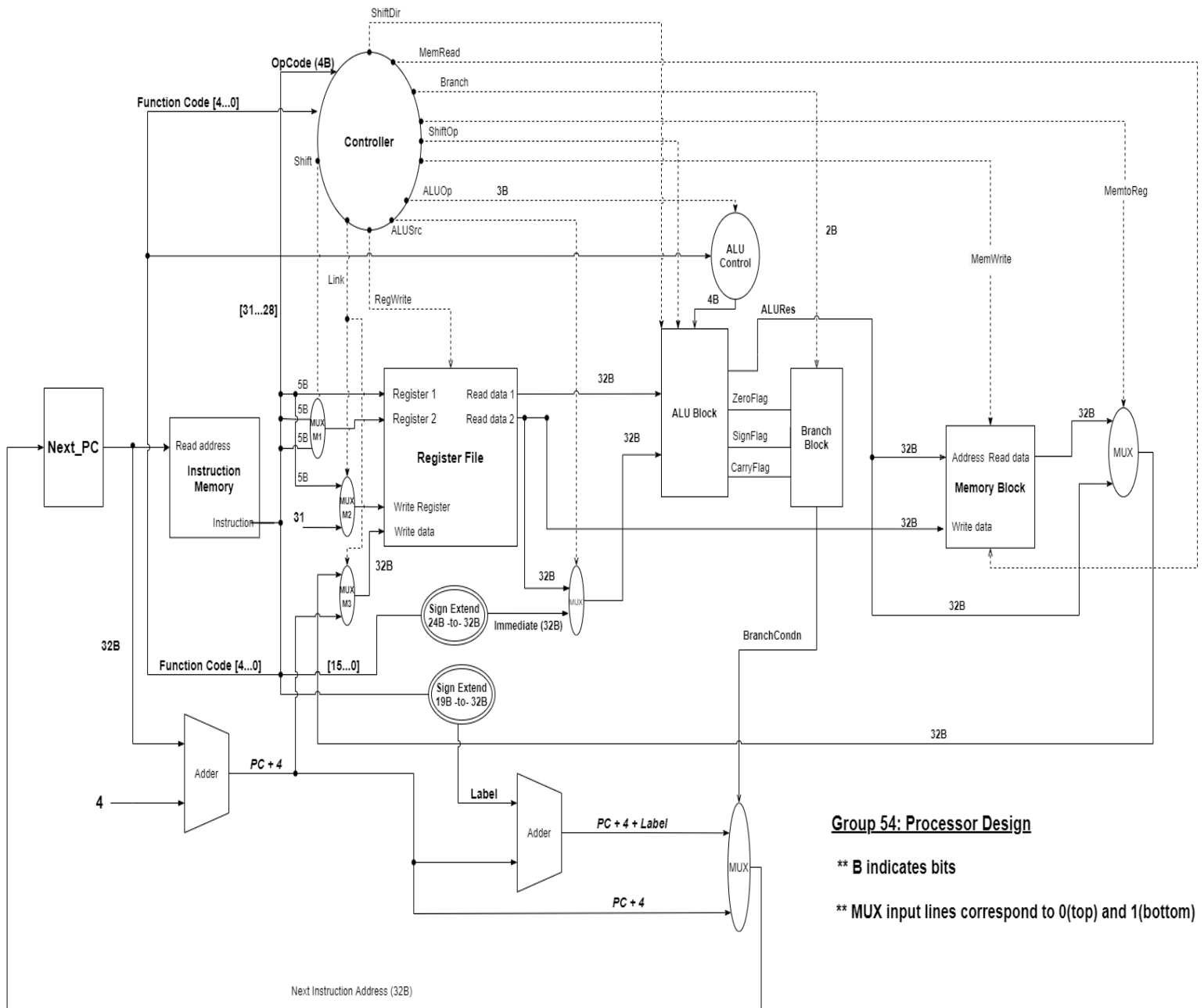
Function	Function Code	Binary Rep.
b	0	0000
bl	1	0001
bcy	2	0010
bncy	3	0011

## 32-bit ALU Design



Group 54: 32-bit ALU Design

# RISC Processor Architecture



## Group 54: Processor Design

**\*\* B indicates bits**

**\*\* MUX input lines correspond to 0(top) and 1(bottom)**

## ALU control table

ALU control	Function
and	0000
add	0001
subtract	0110
xor	1100
comp	0111

ALUOp	Function Code	Functions	ALU_CtrlOp
000	XXXX	lw/sw	1111
001	0000	xor	0000
	0001	and	0001
	0010	add	0010
	0011	comp	0011
010	0000	addi	0010
	0001	compi	0011
011	XXXX	shift	0100
100	XXXX	Branch with reg.	0101
101	XXXX	Branch without reg.	1111

# Control Logic Truth Table

Op Code	Function Code	Function Name	Controller Outputs											
			ShiftDir	ShiftOp	Shift	MemRead	MemWrite	Mem-to-Reg	ALUOp (3B)	ALUSrc	Reg Write	Branch (2B)	Link	Comp
0000	0000	xor	X	X	0	0	0	0	001	0	1	00	0	X
	0001	and	X	X	0	0	0	0	001	0	1	00	0	X
	0010	add	X	X	0	0	0	0	001	0	1	00	0	0
	0011	comp	X	X	0	0	0	0	001	0	1	00	0	1
	0100	shll	0	0	1	0	0	0	011	0	1	00	0	X
	0101	shrl	1	0	1	0	0	0	011	0	1	00	0	X
	0110	shra	1	1	1	0	0	0	011	0	1	00	0	X
	0111	shllv	0	0	1	0	0	0	011	0	1	00	0	X
	1000	shrlv	1	0	1	0	0	0	011	0	1	00	0	X
	1001	shrav	1	1	1	0	0	0	011	0	1	00	0	X
0001	0000	addi	X	X	0	0	0	0	010	1	1	00	0	X
	0001	compi	X	X	0	0	0	0	010	1	1	00	0	X
	00	lw	X	X	0	1	0	1	000	1	1	00	0	X

0010	01	sw	X	X	0	0	1	X	000	1	0	00	0	X
0011	0000	br	X	X	0	0	0	X	100	0	0	01	0	X
	0001	bltz	X	X	0	0	0	X	100	0	0	01	0	X
	0010	bz	X	X	0	0	0	X	100	0	0	01	0	X
	0011	bnz	X	X	0	0	0	X	100	0	0	01	0	X
0100	0000	b	X	X	0	0	0	X	101	0	0	11	0	X
	0001	bl	X	X	0	0	0	X	101	0	0	11	1	X
	0010	bcy	X	X	0	0	0	X	101	0	0	10	0	X
	0011	bncy	X	X	0	0	0	X	101	0	0	10	0	X

### Control Signals:

**ShiftDir:** Indicates the direction of the logical shift operation

**ShiftOp:** Indicates the type of shift operation

**Shift:** Indicates a shift operation

**MemRead:** Indicates a read from the memory

**MemWrite:** Indicates a write to the memory

**MemtoReg:** Indicates a load-word instruction

**RegWrite:** Write data to the destination register

**ALUOp:** Indicates the type of ALU Operation (E.g., and, xor, add)

**ALUSrc:** Indicates immediate instructions

**Link:** Indicates a branch-and-link instruction

**Branch (2B):** Indicates the type of branch instruction

00 -> No branch

01 -> Branch with registers

10 -> Branch with carry

11 -> Branch with no registers

# Testing the processor

## 1) GCD Example

```
# $s3 will store the GCD of A and B
```

```
main:
```

```
1. addi $s1, 2
2. addi $s2, 5
3. addi $s3, 0
```

```
GCD:
```

```
4. add $t0, $s1
5. comp $t1, $s2
6. add $t0, $t1
7. bz $t0, exit      # if(A == B) goto exit
8. bltz $t0, else    # if(A < B) goto else code block
9. add $s1, $t1
10. comp $t3, $t0
11. add $t0, $t3      # Restore t0
12. b GCD
```

```
else:
```

```
13. comp $t2, $s1
14. add $s2, $t2
15. comp $t3, $t0
16. add $t0, $t3      # Restore t0
17. b GCD
```

```
exit:
```

```
18. add $s3, $s1
19. # Exit Program
```

```
memory_initialization_radix=2;
memory_initialization_vector=
00011010100000000010101110010000,
00011011100000000000100100001000,
00011011100000000000000000000000,
0000001111101010000000000000000010,
0000010001011100000000000000000011,
0000001110100000000000000000000010,
0011001111000000000000000011100000,
0011001110000000000000001101000001,
0000101010100000000000000000000010,
0000010100011100000000000000000011,
0000001110101000000000000000000010,
01000000000000000000000000000000,
0000010011010100000000000000000011,
0000101100100100000000000000000010,
0000010100011100000000000000000011,
0000001110101000000000000000000010,
01000000000000000000000000000000,
0000101111010100000000000000000010;
```

This is an example of a GCD computing program to test the processor written according to the ISA format provided. Register conventions are similar to that of 32-bit MIPS register convention. **Module RegFile** contains the 32X32 bit register file initializing each of the 32 registers to 0. The final GCD value is stored in the register \$s3 (Register 23).



## 2) Linear Search

```
1 // Linear Search Algorithm
2 // Answer contained in $t0
3 // Example
4 arr[] = {12, 5, 17, 123}
5 key = 17
6 $t0 = 2 (Result)
7
8 main:
9 1. addi $t0, 0
10 2. addi $t1, 12
11 3. addi $t2, 5
12 4. addi $t3, 17
13 5. addi $t4, 123
14 6. addi $s0, 17      # s0 <-- key
15 7. comp $s1, $s0     # s1 = -17
16 8. addi $s2, 4       # s2 = 4
17 9. sw $t1, 0, $t0
18 10. sw $t2, 4, $t0
19 11. sw $t3, 8, $t0
20 12. sw $t4, 12, $t0
21 13. comp $t5, $s0    # t5 = -key
22
23 searchLoop:
24 14. comp $t6, $s2     # t6 = -4
25 15. add $t6, $s3       # t6 += s3
26 16. bz $t6, Exit      # if(t6 == 0) goto Exit
27 17. lw $t7, 0($t0)     # t7 = Mem[t0]
28 18. add $t7, $s0       # t7 = Mem[t0] - key
29 19. bz $t7, Exit      # if(t7 == 0) goto Exit
30 20. addi $t0, 4        # t0 += 4
31 21. addi $s3, 1        # s3 += 1
32 22. b searchLoop
33
34 Exit:
35 23. # Exit program
```

```
1 memory_initialization_radix=2;
2 memory_initialization_vector=
3 00010011100000000000000000000000,
4 00010100000000000000000001100000,
5 00010100100000000000000001010000,
6 00010101000000000000000010001000,
7 00010101100000000000001111011000,
8 00011010000000000000000100010000,
9 0000101011010000000000000000011,
10 0001101100000000000000001000000,
11 00100100000111000000000000000001,
12 0010010010011100000000000010001,
13 0010010100011100000000000100001,
14 0010010110011100000000000110001,
15 0000011001010000000000000000011,
16 0000011011010100000000000000011,
17 0000011011011000000000000000010,
18 0011011010000000000001011000010,
19 0010011100011100000000000000000,
20 0000011101010000000000000000010,
21 0011011100000000000001011000010,
22 0001001110000000000000001000000,
23 0001101110000000000000000000000,
24 0100000000000000000001110000000,
25 0000000000000000000000000000000,
26
```

Similarly, a linear search algorithm is implemented according to the ISA. The search position is stored in \$t0 (Register 7).

# Loaded Register Values

- Result stored in 2's complement form

The screenshot displays a simulation tool interface with three main panels: Memory, Objects, and a Register Value table.

**Memory Panel:** Shows a tree view of simulation objects for TopModule, including paths like /TopModule\_tb/uut/IM1/inst/native\_mem\_mapped\_mod and /TopModule\_tb/uut/MM1/inst/native\_mem\_mapped\_r.

**Objects Panel:** Lists simulation objects for TopModule, including 'clk' and 'reset', both with a value of 0.

**Register Value Table:** A table showing the loaded values for registers 8 through 31. The values are in 2's complement form.

Register	Value
31	0
30	0
29	0
28	0
27	0
26	0
25	0
24	0
23	408
22	4294966769
21	1462
20	0
19	0
18	0
17	0
16	0
15	0
14	0
13	0
12	0
11	0
10	4294966769
9	4294965834
8	527

**Console Panel:** Displays simulation logs, including messages about the simulator version (Sim R20131013), time resolution (1 ps), and initialization process.

# Synthesis Report

The screenshot displays the ISE Project Navigator interface for a project named 'RISC\_Processor'. The 'Hierarchy' pane on the left shows the project structure, including components like 'TopModule', 'PCM', 'NPC', 'IM1', 'ID1', 'CtrlUnit', 'M1', 'M2', 'REG\_File', and 'M3'. The 'Processes' pane shows the 'Synthesize - XST' process. The 'Console' pane at the bottom displays the synthesis results, indicating that the process completed successfully. The 'TopModule\_tb.v' file is open in the editor, showing a Verilog testbench for the 'TopModule'.

```
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 ///////////////////////////////////////////////////////////////////
24
25 module TopModule_tb;
26
27 // Inputs
28 reg clk;
29 reg reset;
30
31 // Instantiate the Unit Under Test (UUT)
32 TopModule uut (
33     .clk(clk),
34     .reset(reset)
35 );
36
37 always begin
38     #10 clk = ~clk;
39     if (clk)
40         $display($time, " GCD of 697 and 289 = %d", uut.REG_File.ig[21]);
41 end
```

Speed Grade: -1

Minimum period: 7.537ns (Maximum Frequency: 132.679MHz)  
Minimum input arrival time before clock: 0.894ns  
Maximum output required time after clock: No path found  
Maximum combinational path delay: No path found

Process "Synthesize - XST" completed successfully

Ln 26 Col 1 | Verilog

The processor design is correctly synthesized.