EE-739 (Processor Design)

Project

Design and Verilog Code of a 6-Stage Pipelined Processor (IITB RISC)

Nishant Saurabh (193079039)

Ayush Ranjan (193070080)

M Tech EE7(Solid State Devices)

Electrical Engineering Department

IIT Bombay

Problem Statement:-

Design a 6 stage pipelined processor, IITB-RISC, whose instruction set architecture is provided. *IITB-RISC* is a 16-bit very simple computer developed for the teaching that is based on the Little Computer Architecture. The *IITB-RISC* is an 8-register, 16-bit computer system. It should follow the standard 6 stage pipelines (Instruction fetch, instruction decode, register read, execute, memory access, and write back). The architecture should be optimized for performance, i.e., should include hazard mitigation techniques. Hence, it should have forwarding and branch prediction technique.

Features of IITB- RISC

- IITB_RISC is a 16-bit microprocessor, having 8 general purpose registers. R7 is the program counter.
- 6 Stage-Pipelined processors with 5 pipeline registers
- Compatible with a given ISA
- Separate Instruction and Data memory
- Condition Code Registers with two flags Carry Flag(C) and Zero Flag(Z)
- Supports three types of Instruction formats:-
    - R-Type Instruction Format

Opcode (4 bit)	Register A (RA) (3 bit)	Register B (RB) (3-bit)	Register B (RB) (3-bit)	Unused (1 bit)	Condition (CZ) (2 bit)

    - I-Type Instruction Format

Opcode (4 bit)	Register A (RA) (3 bit)	Register C (RC) (3-bit)	Immediate (6 bits signed)

    - J-Type Instruction Format

Opcode (4 bit)	Register A (RA) (3 bit)	Immediate (9 bits signed)

Instructions Encoding:

ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADI:	00_01	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0 + 8 bits corresponding to Reg R7 to R0			
SM:	01_11	RA	0 + 8 bits corresponding to Reg R7 to R0			
BEQ:	11_00	RA	RB	6 bit Immediate		
JAL:	10_00	RA	9 bit Immediate offset			
JLR:	10_01	RA	RB	000_000		

RA: Register A

RB: Register B

RC: Register C

Instruction Description

Mnemonic	Name & Format	Assembly	Action
ADD	ADD (R)	*add rc, ra, rb*	Add content of regB to regA and store result in regC. *It modifies C and Z flags*
ADC	Add if carry set (R)	*adc rc, ra, rb*	Add content of regB to regA and store result in regC, if carry flaf is set. *It modifies C & Z flags*
ADZ	Add if zero set (R)	*adz rc, ra, rb*	Add content of regB to regA and store result in regC, if zero flag is set. *It modifies C & Z flags*
ADI	Add immediate (I)	*adi rb, ra, imm6*	Add content of regA with Imm (sign extended) and store result in regB. *It modifies C and Z flags*
NDU	Nand (R)	*ndu rc, ra, rb*	NAND the content of regB to regA and store result in regC. *It modifies Z flag*
NDC	Nand if carry set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if carry flag is set. *It modifies Z flag*
NDZ	Nand if zero set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if zero flag is set. *It modifies Z flag*
LHI	Load higher immediate (J)	*lhi ra, Imm*	Place 9 bits immediate into most significant 9 bits of register A (RA) and lower 7 bits are assigned to zero.
LW	Load (I)	*lw ra, rb, Imm*	Load value from memory into reg A. Memory address is formed by adding immediate 6 bits with content of red B.

			It modifies zero flag.
SW	Store (I)	*sw ra, rb, Imm*	Store value from reg A into memory. Memory address is formed by adding immediate 6 bits with content of red B.
LM	Load multiple (J)	*lw ra, Imm*	Load multiple registers whose address is given in the immediate field (one bit per register, R7 to R0) in order from right to left, i.e, registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are loaded from consecutive addresses.
SM	Store multiple (J)	sm, ra, Imm	Store multiple registers whose address is given in the immediate field (one bit per register, R7 to R0) in order from right to left, i.e, registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are stored to consecutive addresses.
BEQ	Branch on Equality (I)	beq ra, rb, Imm	If content of reg A and regB are the same, branch to PC+Imm, where PC is the address of beq instruction
JAL	Jump and Link (I)	jalr ra, Imm	Branch to the address PC+ Imm. Store PC+1 into regA, where PC is the address of the jalr instruction
JLR	Jump and Link to Register (I)	jalr ra, rb	Branch to the address in regB. Store PC+1 into regA, where PC is the address of the jalr instruction

Performance Optimization

Verilog code was written using behavioral model.

The following set of Flag were used to include Hazard mitigation technique to optimize the performance.

Pipeline Stall_flag – Every pipeline register has this flag. Whenever we have instruction which has destination as R7 in EX_MEM pipeline register, then we SET this flag to 1 and whatever instruction has entered in pipeline after this instruction will not have any impact on Memory and Write Back stage and will not change their content. They will be simply flushed out of pipeline. In simple words, having a Stall_flag = 1 means a tag which tells each processor stage that it is a waste instruction.

Load_Flag – If we have load instruction followed by Dependent instruction then we SET this flag to 1 which will STALL all preceding instruction for ONE cycle. This flag is set by looking for load instruction at RR_EX pipeline register.

SM_Flag- This flag is only used by Store Multiple Register Instruction. This flag is SET to 1 when we have SM instruction in RR_EX pipeline register to halt all other preceding instruction until this is completed and it is set to 0 after Memory stage.

LM_Flag- This flag is used by Load Multiple Register Instruction to halt all the pipeline until this instruction is executed completely.

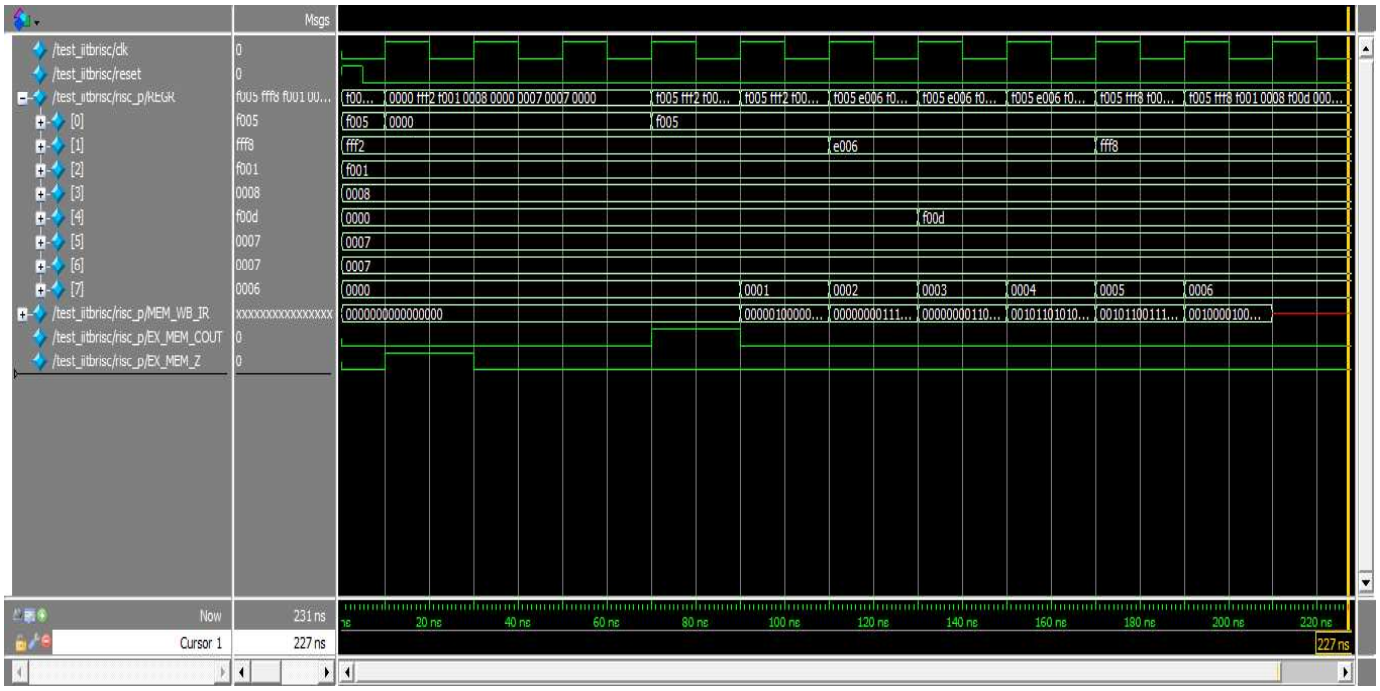
Store Multiple Register Instruction and Load Multiple Register Instruction is executed in single cycle.

Simulation Results

1. Testing ADD, ADC, ADZ, NDU, NDZ, NDC

Program

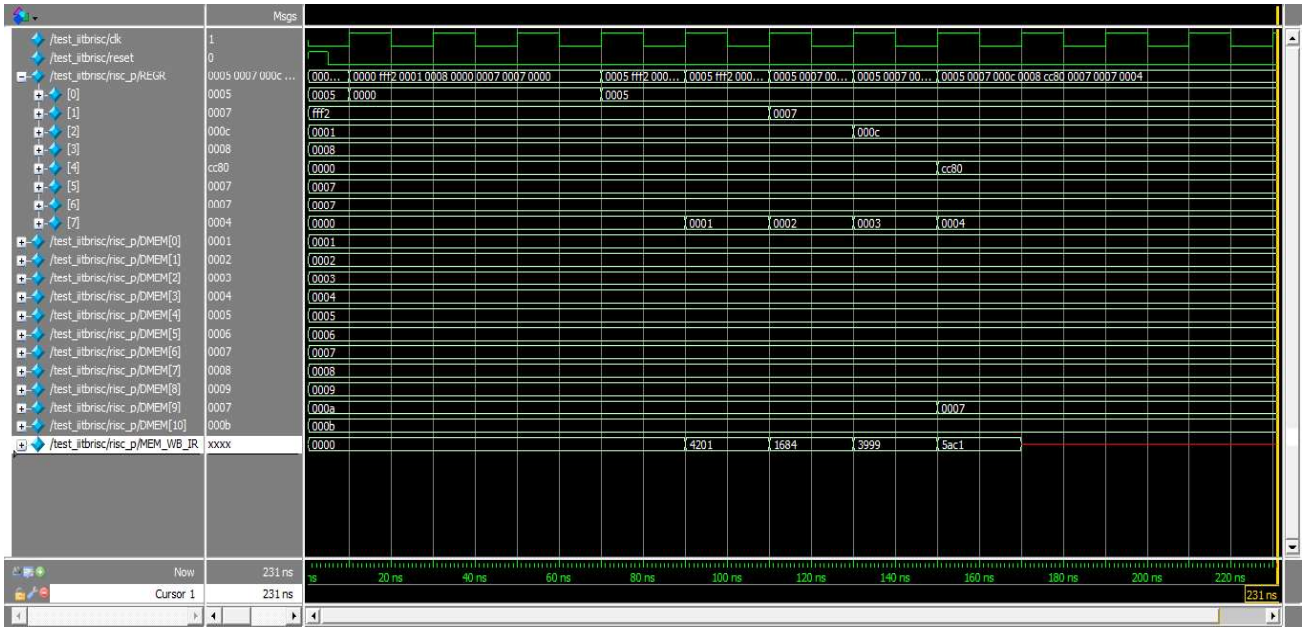
add r1, r2, r0
adc r4, r0, r3 //executed as carry was set
adz r2, r0, r3, // not executed as zero was not set
ndu r1, r6, r5, //do not set z flag as result is non zero
ndz r4, r6, r3, //not execute as Z flag is not set
ndc r2, r0, r4, //will not execute



2. Testing LW, SW, ADI, LHI

Program

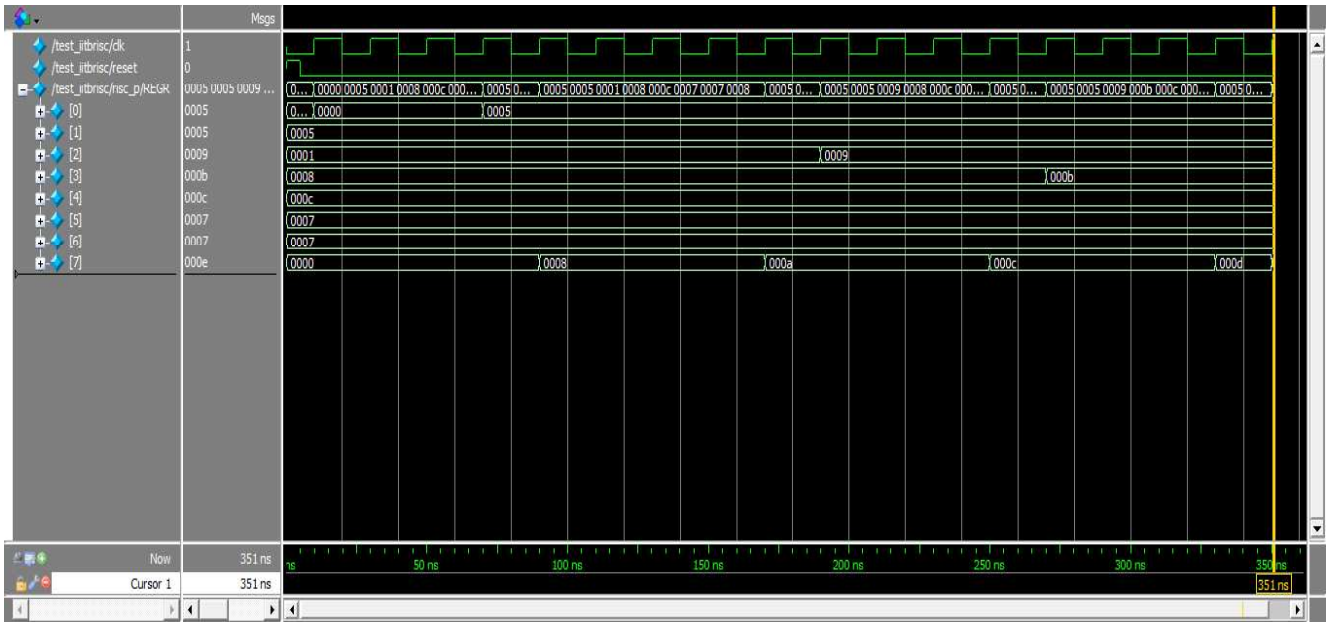
lw r1, r0,000001 // Executed and updated r1 with new value
adi r2, r3 ,000100 // Executed
lhi r4, 110011001// executed
sw r5, r3, 000001 // Executed



3. Testing BEQ, JAL, JLR

Program

beq r0, r1, 001000 // branch as equal
jal r2, 000000010 // compulsory jump
jlr r3, r4 // compulsory jump
beq r0, r5, 001000 // will not execute as r0 not equal r5



4. To Check Data Forwarding and Add after Load

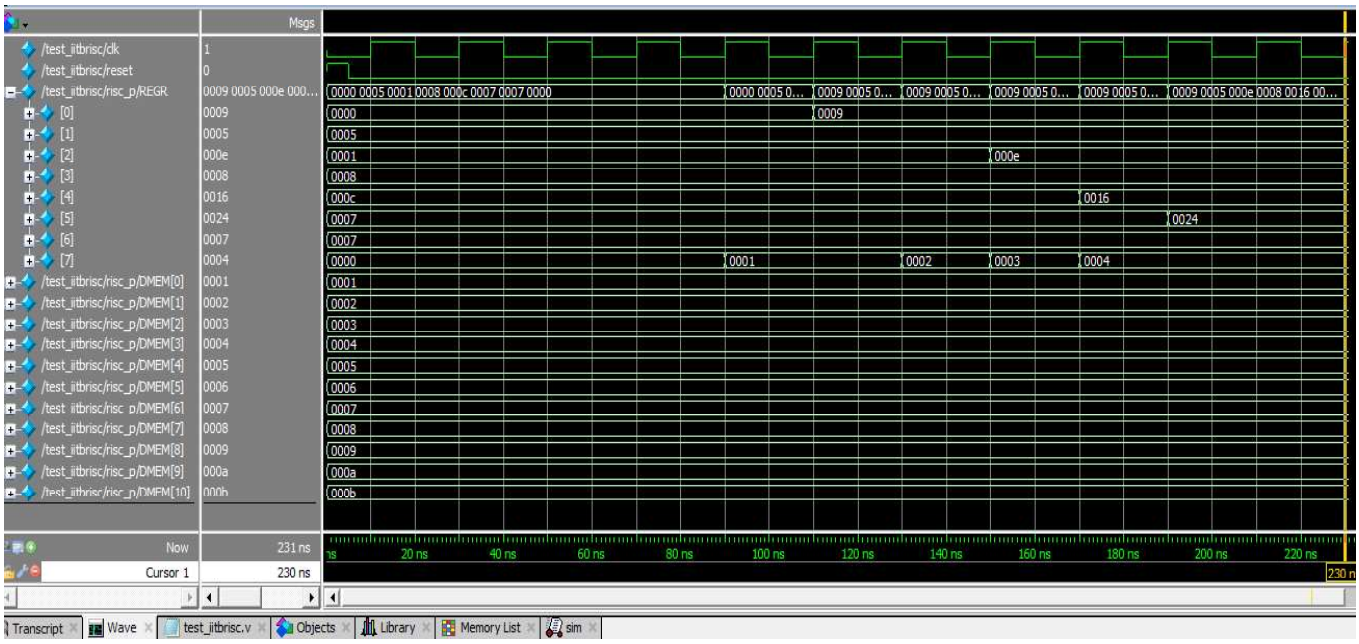
Program

lw r0, r6,000001 //executed and update r0

add r2, r0, r1 // immediate dependence on load, so 1 cycle stall

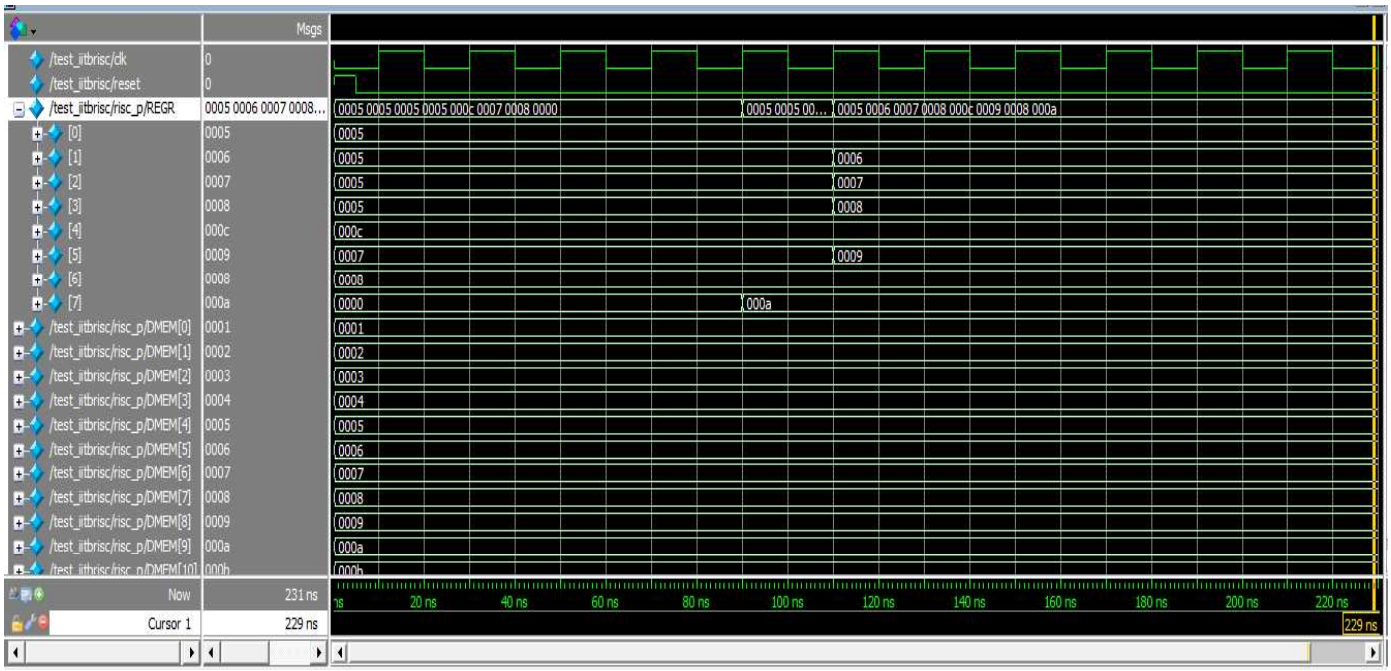
add r4, r2, r3 // source same as destination executed using updated r2

add r5, r2, r4 // both sources on previous two instructions, executed using updated r2 and r4



5. To check LM (load multiple register) with R7 as one of the destination

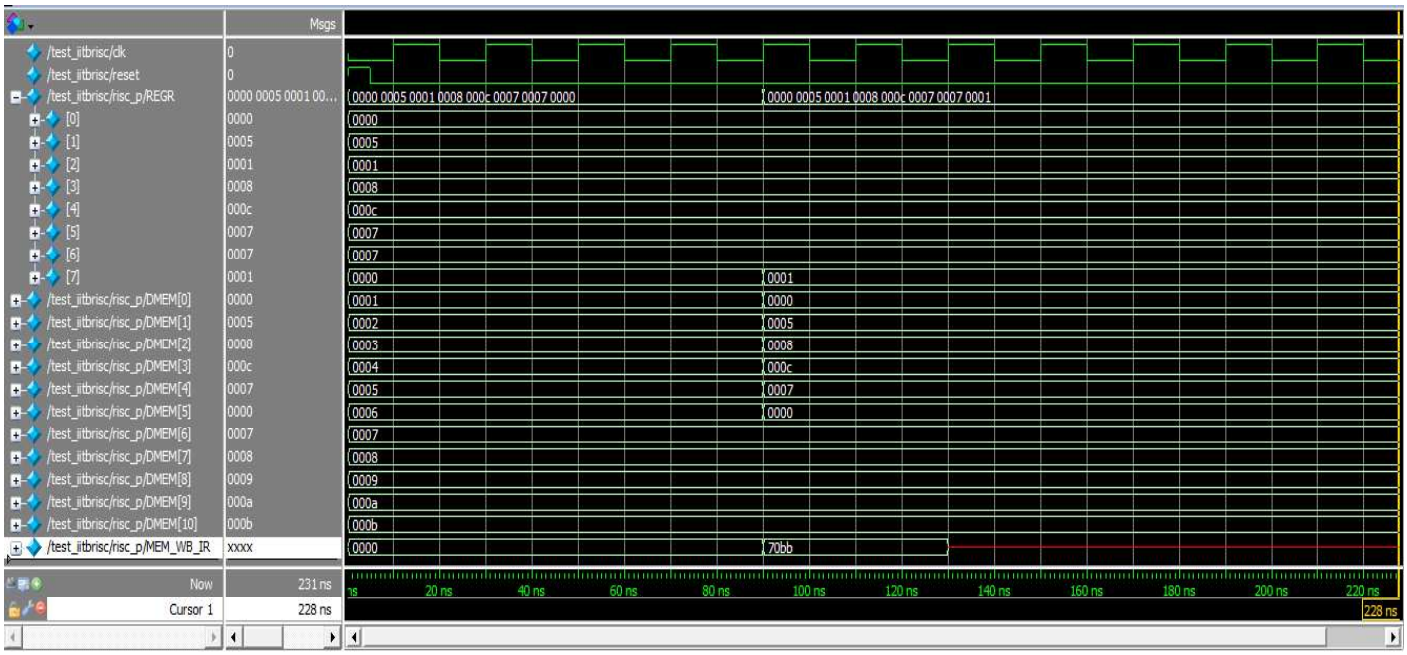
Program
lm r0, 010101110



As we can see the LM Instruction is getting executed in one cycle only. i.e. all register is getting updated in one cycle.

6. To check SM(store multiple Register) with R7 as one of the source

Program
sm r0, 010111011



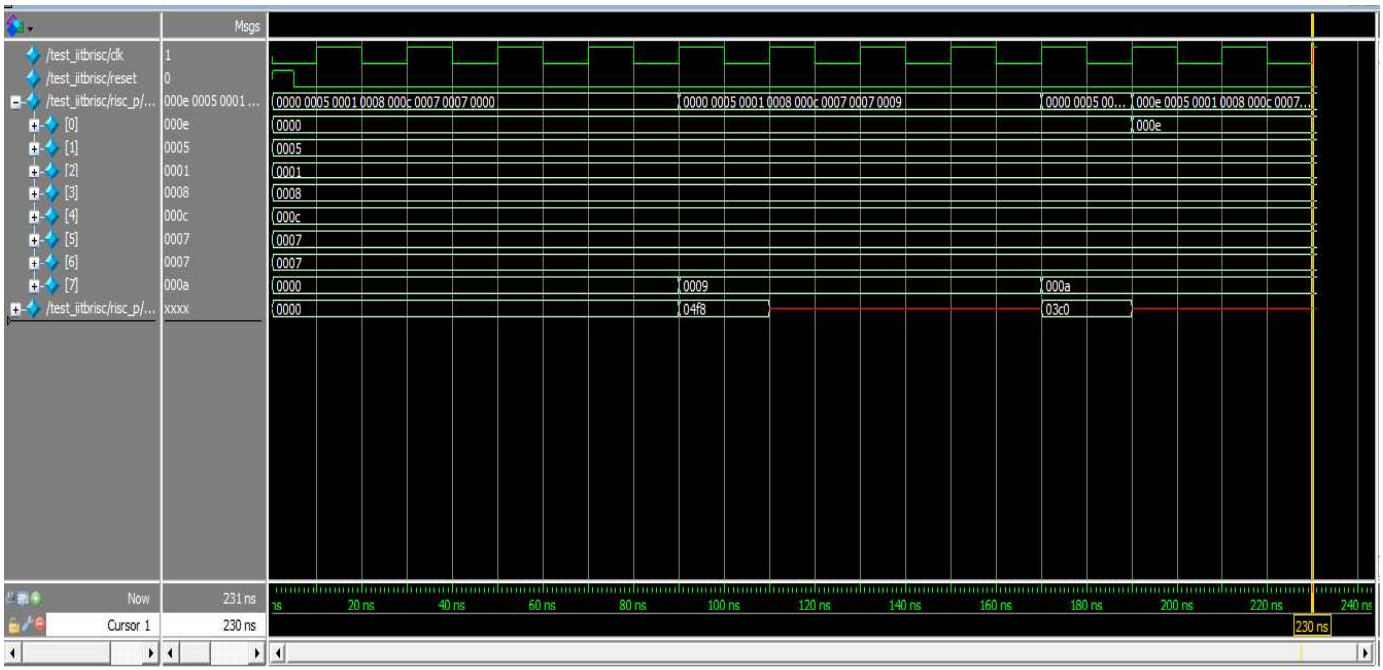
As we can see the SM Instruction is getting executed in one cycle only. i.e. all Data memory is getting updated in one cycle.

7. To check R7 as one of the sources in an instruction. It should take value of R7(PC) which was at the time of instruction under execution. And next instruction is fetched from address which was stored in R7. R7 always holds the value PC+1 usually unless any kind of Jump / Branch / Modification of R7 by user take place in which case it hold the address of the next instruction to be executed.

   Program

   add r7, r2, r3 // update R7

   add r1, r0, r7 // this is executed with updated value of r7


Msgs
/test_iitbrisc/clk
/test_iitbrisc/reset
/test_iitbrisc/risc_p/...
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
/test_iitbrisc/risc_p/...
1
0
000e 0005 0001 ...
000e
0005
0001
0008
000c
0007
0007
000a
xxxx
0000 0005 0001 0008 000c 0007 0007 0000
0000 0005 0001 0008 000c 0007 0007 0009
0000 0005 00...
000e 0005 0001 0008 000c 0007...
0000
000e
0005
0001
0008
000c
0007
0007
0000
0009
000a
0000
04f8
03c0
Now
231 ns
Cursor 1
230 ns
ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
230 ns