Processor Design - EE739

Project Report

Pipelined IITB-RISC Processor

Instructor : Prof. Virendra Singh

Submitted By –

Mohd. Faizaan Qureshi	(203070062)
Kanak Vijay	(203070050)
Muhaamad. Shoaib Iqbaal	(203070058)

Hardware flowchart

R-type instructions

Opcode	Ra	Rb	Rc		condition

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (11-9) → Rf_A1

IWR (8-6) → Rf_A2

Rf_D1 → T1

Rf_D2 → T2

ID_EX_reg

EX

T1 → alu1_A

T2 → alu1_B

alu1_out → T3

EX_MEM_reg

MEM

No operation in MEM

MEM_WB_reg

WB

T3 → Rf_D3

IWR (5-3) → Rf_A3

LW instruction

Opcode	Ra	Rb	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (8-6) → Rf_A1

Rf_D1 → T1

ID_EX_reg

EX

T1 → alu1_A

IWR (5-0) → SE6 → alu1_B

alu1_out → T3

EX_MEM_reg

MEM

T3 → data_memA

data_memD → T4

MEM_WB_reg

WB

T4 → Rf_D3

IWR (11-9) → Rf_A3

SW instruction

Opcode	Ra	Rb	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (8-6) → Rf_A1

IWR (11-9) → Rf_A2

Rf_D1 → T1

Rf_D2 → T2

ID_EX_reg

EX

T1 → alu1_A

IWR (5-0) → SE6 → alu1_B

alu1_out → T3

EX_MEM_reg

MEM

T3 → data_memA

T2 → data_memD

MEM_WB_reg

WB

No Operation

BEQ instruction

Opcode	Ra	Rb	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (11-9) → Rf_A1

IWR (8-6) → Rf_A2

Rf_D1 → T1

Rf_D2 → T2

PC → alu3_A

IWR (5-0) → SE6 → alu3_B

alu3_out → PC_2

ID_EX_reg

EX

T1 → alu1_A

IWR (5-0) → SE6 → alu1_B

alu1_out → T3

if (alu1_zero) then PC_2 → PC_3

else PC_1 → PC_3

EX_MEM_reg

MEM

No Operation

MEM_WB_reg

WB

No Operation

LHI instruction

Opcode	Ra	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

NO Operation

ID_EX_reg

EX

IWR (8-0) → SEM9 → T3

(Here ignore adder result)

EX_MEM_reg

MEM

T3 → data_memA

data_memD → T4

MEM_WB_reg

WB

T4 → Rf_D3

IWR (11-9) → Rf_A3

JAL instruction

Opcode	Ra	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

PC → alu3_A

IWR (8-0) → SE9 → alu3_B

alu3_out → PC_2

ID_EX_reg

EX

No Operation

EX_MEM_reg

MEM

No Operation

MEM_WB_reg

WB

PC_1 → Rf_D3

IWR (11-9) → Rf_A3

JLR instruction

Opcode	Ra	Rb

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (8-6) → Rf_A1

Rf_D1 → PC_2

ID_EX_reg

EX

No Operation

EX_MEM_reg

MEM

No Operation

MEM_WB_reg

WB

PC_1 → Rf_D3

IWR (11-9) → Rf_A3

JRI instruction

Opcode	Ra	Immediate

IF

PC_current_value → instr_memA, alu2_A

+1 → alu2_B

alu2_out → PC_1

instr_memD → IWR

IF_ID_reg

ID & OF

IWR (11-9) → Rf_A1

Rf_D1 → T1

ID_EX_reg

EX

T1 → alu1_A

IWR (8-0) → SE9 → alu1_B

alu1_out → T3 → PC_3

EX_MEM_reg

MEM

No Operation

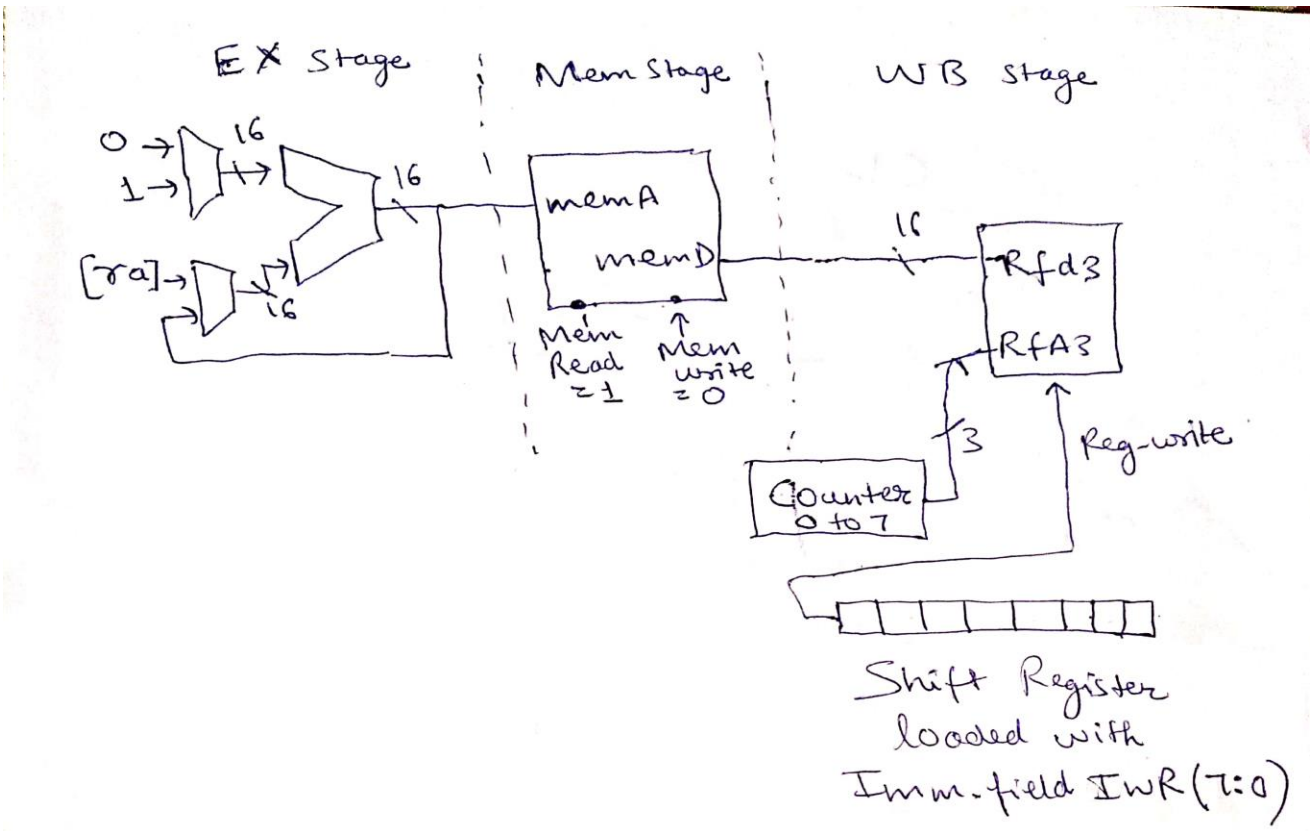
MEM_WB_reg

WB

No Operation

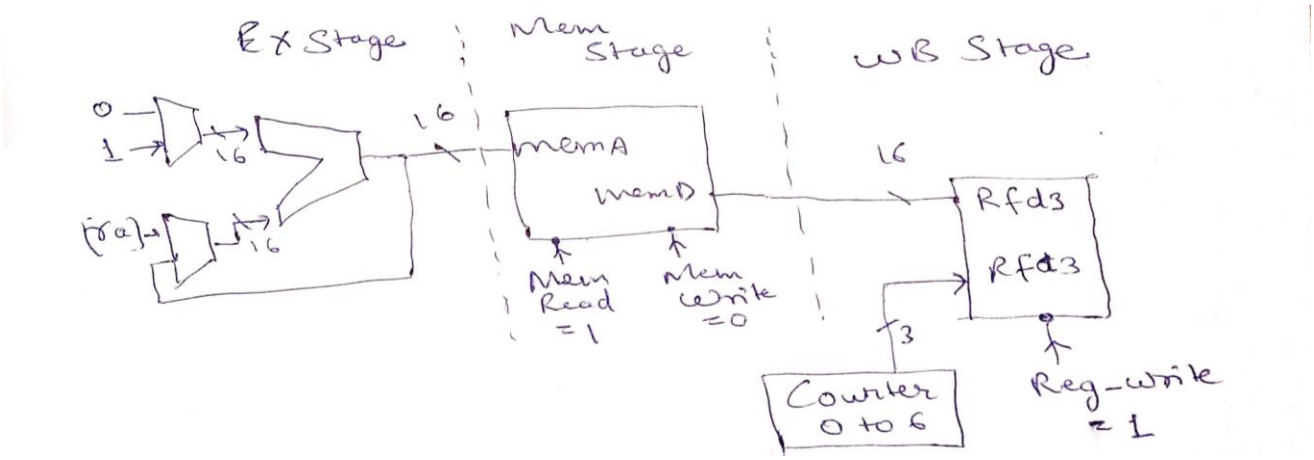
LM Instruction

Opcode	Ra	Immediate



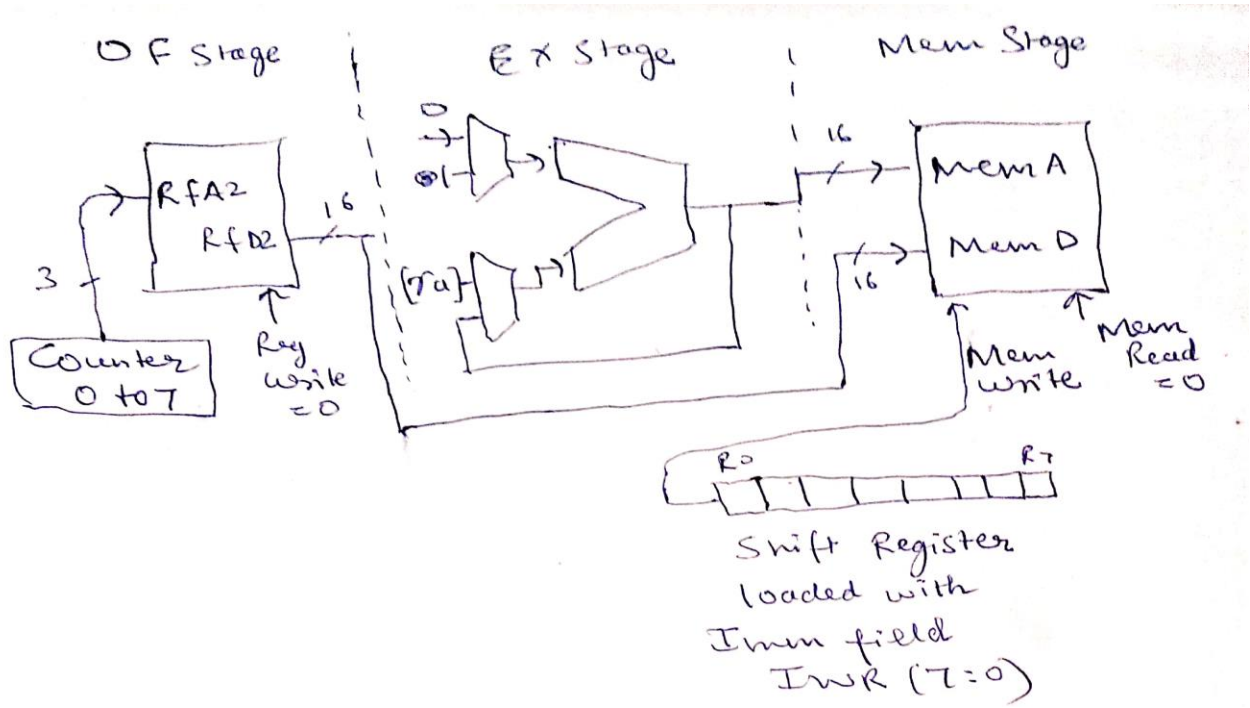
LA Instruction

Opcode	Ra	Immediate



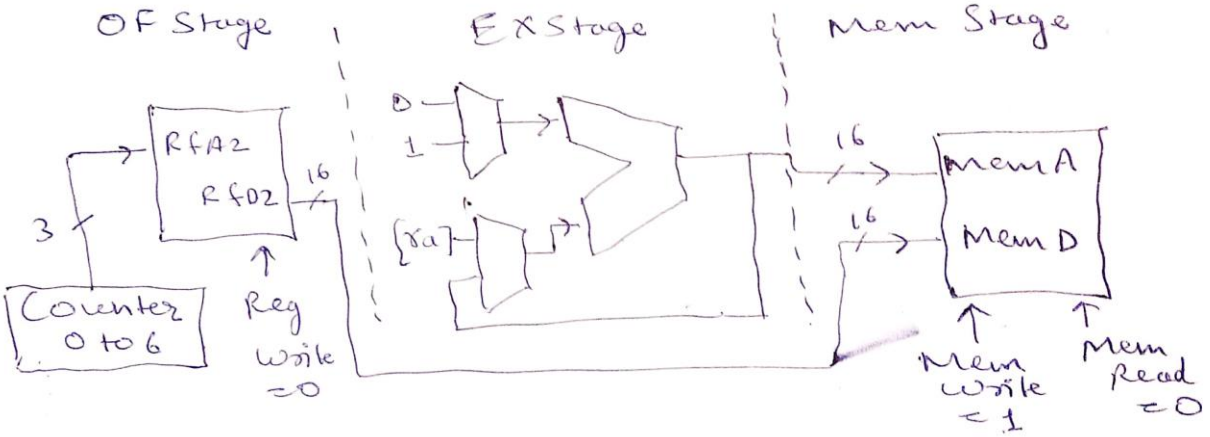
SM Instruction

Opcode	Ra	Immediate



SA Instruction

Opcode	Ra	Immediate



Control Signals

Signal Name	Size
alu1_src	2 bit
alu1_op	2 bit
t3_sel	1 bit
load	1 bit
sig_multiple	1 bit
sig_all	1 bit
branch	1 bit
jump	1 bit
jump_type	2 bit
mem_read	1 bit
mem_write	1 bit
reg_write	1 bit
reg_write_data_sel	2 bit
reg_write_addr_sel	2 bit

Pipeline Registers

Register Name	Size
IF_ID_reg	49 bits
ID_EX_reg	116 bits
EX_MEM_reg	97 bits
MEM_WB_reg	78 bits

Look Up table we have used has 8 entries and each has width 34 bits.

Data Hazards

Data Hazards occur when there is a violation in producer – consumer relationships.

All instructions that read operands which is being written by previous instruction, then there may be hazards.

All instructions which are writing in to register file =

All R-type instructions, LW, LHI, JAL, JLR

All instructions which read from register file in decode and operand read stage =

All R-type instructions, LW, SW, BEQ, JLR, JRI, SM, SA, LM, LA

So, there can be many possible dependencies which can occur, and we have covered all possible dependencies.

Let's test the processor

TEST1

Instructions =>

LW R1, R0, 1

LW R2, R0, 2

ADD R3, R1, R2 // there is a dependency, need R1, R2

ADD R4, R1, R3 // there is a dependency, need R1, R3

ADD R5, R2, R3 // there is a dependency, need R2, R3

ADD R6, R3, R4 // there is a dependency, need R3, R4

NOP

G: > intelFPGAlite_workspace > pd_project_pipeline_up > program5.hex

```
0100001000000001
0100010000000010
0001001010011000
0001001011100000
0001010011101000
0001011100110000
0110000000000000
0110000000000000
0110000000000000
0110000000000000
0110000000000000
```

Data Memory content =>

Memory Data - /processor7/data_memory_dut/ram_array - Default

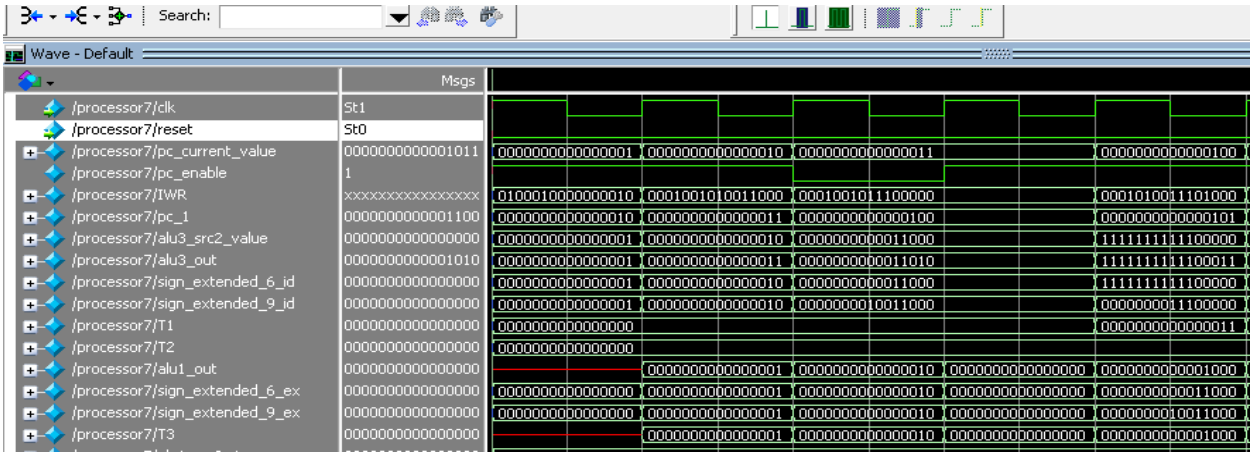
0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	7
8	17
9	19
10	x
11	x
12	x
13	x

Reg file content at the end of execution in modelsim =>

Memory Data - /processor7/reg_file/reg_array - Default

0	0
1	3
2	5
3	8
4	11
5	13
6	19
7	11

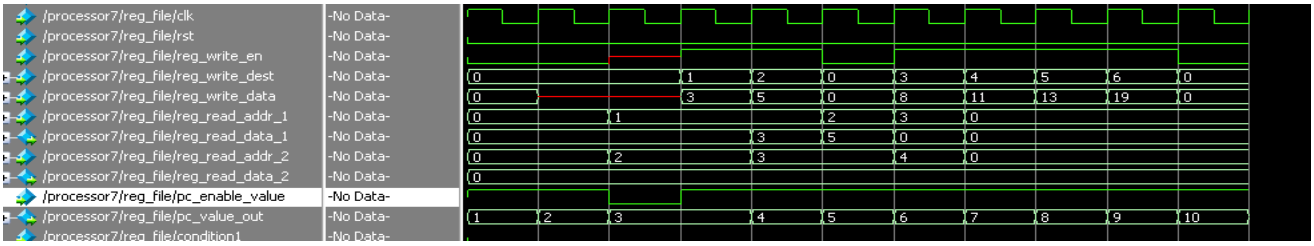
R7 is program counter.



Due to immediate dependency after load, there will be necessary STALL, which can be seen in above figure, pc_enable went to logic 0 for a while.

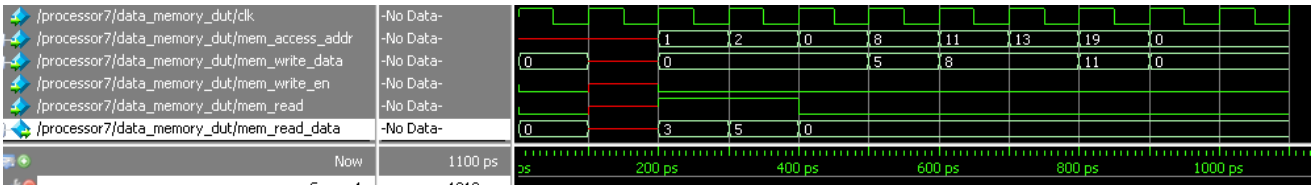
Simulation Waveforms =>

RegFile data flow from ports =>



Here in above figure, pc_value_out is R7 which stores program counter , and it is inside Register File module.

Data Memory data flow from ports =>



TEST2

Instructions =>

LM R1, 11111110

ADD R2, R6, R7 //there is a dependency, need R6

ADD R3, R7, R5 // there is a no dependency (R5 has been written)

ADD R4, R6, R5

ADD R5, R4, R5 //there is a dependency, need R4

ADD R6, R5, R3 //there is a dependency, need R5, R3

ADD R7, R6, R2 //there is a dependency, need R6, and there will be jump

SM R2, 11111111 //skipped

NOP

intelFPGAlite_workspace > pd_project_pipeline_up > program12.hex

```
1100001011111110
0001110111010000
0001111101011000
0001110101100000
0001100101101000
0001101011110000
0001110010111000
1101010011111111
0110000000000000
0110000000000000
0110000000000000
0110000000000000
0110000000000000
```

Data Memory Content =>

Memory Data - /processor7/data_memory_dut/ram_array - Default

0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	7
8	17
9	19
10	x
11	x
12	x
13	x

Reg file content at the end of execution in modelsim =>

Memory Data - /processor7/reg_file/reg_array

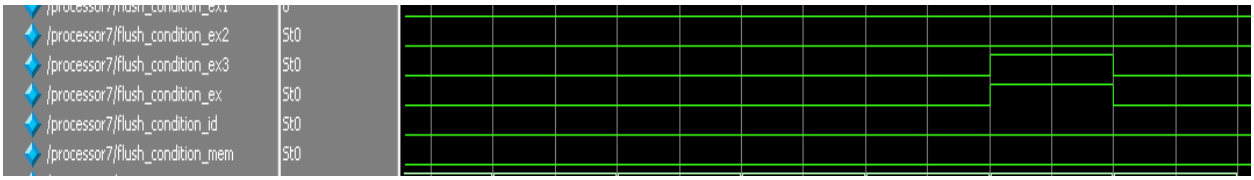
0	1
1	3
2	15
3	14
4	24
5	35
6	49
7	67

Here in this case ADD instruction is writing into R7, so there will be entry in look table =>

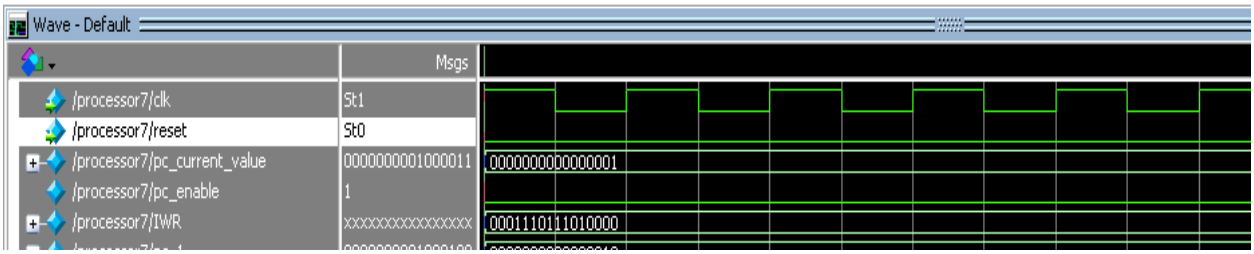
Memory Data - /processor7/look_up_table_branch - Default

00000000	00000000000000110000000000100000011
00000001	00000000000000000000000000000000000
00000002	00000000000000000000000000000000000
00000003	00000000000000000000000000000000000
00000004	00000000000000000000000000000000000
00000005	00000000000000000000000000000000000
00000006	00000000000000000000000000000000000
00000007	00000000000000000000000000000000000

Flush signal is also generated on encountering jump instruction =>

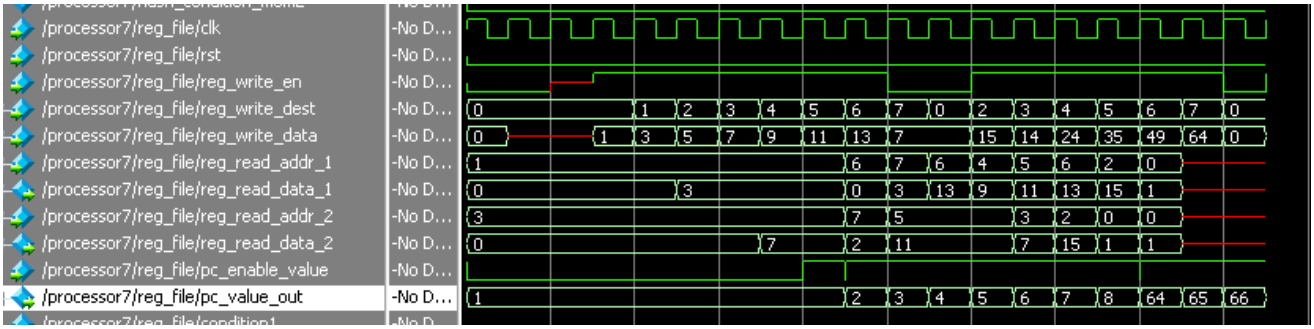


For LM instruction there would be STALL till its complete execution =>



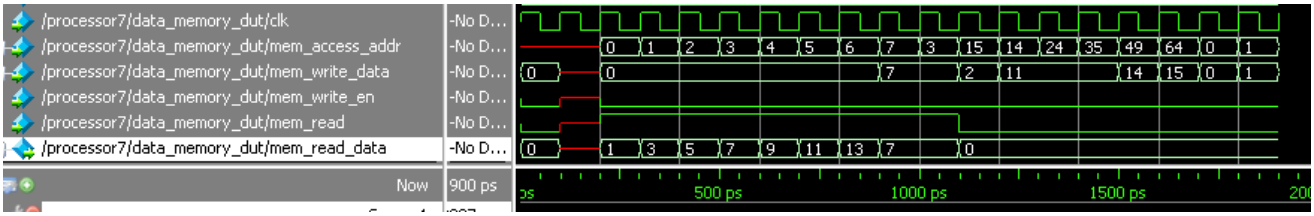
Simulation Waveforms =>

Register File data flow from it's ports =>



Here pc_value_out is R7. And clearly we can see PC=R7=8, it went to PC=R7=64

Data Memory data flow from it's ports =>



TEST3

Instructions =>

LA R1 //LA loads only R0 to R6

ADD R2, R5, R6 // there is a dependency , need R6, (R5 has been written)

ADD R3, R6, R4

ADD R4, R5, R4

ADD R5, R3, R4 // there is a dependency, need R4, R3

ADD R6, R4, R5 // there is a dependency, need R5, R4

SA R4

NOP

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
1    1110001000000000
2    0001101110010000
3    0001110100011000
4    0001101100100000
5    0001011100101000
6    0001100101110000
7    1111100000000000
8    0110000000000000
9    0110000000000000
10   0110000000000000
11   0110000000000000
12   0110000000000000
```

Data Memory Content =>

Memory Data - /processor7/data_memory_dut/ram_array - Default

0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	7
8	17
9	19
10	x
11	x
12	x
13	x

Reg file content at the end of execution in modelsim =>

Memory Data - /processor7/reg_file/reg_array

0	1
1	3
2	24
3	22
4	20
5	42
6	62
7	10

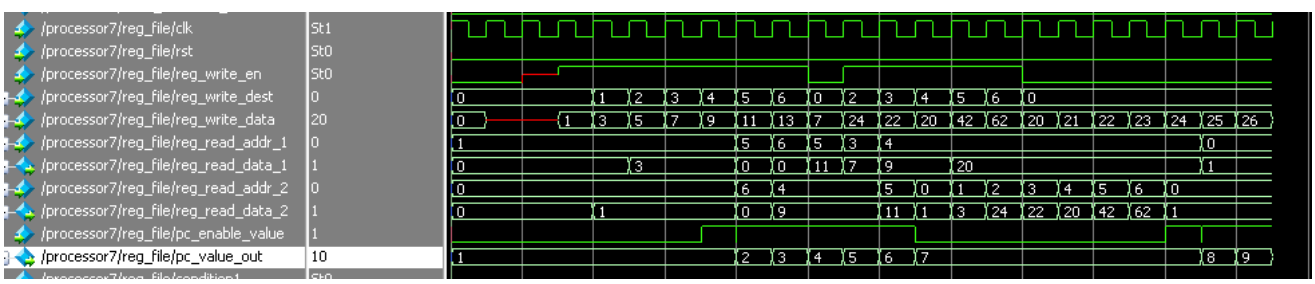
Data Memory Content after executon (since SA stored all registers from R0 to R6)=>

Memory Data - /processor7/data_memory_dut/ram_array

3	7
4	9
5	11
6	13
7	7
8	17
9	19
10	x
11	x
12	x
13	x
14	x
15	x
16	x
17	x
18	x
19	x
20	1
21	3
22	24
23	22
24	20
25	42
26	62
27	x
28	x

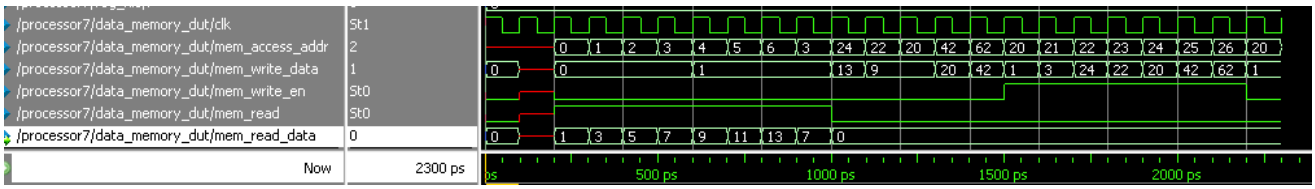
Simulation Waveforms =>

Register File data flow from it's ports =>



Here as always, R7 = PC Counter, represented by pc_value_out in above figure.

Data Memory data flow from it's ports =>



Here mem_read and mem_write signals are clearly shown.

TEST4

Instructions =>

LW R1, R0, 1

LW R2, R0, 2

LW R3, R0, 3

BEQ R2, R3, 3 //there is a dependency, need R2, R3

ADD R2, R2, R1 // there is a dependency, need R2

JRI R0, 3

NOP

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
0100001000000001
0100010000000010
0100011000000011
1000010011000011
0001010001010000
1011000000000011
0110000000000000
0110000000000000
0110000000000000
0110000000000000
0110000000000000
```

Data Memory Content =>

Memory Data - /processor7/data_memory_dut/ram_array

0	x
1	2
2	3
3	15
4	x
5	x
6	x
7	x

Register File content after first time encountering of Branch and jump instructions =>

Memory Data - /processor7/reg_file/reg_array

0	0
1	2
2	3
3	15
4	0
5	0
6	0
7	3

Look up table after first time encountering of Branch and jump instructions =>

Memory Data - /processor7/look_up_table_branch - Default

00000000	00000000000000110000000000000011001
00000001	00000000000001010000000000000001111
00000002	00000000000000000000000000000000000
00000003	00000000000000000000000000000000000
00000004	00000000000000000000000000000000000
00000005	00000000000000000000000000000000000
00000006	00000000000000000000000000000000000
00000007	00000000000000000000000000000000000

Here Branch has History Bit = 01, initially. Further it can be changed based on taken or not taken branch.

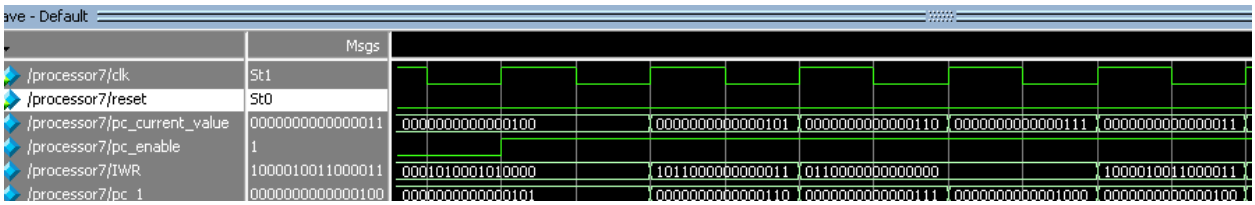
But Jump instruction is always taken branch instruction, so History Bit = 11 always.

Look up table after 2nd time encountering Branch instruction (1st entry in look table), History Bit has been changed to 00. =>

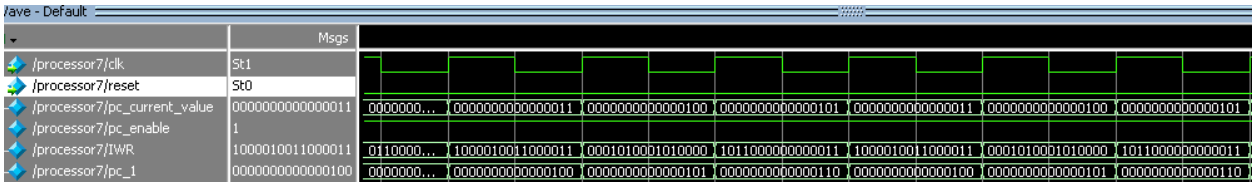
Memory Data - /processor7/look_up_table_branch - Default

00000000	00000000000000110000000000000011000
00000001	00000000000001010000000000000001111
00000002	00000000000000000000000000000000000
00000003	00000000000000000000000000000000000
00000004	00000000000000000000000000000000000
00000005	00000000000000000000000000000000000
00000006	00000000000000000000000000000000000
00000007	00000000000000000000000000000000000

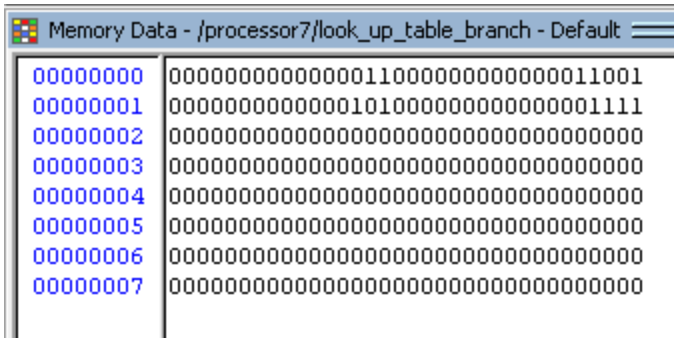
Here , in below image, we can see first time after jump it has to go beyond, because there is no entry in look table, pc_current_value = 5,6,7, then 3



But after entry in look up table, pc_current_value = 5,3,4,5,3…… So, here look table table is saving some cycles per instructions (CPI)

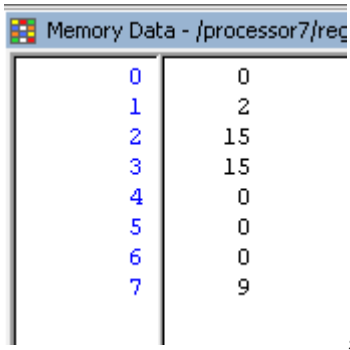


Look up table after branch taken to exit the loop =>



History bit here now is 01

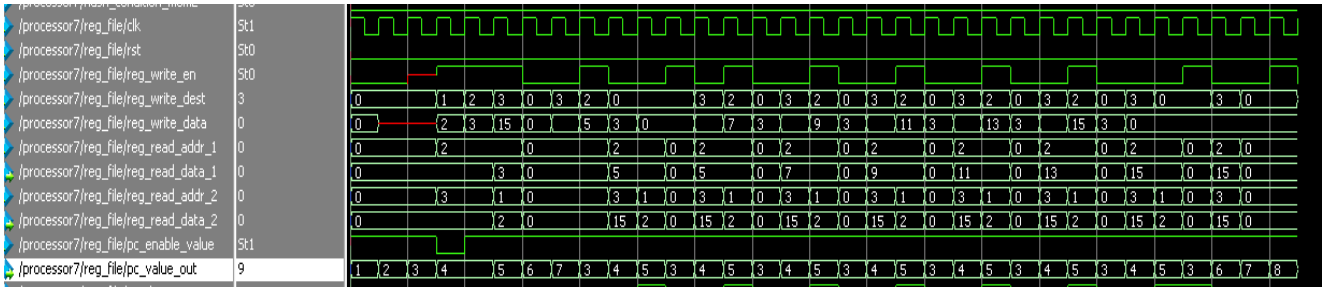
Register File Content after complete execution = >



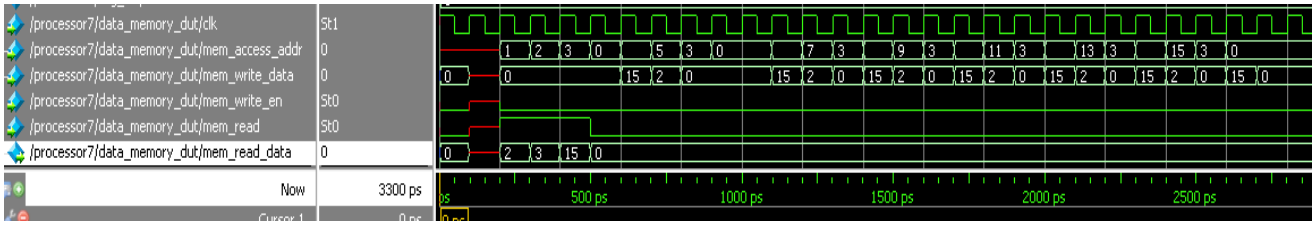
, Here loop breaks after R2 = R3 = 15.

Simulation Waveforms =>

Register File data flow from it's ports =>



Data Memory data flow from it's ports =>



Here there can be unnecessary data at it's ports, but mem_read and mem_write knows when to read and when to write, which is given by processor.

TEST5

Instructions =>

LM R0, 11111000

ADD R3, R1, R2 // Here R3 would be 12

NDU R0, R3, R4 // (12 NAND 26) = 1111 1111 1111 0111 = -9

ADZ R1, R3, R0 // Zero not asserted before, so no write here, but carry will be 1

ADC R5, R1, R3 //here carry was 1, so write R5 = 21

ADC R5, R1, R3 //here carry was 0 from previous, so not write R5=21

NOP

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
1    1100000011111000
2    0001001010011000
3    0010011100000000
4    0001011000001001
5    0001001011101010
6    0001001011101010
7    0110000000000000
8    0110000000000000
9    0110000000000000
10   0110000000000000
11   0110000000000000
```

Data Memory Content =>

Memory Data - /processor7/data_memory_dut/ram_array -

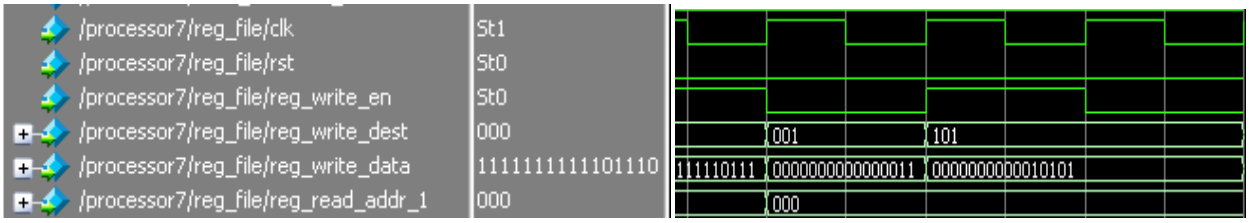
0	8
1	9
2	3
3	4
4	26
5	6
6	7
7	8
8	9
9	10
10	11
11	x
12	x

Register File Content after execution =>

Memory Data - /processor7/reg_file/reg_array

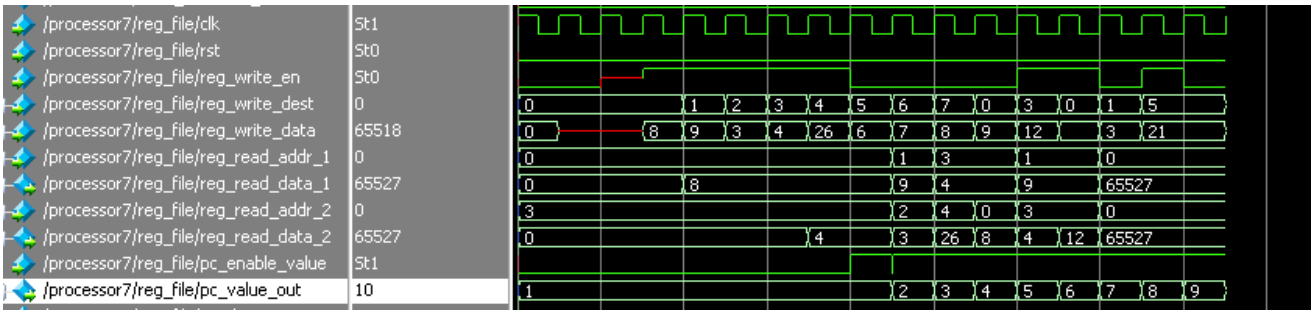
0	-9
1	9
2	3
3	12
4	26
5	21
6	0
7	10

Reg_write = 0 for last ADC instruction =>

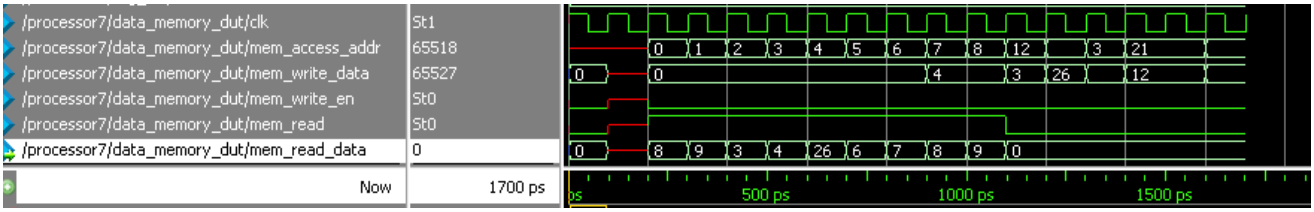


Simulation Waveforms = >

Register File data flow from it's ports =>



Data Memory data flow from it's ports =>



TEST6

Instructions =>

LW R1, R0, 1

LW R2, R0, 2

ADD R3, R1, R2 // there is a dependency, need R1, R2

ADD R4, R2, R3 // there is a dependency, need R2, R3

JAL R5, 7 // It is function call, so make a new entry in look up table

ADD R5, R3, R4 // skipped for the first time

ADD R5, R5, R4 // skipped for the first time

NOP

NOP

NOP

NOP

LW R6, R0, 3 // starting of function

ADD R4, R6, R4 // there is a dependency, need R6

ADD R4, R4, R3 // there is a dependency, need R4

ADD R3, R4, R2 // there is a dependency, need R3, R4

JRI R5, 0 // return to next instruction of function call instruction

```
: > intelFPGAlite_workspace > pd_project_pipeline_up
1    0100001000000001
2    0100010000000010
3    0001001010011000
4    0001010011100000
5    1001101000000111
6    0001011100101000
7    0001101100101000
8    0110000000000000
9    0110000000000000
10   0110000000000000
11   0110000000000000
12   0100110000000011
13   0001110100100000
14   0001100011100000
15   0001100010011000
16   1011101000000000
```

Data Memory Content =>

Memory Data - /processor7/data_memory_dut/ram_array -

0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	7
8	17
9	19
10	x
11	x

Look up table having entry of functional call from JAL and return instruction from JRI =>

Memory Data - /processor7/look_up_table_branch

0	00000000000001000000000000000101111
1	00000000000011110000000000000010111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

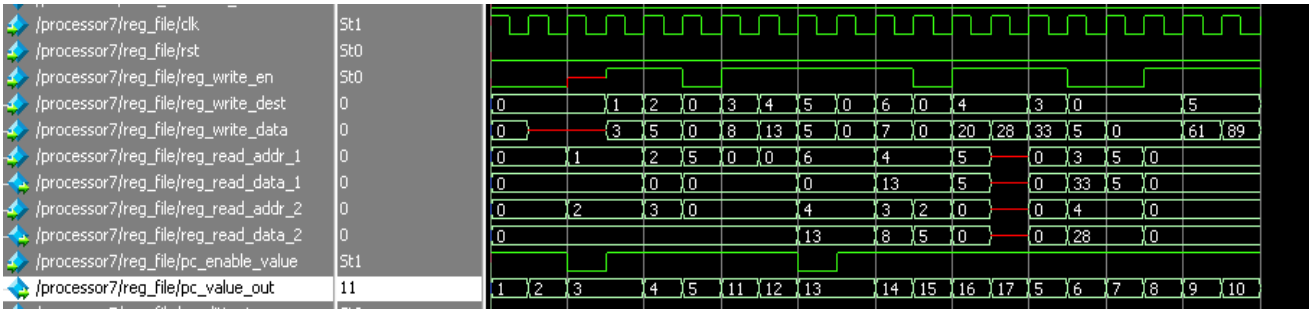
Register file content after execution =>

Memory Data - /processor7/reg_file/reg_

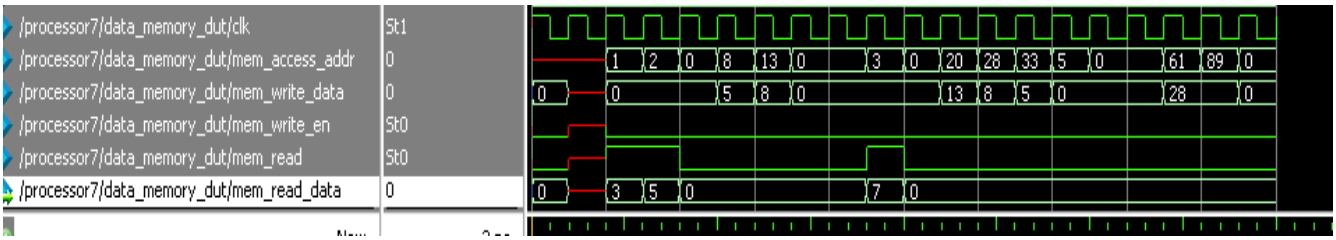
0	0
1	3
2	5
3	33
4	28
5	89
6	7
7	12

Simulation Waveforms =>

Register File data flow from it's ports =>



Data Memory data flow from it's ports =>



TEST7 (Fibonacci Sequence generating)

Instructions =>

LW R1, R0, 1

LW R2, R0, 1

LW R5, R0, 2

BEQ R4, R5, 7 // there is a dependency, need R5

ADD R3, R1, R2 //there is a dependency, need R2

ADI R1, R2, 0

ADI R2, R3, 0 // there is a dependency, need R3

SW R3, R4, 5 // there is a dependency, need R3

ADI R4, R4, 1

JRI R0, 3

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
1     0100001000000001
2     0100010000000001
3     0100101000000010
4     1000100101000111
5     0001001010011000
6     0000010001000000
7     0000011010000000
8     0101100011000101
9     0000100100000001
10    1011000000000011
11    0110000000000000
12    0110000000000000
13    0110000000000000
14    0110000000000000
15    0110000000000000
```

Data Memory before execution =>

Memory Data - /processor7/data_memory_dut/ram_array -

0	x
1	1
2	15
3	x
4	x
5	x
6	x
7	x

Look up table after encountering first time Branch and jump =>

y Data - /processor7/look_up_table_branch - Default

0	00000000000000110000000000000101001
1	00000000000000000000000000000000000
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

y Data - /processor7/look_up_table_branch

0	00000000000000110000000000000101001
1	00000000000001001000000000000001111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Register File Content initially =>

Memory Data - /processor7/reg_file/reg_arra

0	0
1	1
2	2
3	2
4	1
5	15
6	0
7	5

Look up table after again encountering branch, making it confirm not to take branch, i.e. History Bit = 00, =>

Memory Data - /processor7/look_up_table_branch

0	00000000000000110000000000000101000
1	00000000000001001000000000000001111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Look up table after loop exit, here Branch instruction History Bit would become 01=>

ory Data - /processor7/look_up_table_branch

0	00000000000000110000000000000101001
1	00000000000001001000000000000001111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Register File Content after execution =>

Memory Data - /processor7/reg_file/reg_array

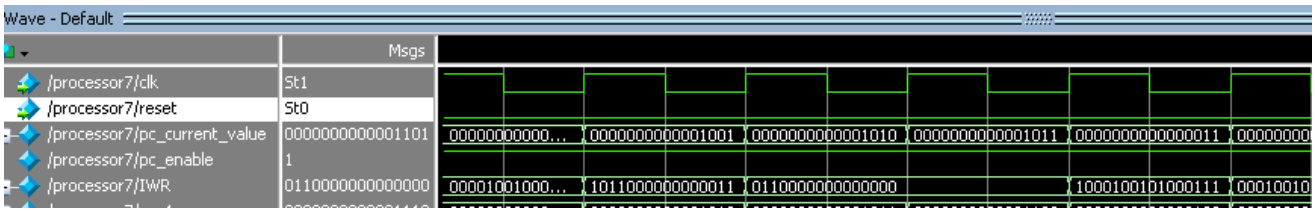
0	0
1	987
2	1597
3	1597
4	15
5	15
6	0
7	13

Data Memory Content after written 15 Fibonacci no from memory address 5 =>

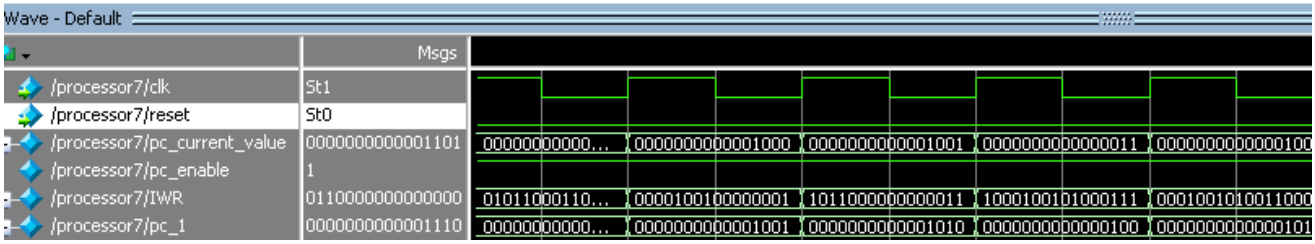
Memory Data - /processor7/data_memory_dut/ram_array

0	x
1	1
2	15
3	x
4	x
5	2
6	3
7	5
8	8
9	13
10	21
11	34
12	55
13	89
14	144
15	233
16	377
17	610
18	987
19	1597
20	x
21	x
22	x

Waveforms showing CPI improvement by look up table =>



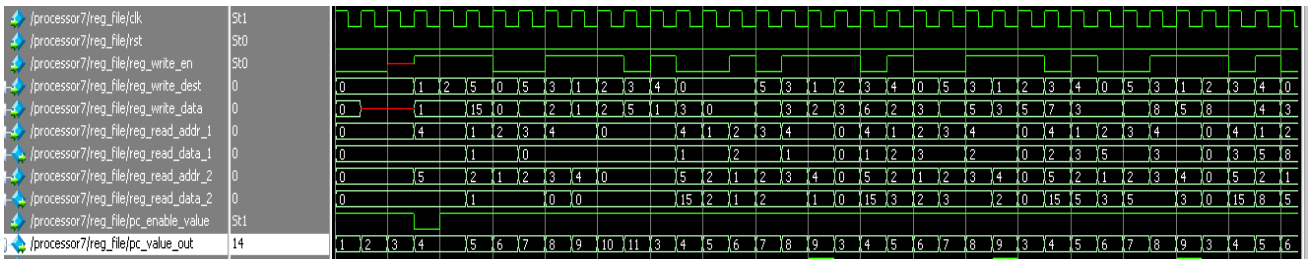
Here in waveforms, we can see that first time after jump (PC=9), it had fetched PC=10, 11. But after again fetching same jump instruction, then it did not fetch PC=10 instr. instead it looked in look up table and got Branch Target Address, and fetched PC=3 directly =>

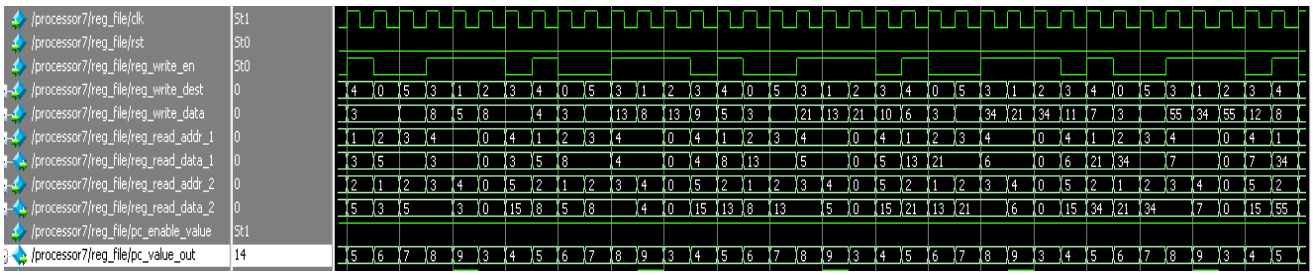


So, in the loop, with the help of look up table, PC is going like = 3,4,5,6,7,8,9,3,4…..

Simulation Waveforms =>

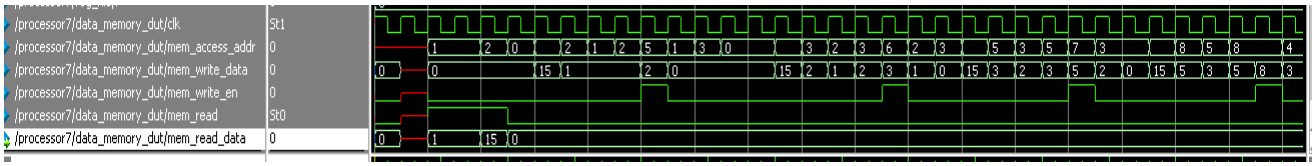
Register file data in and out of port =>

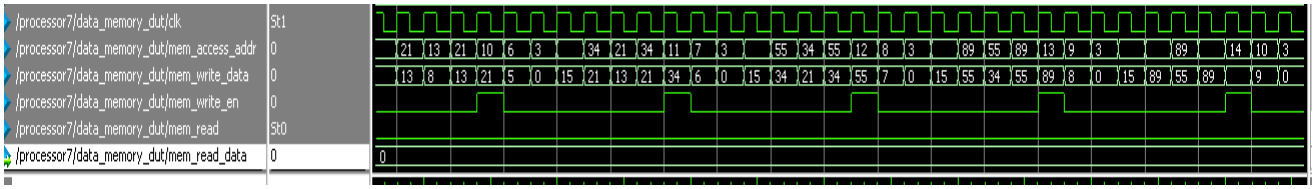


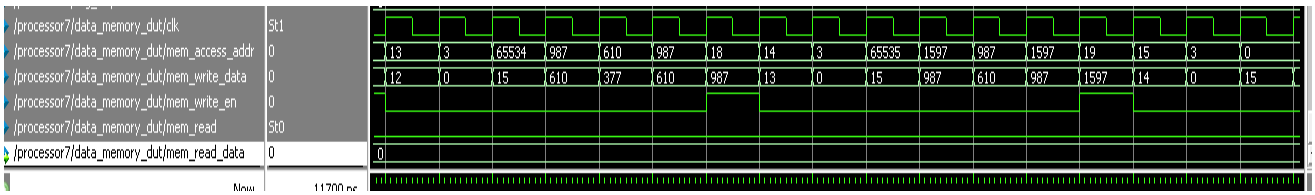


Here pc_value_out is R7.

Data Memory Waveform =>

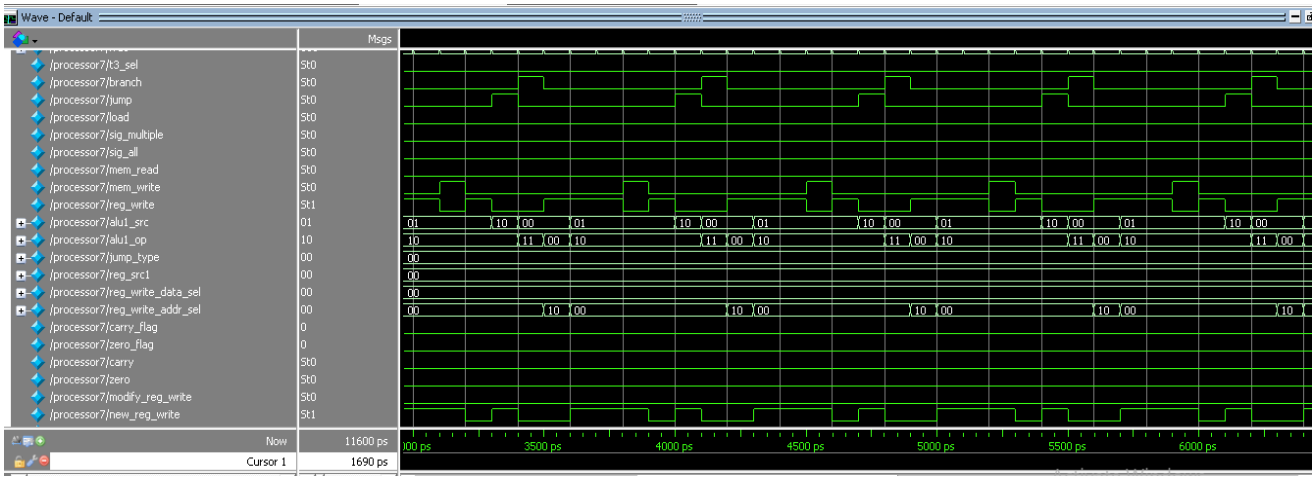




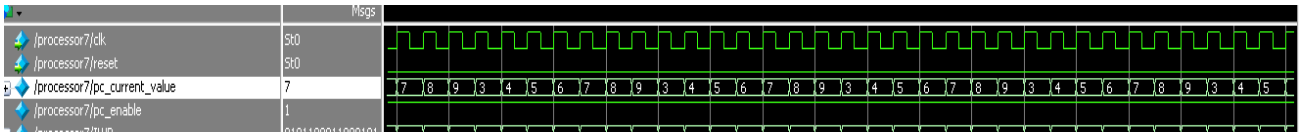


Clearly it can be seen, that Fibonacci numbers are being written on successive memory address.

Control Signals =>



PC current changing as required =>



TEST8 (Addition of 2 arrays)

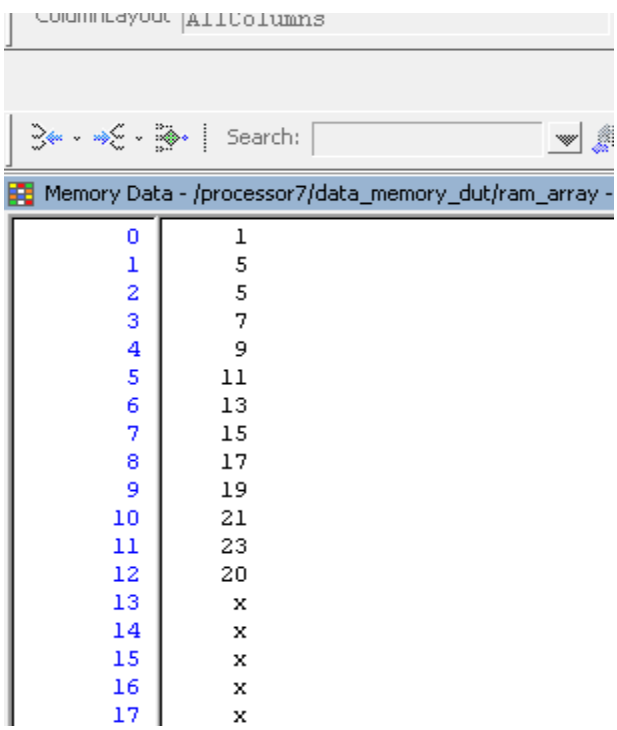
Instructions =>

LW R2, R0, 1	// [R2] ← Mem([R0] + 1)
BEQ R1, R2, 7	//there is a dependency, need R2
LW R3, R1, 2	// [R3] ← Mem([R1] + 2)
LW R4, R1, 7	// [R4] ← Mem([R1] + 7)
ADD R5, R3, R4	// there is a dependency , need R3, R4 (It is R5=R3+R4)
SW R5, R1, 12	//Mem([R1] + 12) ← [R5], dependency here, need R5
ADI R1, R1, 1	// [R1] ← [R1] + 1
JRI R0, 1	// jump to [R0] + 1
NOP	

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
0100010000000001
1000001010000111
0100011001000010
0100100001000111
0001011100101000
0101001101001100
0000001001000001
1011000000000001
0110000000000000
0110000000000000
0110000000000000
0110000000000000
```

Data Memory Content at the starting of execution =>

We have 2 arrays or vectors of length 5 each. In memory, one array is from address 2, and other array is from address 7



0	1
1	5
2	5
3	7
4	9
5	11
6	13
7	15
8	17
9	19
10	21
11	23
12	20
13	x
14	x
15	x
16	x
17	x

Register File Content during and at the end of execution =>

Memory Data - /processor7/reg_file/reg_array

0	0
1	0
2	5
3	5
4	15
5	20
6	0
7	9

ory Data - /processor7/reg_file/reg_array

0	0
1	2
2	5
3	9
4	19
5	28
6	0
7	2

ory Data - /processor7/reg_file/reg_array -

0	0
1	3
2	5
3	11
4	21
5	32
6	0
7	3

ory Data - /processor7/reg_file/reg_array

0	0
1	5
2	5
3	13
4	23
5	36
6	0
7	10

Look Up table Content during and at end of execution =>

y Data - /processor7/look_up_table_branch - Default

0	00000000000000010000000000000100001
1	00000000000000000000000000000000000
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Data - /processor7/look_up_table_branch - Default

0	00000000000000010000000000000100001
1	00000000000000111000000000000000111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Search:

y Data - /processor7/look_up_table_branch - Default

0	00000000000000010000000000000100000
1	00000000000001110000000000000000111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Search:

mory Data - /processor7/look_up_table_branch

0	00000000000000010000000000000100001
1	00000000000001110000000000000000111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Data Memory Content after execution =>

Search:

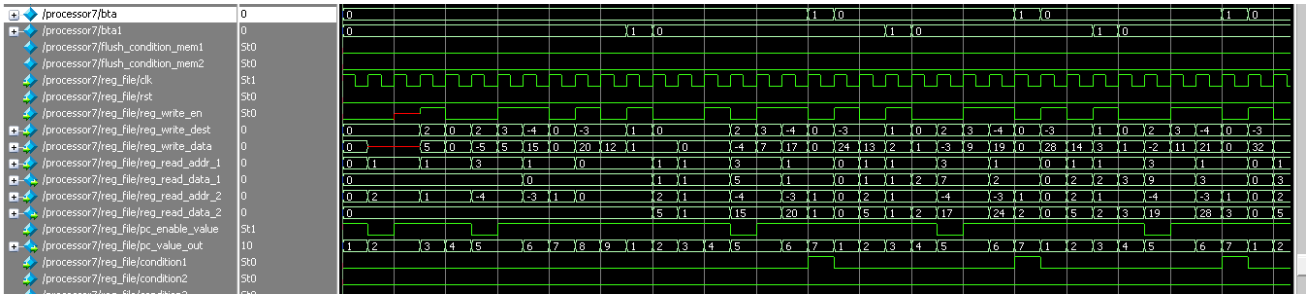
nory Data - /processor7/data_memory_dut/ram_array

0	1
1	5
2	5
3	7
4	9
5	11
6	13
7	15
8	17
9	19
10	21
11	23
12	20
13	24
14	28
15	32
16	36
17	x
18	x
19	x
20	x

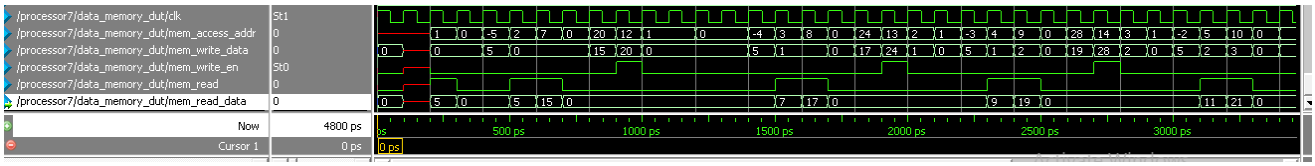
Here we can see from memory address 12 onwards, we are getting result of 2 vectors/arrays of length 5.

Simulation Waveforms =>

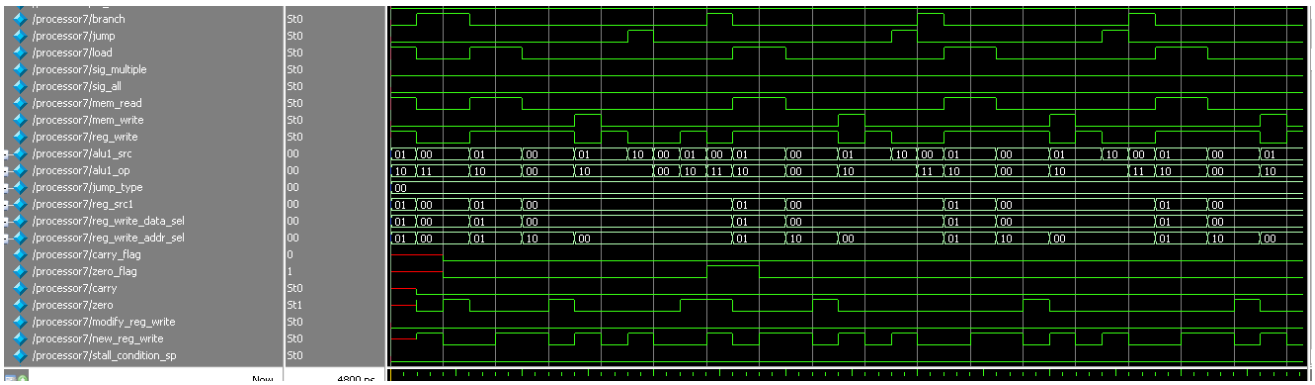
Reg_file ports data



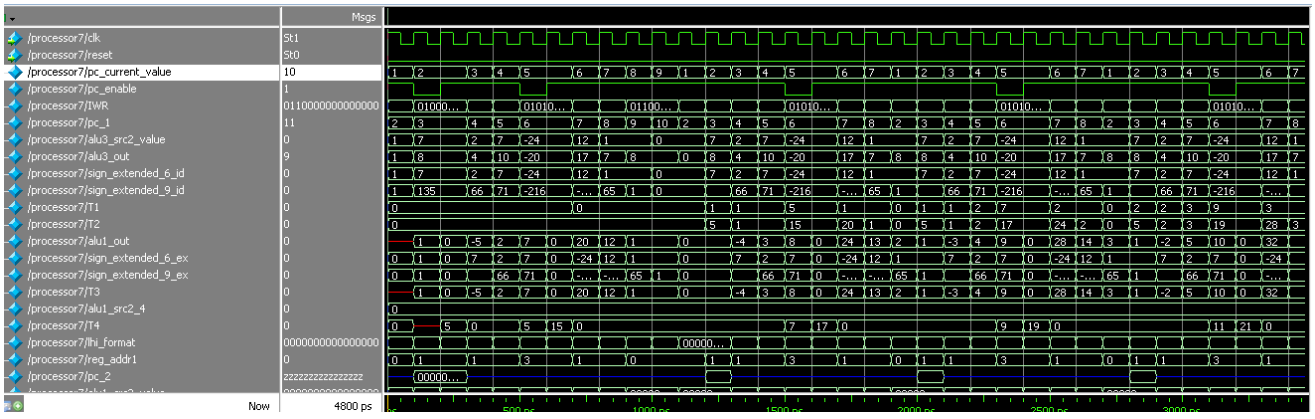
Data Memory data flow =>



Control Signals and stall condition activation =>



Some intermediate Signals and pc_current_value =>



TEST9 (Minimum number finder)

Instructions =>

```
LW  R5,  R0,  1          // R5 ← 1
LW  R2,  R0,  2          // R2 ← 5 ( i.e size of array )
LW  R4,  R0,  3          //  R4 ← 3 ( i.e first element of an array )
BEQ  R2,  R5,  8   // there is a dependency, need R2, R5, which has not been written
LW  R6,  R5,  3          //R6 ←mem(1 + 3) = 5
NDU  R3,  R6,  R6        // nand all R6 with R6  to get inverted bits of R6 in R3
ADI  R3,  R3,  1         // add 1 to get 2's compliment of no.
ADD  R3,  R4,  R3        // R3 ← R4 + R3, if  +ve, carry generated, else not
ADC  R4,  R6,  R0        // if carry generated previously , then only work
ADI  R5,  R5,  1         // R5 ← R5 + 1,  increment the loop counter
JRI  R0,  3              // jump to branch instr.
SW  R2,  R4,  4          // store the result of minimum value after loop ends
NOP
```

```
> intelFPGAlite_workspace > pd_project_pipeline_up >
1    0100101000000001
2    0100010000000010
3    0100100000000011
4    1000010101001000
5    0100110101000011
6    0010110110011000
7    0000011011000001
8    0001100011011000
9    0001110000100010
10   0000101101000001
11   1011000000000011
12   0101010100000100
13   0110000000000000
14   0110000000000000
15   0110000000000000
16   0110000000000000
```

Data Memory content before execution =>

Memory Data - /processor7/data_memory_dut/ram_array - Defau

0	x
1	1
2	5
3	3
4	5
5	1
6	2
7	4
8	x
9	x
10	x

As we can see, there is no result at address 9, as of now.

We have array of 5 length (size written at memory address 2, which program would need during execution), which has numbers as we can see = {3, 5, 1, 2, 4}, starting from memory address 3.

Register File contents during execution =>

Search:

Data - /processor7/reg_file/reg_array

0	0
1	0
2	5
3	0
4	3
5	1
6	5
7	8

Search:

Data - /processor7/reg_file/reg_array

0	0
1	0
2	5
3	-3
4	3
5	2
6	5
7	6

Search:

Data - /processor7/reg_file/reg_array

0	0
1	0
2	5
3	-2
4	1
5	3
6	2
7	4

Search:

Data - /processor7/reg_file/reg_array

0	0
1	0
2	5
3	-4
4	1
5	5
6	4
7	15

Look up table showing entries of branch instructions, conditional branch, and unconditional jump instr. =>

Search:

Memory Data - /processor7/look_up_table_branch - Default

0	00000000000000110000000000000101101
1	00000000000010100000000000000001111
2	00000000000000000000000000000000000
3	00000000000000000000000000000000000
4	00000000000000000000000000000000000
5	00000000000000000000000000000000000
6	00000000000000000000000000000000000
7	00000000000000000000000000000000000

Data Memory content after execution =>

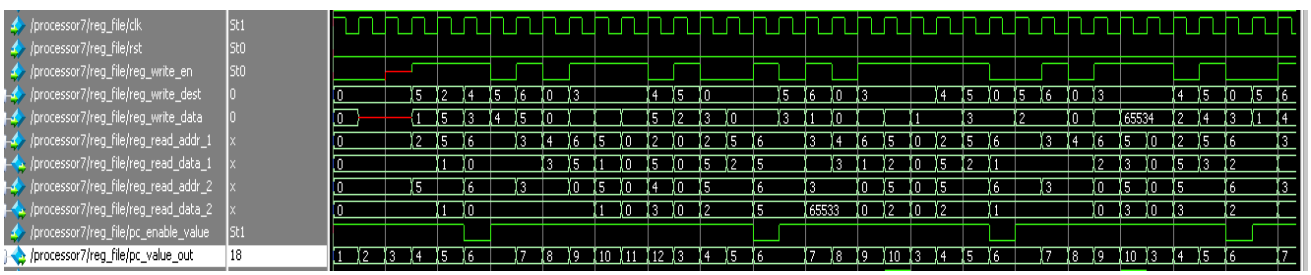
Memory Data - /processor7/data_memory_dut/ram_array - [

0	x
1	1
2	5
3	3
4	5
5	1
6	2
7	4
8	x
9	1
10	x
11	x

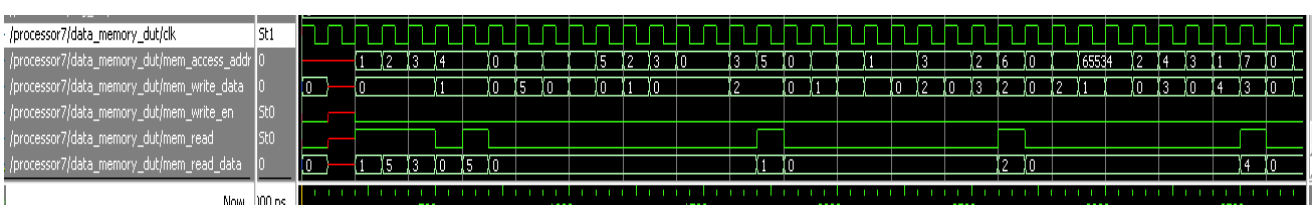
Here we can see, memory address 9 has minimun value "1", among numbers (3,5,1,2,4).

Simulation Waveforms =>

Register File data flow from it's ports =>



Data Memory data flow from it's ports =>



Conclusion

We made a processor based on given Instruction Set Architecture, and it is required that it should overcome all dependency issues , so that no pipelining hazards should occur and also to improve CPI, it should have Branch Prediction system and also R7 register should store PC value. So after making whole design, we picked couple of test examples and run on our design of IITB-RISC processor, and **all test cases successfully passed** and all simulation waveforms and modelsim output screenshots are attached with the test examples.