ECE 552 - Introduction to Computer Architecture

THE UNIVERSITY of WISCONSIN MADISON

Aman Rakesh Chadha
9068354597

1 Overview

a Introduction

Wisc F12 is a *16 bit 5 stage pipelined processor* which is designed to satisfy the specified instruction set. It consists of 5 pipeline stages to increase the *throughput* of the processor as compared to a single cycle processor.

Wisc F12 has been designed to tackle various data hazards and control hazards by using techniques like *forwarding*, *stalling* and *flushing* of the pipeline stages.

The implemented pipeline stages are:

- ✓ IF/ID
- ✓ ID/EX
- ✓ EX/MEM
- ✓ MEM/WB

b List of components generated by MegaWizard

- ✓ Data Memory (16 Flipflops each of 16 Bits)
- ✓ Instruction Memory (16 Flipflops each of 16 Bits)
- ✓ Register File (16 Flipflops each of 16 Bits)
- ✓ Multiplexers (2 to 1, 3 to 1, 4 to 1, 8 to 1 and 16 to 1)
- ✓ Arithmetic Right Shifter (16 bit)
- ✓ Logical Right Shifter (16 bit)
- ✓ Logical Left Shifter (16 bit)
- ✓ Adder_subtractors (16 bit)
- ✓ AND, NOT, OR, EX-OR gates (2, 3, 4, 5 inputs)

c Efforts made to optimize the design

Branching and calling was handled in the *ID/EX stage* to prevent flushing of too many instructions. HALT was handled in the *IF/ID stage* to avoid flushing of too many instructions.

d Does the design meet all the requirements?

The design does well in most of the cases. However, it does not do well in case of *consecutive calls and returns*.

2 Machine Organization

a Overview of the pipeline stages

IF Stage:
The instruction is *fetched* from the instruction memory into the *IF/ID pipeline* register. The PC value and incremented PC value are also saved for further use in later stages. Hence, the *primary objective* of this stage is to deal with the *fetching* of instructions.

ID Stage:
The *control logic* has been implemented in the *ID* stage. Select signals to most multiplexers used are given out from this control logic on basis of Instruction OPCODE.

The *General Purpose Registers*, in the form of a *Register File,* also reside in this stage. Thus, this stage deals with the *decoding* of the instructions. The values of the control signals and instruction value/PC value are saved into the *ID/EX* pipelined register for use by the later stages.

EX Stage:
The EX Stage is powered by the ALU which performs the various arithmetic and logical operations. Branch and Call are decided in this stage on basis of the condition specified by the control logic as well as the offset provided by the instruction.

When a branch or call condition is true, the control signals are made *zero* in order to *flush the pipeline*. The relevant control and data values are pipelined into the *EX/MEM register* on the next clock cycle.

MEM Stage:
This stage deals with issues like data forwarding and also stalls the pipeline if a load instruction followed by a RAW dependent instruction appears. Data memory resides in this stage and control signals like *RegFileWR* and *LW_MemRead* are passed on to this stage for future use. Return instruction is executed in this stage.

WB Stage:
In this stage, the values are *written back* to the registers during the positive clock cycle edge. It also functions as a *data selecting unit* (implemented as a multiplexor) wherein it feeds in either the *output of the ALU* or the *data obtained from the Data Memory* back to the Register File.

b Implementation of instruction memory, registers, data memory

A *16 × 16 Data Memory* and *Instruction Memory* was created using 16 bit flip flops (16 nos.) using the *MegaWizard function*. Similarly, a *16 × 16 register file* was created. Registers were initialized to suitable values to debug and test the processor.

c Strategies to handle pipeline hazards - data and control hazards

Pipeline hazards, viz. *data* and *control* hazards, were handled by using forwarding, stalling and flushing techniques.

Data dependencies such as *RAW* were resolved using the forwarding technique. Data was pre-fetched from the EX/MEM and WB stages to supply to the ALU inputs. Hence, data forwarding was implemented at:

- ✓ *EX Stage*
- ✓ *MEM Stage*

d Strategies to handle the Load Use data hazard

In case of a Load Use data hazard where a RAW dependency cannot be resolved by forwarding, pipeline was stalled to make sure that the *correct data was fetched*. Pipeline registers were *disabled* to achieve this and the *PC was frozen* in order to avoid an instruction miss.

Control Hazards like *branching* and *call* were dealt with by making a pseudo control logic block in the ID/EX state which flushes out all the control signals in the upcoming undesirable instructions until the desired PC value reaches the IF stage. Calls were also handled in the *ID/EX stage* where the program counter value is saved into the *location pointed by RF[15]* and SP is then *incremented* by 1. Return happens in *EX/MEM stage* where SP is *decremented* by 1 and the saved program counter value is then *restored* into PC.

e Description of the structure of the four pipeline registers

IF/ID Register

1	IF/ID.PC + 1	Incremented value of PC
2	IF/ID.External_DataLoad_Reg[15:0]	External Load Signal to load registers in the beginning
3	IF/ID.External_load	External Load Control Signal to switch on the external loading process
4	IF/ID.External_DestAddr[3:0]	External Load Address to specify where the external data is to be fed
5	IF/ID.DM_Load[15:0]	External Data Memory Load Signal to specify the data is to be written in the address specified by DM_Addr
6	IF/ID.DM_Addr[3:0]	External Data Memory Address Signal which is used to indicate the address in the Data Memory where the data in DM_Load is to be written to

ID/EX Register

1	ID/EX.rd[3:0]	Value of RD passed through the ID stage
2	ID/EX.PC+1	Incremented value of PC
3	ID/EX.RET2	Value of RET passed through the ID stage
4	ID/EX.A[15:0]	Value of first operand
5	ID/EX.B[15:0]	Value of second operand
6	ID/EX.Ins_out_RF[15:0]	Value of the Instruction at the output of the RF
7	ID/EX.SignEx_Imm[15:0]	Value of the sign extended 8-bit data
8	ID/EX.Control[7:0]	Value of the Control Signals
9	ID/EX.8bitimm_In	Value of the 8bitimm signal passed to the Control Signals unit of the ID/EX pipeline. Set if the instruction has 8-bit data coming in or else reset.
10	ID/EX.lhborllb_in	Value of the lhborllb signal passed to the Control Signals unit of the ID/EX pipeline. Set if the instruction is either a LHB or LLB

		or else reset.
11	ID/EX.lhbllbbar_In	Value of the lhbllbbar signal passed to the Control Signals unit of the ID/EX pipeline. Set if the instruction is either a LHB or else reset if it's a LLB.
12	ID/EX.AddrCalc_In	Value of the AddrCalc signal passed to the Control Signals unit of the ID/EX pipeline. Set if the instruction needs internal address calculation from the ALU.
13	ID/EX.IncDecSP_In	Value of the IncDecSP signal passed to the Control Signals unit of the ID/EX pipeline. Set if the SP is to be incremented or else reset if it is to be decremented.
14	ID/EX.SPsel_In	Value of the IncDecSP signal passed to the Control Signals unit of the ID/EX pipeline. Used to select the MUX which feeds in the value of SP-1 or DOUT (ALU Result) to the Data Memory Address. Also used to select the MUX which feeds in the value of PC+1 or B to the Data Memory.
15	ID/EX.LW_MemRead_In	Value of the LW_MemRead signal passed to the Control Signals unit of the ID/EX pipeline.
16	ID/EX.MemWrite_In	Value of the LW_MemRead signal passed to the Control Signals unit of the ID/EX pipeline.
17	ID/EX.RegFileWR_In	Value of the LW_MemRead signal passed to the Control Signals unit of the ID/EX pipeline.
18	ID/EX.ForwardEX_A	High when dataA at the input of the ALU is to be forwarded from the EX stage of the previous instruction.
19	ID/EX.ForwardEX_B	High when dataB at the input of the ALU is to be forwarded from the EX stage of the previous instruction.
20	ID/EX.ForwardMEM_A	High when dataA at the input of the ALU is to be forwarded from the MEM stage of the previous instruction.
21	ID/EX.ForwardMEM_B	High when dataB at the input of the ALU is to be forwarded from the MEM stage of the previous instruction.

EX/MEM Register

1	EX/MEM.rd[3:0]	Value of RD passed through the EX stage
2	EX/MEM.PC+1	Incremented value of PC
3	EX/MEM.RET3	Value of RET passed through the EX stage
4	EX/MEM.A[15:0]	Value of first operand
5	EX/MEM.B[15:0]	Value of second operand
6	EX/MEM.ans[15:0]	Output of the ALU
7	EX/MEM.OutALU[15:0]	Value of the Instruction at the output of the ALU
8	EX/MEM.IncDecSP_In	Value of the IncDecSP signal passed to the Control Signals unit of the EX/MEM pipeline. Set if the SP is to be incremented or else reset if it is to be decremented.
9	EX/MEM.SPsel_In	Value of the IncDecSP signal passed to the Control Signals unit of the EX/MEM pipeline. Used to select the MUX which feeds in the value of SP-1 or DOUT (ALU Result) to the Data Memory Address. Also used to select the MUX which feeds in the value of PC+1 or B to the Data Memory.
10	EX/MEM.LW_MemRead_In	Value of the LW_MemRead signal passed to the Control Signals unit of the EX/MEM pipeline.
11	EX/MEM.MemWrite_In	Value of the LW_MemRead signal passed to the Control Signals unit

		of the EX/MEM pipeline.
12	EX/MEM.RegFileWR_In	Value of the LW_MemRead signal passed to the Control Signals unit of the EX/MEM pipeline.

MEM/WB Register

1	ID/EX.rd[3:0]	Value of RD passed through the MEM stage
2	ID/EX.SP[15:0]	Value of the Stack Pointer passed on from the EX stage
3	ID/EX.ans[15:0]	Output of the ALU
4	ID/EX.DM_dataOut[15:0]	Output of the Data Memory
5	ID/EX.Ins4[15:0]	Value of the Instruction passed on from the MEM stage
6	ID/EX.IncDecSP_In	Value of the IncDecSP signal passed to the Control Signals unit of the EX/MEM pipeline. Set if the SP is to be incremented or else reset if it is to be decremented.
7	ID/EX.SPsel_In	Value of the IncDecSP signal passed to the Control Signals unit of the EX/MEM pipeline. Used to select the MUX which feeds in the value of SP-1 or DOUT (ALU Result) to the Data Memory Address. Also used to select the MUX which feeds in the value of PC+1 or B to the Data Memory.
8	ID/EX.LW_MemRead_In	Value of the LW_MemRead signal passed to the Control Signals unit of the EX/MEM pipeline.
9	ID/EX.RegFileWR_In	Value of the LW_MemRead signal passed to the Control Signals unit of the EX/MEM pipeline.

Hand-Simulated TestBench Programs

a Program 1

RF[1]=1, RF[2]=12, RF[3] = 13, rest of RF are cleared

IM Addr	Program	Result
0	ADD $1, $2, $3	RF[1] ← 12+13=25 (or 0x19)
1	SLL $4, $1, 2	RF[4] ← SLL(0x19) = 100 (or 0x64)
2	XOR $5, $1, $4	RF[5] ← 125 (or 0x7D)
3	SUB $6, $5, $1	RF[6] ← 0x7D – 0x19 = 0x64
4	HALT	Halt processor
5	INC $3, $2, 1	Cannot be fetched as processor is in a Halt State

b Program 2

RF[1]=1, RF[2]=2, RF[3] = 3, rest of RF are cleared. DM[2]=11, rest DM are cleared.

IM Addr	Program	Result
0	LW $4, 2	RF[4] ← 11
1	SUB $14, $4, $2	RF[14] ← RF[4] – RF[2] = 11-2 = 9
2	SW $3, 0	DM[0] ← RF[3]
3	LW $4, 0	RF[4] ← DM[0]
4	SUB $4, $1, $4	RF[4] ← 1 – 3 = 0001 - 0011 = FFFE
5	SRA $4, $4, 3	RF[4] ← SRA(FFFE) by 3 = FFFF
6	LLB $4, 0	RF[4] ← {FF,00} = FF00
7	HALT	Halt processor

c Program 3

RF[1]=1, RF[2]=2, RF[3] = 3, rest of RF are cleared. DM[2]=11, rest DM are cleared.

IM Addr	Program	Result
0	LLB $1, 255	RF[1] ← 0x00FF (or 255)
1	LOOP0: INC $5, $3, 0	RF[5] ← RF[3] + 0 = 3
2	LOOP1: ADD $6, $5, $1	RF[6] ← RF[5] + RF[1] = 0x0102 (or 258)
3	LHB $6, 0	RF[6] ← {00,LowerByte(RF[6])} = 0x0002
4	INC $5, $5, -1	RF[5] ← RF[1] – 1 = 0x0002

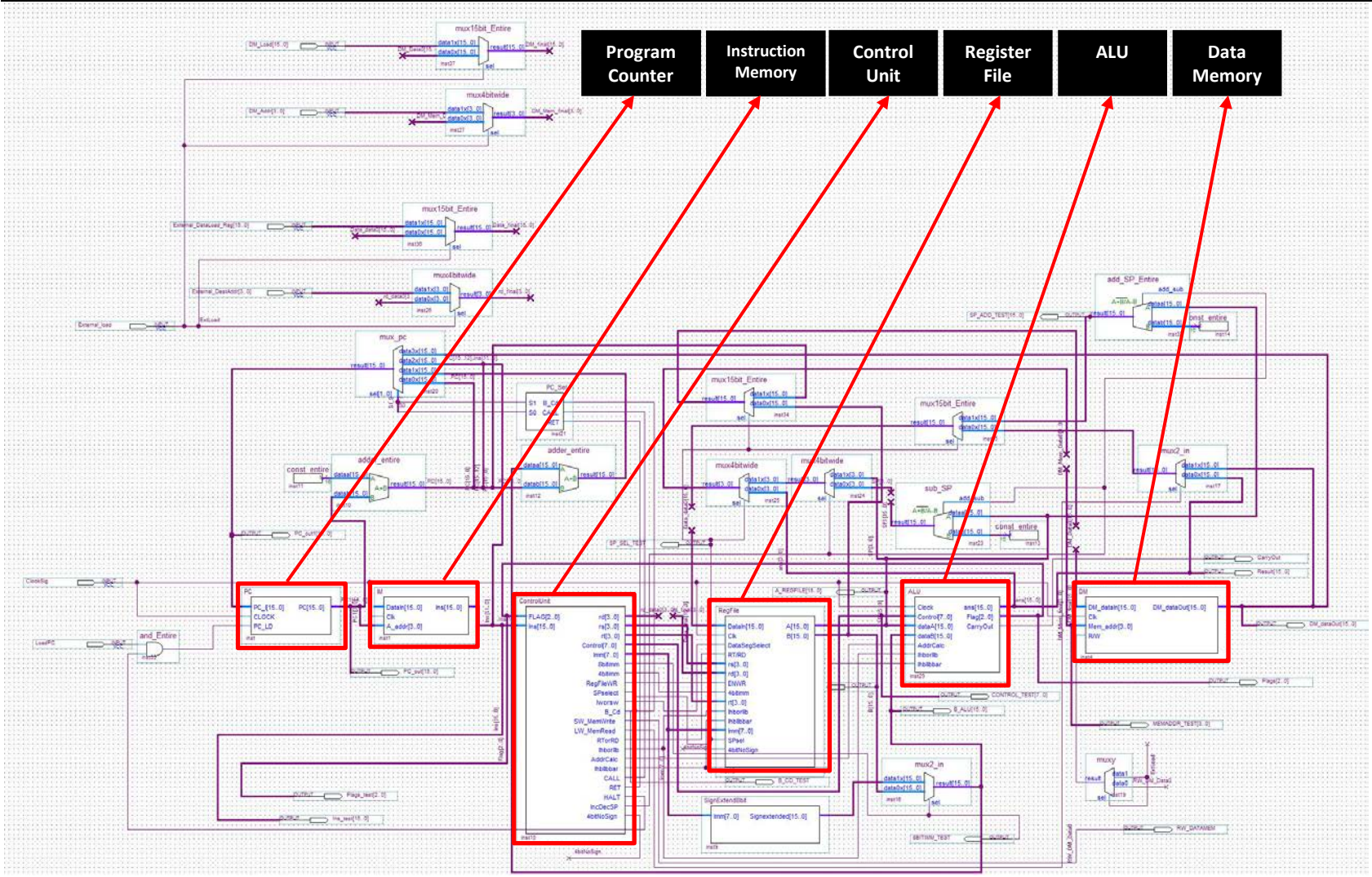
5	B GT, LOOP1	Branch to Loop1 taken as 0x0002 is positive. RF[6] ← RF[5] + RF[1] = 0x0101 (or 257) RF[6] ← {00,LowerByte(RF[6])} = 0x0001 RF[5] ← RF[1] – 1 = 0x0001 Branch to Loop1 taken as 0x0001 is positive. RF[6] ← RF[5] + RF[1] = 0x0100 (or 256) RF[6] ← {00,LowerByte(RF[6])} = 0x0000 RF[5] ← RF[1] – 1 = 0x0000 Branch to Loop1 not taken as 0 is not positive.
6	INC $2, $2, -1	RF[2] ← RF[2] - 1 = 2-1 = 1
7	B GT, LOOP0	Branch to Loop0 taken as 0x0001 is positive. RF[5] ← RF[3] + 0 = 3 RF[6] ← RF[5] + RF[1] = 0x0102 (or 258) RF[6] ← {00,LowerByte(RF[6])} = 0x0002 RF[5] ← RF[1] – 1 = 0x0002 Branch to Loop1 taken as 0x0002 is positive. RF[6] ← RF[5] + RF[1] = 0x0101 (or 257) RF[6] ← {00,LowerByte(RF[6])} = 0x0001 RF[5] ← RF[1] – 1 = 0x0001 Branch to Loop1 taken as 0x0001 is positive. RF[6] ← RF[5] + RF[1] = 0x0100 (or 256) RF[6] ← {00,LowerByte(RF[6])} = 0x0000 RF[5] ← RF[1] – 1 = 0x0000 RF[2] ← RF[2] - 1 = 0
8	HALT	Halt processor

c Program 4

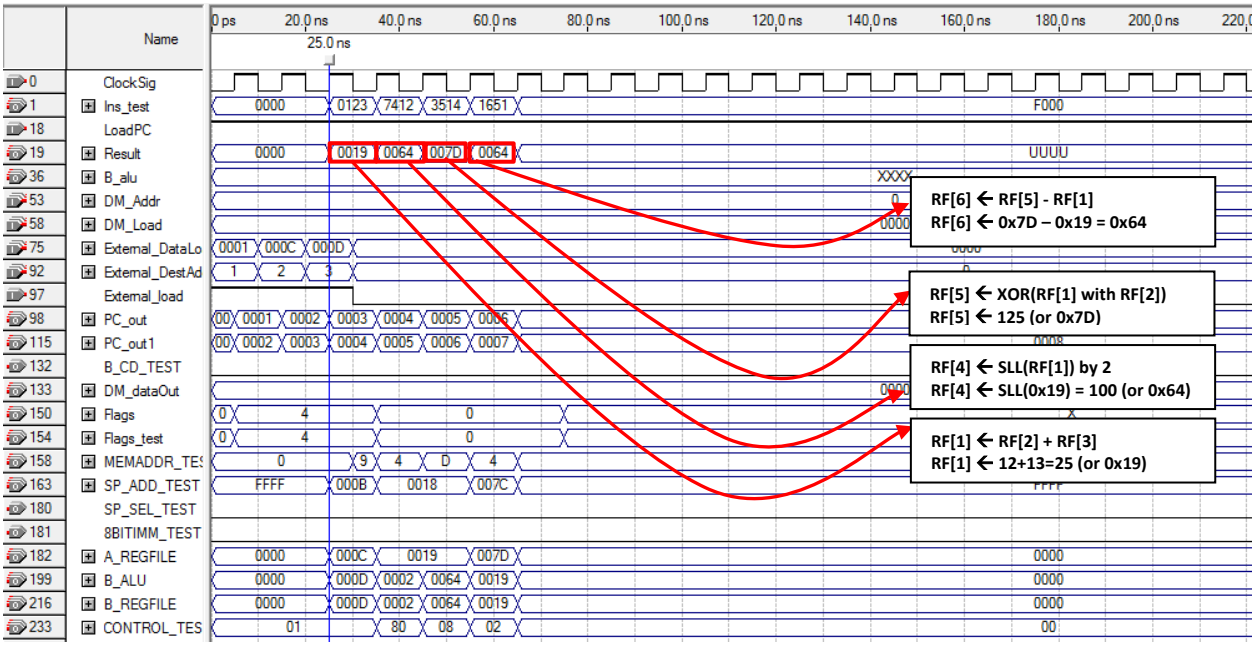
RF[1]=1, RF[2]=2 RF[3] = 3, rest of RF are cleared. DM[2]=11, rest DM are cleared.

IM Addr	Program	Result
0	LW $4, 2	RF[4] ← DM[2]; RF[4] ← 11
1	LOOP: SUB $3, $3, $2	RF[3] ← RF[3] – RF[2] = 3 -2 = 1
2	B GT, LOOP	Branch to Loop taken as 0x0001 is positive. LOOP: RF[3] ← RF[3] – RF[2] = 1-2 = FFFF Branch not taken as FFFF is negative.
3	CALL TGT	Call TGT. Program control transferred to IM[5]
4	HALT	Processor halted
5	TGT: INC $15, $15, -1	TGT: RF[15] = RF[15]-1 = 1-1 = 0
6	RET	Return back to IM[4]

Single Cycle Data Path

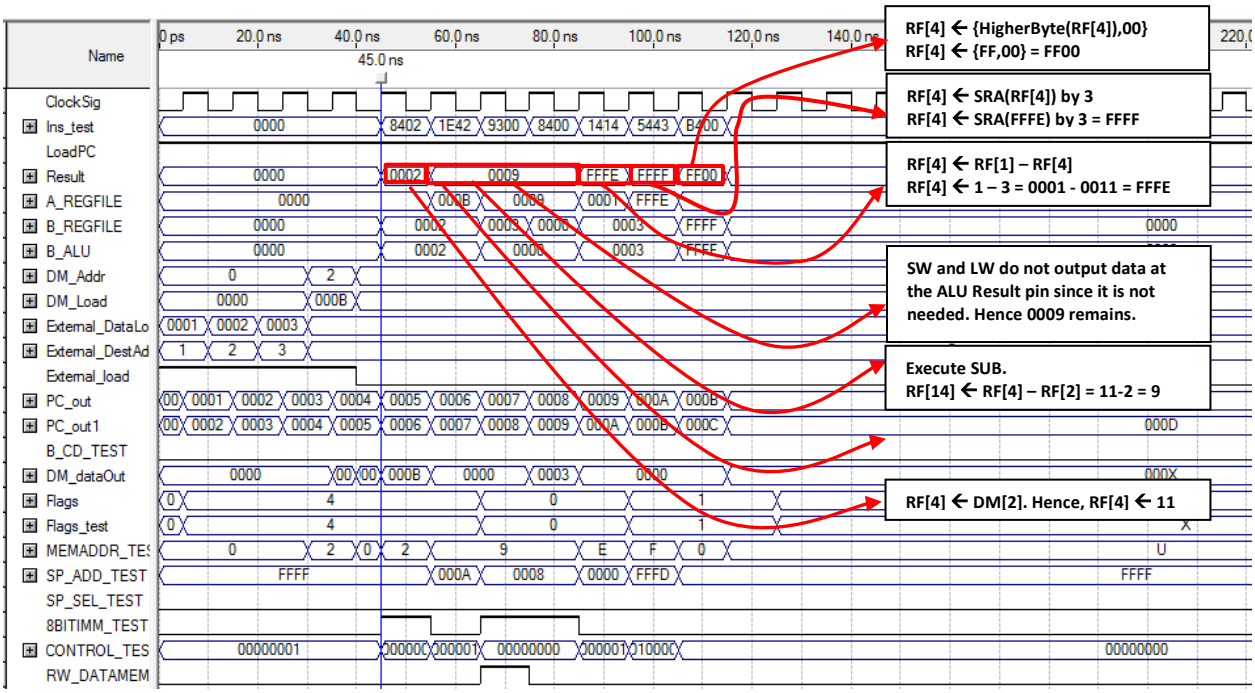


TestBench Program 1 – RAW Data Hazards – *Single Cycle*



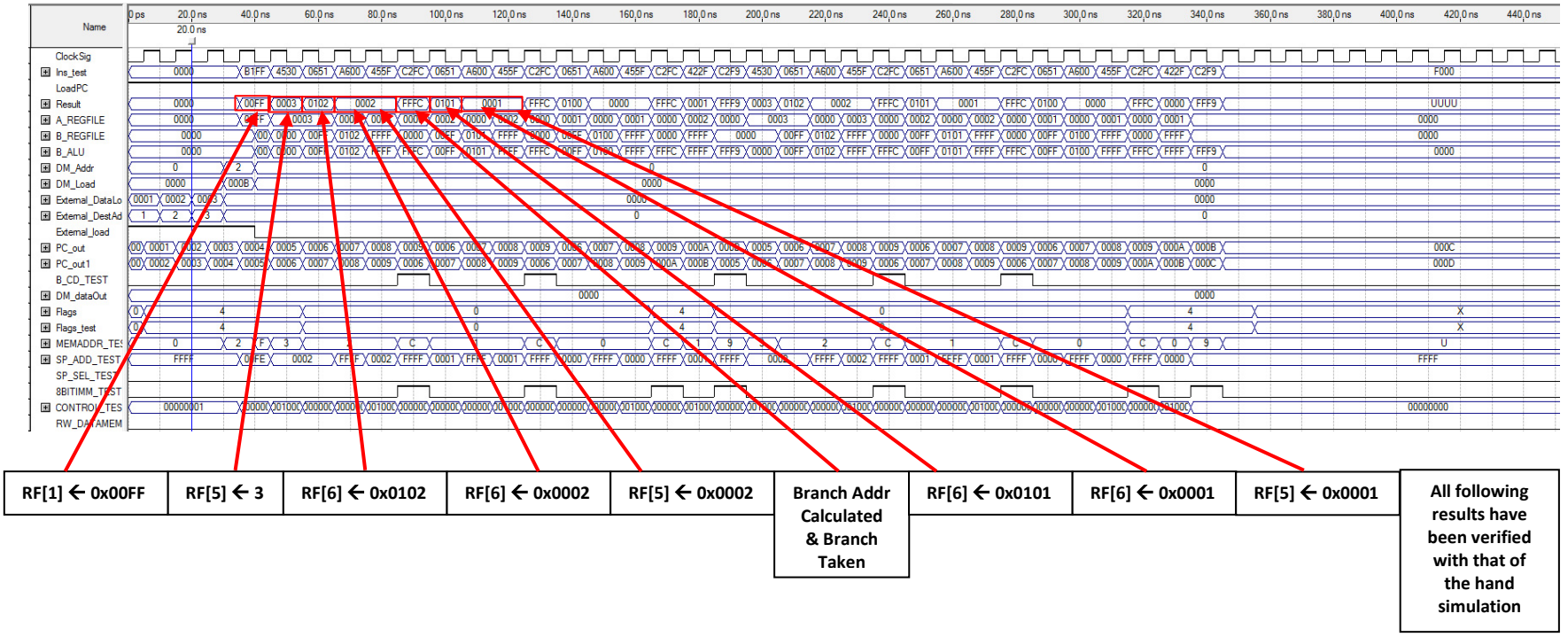
TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 12 and RF[3] = 13	2
STEP 2	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT – ***considering the instruction after HALT is not fetched***)	4
	TOTAL CYCLES →	**6**

TestBench Program 2 – Load Use Hazard – *Single Cycle*



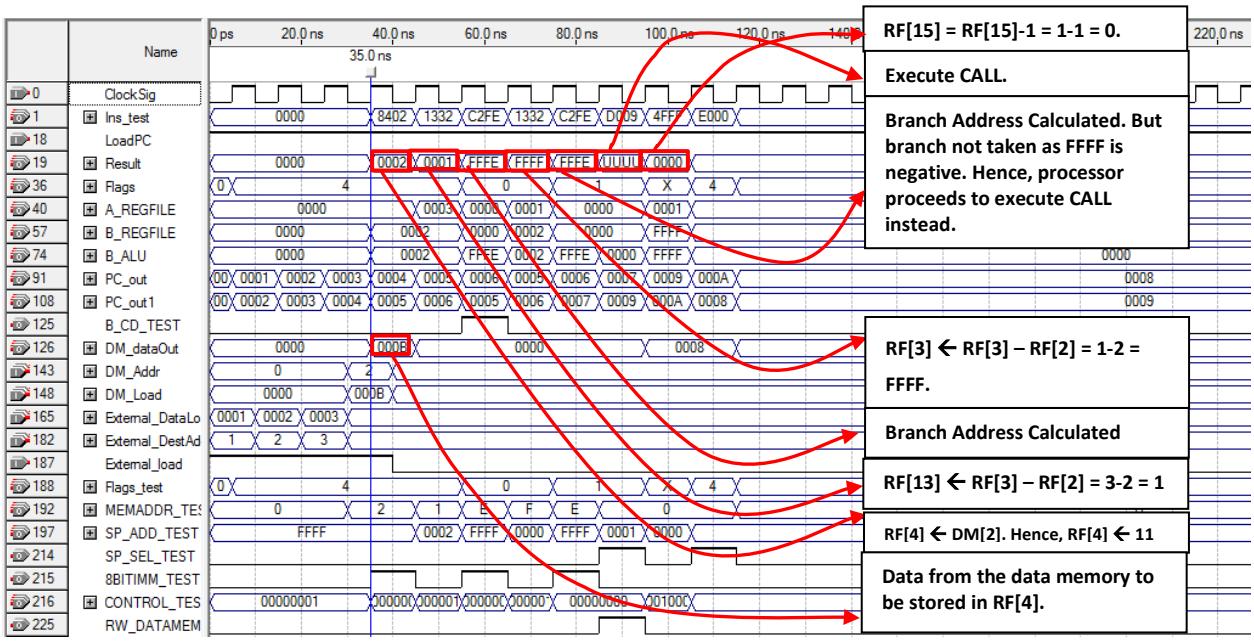
TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	4
STEP 2	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	8
	TOTAL CYCLES →	**12**

TestBench Program 3 – Branch/Control Hazard – *Single Cycle*



TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	3
STEP 2	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	32
	TOTAL CYCLES →	**35**

TestBench Program 4 – Call-Ret/Control Hazard – *Single Cycle*



TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	3
STEP 2	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	9
	TOTAL CYCLES →	**12**

Flow Summary – *Single Cycle*

Flow Status	Successful - Thu Dec 13 18:19:55 2012
Quartus II 64-Bit Version	9.0 Build 235 06/17/2009 SP 2 SJ Full Version
Revision Name	552HW2
Top-level Entity Name	Entire
Family	Stratix
Device	EP1S80F1508C5
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1,098 / 79,040 (1 %)
Total pins	210 / 1,212 (17 %)
Total virtual pins	0
Total memory bits	0 / 7,427,520 (0 %)
DSP block 9-bit elements	0 / 176 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

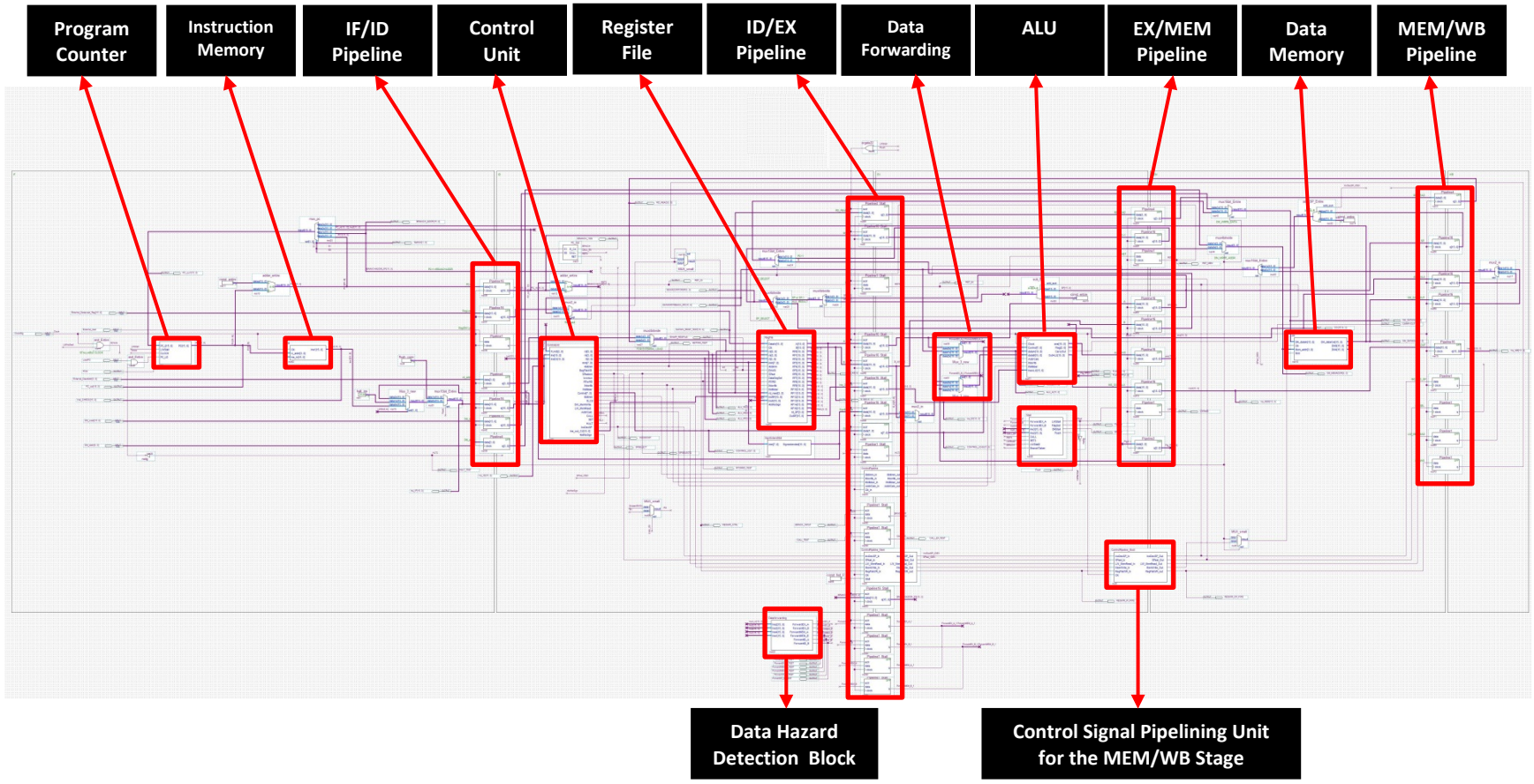
Timing Analyzer Summary - *Single Cycle*

Timing Analyzer Summary

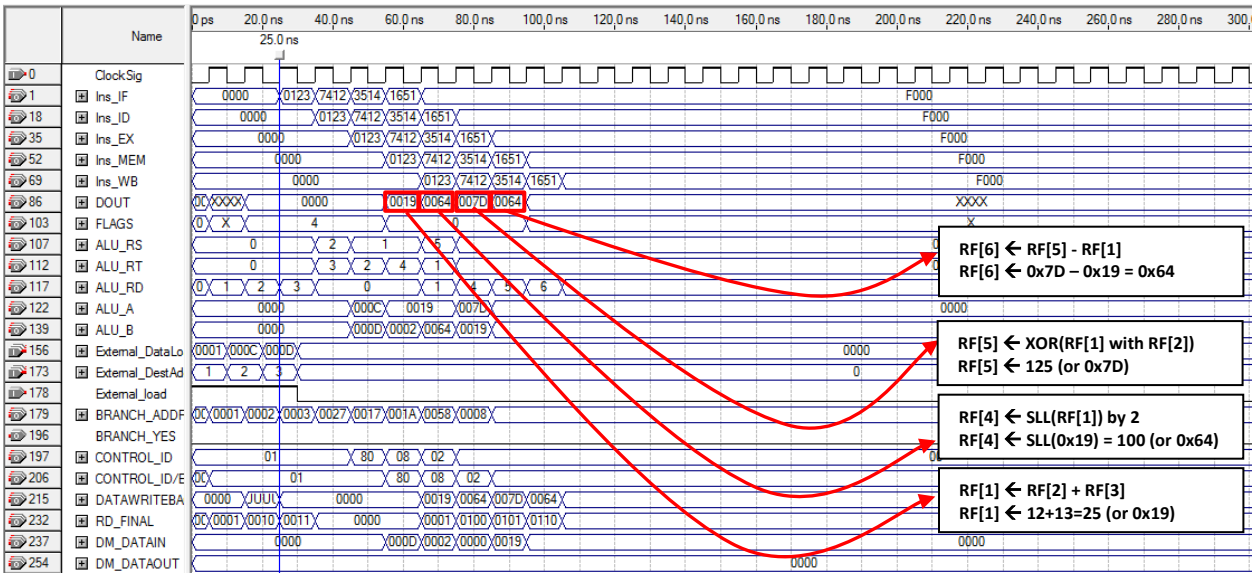
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	12.079 ns	External_load	DM:inst4|dff_reg:inst8|lpm_ff:lpm_ff_component|dffs[11]	--	ClockSig	0
2	Worst-case tco	N/A	None	16.348 ns	PC:inst|pcounter:inst1|lpm_ff:lpm_ff_component|dffs[1]	DM_dataOut[9]	ClockSig	--	0
3	Worst-case tpd	N/A	None	19.198 ns	External_load	DM_dataOut[9]	--	--	0
4	Worst-case th	N/A	None	-2.355 ns	DM_Load[0]	DM:inst4|dff_reg:inst11|lpm_ff:lpm_ff_component|dffs[0]	--	ClockSig	0
5	Clock Setup: 'ClockSig'	N/A	None	108.89 MHz (period = 9.184 ns)	PC:inst|pcounter:inst1|lpm_ff:lpm_ff_component|dffs[3]	DM:inst4|dff_reg:inst8|lpm_ff:lpm_ff_component|dffs[11]	ClockSig	ClockSig	0
6	Total number of failed paths								0

- ✓ **Max Working Frequency : <u>108.89 MHz</u>**
- ✓ **Period (Critical Path Delay or Max Delay Path): <u>9.184 ns</u>**

Pipelined Data Path

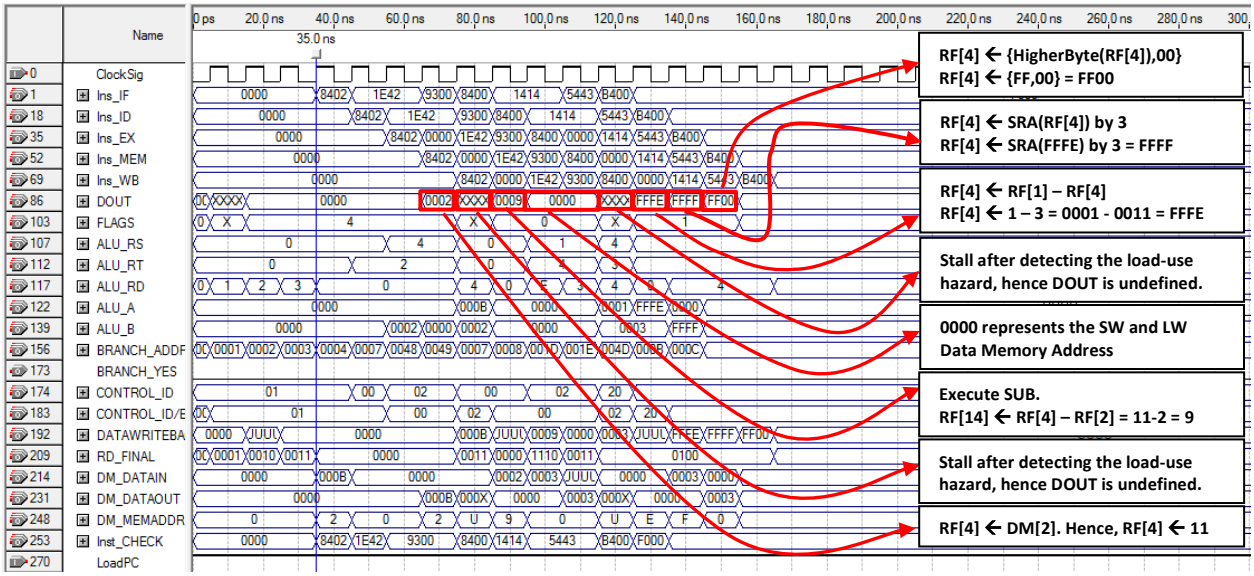


TestBench Program 1 – RAW Data Hazards - *Pipelined*



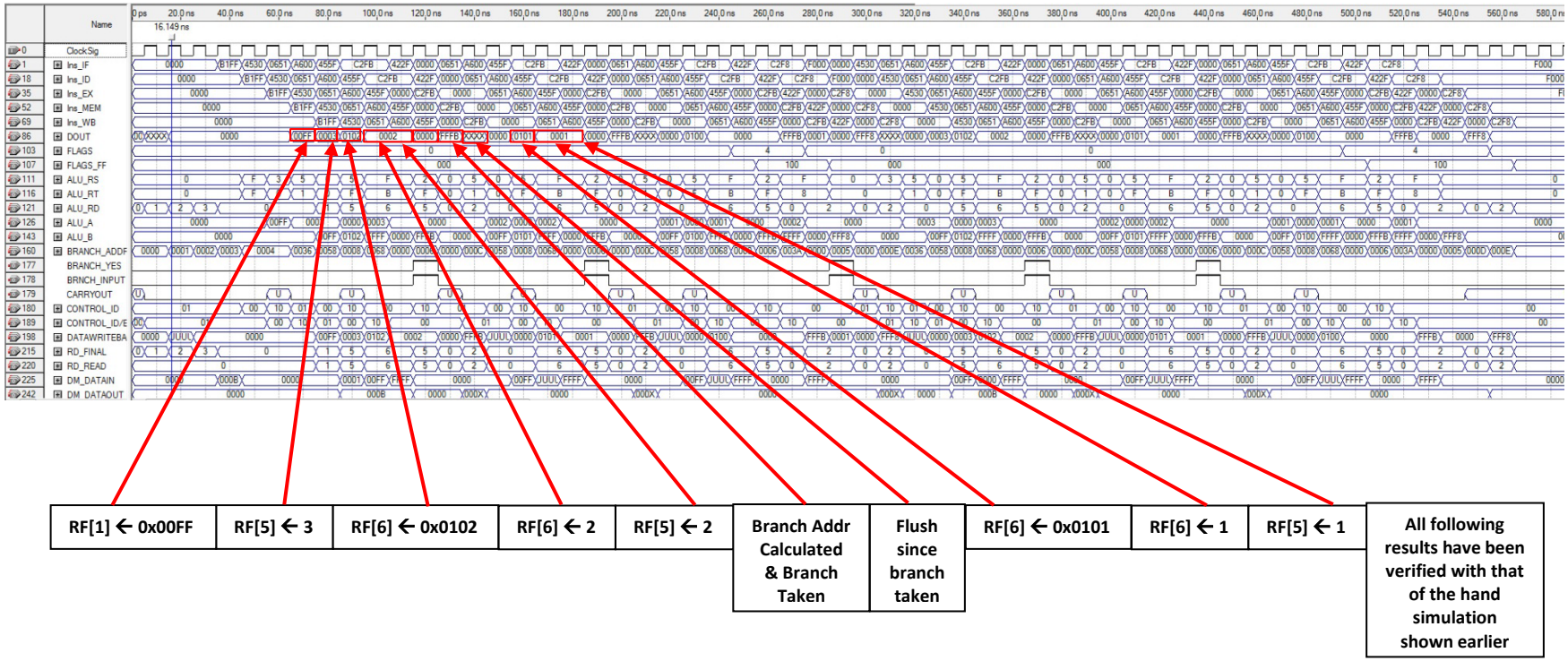
TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 12 and RF[3] = 13	3
STEP 2	Number of cycles before processor executes first instruction after Step 1	3
STEP 3	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT – ***considering the instruction after HALT is not fetched***)	4
	TOTAL CYCLES →	**10**

TestBench Program 2 – Load Use Hazard - *Pipelined*



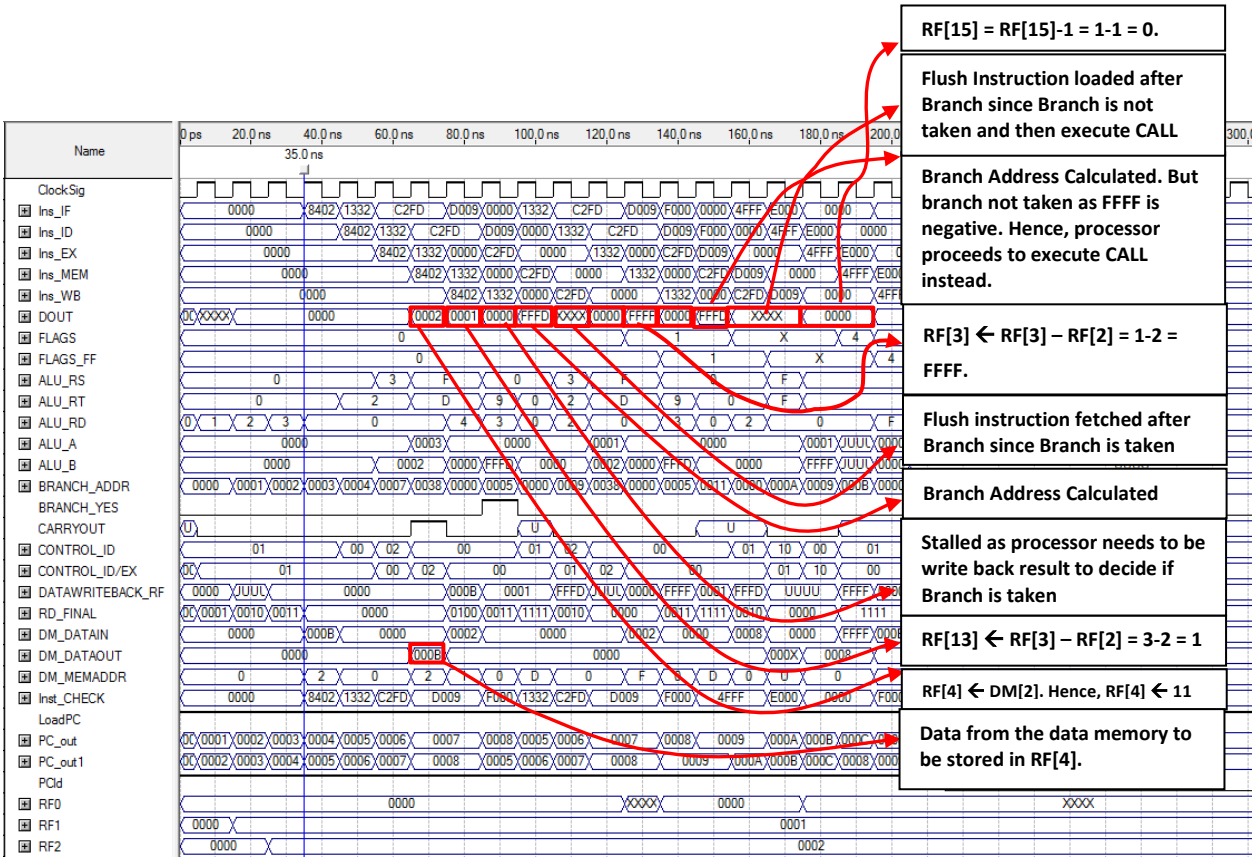
TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	4
STEP 2	Number of cycles before processor executes first instruction after Step 1	3
STEP 3	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	8
	TOTAL CYCLES →	**15**

TestBench Program 3 – Branch/Control Hazard - *Pipelined*



TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	3
STEP 2	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	51
	TOTAL CYCLES →	**54**

TestBench Program 4 – Call-Return/Control Hazard - *Pipelined*



TOTAL NUMBER OF CYCLES NEEDED FOR EXECUTION		
STEP 1	Initial cycles needed to load RF[1] = 1, RF[2] = 2, RF[3] = 3 and DM[2] = 11	4
STEP 2	Number of cycles before processor executes first instruction after Step 1	3
STEP 3	Number of cycles needed to execute all instructions in the program (this is equal to the number of instructions till and including HALT)	8
	TOTAL CYCLES →	**15**

Flow Summary - *Pipelined*

Flow Status	Successful - Thu Dec 13 18:28:32 2012
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Full Version
Revision Name	552HW2
Top-level Entity Name	Entire
Family	Stratix
Device	EP1S80F1508C5
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1,574 / 79,040 (2 %)
Total pins	668 / 1,212 (55 %)
Total virtual pins	0
Total memory bits	0 / 7,427,520 (0 %)
DSP block 9-bit elements	0 / 176 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

Timing Analyzer Summary - *Pipelined*

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock
1	Worst-case tsu	N/A	None	4.831 ns	LoadPC	PC:inst91|pcounter:inst|lpm_ff:lpm_ff_component|dffs[3]	--	ClockS
2	Worst-case tco	N/A	None	12.692 ns	Pipeline16:inst57|lpm_ff:lpm_ff_component|dffs[15]	Ins_IF[9]	ClockSig	--
3	Worst-case tpd	N/A	None	9.074 ns	LoadPC	PCId	--	--
4	Worst-case th	N/A	None	0.226 ns	DM_Load[0]	Pipeline16:inst42|lpm_ff:lpm_ff_component|dffs[0]	--	ClockS
5	Clock Setup: 'ClockSig'	N/A	None	137.04 MHz (period = 7.297 ns)	Pipeline1:inst70|lpm_ff:lpm_ff_component|dffs[0]	ALU:inst77|flagff:inst17|lpm_ff:lpm_ff_component|dffs[0]	ClockSig	ClockS
6	Total number of failed paths							

- ✓ **Max Working Frequency : 137.04 MHz**
- ✓ **Period (Critical Path Delay or Max Delay Path): 7.297 ns**