IITB-RISC PIPELINED

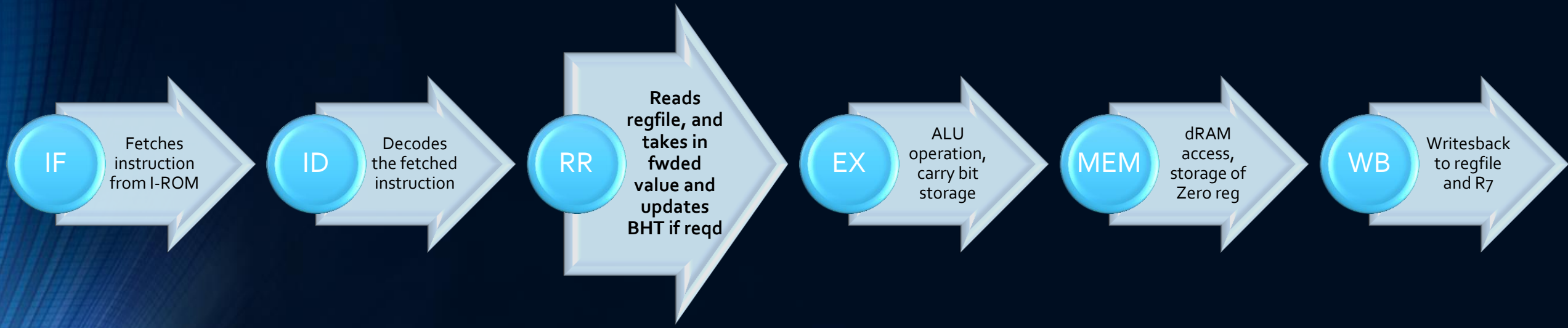
AMRITESH SHARMA 14D260012
AJINKYA GORAD 140110033
AGRIM GUPTA 140020003
HARSHVARDHAN TIBREWAL 140020023

PROJECT DESIGN

- Development of custom memory and regfile so that LM and SM get done with in one cycle, and finalization of datapath components
- Division of datapath into equal parts, for achieving better efficiency
- Recording all hazards possible, and developing a hazard detection unit to rectify them all
- Developing an instruction decoder and merging it with datapath
- Testing and Analysis of the project
- Future Work Possible

Salient Features of Datapath

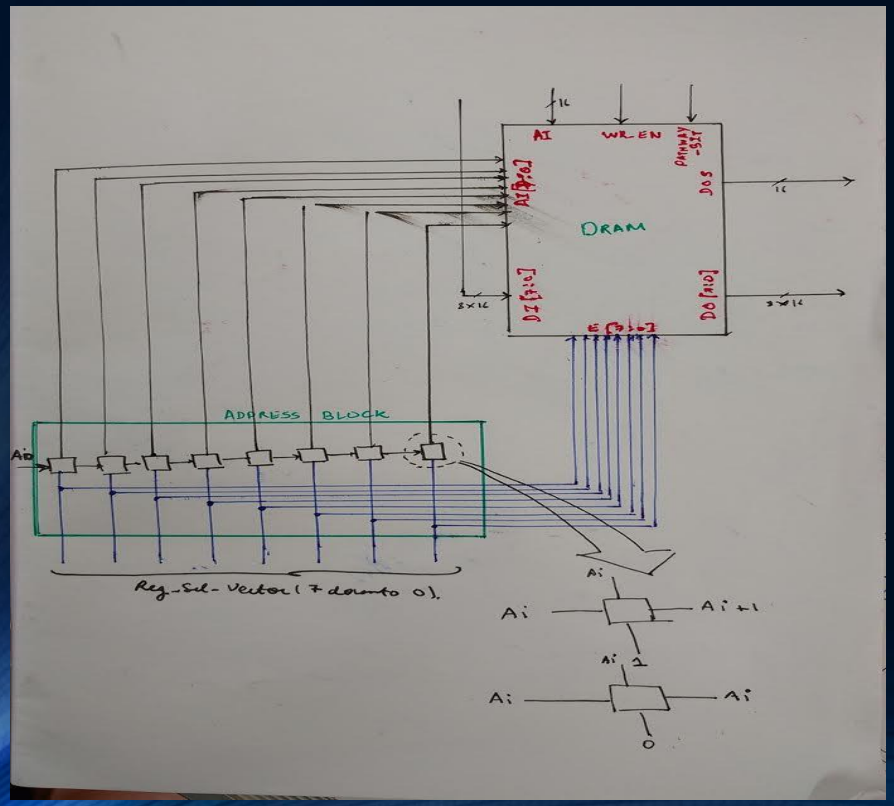
- Divided into six stages, namely,

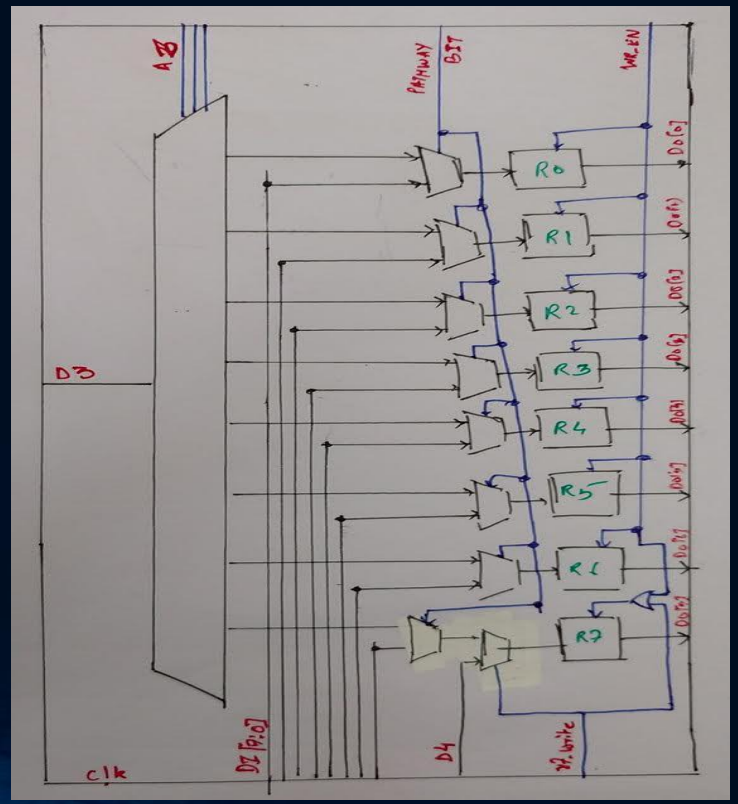


- Proxy PC : A hidden register used to store temporary value of next fetching isntruction's address, and always performs a writeback to R7, to update PC properly [IRF-IRE concept in CISC design]

Datapath: Memory and Regfile Design

- Inspired from spatial locality concepts used in cache memory, useful for doing LM and SM in one cycle, and thus reducing potential hazards that would've been created.





Instruction Decoder

- Instruction Decoder decodes the opcodes (instruction[15:12]) and CZ (instruction[1:0]) and provides the corresponding Control Signals for the Multiplexers, Register File, Data Memory and Arithmetic Logic Unit (ALU).
- It assumes independent execution from the hazard Control and Forwarding Logic. And in event of hazards Instruction controller signals are overwritten by Hazard Control Logic.

Pipeline Register Description

- IF-ID : Instruction word, PC , PC+1
- ID-RR: Instruction word, PC , PC+1, SignExt, Padder, Pc+1mm6, PC+imm9, RegFile select signals, Mux 3,4,6,7 select signals, ALU select signals
- RR-EX: Instruction word, PC , PC+1, SignExt, Padder, Register Data(R0-R7) , RegFile select signals, Mux 3,4,6,7 select signals, ALU select signals, RAM control
- EX-MEM: Instruction word, PC,PC+1,Carry, Zero, Rfcontrol,RAM control, RegFile data, multiple address input (LM)
- MEM-WB: Instruction word, PC,PC+1, ALU_out, Padder, Carry, Zero, Memory_load output, LM_output, RF select signals

Hazard Detection Logic

- Excel file created for systematic detection of hazards.
- We had obtained a total of 38 cases of data hazard detection, which were simplified to 12 unified rules, to be implemented by the hazard detection logic.
- For Control Hazards we got 1 rule for R type instr'n having R7 as destination, 2 for LHI with R7 as destination [1 w/o branch prediction, 1 with branch prediction], 1-1 each for LM/LW, 2+4 for BEQ[2 w/o branch prediction, 4 with branch prediction], 1+1 for JAL and 1+1+(2 obsolete) for JLR.

Forwarding Unit for Data Hazards

- Forwarding Unit:
  - Handles hazards where data needed hasn't been yet written and is still processing in further pipelines.
  - Located in RR stage of pipeline, compares dependency of instructions across RR-EX, RR-MEM, RR-WB with priority given to closest proximity, along with resolving latest carry and zero for RC/RZ type of instructions.
  - Carry write is in Execute stage and Zero flag write in MEM stage. Value of carry is preserved along the pipeline for decisions in stages MEM and writeback. Similar procedure for Zero flag, but since zero is written in MEM stage, value of zero flag needed in execute stage is forwarded from sources load_zero_flag, alu_zero_flag, zeroflag_reg.

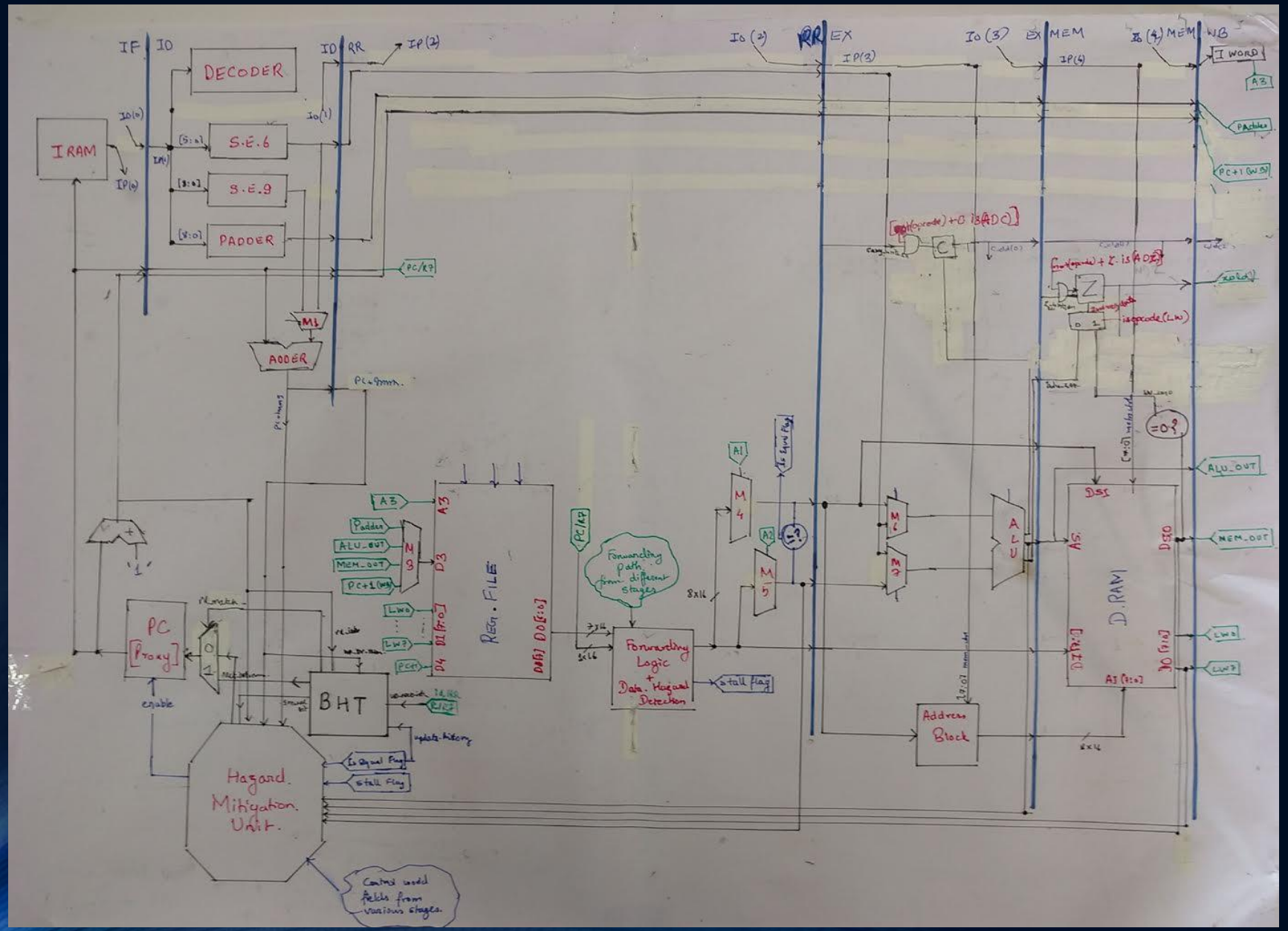
Hazard Detection Hardware

- Hazard Mitigation Unit:
  - This takes input of all the pipeline registers and computes stall/flushes and writes them back to the pipeline registers
  - Stall: Logic comes from forwarding unit and the pipeline registers (IF-ID and ID-RR) are disabled and stalled by one state (LW/LM type dependency)
  - Flush: Instructions causing jumps in PC contribute to control hazards. If destination is R7 , then depending on the instruction, desired flushes are implemented (place 1111 in opcode).
  - See Hazards.ods for all dependencies.

BRANCH HISTORY TABLE INTEGRATION

- Fully Associative Table with capacity of 20 records, one record consisting of Instruction number, Branch to Address , History bit and Target Register.
- The branch history table has inputs from IF stage (rd_instr_num) and from RR stage (update_next_instr, update_instr_num, update_history), on which it acts to produce 3 outputs, first one being rd_match, which is stated high when rd_instr_num is found in the table, second one being the predicted next state which goes to the instruction memory if rd_match .(not screwed_bit) is high, and lastly the screwed_bit, which goes to the Hazard Mitigation unit, and if stated high, indicates an incorrect prediction.
- The Inputs from IF stage are given to BHT for every instruction, whereas, Inputs from RR stage are given only for 3 instructions for which record is maintained (BEQ, JAL and LHI R7). The predicted instruction is PC+imm6 for BEQ, PC+imm9 for JAL and LHI.

THE FINAL DATAPATH



Analysis of Design-Flushes

- R Type instructions involving R7 :- 3 flushes : Fr'n of such r instructions: R7R, where R7R is the fraction of instructions of r type with destination as r7. Overhead: R7R*3

IF	ID	RR	EX	M	WB
RC/Z/U R7 R5 R1 (30)	-x	-x	-x	-x	-x
-x (31)	RC/Z/U R7 R5 R1 (30)	-x	-x	-x	-x
-x (32)	-x (31)	RC/Z/U R7 R5 R1 (30)			
-x (33)	-x (32)	-x (31)	RC/Z/U R7 R5 R1 (30)		
-x (R5+R1)	flushed	flushed	flushed	RC/Z/U R7 R5 R1 (30)	
-x (R5+R1+1)	-x (R5+R1)	flushed	flushed	flushed	RC/Z/U R7 R5 R1 (30)

- For LM7 and LW7, there are 4 flushes unconditionally. Overhead: 4*R7LM and 4*R7LW

LM Ri,#1xx (30)	-x	-x	-x	-x	-x
-x (31)	LM Ri,#1xx (30)				
-x (32)	-x (31)	LM Ri,#1xx (30)			
-x (33)	-x (32)	-x (31)	LM Ri,#1xx (30)		
-x (34)	-x (33)	-x (32)	-x (31)	LM Ri,#1xx (30)	
-x (new PC)	flushed	flushed	flushed	flushed	LM Ri,#1xx (30)
LW R7,R1,Imm6 (30)					
-x (31)	LW R7,R1,Imm6 (30)				
-x (32)	-x (31)	LW R7,R1,Imm6 (30)			
-x (33)	-x (32)	-x (31)	LW R7,R1,Imm6 (30)		
-x (34)	-x (33)	-x (32)	-x (31)	LW R7,R1,Imm6 (30)	
-x (new PC)	flushed	flushed	flushed	flushed	LM Ri,#1xx (30)

Analysis of Design-Flushes

- Now, for JAL and LHI, there is one flush if entry is not in branch target table, but with branch target table, there is no flush. Thus in the long run, the flushes are dead. Net increase in CPI -> 0
- For JLR Ra,Rb if there is no dependent on Rb instruction before execute stage, we can get the branched instruction in IF stage by forwarding from execute stage, which will be the next instruction. Thus, net increase in CPI -> $2*jlr_b + jlr_b'$, where jlr_b is fraction of JAL ra,rb instructions, with a depedent instruction nearby.

JLR.A	-x (old PC+1)	JLR R1,R2 (PC)	-x	-x	-x	-x	w/o any entries of instr'n in branch history table
	-x (old PC+2)	-x (old PC+1)	JLR R1,R2 (PC)	-x	-x	-x	add to be branched to is det in RR, 2 instr'n are flus
	-x (new PC)	**flushed**	**flushed**	JLR R1,R2 (PC)	-x	-x	new PC is stored in PC, new pc to be fwded
	-x (new PC+1)	-x (new PC)	**flushed**	**flushed**	JLR R1,R2 (PC)	-x	As there will be PC+2 in PC now,normal op
JLR.B	-x (old PC+1)	JLR R1,R2 (PC)	D(R2)	-x	-x	-x	forward R2 to the BHT, and also store it in PC
	-x (new PC fwded)	**flushed**	JLR R1,R2 (PC)	D(R2)		-x	no other forwardiing required here
JLR.C	-x (new PC fwded)	JLR R1,R2 (PC)	-x	D(R2)		-x	forward R2 to the BHT, and also store it in PC
	-x (new PC+1)	-x (new PC fwded)	JLR R1,R2 (PC)	-x	D(R2)		no other forwarding required here

Implementation of BHT for JLR doesn't make much sense, as register value may change dynamically. Hence it is ignored fpor implementation purposes

Analysis of Design-Flushes

- For BEQ, there are 6 possible cases, 2 w/o entry in BHT, 4 with entries in BHT (See right pic)
- In long run, we ignore the latency of first 2 instructions
- Thus, net overhead will be :
  - (1-p)*3*BEQ

5a	BEQ ra,rb,#10 (30)						**Branch taken, and no entry n BHT**
	-x (31)	BEQ ra,rb,#10 (30)					BEQ detected, new PC fwded, PC updated
	-x (40)	**flushed**	BEQ ra,rb,#10 (30)				
	-x (41)	-x (40)	**flushed**	BEQ ra,rb,#10 (30)			Ra-Rb executed, BHT updated
5b	BEQ ra,rb,#10 (30)						**Branch not taken, and no entry n BHT**
	-x (31)	BEQ ra,rb,#10 (30)					BEQ detected, new PC fwded, PC updated
	-x (40)	**flushed**	BEQ ra,rb,#10 (30)				
	-x (41)	-x (40)	**flushed**	BEQ ra,rb,#10 (30)			Ra-Rb executed, BHT updated, #31 stored and fwde
	-x (31)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)		
	-x (32)	-x (31)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)	
5c	BEQ ra,rb,#10 (30)						**BHT=> TAKE, BEQ=>TAKE**
	-x (40)	BEQ ra,rb,#10 (30)					
	-x (41)	-x (40)	BEQ ra,rb,#10 (30)				
	-x (42)	-x (41)	-x (40)	BEQ ra,rb,#10 (30)			UPDATE BHT
5d	BEQ ra,rb,#10 (30)						**BHT=> TAKE, BEQ=>NOT TAKE**
	-x (40)	BEQ ra,rb,#10 (30)					
	-x (41)	-x (40)	BEQ ra,rb,#10 (30)				
	-x (42)	-x (41)	-x (40)	BEQ ra,rb,#10 (30)			BHT updated, 31 fwded to PC and also stored
	-x (31)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)		
	-x (32)	-x (31)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)	
5e	BEQ ra,rb,#10 (30)						**BHT=>not TAKE, BEQ=>not TAKE**
	-x (30)	BEQ ra,rb,#10 (30)					
	-x (31)	-x (30)	BEQ ra,rb,#10 (30)				
	-x (32)	-x (31)	-x (30)	BEQ ra,rb,#10 (30)			UPDATE BHT
5f	BEQ ra,rb,#10 (30)						**BHT=>not TAKE, BEQ=> TAKE**
	-x (30)	BEQ ra,rb,#10 (30)					
	-x (31)	-x (30)	BEQ ra,rb,#10 (30)				
	-x (32)	-x (31)	-x (30)	BEQ ra,rb,#10 (30)			UPDATE BHT, 40 fwded and stored
	-x (40)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)		
	-x (41)	-x (40)	**flushed**	**flushed**	**flushed**	BEQ ra,rb,#10 (30)	and so on

Analysis of Design-Stalls

- We have stalling of one instruction if there is a dependent instruction following a LM/LW instruction. Net overhead thus : LWd+LMd

I(R1)	LM r1,imm#9	-x	-x	-x	-x	We get value in R1 after end Of M stage, so wait
-x	I(R1)	LM r1,imm#9				Hazard detected, and pipeline is stalled for
-x	I(R1)	LM r1,imm#9	LM r1,imm#9			One cycle stall is there now, to prevent hazard
-X+1	-x	I(R1)		LM r1,imm#9		Data forwarded from M stage to RR end, for normal op

I(R1)	LW R1,R2,Imm#6	-x	-x	-x	-x	LW data can be avaiable only after M stage
-x	I(R1)	LW R1,R2,Imm#6	-x	-x	-x	Stall one instruction unconditionally,if hazard detected
-x	I(R1)	LW R1,R2,Imm#6	LW R1,R2,Imm#6	-x	-x	Due to stalling, now offset will be in range of forwarding
-x	-x	I(R1)	-x	LW R1,R2,Imm	-x	Forward from Mem-op to RR end

- **NET CPI = 1+(LWd+ 4*R7LW + 2*JLR + (1-p)*3*BEQ) + (LMd+ + 4*R7LM) + 3*R7R**

Analysis: Sample CPI

NET CPI = 1+(LWd+ 4*R7LW + 2*JLR + (1-p)*3*BEQ)*j + (LMd+ + 4*R7LM)*i + 3*R7R*r

For sample instruction set as shown,

CPI : 1+0.01+0.04+0.2+(1-p)*0.3+0.02+0.04+0.45

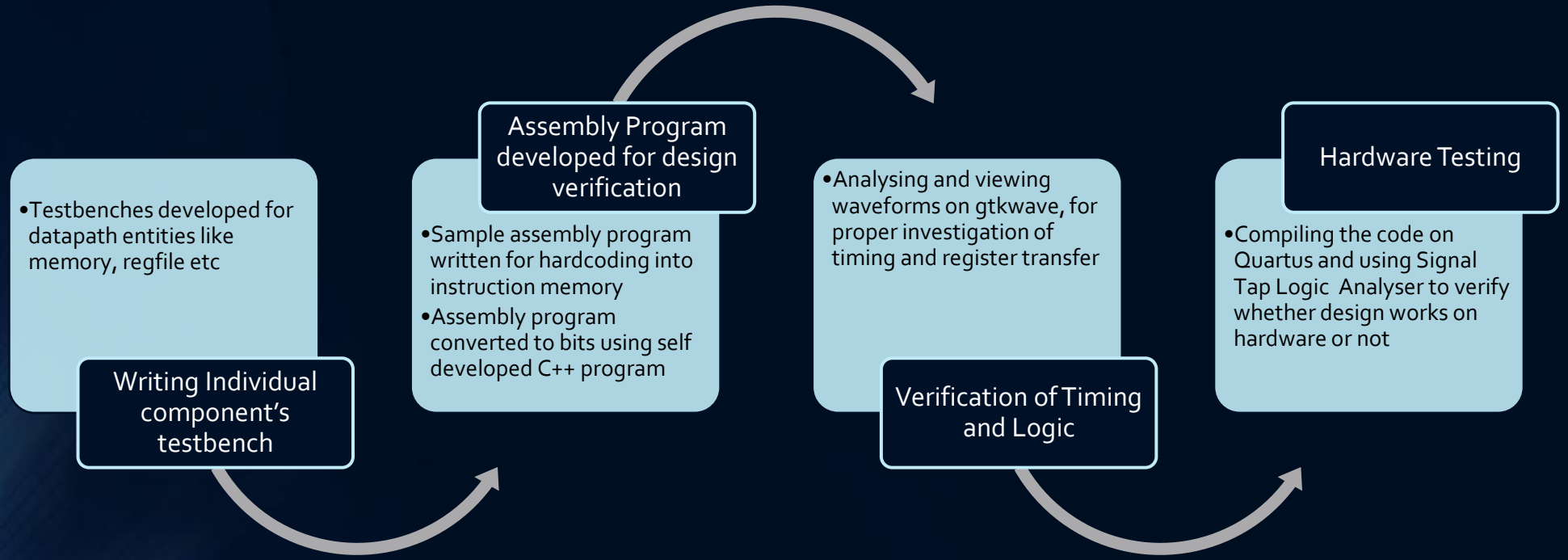
CPI : 1.76+0.3*(1-p)

As Cycle time can be approx. 1/6 of single cycle, and

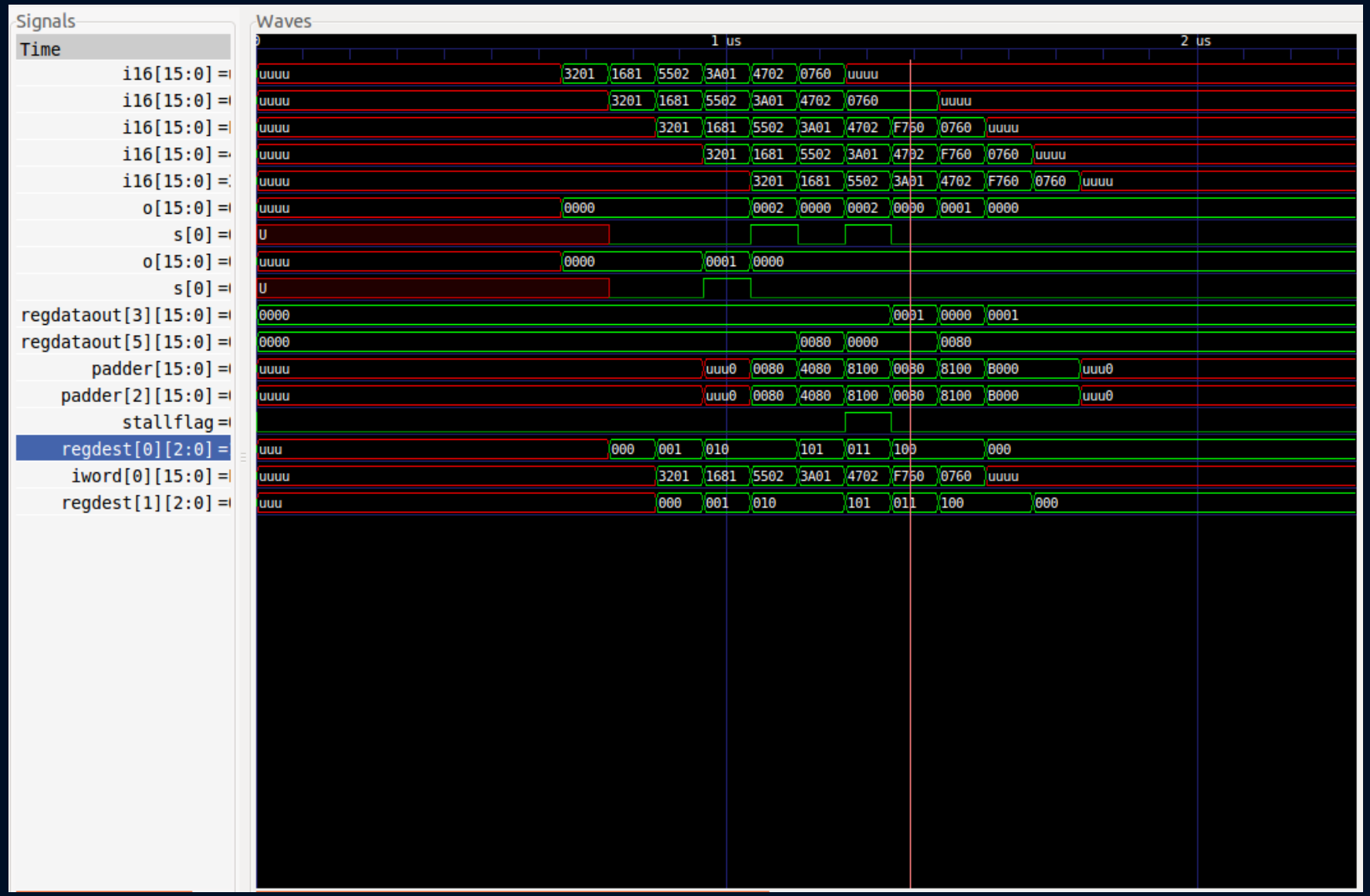
Performance= CPI*Cycle Time*Code Size
Performance ratio of pipeline to single cycle will be approx. : **3**

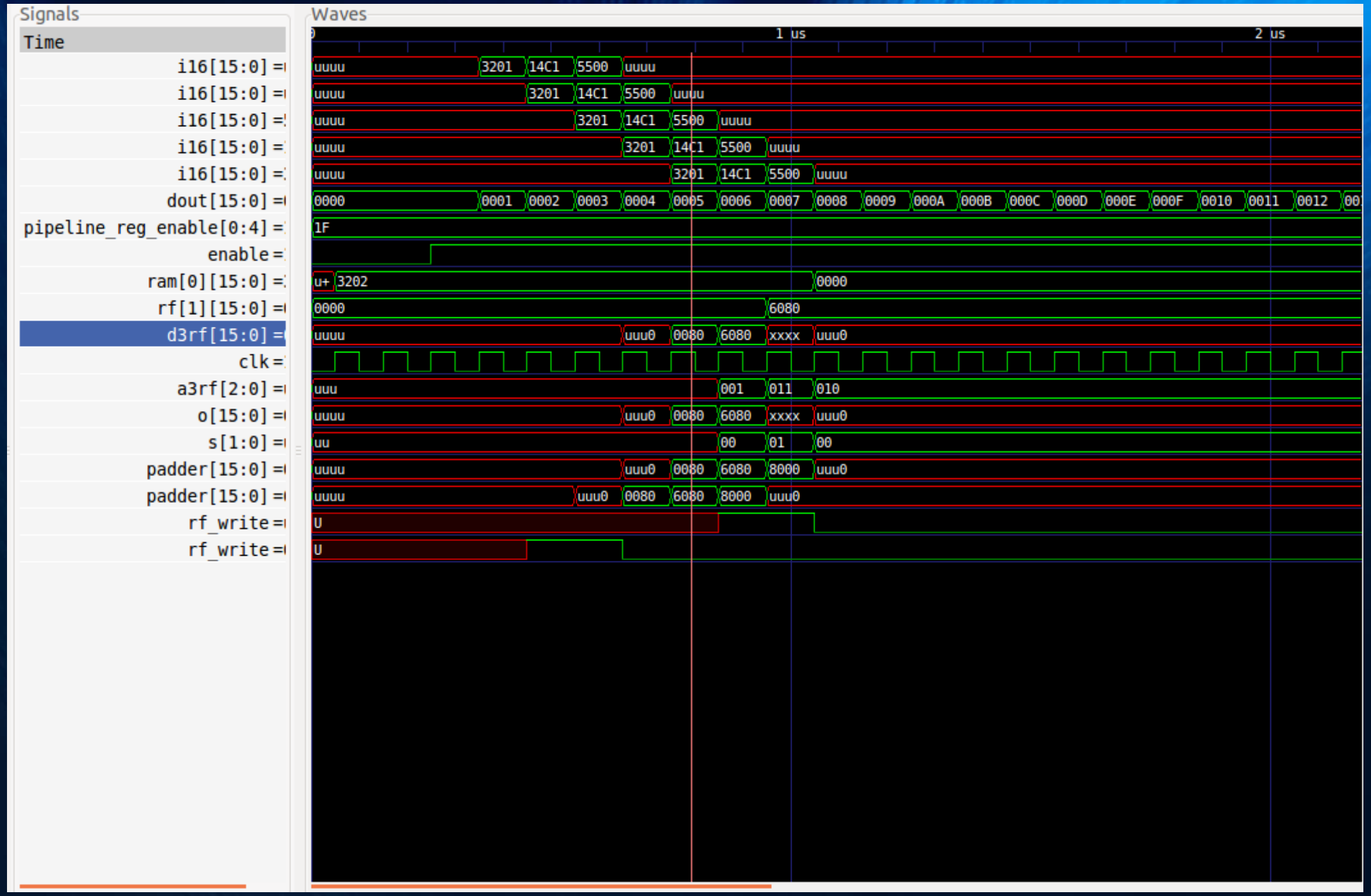
Instruction	%	Extra
Rtype+ADI	50	R7R approx 15 %
LW	5	LWd approx 1% R7LW approx 1%
SW	5	-
LHI	10	-
LM	2	LMd approx. 1% R7LM approx 1%
SM	3	-
JAL	5	-
JLR	10	-
BEQ	10	-

Debugging Procedures Used

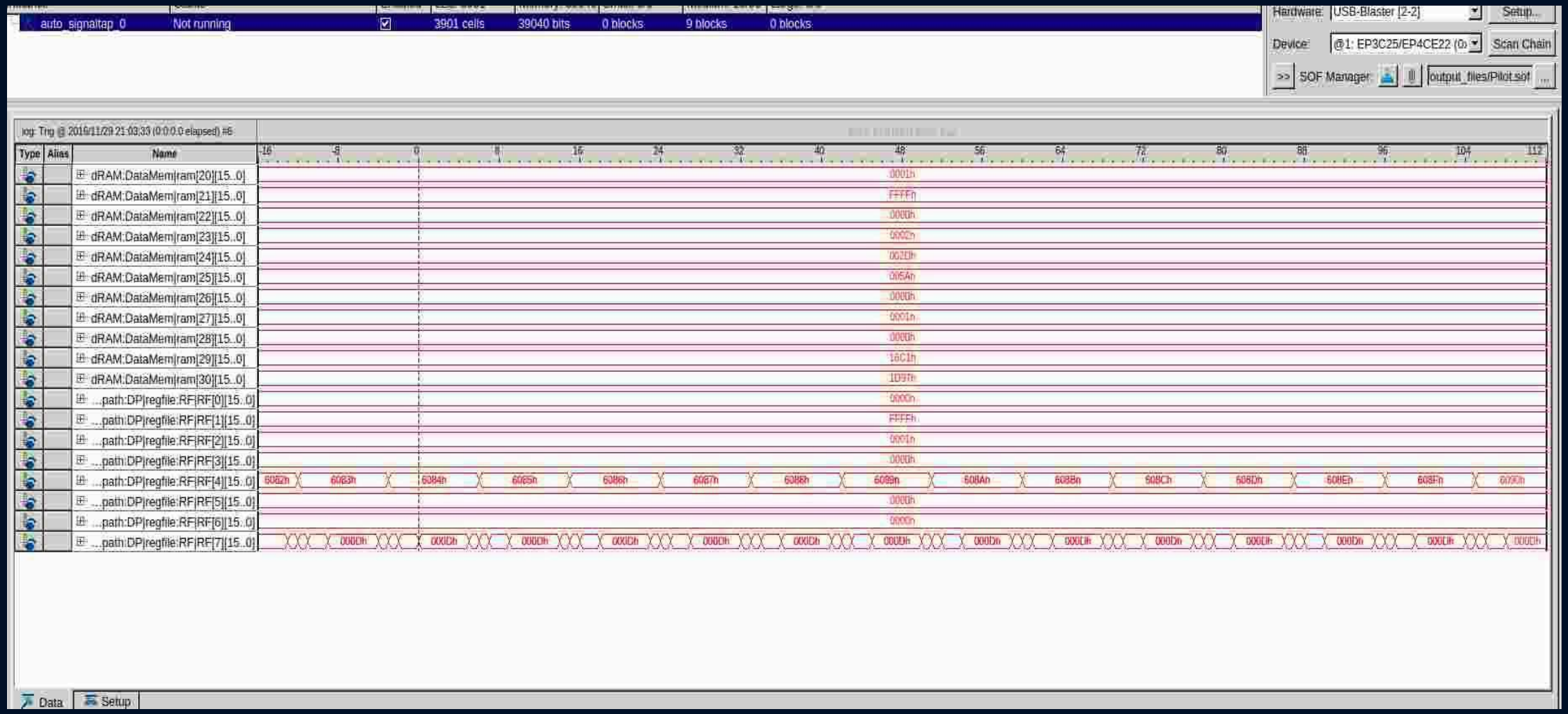


SIMULATION RESULTS-GTKwave




Signals
Waves
Time
i16[15:0]
i16[15:0]
i16[15:0]
i16[15:0]
i16[15:0]
dout[15:0]
pipeline_reg_enable[0:4]
enable
ram[0][15:0]
rf[1][15:0]
d3rf[15:0]
clk
a3rf[2:0]
o[15:0]
s[1:0]
padder[15:0]
padder[15:0]
rf_write
rf_write
1 us
2 us

SIMULATION RESULTS-SignalTap



Future Work to be Done

- Implementing already developed BHT into datapath, for BEQ, JLR, JAL, LHI R7 instructions.
- Developing a more efficient BHT, in terms of both space and access time, by using different strategies.
- Making the code more neat and and resolving errors in port mapping
- Improving CPI by incorporating branch history table with memory using caches.

THANK YOU FOR YOUR ATTENTION !!