TRƯỜNG ĐẠI HỌC BÁCH KHOA
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
BỘ MÔN ĐIỆN TỬ

BÁO CÁO BÀI TẬP LỚN

THIẾT KẾ CPU RICSV 32

GVHD: **TRẦN HOÀNG LINH**

SVTH:

Trương Tấn Sang	1813818
Mã Kim So	1813833

TP HCM, 28 tháng 11 năm 2021

Contents

Chapter 1

Cơ sở lý thuyết về CPU RICSV 32

1.1 Giới thiệu tổng quát về CPU RICSV 32

CPU RICSV 32 có tổng cộng 32 lệnh hợp ngữ, mỗi lệnh có độ dài 32 bits và 7 bits [6:0] (opcode) để xác định loại lệnh.

Tập lệnh của RICSV 32 còn được gọi là tập lệnh kiểu load-store, nghĩa là data trong bộ nhớ muốn được thực thi thì trước hết phải được lấy ra bỏ vào băng thanh ghi rồi mới được tính toán. Sau khi tính toán, data sẽ được lưu lại vào memory.

Các thanh ghi trong băng thanh ghi (Register Bank) có độ dài 32 bits và có 32 thanh ghi (từ $x_0 - x_{31}$), cần có 5 bits để xác định địa chỉ của các thanh ghi trong băng thanh ghi. Trong đó, chức năng của 32 thanh ghi được cho như bảng bên dưới.

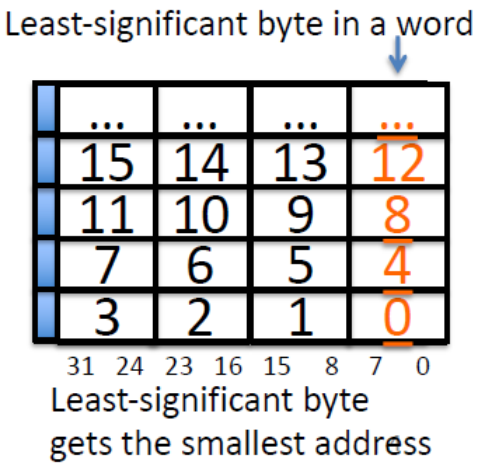
Register	**ABI name**	**Description**	**Saver**
x0	zero	Hard-wired zero	_ _ _
x1	ra	Return address	Caller
x2	sp	Stack pointer	Caller
x3	gp	Global pointer	_ _ _
x4	tp	Thread pointer	_ _ _
x5	t0	Temporary/alternate link register	Caller
x6-7	t1-2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Caller
x9	s1	Saved register	Caller
x10-11	a0-1	Function arguments/return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved registers	Caller
x28-31	t3-6	Temporaries	Caller

Lưu ý: Thanh ghi x_0 luôn có giá trị bằng 0x000000000 và không thay đổi giá trị này.

Dữ liệu trong cả bộ nhớ dữ liệu (DMEM) và bộ nhớ chương trình (IMEM) đều có độ dài 32bit và được sắp xếp theo kiểu **little edian**.

DMEM được định địa chỉ theo từng byte (= 8 bits) chứ không theo word (= 32 bits). Nếu định địa chỉ theo word thì lấy địa chỉ của byte có trọng số thấp nhất.

Điều này được trình bày như ở hình dưới.



1.2 Nhóm lệnh R-Format

Nhóm lệnh này bao gồm các lệnh có cấu trúc như ở hình sau:

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Nhóm lệnh này có opcode là [6:0] = 0110011

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi **rs1** và **rs2** thực hiện đưa vào khối ALU để tính toán, sau đó lưu kết quả vào thanh ghi **rd**.

1.3 Nhóm lệnh I (tính toán)

Nhóm lệnh này có opcode là [6:0] = 0010011

Nhóm lệnh này (trừ 3 lệnh SRAI, SRLI, SLLI) thực hiện lấy giá trị lưu ở thanh ghi **rs1** và giá trị lưu ở **imm[11:0]** (được mở rộng dấu), thực hiện đưa vào khối ALU để tính toán. Kết quả được lưu vào thanh ghi **rd**.

imm[11:0]		rs1	000	rd	0010011	addi
imm[11:0]		rs1	010	rd	0010011	slti
imm[11:0]		rs1	011	rd	0010011	sltiu
imm[11:0]		rs1	100	rd	0010011	xori
imm[11:0]		rs1	110	rd	0010011	ori
imm[11:0]		rs1	111	rd	0010011	andi
0000000	shamt	rs1	001	rd	0010011	slli
0000000	shamt	rs1	101	rd	0010011	srli
0100000	shamt	rs1	101	rd	0010011	srai

One of the higher-order immediate bits is used to distinguish "shift right logical" (SRLI) from "shift right arithmetic" (SRAI)

"Shift-by-immediate" instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions)

1.4 Nhóm lệnh L (Load data)

imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	010	rd	0000011	lh
imm[11:0]	rs1	011	rd	0000011	lw
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	110	rd	0000011	lhu

funct3 field encodes size and 'signedness' of load data

Nhóm lệnh này có opcode là [6:0] = 0000011

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi **rs1** và giá trị lưu ở **imm[11:0]** (được mở rộng dấu) để tính tổng **rs1 + ext(imm[11:0])**. Sau đó lấy giá trị lưu trong DMEM tại địa chỉ **rs1 + ext(imm[11:0])**, lưu vào thanh ghi **rd**.

1.5 Nhóm lệnh S (Store data)

Imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	sb
Imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	sh
Imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw

width

Nhóm lệnh này có opcode là [6:0] = 0100011

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi **rs1** và giá trị lưu ở **imm[11:5]** và **imm[4:0]** (ghép lại và mở rộng dấu) để tính tổng **rs1 + ext(imm[11:5])imm[4:0]**. Sau đó lấy giá trị lưu trong thanh ghi **rs2** lưu vào DMEM tại địa chỉ **rs1 + ext(imm[11:5]imm[4:0])**.

1.6 Nhóm lệnh B (Rẽ nhánh)

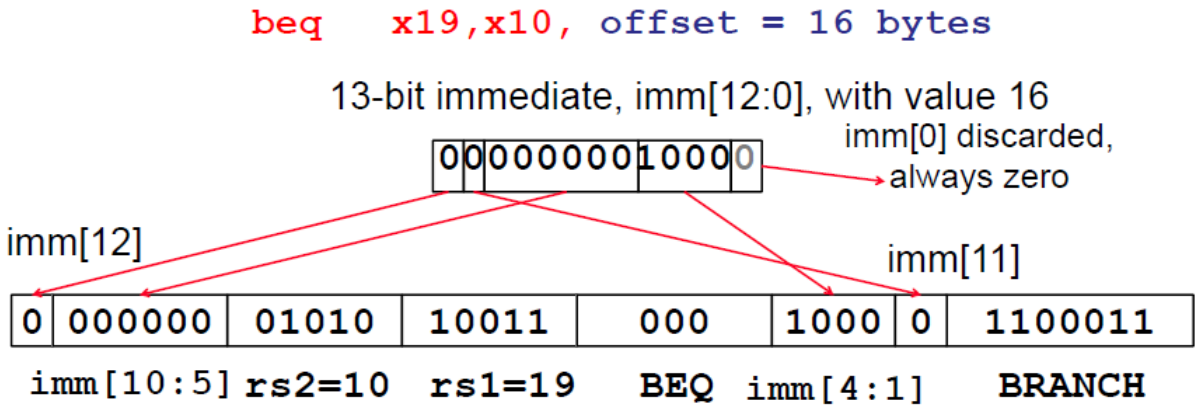
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGE

Nhóm lệnh này có opcode là [6:0] = 1100011

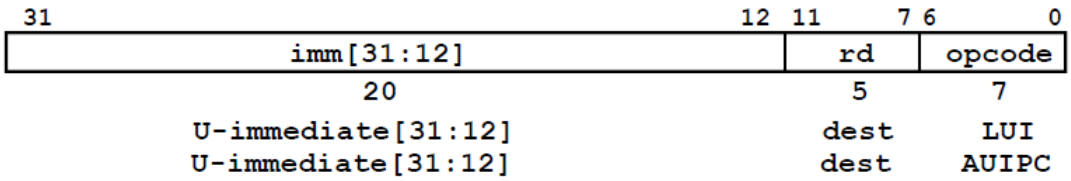
Nhóm lệnh này sẽ thực hiện chuyển giá trị của thanh ghi PC thành giá trị được lưu trong các phần **imm** giá trị lưu trong **rs1** và **rs2** thỏa điều kiện câu lệnh (bằng, không bằng, lớn hơn hoặc bằng,...).

Khi lấy giá trị lưu ở phần **imm** ta phải ghép lại cho đúng thứ tự và mở rộng dấu, bit LSB luôn luôn bằng 0.

Lấy ví dụ như ở hình dưới:



1.7 Nhóm lệnh U



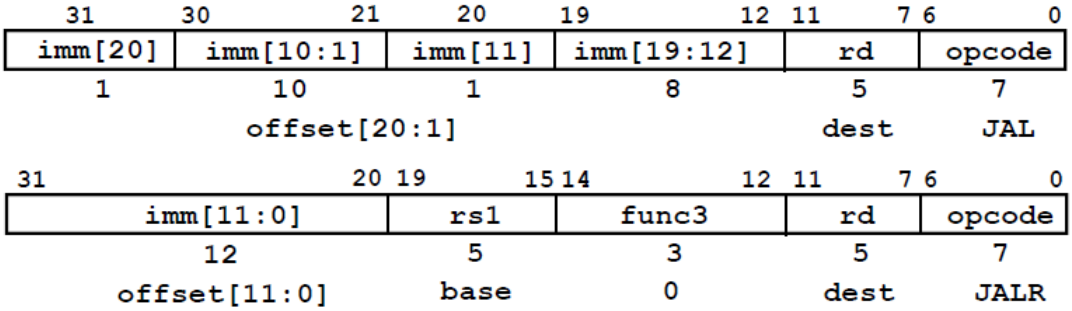
Với lệnh LUI

- Opcode = 0110111
- Lệnh này load giá trị **imm[31:12]00000000000** vào thanh ghi **rd**.

Với lệnh AUIPC

- Opcode = 0010111
- Lệnh này load giá trị ở **PC** vào thanh ghi rd.

1.8 Nhóm lệnh J (nhảy không điều kiện)

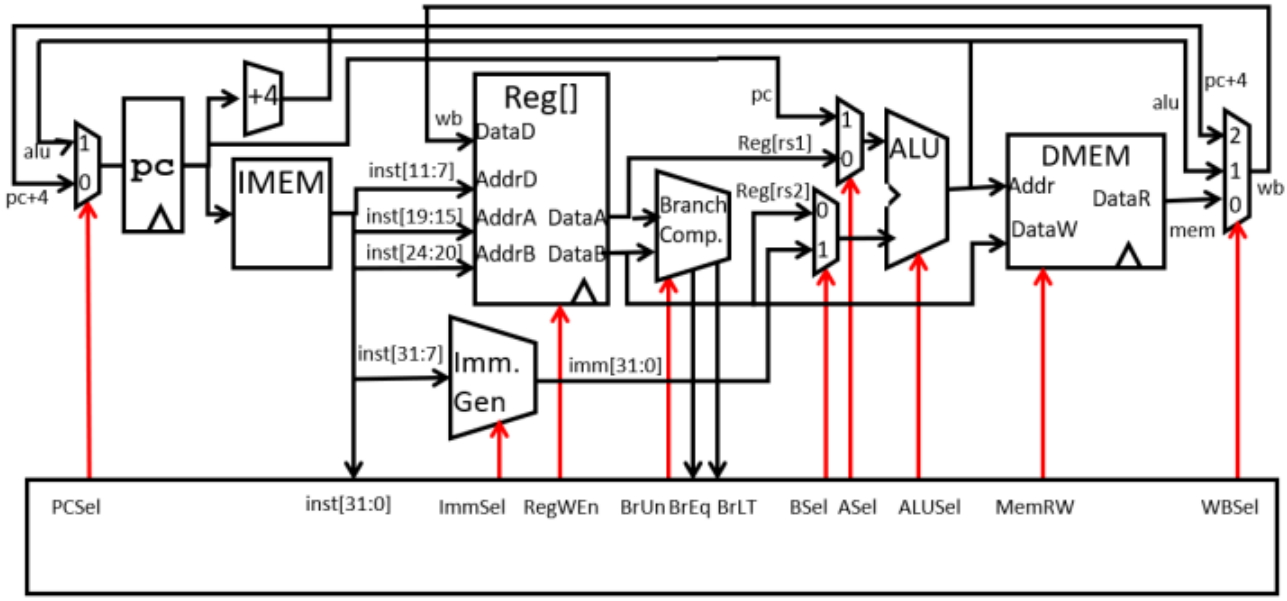


Chapter 2

Thiết kế CPU RICSV 32 đơn chu kỳ

2.1 Thiết kế phần cứng

Phần cứng được thiết kế theo sơ đồ sau:



Trong đó, các tín hiệu kết nối được đặt tên theo bảng sau:

Name	Khối bắt đầu	Khối đích	Ý nghĩa
clk	Input của CPU	PC, Register bank, DMEM	Xung clock điều khiển chu kỳ lệnh
pc_in	PCmux	PC	PC của lệnh tiếp theo
pc_out	PC	IMEM, PC+4, ALUmux1	PC hiện tại
pc_plus4_out	PC+4	PCmux	PC<-PC+4
rs1	IMEM	Register Bank	Địa chỉ của rs1
rs2	IMEM	Register Bank	Địa chỉ của rs2
rd	IMEM	Register Bank	Địa chỉ của rd
rs1_out	Register Bank	BranchComp, ALUmux1	Data của rs1
rs2_out	Register Bank	BranchComp, ALUmux2, DMEM	Data của rs2
imm_in	IMEM	ImmGen	Data vào ImmGen
imm_out	ImmGen	ALUmux2	Data sau khi qua khối ImmGen
alumux1_out	ALUmux1	ALU	Toán hạng 1 vào ALU
alumux2_out	ALUmux2	ALU	Toán hạng 2 vào ALU
aluout	ALU	DMEM, Wbmux, PCmux	Ngõ ra của ALU
dmem_out	DMEM	Wbmux	Data đọc của DMEM
wb_out	Wbmux	Register Bank	Data ghi ngược

Thiết kế khối Control

Tiến hành lập bảng bao gồm các lệnh, các tín hiệu vào và tín hiệu ra, sau đó phân tích từng lệnh vè điền vào bảng như hình sau:

Index	No	Type	MNEMONIC	Inst[30]	Inst[14:12]	Inst[6:2]	BrEq	BrLT	PcSel	ImmSel	RegWEN	BrUn	Bsel	Asel	ALUsel	MemRW	DataIn	DataOutAddj	WBSel
									19	18:16	15	14	13	12	11:8	7	6:5	4:2	1:0
							0 => Not EQ	0 => Not LT	0 => PC + 4	000 => ins[31:20], ext sign, 12	0 => Read	0 => Signed	0 => Data B	0 => Data	0000 => ADD	0 => Read Only	00 => SB	000 => exB	00 => DMEM
							1 => EQ	1 => LT	1 => ALU	001 => ins[31:20], ext usign, 12	1 => Write	1 => Usigned	1 => Imm Gen	1 => PC	0001 => SUB	1 => Read Write	01 => SHW	001 => exHW	01 => ALU
										010 => ins[24:20], ext usign, 5					0010 => SLL		11 => SW	010 => W	10 => PC +4
										011 => ins[31:25][11:7], ext sign, 12					0011 => SLT			011 => exBU	
										100 => ins[31][7][30:25][11:8]0, ext sign, 12					0100 => SLTU			100 => exHU	
										101 => ins[31:12]0..0,					0101 => XOR				
										110 => ins[31][19:12][20][30:21]0, ext sign, 20					0110 => SRL				
															0111 => SRA				
															1000 => OR				
															1001 => AND				
															1110 => PC + ImmGen + 4				
															1111 => Sel B				
0	0	R	ADD	0	000	01100	x	x	0	x	1	x	0	0	0000	0	x	x	01
1	1	R	SUB	1	000	01100	x	x	0	x	1	x	0	0	0001	0	x	x	01
2	2	R	SLL	0	001	01100	x	x	0	x	1	x	0	0	0010	0	x	x	01
3	3	R	SLT	0	010	01100	x	x	0	x	1	x	0	0	0011	0	x	x	01
4	4	R	SLTU	0	011	01100	x	x	0	x	1	x	0	0	0100	0	x	x	01
5	5	R	XOR	0	100	01100	x	x	0	x	1	x	0	0	0101	0	x	x	01
6	6	R	SRL	0	101	01100	x	x	0	x	1	x	0	0	0110	0	x	x	01
7	7	R	SRA	1	101	01100	x	x	0	x	1	x	0	0	0111	0	x	x	01
8	8	R	OR	0	110	01100	x	x	0	x	1	x	0	0	1000	0	x	x	01
9	9	R	AND	0	111	01100	x	x	0	x	1	x	0	0	1001	0	x	x	01

Sau đó, chuyển bảng vừa lập thành khối Control. Viết theo kiểu ROM

Thiết kế các khối chức năng

Phân tích từng khối: chức năng, ngõ vào, ngõ ra và viết riêng từng module.

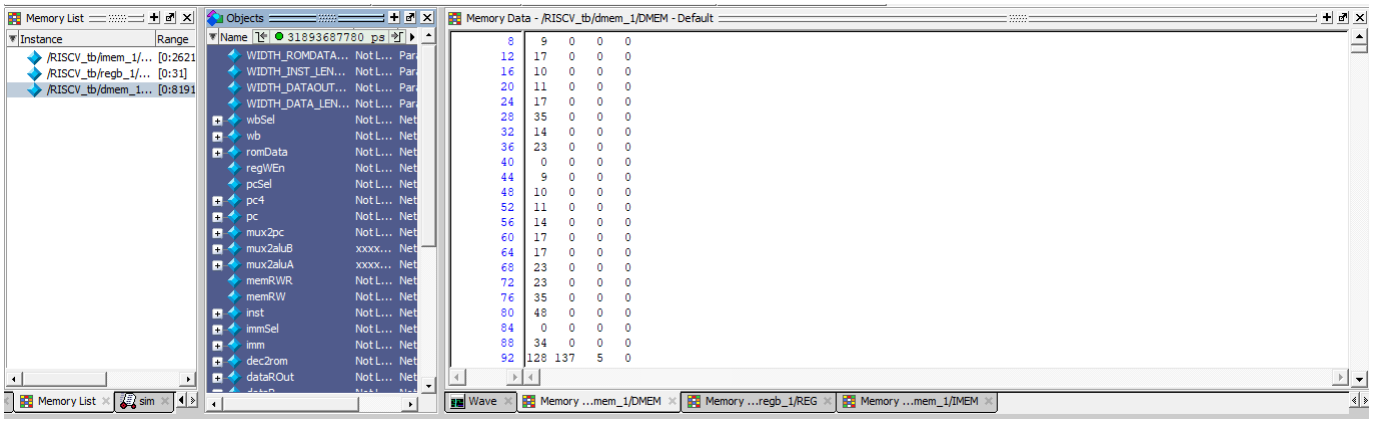
2.2 Thực hiện mô phỏng

2.2.1 Viết đoạn chương trình test

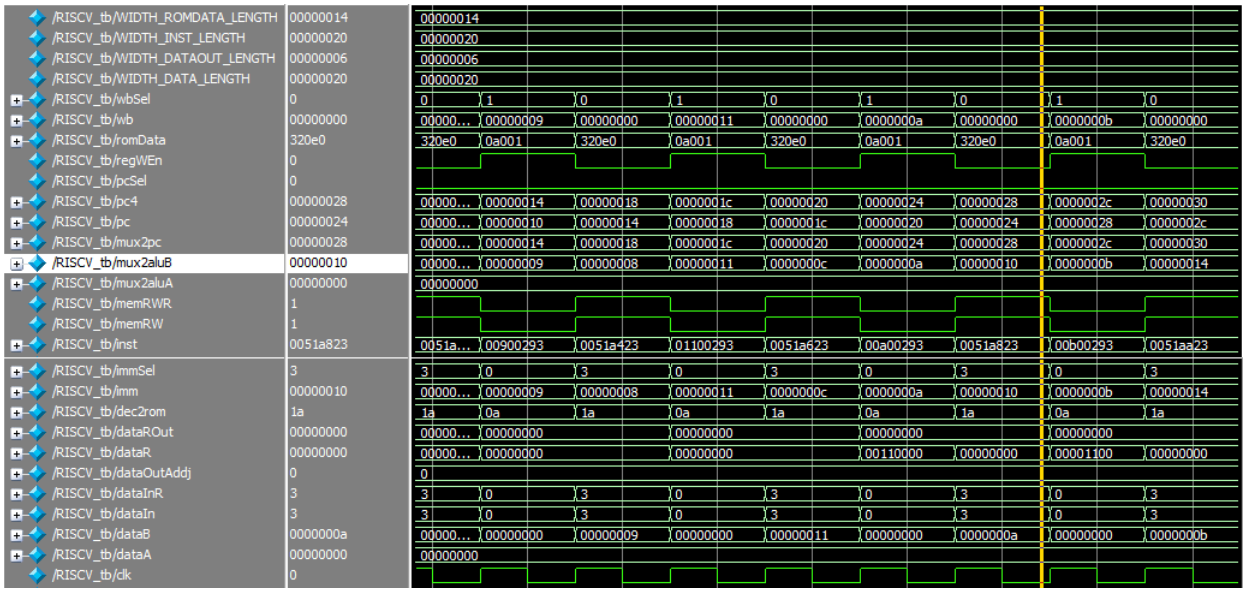
- Lấy 10 số lưu trong DMEM và sắp xếp lại rồi lưu vào DMEM ở 10 vị trí tiếp theo.
- Tính giai thừa số lớn nhất và lưu ở vị trí tiếp theo.
- Tính số Fibonanci của số lớn nhất và lưu ở vị trí tiếp theo.

2.2.2 Test dạng sóng trên ModelSim và kết quả mô phỏng

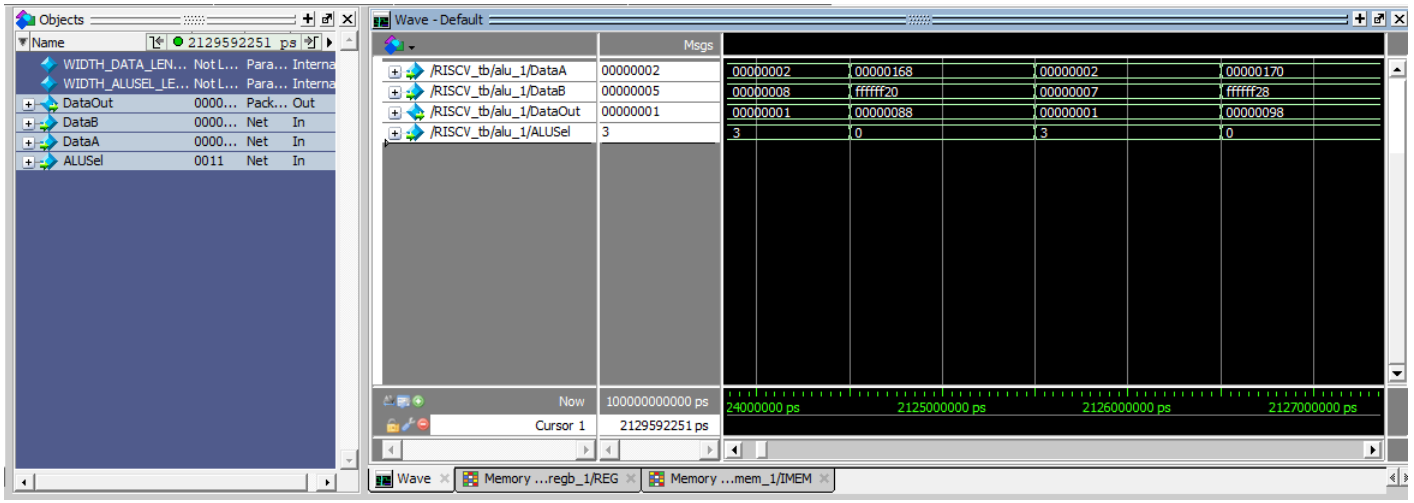
Kết quả sau khi chạy đoạn code trên:



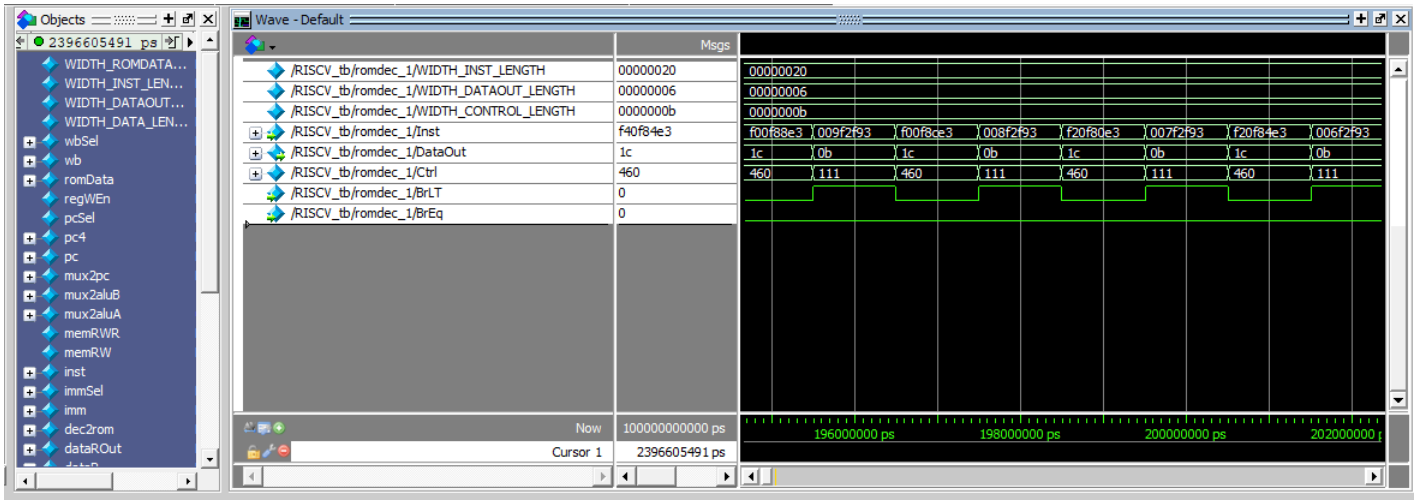
Dạng sóng của đoạn code trên:



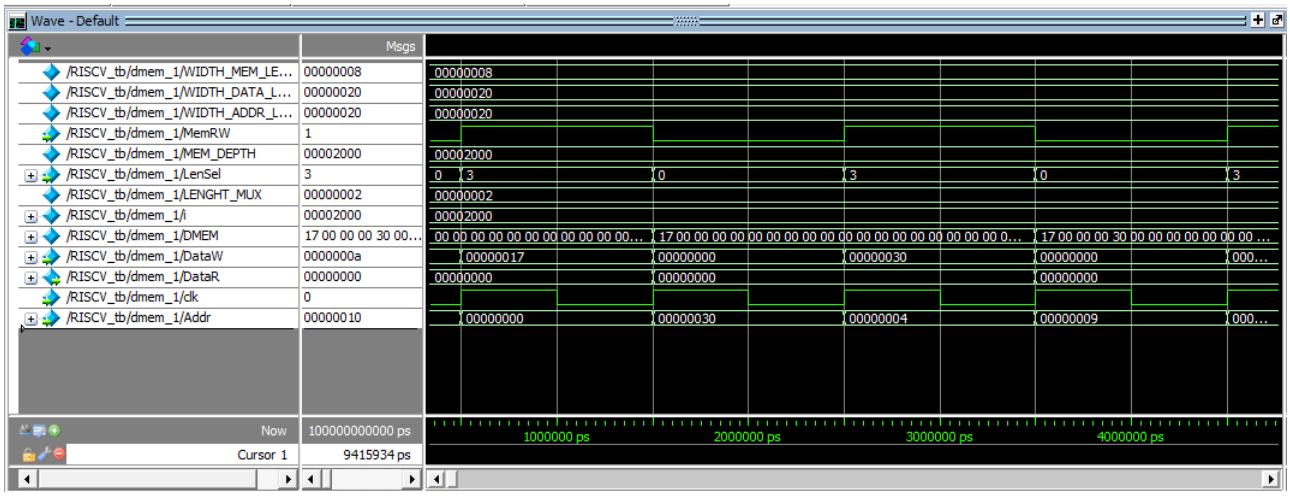
Kết quả mô phỏng của khối ALU:



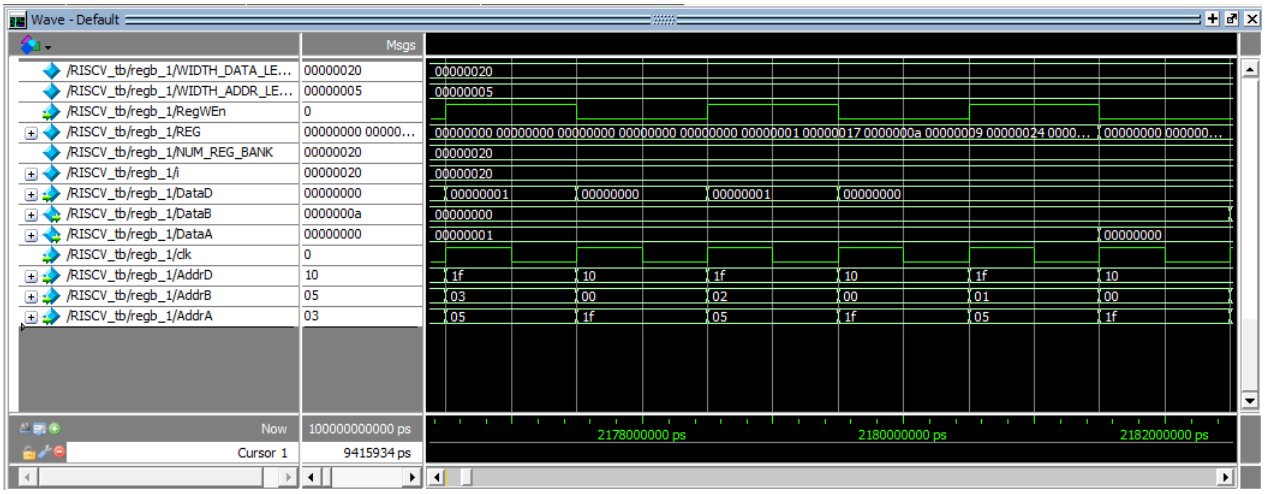
Kết quả mô phỏng khối Branch Comp:



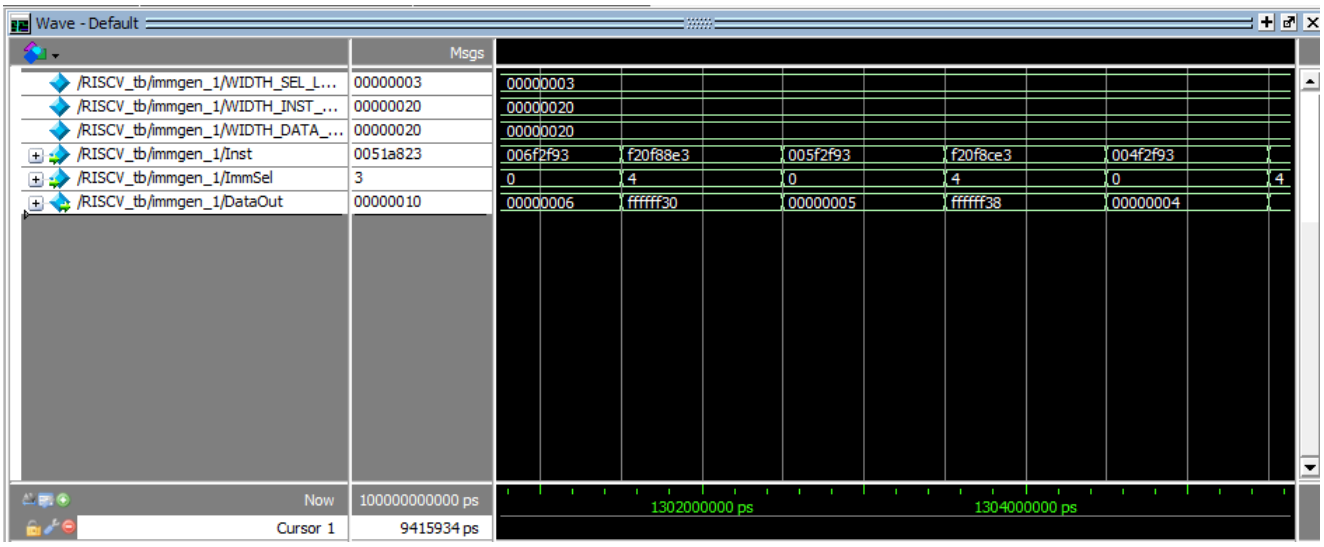
Kết quả mô phỏng khối DMEM:



Kết quả mô phỏng khối Reg:



Kết quả mô phỏng khối Imm:



Chapter 3

Thiết kế CPU RICSV 32 Pipeline 5 tầng

3.1 Tổng quát về Pipeline

Pipeline là một kỹ thuật mà trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap).

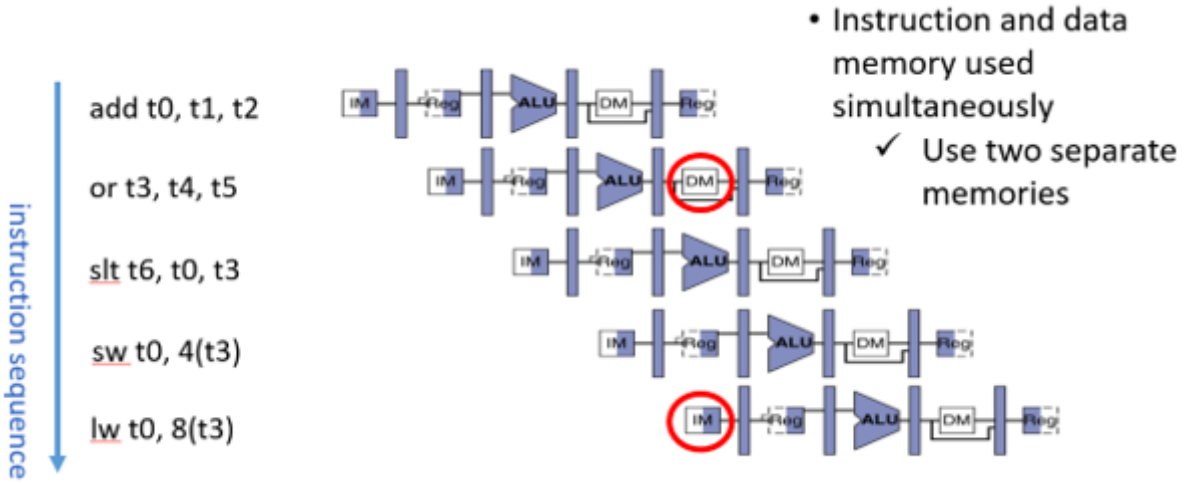
Dựa trên thiết kế của đơn chu kỳ ta thấy rằng, để thực hiện một câu lệnh cần ít nhất 5 trạng thái. Các trạng thái được nêu chi tiết trong bảng sau:

IF	ID	EX	MEM	WB
Instruction Fetch	Instruction decode/register file read	Execute/Address calculation	Memory access	Write back
Nạp lệnh	Decode	Tính toán	Truy cập MEM	Ghi vào RD

3.2 Pipelining Hazards

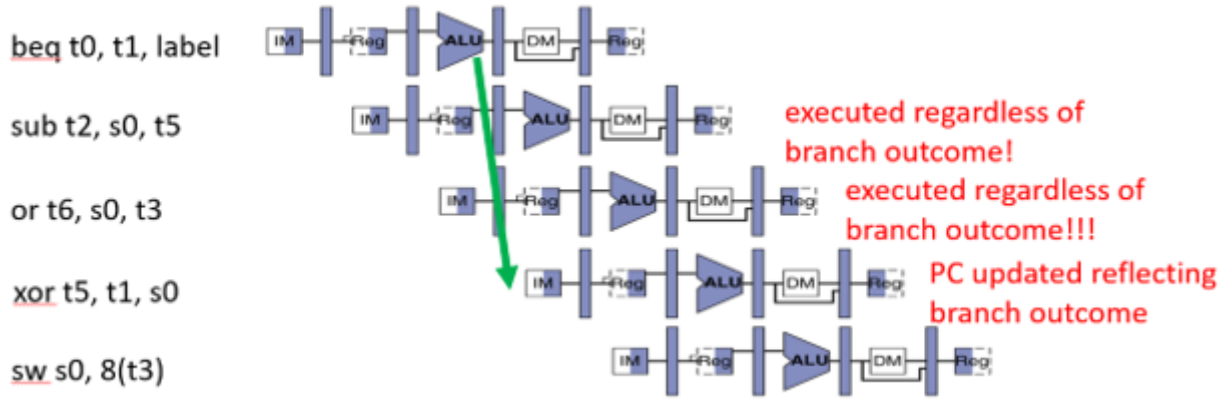
3.2.1 Structural Hazard

Hai câu lệnh cùng truy cập vào bank thanh ghi cùng lúc nhưng nó chỉ có thể đọc hoặc ghi dựa vào Clock.



3.2.2 Control Hazard

Hazard control sẽ xảy ra khi dùng các lệnh nhảy như beq, bne,... Sau khi lệnh nhảy đến tầng ALU nhưng địa chỉ kế tiếp chứa câu lệnh thực hiện kế tiếp không biết là nhảy hay không nhảy và nhảy đến đâu

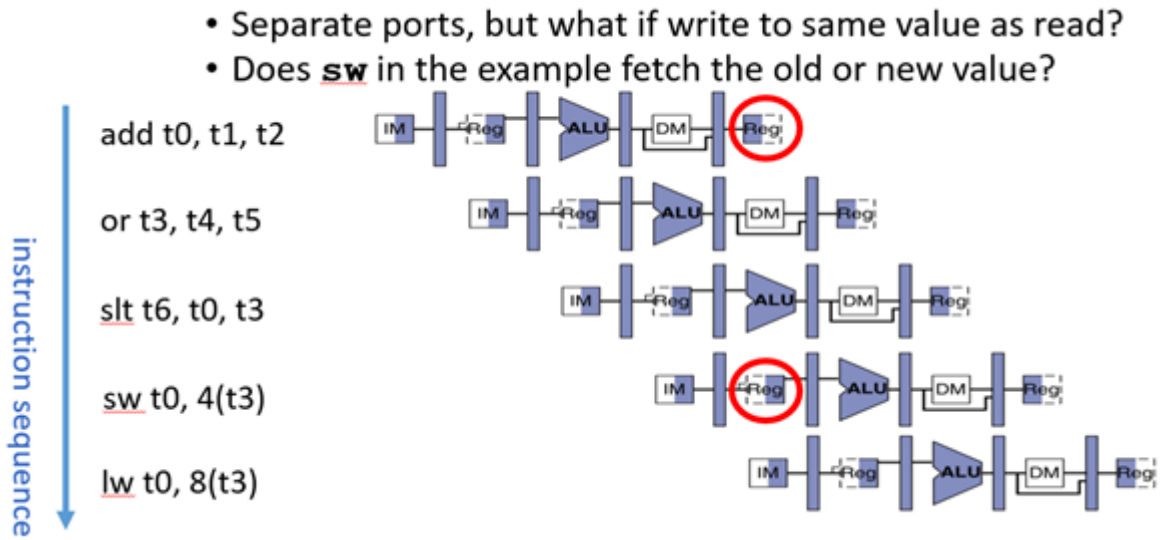


3.2.3 Data hazard

Có hai loại Data Hazard

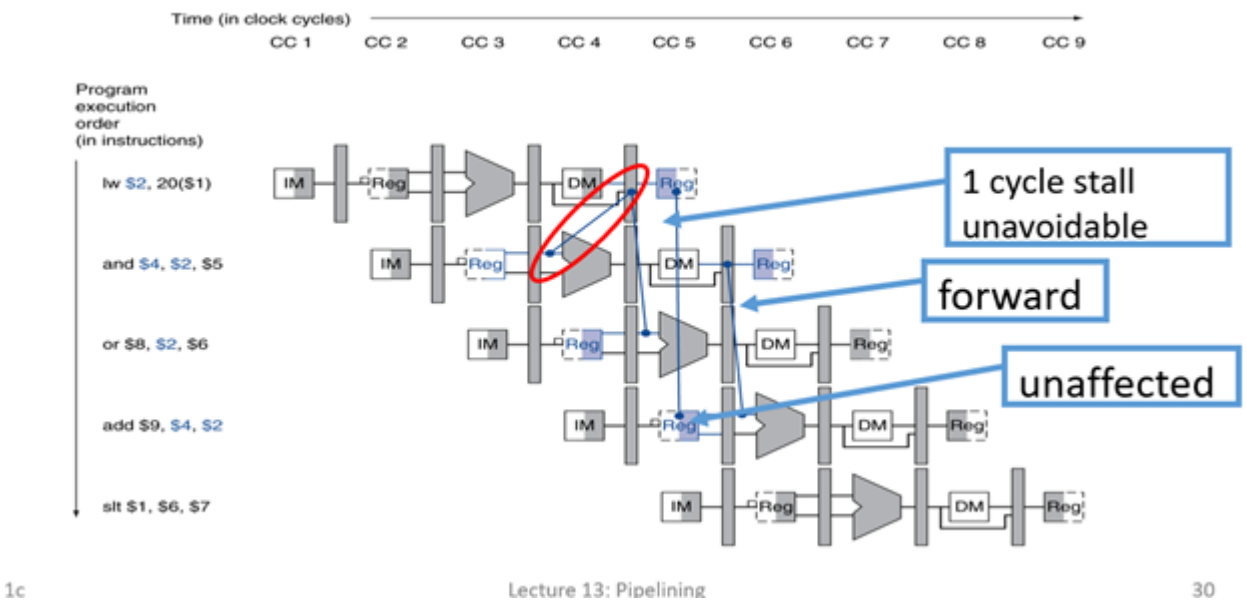
- **Hazard R type instruction**

  Ở đây ta dùng phương pháp Forwarding để lấy giá trị ở các tầng khác đưa về tầng đang cần dùng giá trị đó. Cấu trúc như sau:



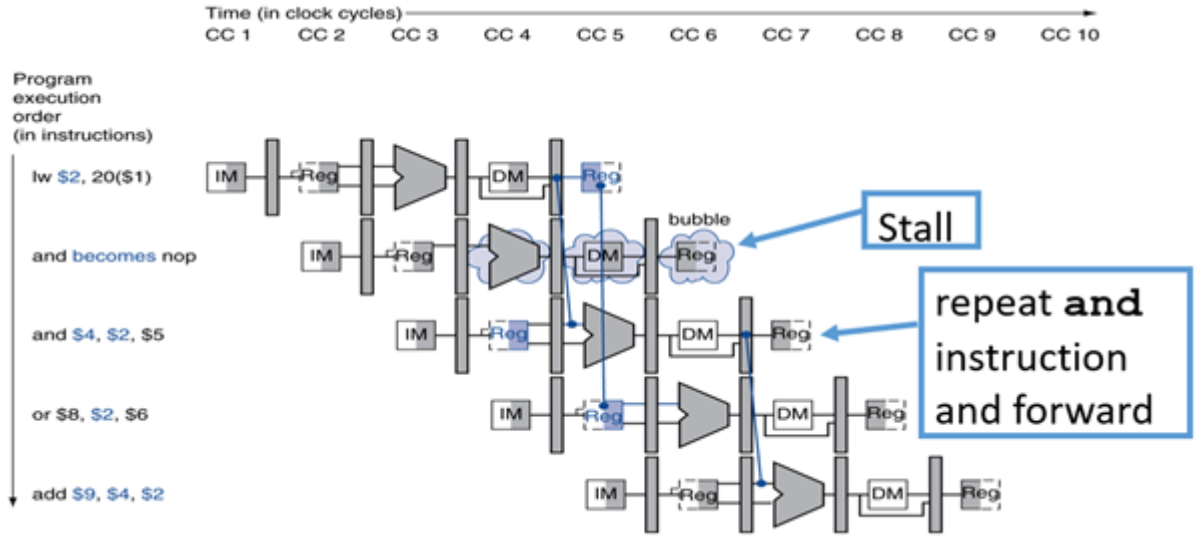
Việc Forward này có tác dụng đưa các giá trị cần dùng về tầng X khi ở tầng X cần dùng kết quả được tính trong cùng một thanh ghi nhưng nó vẫn chưa Writeback được về băng thanh ghi, ở đây ta cí thể có hai vị trí Writeback về đó là ở tầng M và tầng W. Việc Forward này đều có thể xảy ra được trên cả rs1 và rs2.

- **Load data hazard**



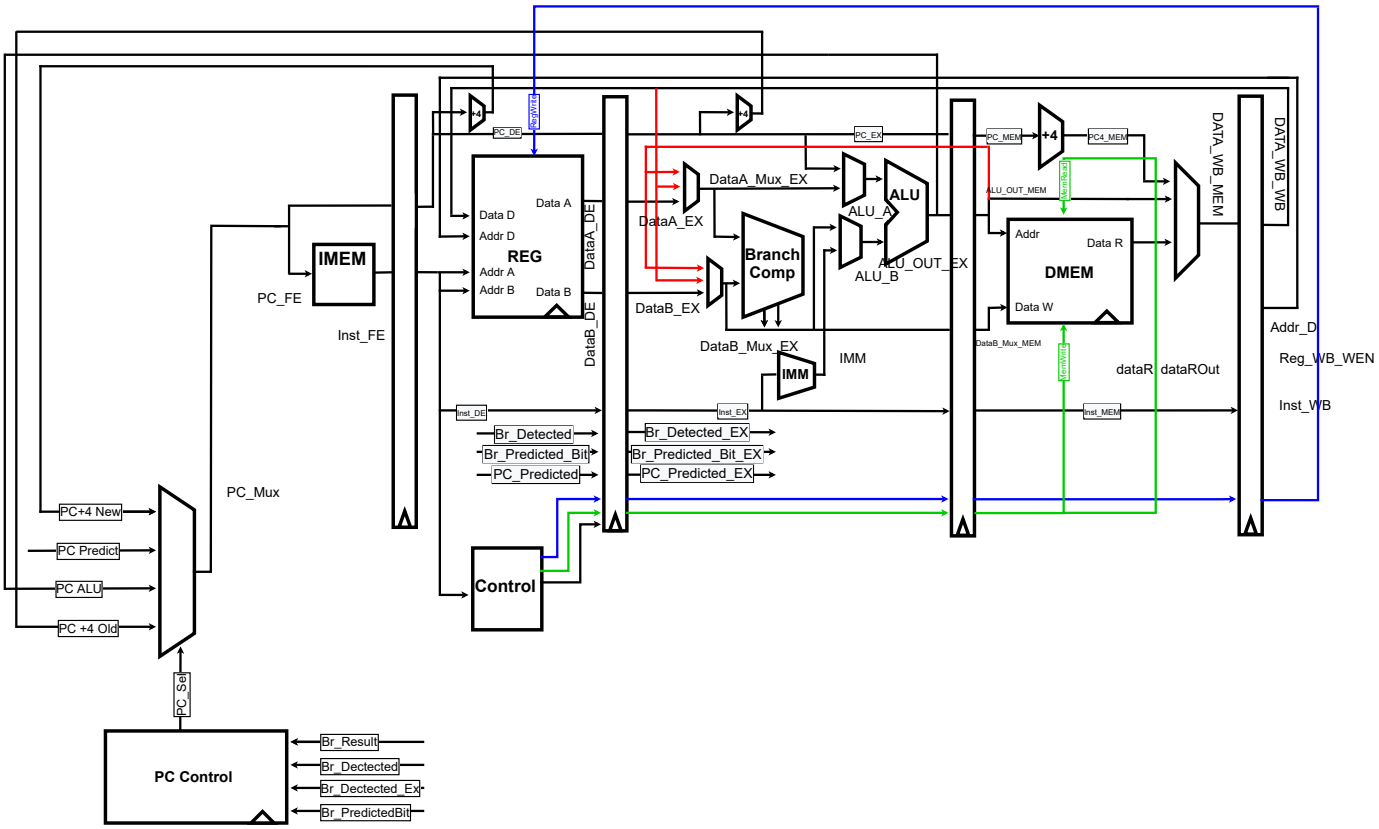
1c Lecture 13: Pipelining 30

Lỗi ở đây là việc khi dùng lệnh load vào một thanh ghi nào đó nếu tại thời điểm đó mà cũng chính là thời điểm bộ ALU cần giá trị của thanh ghi đó để tính toán nên ta phải dừng quá trình này lại 1 chu kỳ hay chèn một lệnh NOP vào giữa hai lệnh đó.



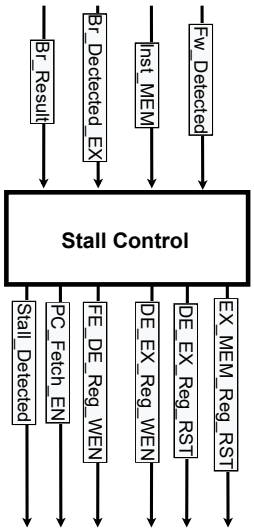
3.3 Thiết kế Pipeline 5 tầng

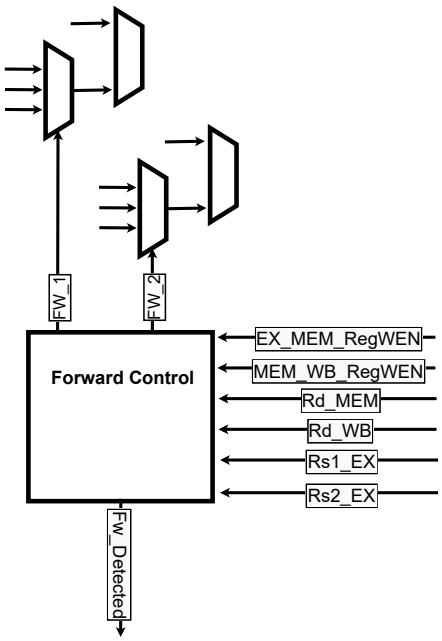
Kết quả phần cứng được thiết kế lại sau khi Pipeline:



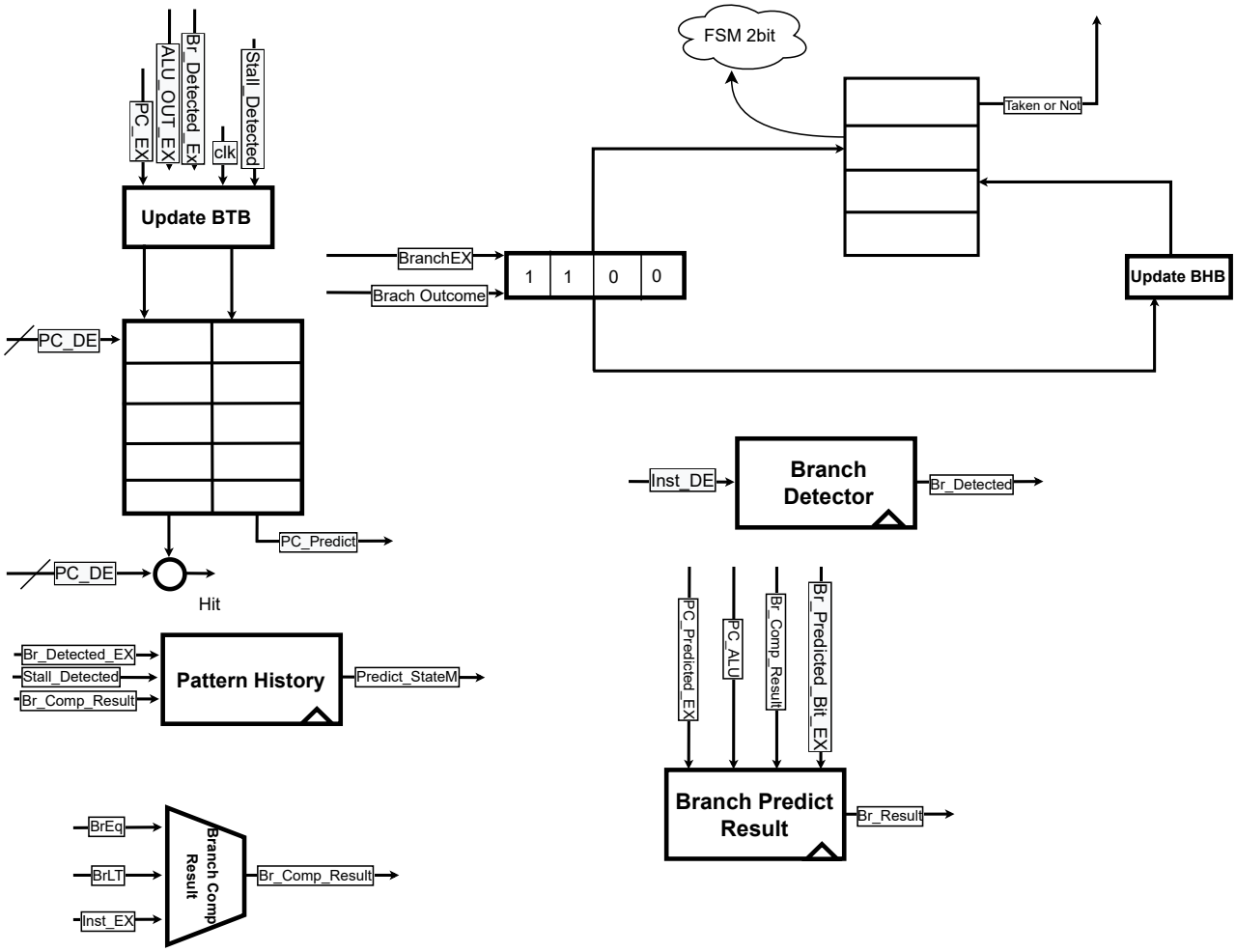
Ngoài ra, một số khối được thiết kế thêm để giải quyết các vấn đề về Hazard:

- **Bộ Stall và Forward**





– **Bộ Branch predict**



3.4 Thực hiện mô phỏng

3.4.1 Viết đoạn chương trình test

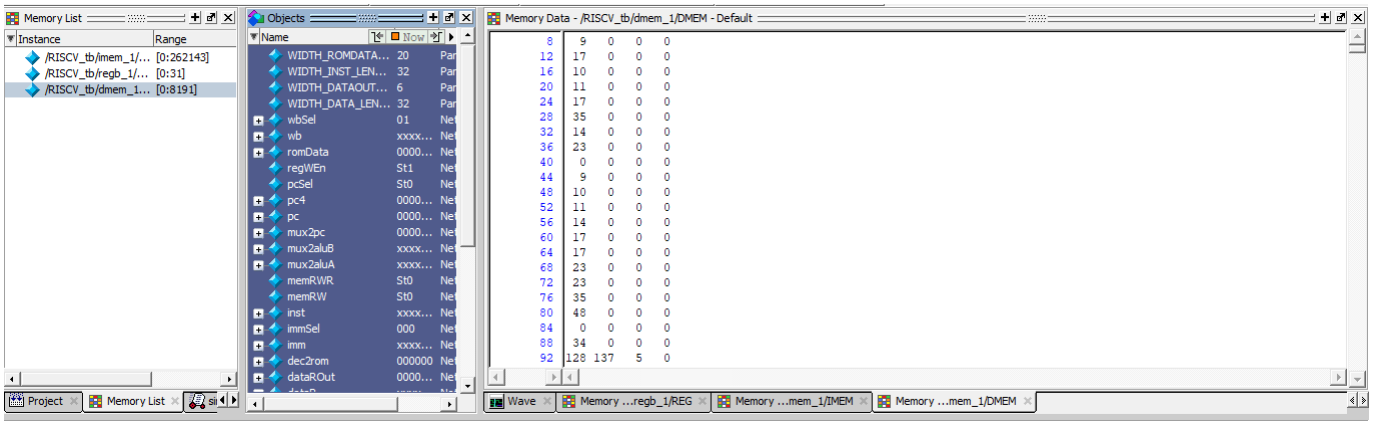
– Lấy 10 số lưu trong DMEM và sắp xếp lại rồi lưu vào DMEM ở 10 vị trí tiếp theo.

– Tính giai thừa số lớn nhất và lưu ở vị trí tiếp theo.

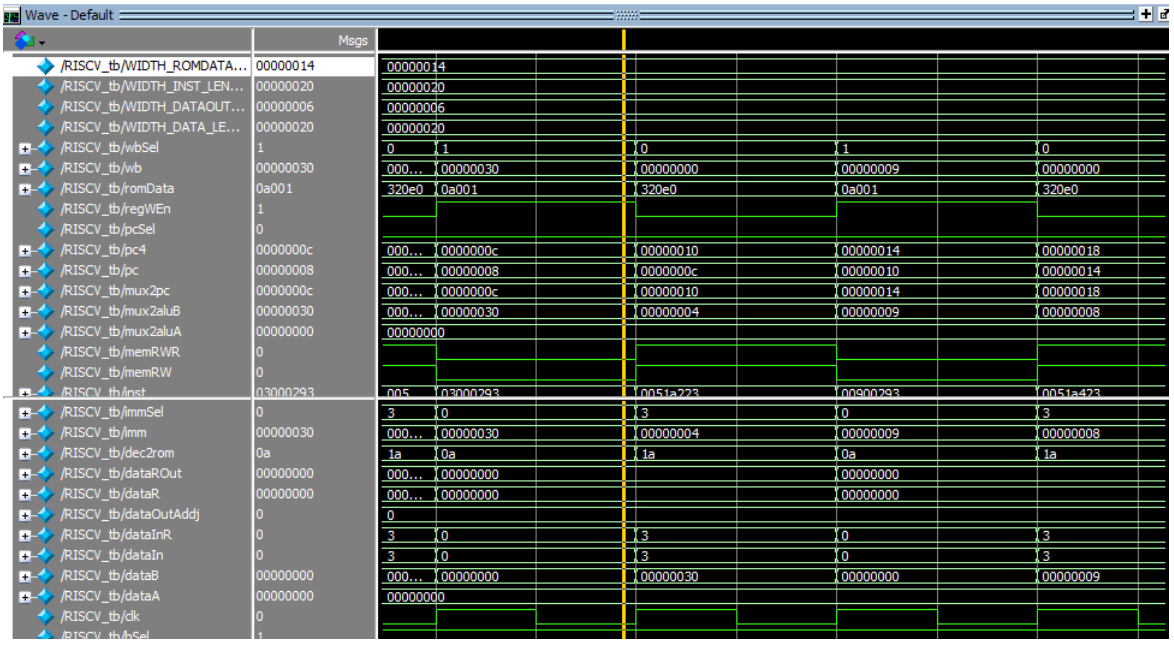
– Tính số Fibonanci của số lớn nhất và lưu ở vị trí tiếp theo.

3.4.2 Test dạng sóng trên ModelSim và kết quả mô phỏng

Kết quả sau khi chạy đoạn code trên:



Dạng sóng của đoạn code trên:



Chapter 4

Kết Luận

Tính chất quyết định cho viêc pipeline có hiệu quả hay không nằm ở tốc độ tính toán.

Ở đây ta sẽ dùng hai thông số **speech up** và **throughput** để thể hiện.

- **speech up** thể hiện tỉ lệ giữa thời gian thực hiện single và thời gian thực hiện pipeline, qua đó cho thấy mức độ cải thiện tốc độ của pipeline

$$speechup = \frac{TimeUsingSequentialProcessing}{TimeUsingPipelineProcessing} = \frac{m_1 * t_1}{(n + m_2 - 1) * t_2}$$

  - Trong đó:

  n: số tầng pipeline

  m_1, m_2: lần lượt là số lệnh được thực hiện trong chương trình single và pipeline (cách lệnh lặp lại được tính riêng).

  t_1: thời gian thực hiện 1 lệnh trong đơn chu kì.

  t_2: thời gian thực hiện 1 tầng trong pipeline.

- **throughput**: thể hiện số lệnh có thể thực hiện được trong 1 s.

$$throughput = \frac{m}{(n + m - 1) * t}$$

  - Trong đó:

  m: là số lệnh được thực hiện trong chương trình.

  n: là số tầng pipeline (bằng 1 với single).

  t: thời gian thực hiện 1 câu lệnh với single và là thời gian thực hiện 1 tầng với pipeline.

Từ cơ sở trên, ta có:

- Single cycle

$$speechup = \frac{2457 * 800}{(5 + 3179 - 1) * 200} = 3.08765$$

$$throughput = \frac{2457}{2457 * 800} = 1.25Gtask/s$$

- Pipiline 5 tầng

$$throughput = \frac{3179}{(3179 + 5 - 1) * 800} = 5Gtask/s$$

Vậy tóm lại:

- Tốc độ xử lý tăng lên đáng kể gần gấp 4 lần so với trước khi pipeline.
- Tỷ lệ thời gian single và pipeline chưa đủ lớn do chưa có các bộ dự đoán nhảy cũng như là bộ cache để lưu các dữ liệu cần dùng lại.