

**ĐẠI HỌC QUỐC GIA TP. HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ**



**BÁO CÁO BÀI TẬP LỚN MÔN HỌC
CẤU TRÚC MÁY TÍNH
THIẾT KẾ CPU RISC-V**

GVHD: T.S Trần Hoàng Linh

SVTH: Phan Nguyên Trung

MSSV: 1814519

Hoàng Đình Toàn

MSSV: 1814379

TP.HCM, 11/2021

Nội dung

I.	Tổng quát về CPU RV32:.....	5
1.	Tập lệnh:.....	5
2.	Thanh ghi:	7
3.	Bộ nhớ:	8
II.	Thiết kế CPU RISC-V đơn chu kỳ:	9
1.	Mục tiêu.....	9
2.	Sơ đồ khối :	9
3.	Code Verilog	19
4.	Code test.....	19
5.	Kết quả:	19
6.	Kết luận	24

Mục lục ảnh.

Hình 1: Nhóm lệnh R.	5
Hình 2: Nhóm lệnh I (tính toán).	6
Hình 3: Nhóm lệnh I (Load data).	6
Hình 4: Nhóm lệnh S (Store data).	7
Hình 5: Nhóm lệnh B (rẽ nhánh).	7
Hình 6: Nhóm lệnh J (nhảy không điều kiện).	7
Hình 7: Thanh ghi của CPU RISCV-32.	8
Hình 8: Tổ chức bộ nhớ CPU RISC-V 32.	8
Hình 9: Sơ đồ CPU RISC -V.	9
Hình 10: Sơ đồ khối PC.	10
Hình 11 Sơ đồ khối Plus1.	11
Hình 12: Sơ đồ khối IMEM,	11
Hình 13: Khối Register.	12
Hình 14: Sơ đồ khối ImmGen.	13
Hình 15: Khối Brand Comp.	14
Hình 16: Khối ALU.	15
Hình 17 Khối DMEM.	16
Hình 18: Khối Control Unit.	17
Hình 19: Khối Mux 2.	18
Hình 20: Khối Mux 3.	18
Hình 21: Câu lệnh thử nghiệm.	19
Hình 22: Dạng sóng.	20
Hình 23: Các bộ nhớ.	20
Hình 24: Chương trình thử nghiệm.	21
Hình 25: Dạng sóng.	22
Hình 26: Các bộ nhớ.	22
Hình 27: Chương trình test.	23
Hình 28: Dạng sóng.	23
Hình 29: Các bộ nhớ.	24

I. Tổng quát về CPU RV32:

RISC-V là một kiến trúc tập lệnh (ISA) phần cứng nguồn mở dựa trên các nguyên tắc máy tính với tập lệnh đơn giản hóa (RISC) đã thiết lập.

1. Tập lệnh:

CPU RISC-V 32 có tổng cộng 32 lệnh hợp ngữ trong đó mỗi lệnh có độ dài 32bit và có 7 bit [6:0] (opcode) để xác định loại lệnh, Tập lệnh RV32 còn được gọi là tập lệnh kiểu load-store vì các câu lệnh tính toán sẽ được thực hiện trên thanh ghi, nghĩa là dữ liệu trong bộ nhớ muốn được thực thi trước hết phải được nạp vào thanh ghi, sau khi tính toán trên các thanh ghi thì dữ liệu sẽ được lưu lại vào bộ nhớ. Các lệnh trong RICS-V được chia thành các nhóm như sau:

- Nhóm lệnh R:

R-Format có Opcode [6:0] = 0110011 .Nhóm lệnh này có chức năng thực hiện các phép tính toán. Khi thực thi lệnh , các lệnh trong nhóm này sẽ 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.

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

Hình 1: Nhóm lệnh R.

- Nhóm lệnh I (tính toán):

Nhóm lệnh có opcode [6:0] = 0010011 . Trừ 3 lệnh SRAI, SRLI, SLLI, chức năng của nhóm này là 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

Hình 2: Nhóm lệnh I (tính toán).

Nhóm lệnh I (Load data):

I (Load data) có opcode [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 $rs1 + \text{ext}(\text{imm}[11:5]\text{imm}[4:0])$. Sau đó lấy giá trị lưu trong thanh ghi rs2 lưu vào DMEM tại địa chỉ $rs1 + \text{ext}(\text{imm}[11:5]\text{imm}[4:0])$.

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

Hình 3: Nhóm lệnh I (Load data).

- Nhóm lệnh S (Store data):

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 $rs1 + \text{ext}(\text{imm}[11:5]\text{imm}[4:0])$. Sau đó lấy giá trị lưu trong thanh ghi rs2 lưu vào DMEM tại địa chỉ $rs1 + \text{ext}(\text{imm}[11:5]\text{imm}[4:0])$.

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

Hình 4: Nhóm lệnh S (Store data).

- Nhóm lệnh B (rẽ nhánh):

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.

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	BLTU
imm[12 10:5]	rs2	rs1	111	imm[4:1 11]	1100011	BGEU

Hình 5: Nhóm lệnh B (rẽ nhánh).

- Nhóm lệnh J (nhảy không điều kiện):

Nhóm lệnh này có opcode là [6:0] = 1101111. Chức năng nhảy không có điều kiện.

imm[20 10:1 11 19:12]			rd	1101111	JAL
imm[11:0]	rs1	000	rd	1100111	JALR

Hình 6: Nhóm lệnh J (nhảy không điều kiện).

2. Thanh ghi:

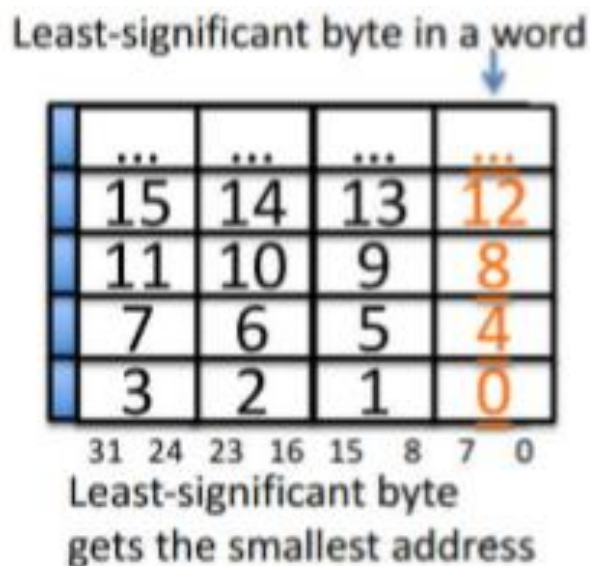
Có 32 thanh ghi trong bang thanh ghi (Register Bank) được đánh dấu từ x₀ - x₃₁. Mỗi thanh ghi có độ dài 32 bits. Do đó cần 5 bits địa chỉ của thanh ghi trong bang thanh ghi. Chức năng của mỗi thanh ghi được quy định như sau:

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
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	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Hình 7: Thanh ghi của CPU RISC-V-32.

3. Bộ nhớ:

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.



Hình 8: Tổ chức bộ nhớ CPU RISC-V 32.

II. Thiết kế CPU RISC-V đơn chu kỳ:

1. Mục tiêu

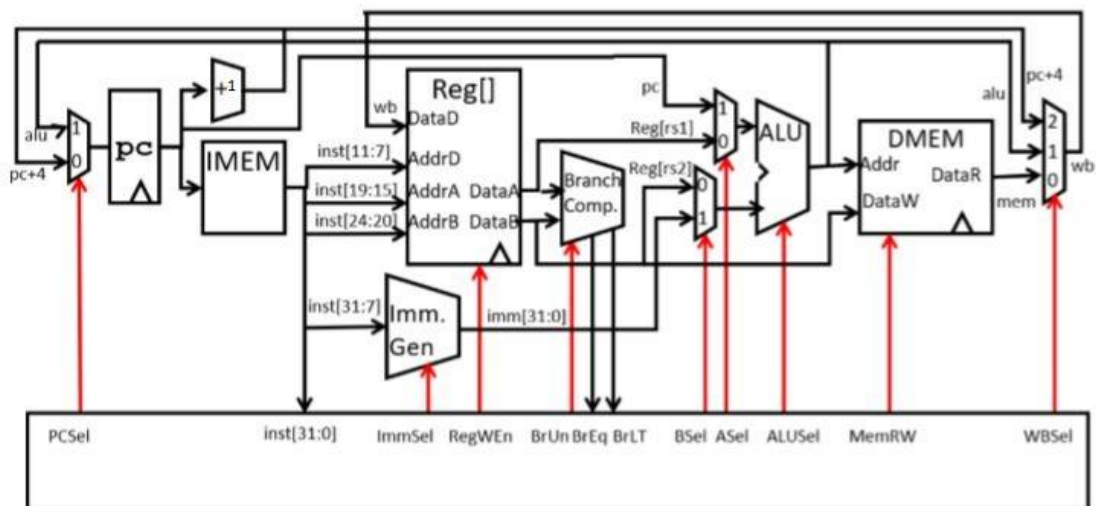
Nhóm tiến hành thiết kế CPU RISC-V đơn chu kỳ với một số tính chất như sau:

Mỗi câu lệnh được thực thi trong một chu kỳ và trong mỗi chu kỳ chỉ có 1 câu lệnh được thực hiện. Mỗi chu kỳ có độ dài 1ns.

Dữ liệu trong bộ nhớ IMEM và DMEM sẽ được định địa chỉ theo word (32 bits). Do đó sau mỗi câu lệnh, khối PC (Program Counter) tăng thêm 1 thay vì tăng thêm 4.

2. Sơ đồ khối :

- Sơ đồ CPU RICS –V.



Hình 9: Sơ đồ CPU RISC -V.

Hình 9: Sơ đồ khối CPU thiết kế

Để thiết kế được kiến trúc CPU như trên, ta sẽ tách thành từng khối nhỏ. Cụ thể ta sẽ tách thành các khối như sau.

- Khối PC

Có nhiệm vụ lấy địa chỉ của ô nhớ chứa lệnh. Khi có xung Clock tích cực cạnh lên thì giá trị của ngõ ra Out lập tức bằng bằng ngõ ra In.

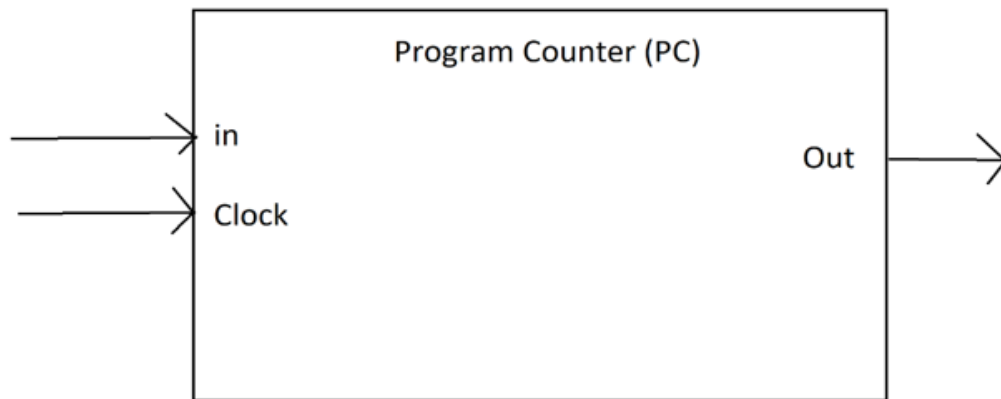
Ngõ vào :

- In [31:0] là địa chỉ ô nhớ lấy lệnh.
- Clock là tín hiệu xung Clock

Ngõ ra:

- Out [31:0] là địa chỉ ô nhớ lấy lệnh

Sơ đồ khối như sau:



Hình 10: Sơ đồ khối PC.

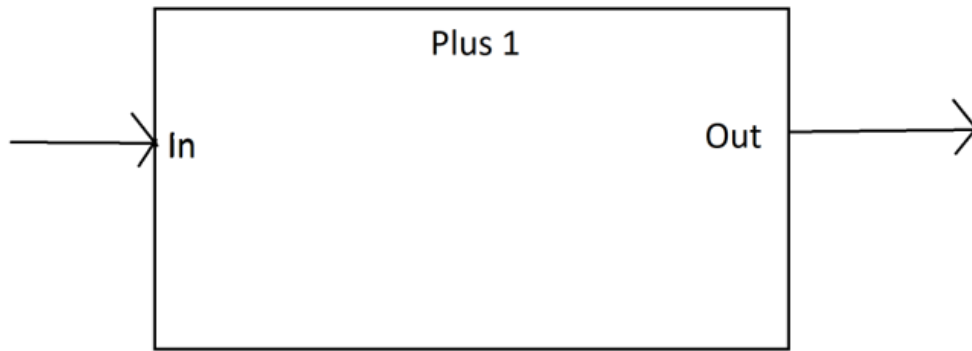
- Khối Plus1:

Chức năng: Ngõ ra Out sẽ bằng ngõ In tăng thêm 1. Khối này tạo bộ đếm cho bộ PC ở trên.

Ngõ vào : In [31:0]

Ngõ ra: Out [31:0]

Sơ đồ khối như sau:



Hình 11 Sơ đồ khối Plus1.

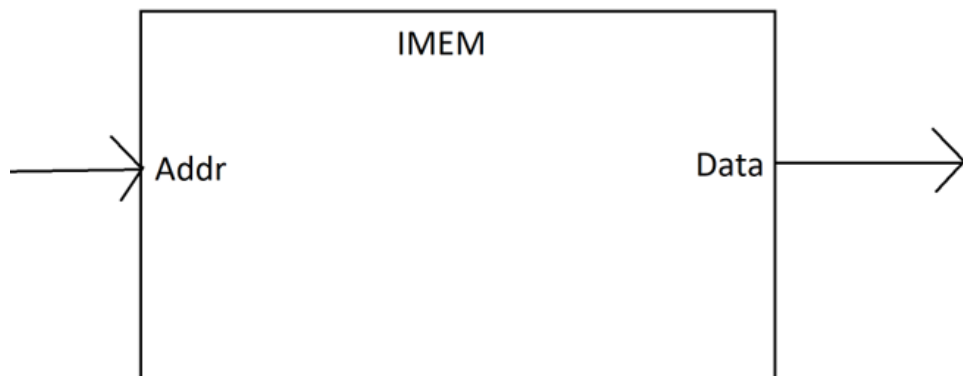
- Khối IMEM

Chức năng: Đây là khối chứa toàn bộ lệnh thực thi.

Ngõ vào: Addr [31:0] là địa chỉ của ô nhớ chứa lệnh.

Ngõ ra : Data [31:0] là nội dung có trong ô nhớ với địa chỉ Addr đầu vào

Sơ đồ khối như sau:



Hình 12: Sơ đồ khối IMEM,

- Khối Register

Là thanh ghi chứa dữ liệu:

Ngõ vào :

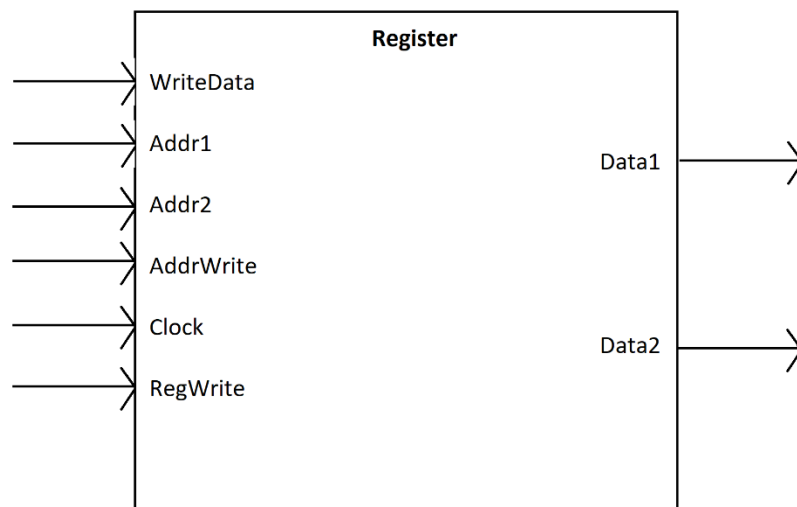
- Addr1 [5:0] : Địa chỉ thanh ghi được đọc thứ nhất.
- Addr 2 [5:0] : Địa chỉ thanh ghi được đọc thứ hai.
- AddrWrite [5:0] : Địa chỉ thanh ghi được ghi.

- WriteData [31:0] : Nội dung dữ liệu sẽ được ghi.
- RegWrite : Tín hiệu cho phép lệnh ghi được thực hiện (tích cực cao).
- Clock : Tín hiệu xung clock.

Ngõ ra:

- Data1: Nội dung của thanh ghi được đọc thứ nhất.
- Data2: Nội dung của thanh ghi được đọc thứ hai.

Sơ đồ khối như sau:



Hình 13: Khối Register.

- Khối ImmGen

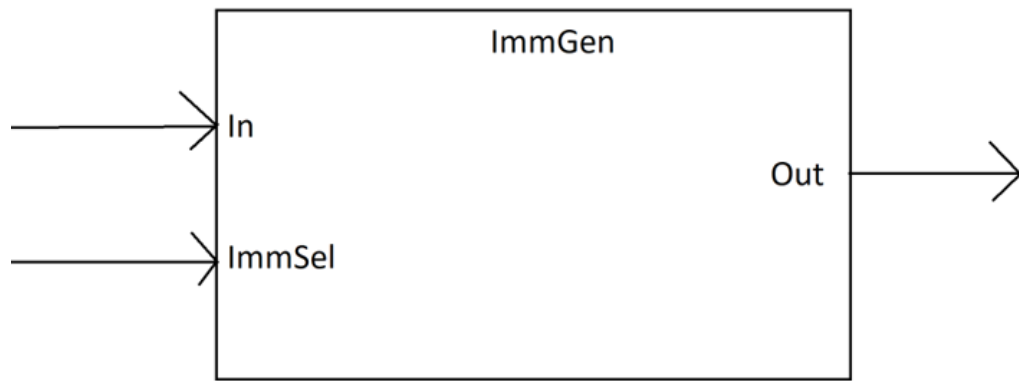
Chức năng: Tạo và mở rộng dấu

Ngõ vào:

- In [24:0] : Dữ liệu sẽ được mở rộng dấu.
- ImmSel [2:0]: Tín hiệu quy định cách mở rộng dấu

Ngõ ra:

- Out [31:0] là dữ liệu sau khi được mở rộng dấu.



Hình 14: Sơ đồ khối ImmGen.

- Khối BranchComp:

Ngõ vào:

- A [31:0] : Dữ liệu so sánh.
- B [31:0] : Dữ liệu so sánh.
- BrUn: Tín hiệu điều khiển phương thức so sánh. BrUn =1 thì so sánh dữ liệu không dấu, BrUn so sánh dữ liệu có dấu.

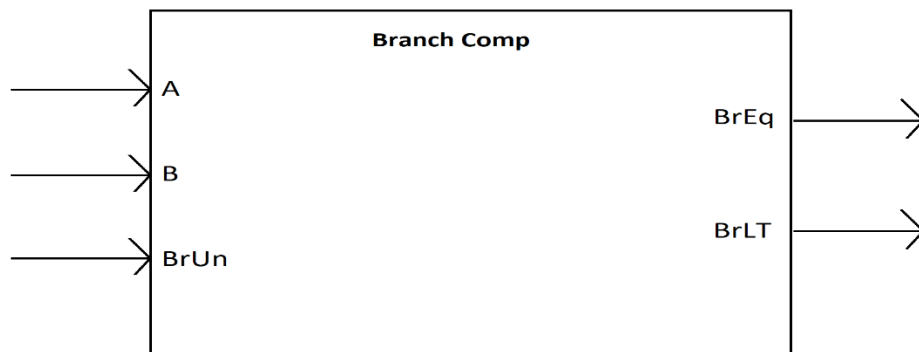
Ngõ ra:

- BrEq : Tín hiệu ra
- BrLT: Tín hiệu ra

Đây là bộ so sánh. Hai dữ liệu đưa vào sẽ được so sánh với nhau. Bảng sự thật như sau:

Các trường hợp	BrUn	BrEq	BrLT
A = B	x	1	0
A > B	x	0	0

$A < B$	x	0	1
$A \leq B$	x	1	1
$A \geq B$	x	1	0



Hình 15: Khối Branch Comp.

- Khối ALU

Chức năng: Đây là bộ ALU thực hiện các lệnh tính toán, logic của CPU.

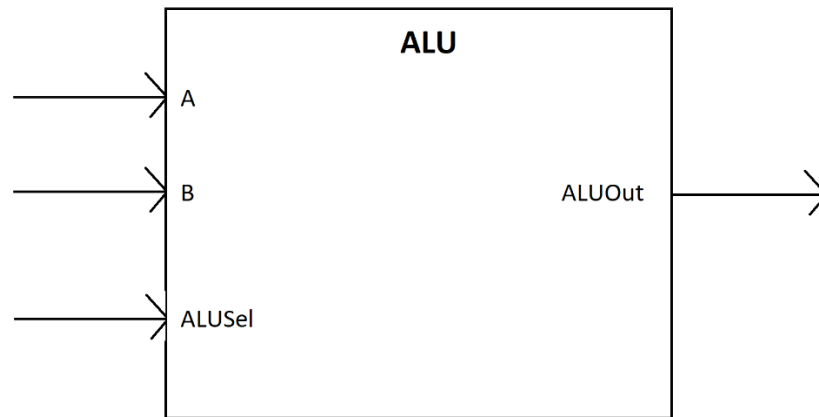
Ngõ vào :

- A [31:0] : Dữ liệu phép tính.
- B [31:0] : Dữ liệu phép tính.
- ALUSel [3:0]: Tín hiệu chọn loại phép tính.

Ngõ ra:

- ALUOut [31:0] là kết quả sau khi thực hiện phép toán.

Sơ đồ khối như sau:



Hình 16: Khối ALU.

- Khối DMEM

Chức năng: Đây là khối chứa dữ liệu của CPU. Cho phép CPU đọc và ghi dữ liệu vào.

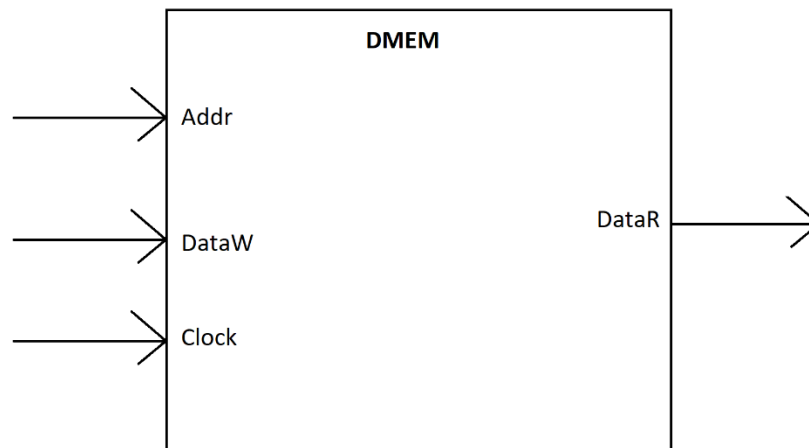
Ngõ vào:

- Addr [31:0]: Địa chỉ dữ liệu được ghi hoặc đọc.
- DataW [31:0] : Dữ liệu ghi.
- Clock: Tín hiệu Clock

Ngõ ra:

- DataR [31:0] là dữ liệu được đọc.

Sơ đồ khối như sau:



Hình 17 Khối DMEM.

- Khối Control Unit

Chức năng: Là bộ điều khiển của CPU. Thiết kế theo kiểu ROM với bảng sự thật được đính kèm trong tệp nộp bài.

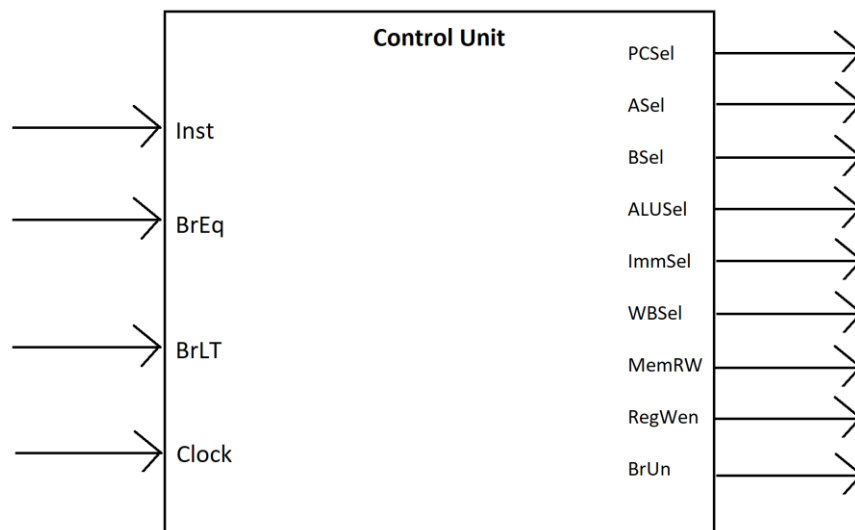
Ngõ vào:

- Inst [31:0]: Nội dung của câu lệnh thực thi.
- BrEq: Tín hiệu so sánh bằng từ bộ Branch comp.
- BrLT: Tín hiệu so sánh bé hơn từ bộ Branch comp.
- Clock: Xung clock.

Ngõ ra:

- PCSel: Tín hiệu điều khiển mux PC.
- ASel: Tín hiệu điều khiển muxALU.
- BSel: Tín hiệu điều khiển muxBranch.
- ALUSel: Tín hiệu điều khiển bộ ALU.
- ImmSel: Tín hiệu điều khiển khối ImmGen.
- WBSel: Tín hiệu điều khiển muxDMEM.
- MemRW: Tín hiệu điều khiển khối DMEM.
- RegWen: Tín hiệu điều khiển lệnh ghi của thanh ghi Resgister.
- BrUn: Tín hiệu điều khiển bộ Branch comp.

Sơ đồ khối như sau:



Hình 18: Khối Control Unit.

- **Khối Mux 2**

Chức năng: Đây là khối mux 2 to 1.

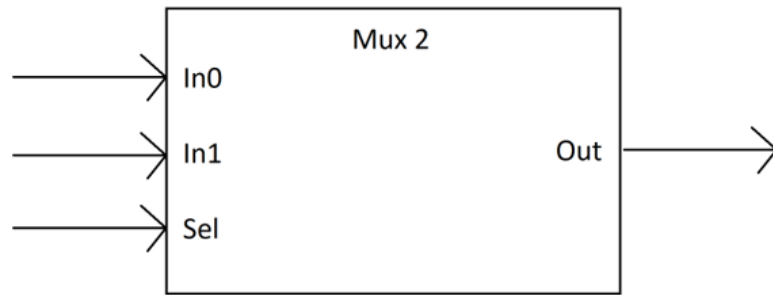
Ngõ vào:

- In0 [31:0]: Dữ liệu thứ nhất.
- In1 [31:0]: Dữ liệu thứ hai.
- Sel: Tín hiệu chọn dữ liệu.

Ngõ ra: Out [31:0] là dữ liệu ngõ ra phụ thuộc vào tín hiệu Sel.

Thiết kế CPU theo sơ đồ trên cần 3 bộ mux 2 gồm:

- MuxPC: Bộ mux gần khối PC có chức năng chọn tín hiệu đầu vào cho thanh ghi PC.
- MuxBranch: Bộ mux sau và gần khối Branch comp nhất có chức năng chọn dữ liệu đầu vào thứ hai cho bộ ALU.
- MuxALU: Bộ mux trước và gần khối ALU nhất có chức năng chọn dữ liệu đầu vào thứ nhất cho bộ ALU.



Hình 19: Khối Mux 2.

- Khối Mux 3:

Chức năng: Đây là khối mux 3 to 1.

Ngõ vào:

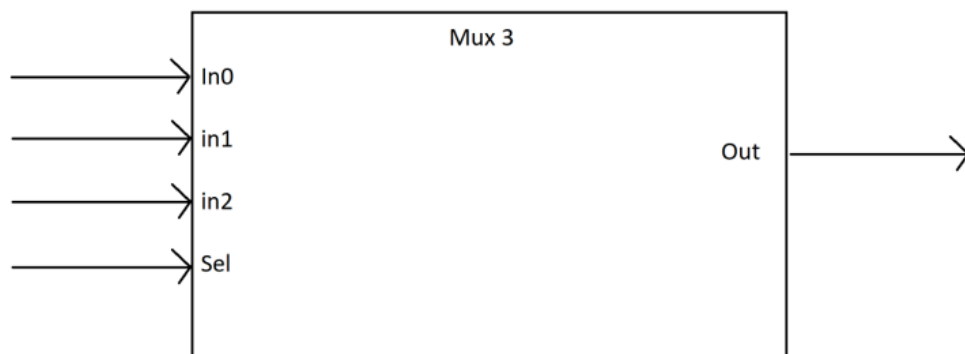
- In0 [31:0]: Dữ liệu thứ nhất.
- In1 [31:0]: Dữ liệu thứ hai.
- In2 [31:0]: Dữ liệu thứ ba.
- Sel: Tín hiệu chọn dữ liệu.

Ngõ ra:

- Out [31:0] là dữ liệu ngõ ra phụ thuộc vào tín hiệu Sel.

Trong bài này ta cần 1 bộ Mux 3 nằm ngay sau khối DMEM có chức năng chọn dữ liệu để được ghi vào thanh ghi Register.

Sơ đồ khối như sau:



Hình 20: Khối Mux 3.

3. Code Verilog

Được đính kèm trong tệp nộp bài.

4. Code test

Được đính kèm trong tệp nộp bài.

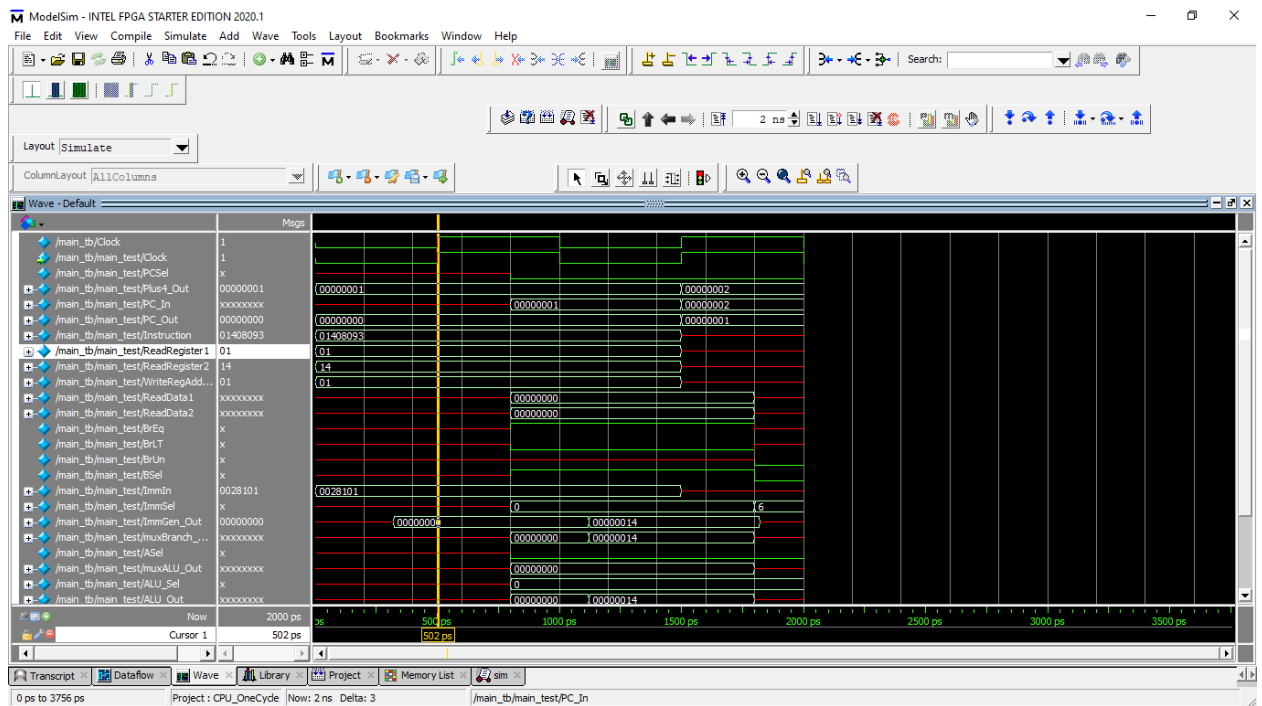
5. Kết quả:

- Thử nghiệm với từng lệnh:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00003000	0x01408093	addi x1,x1,0x00000014	l: addi x1,x1,20

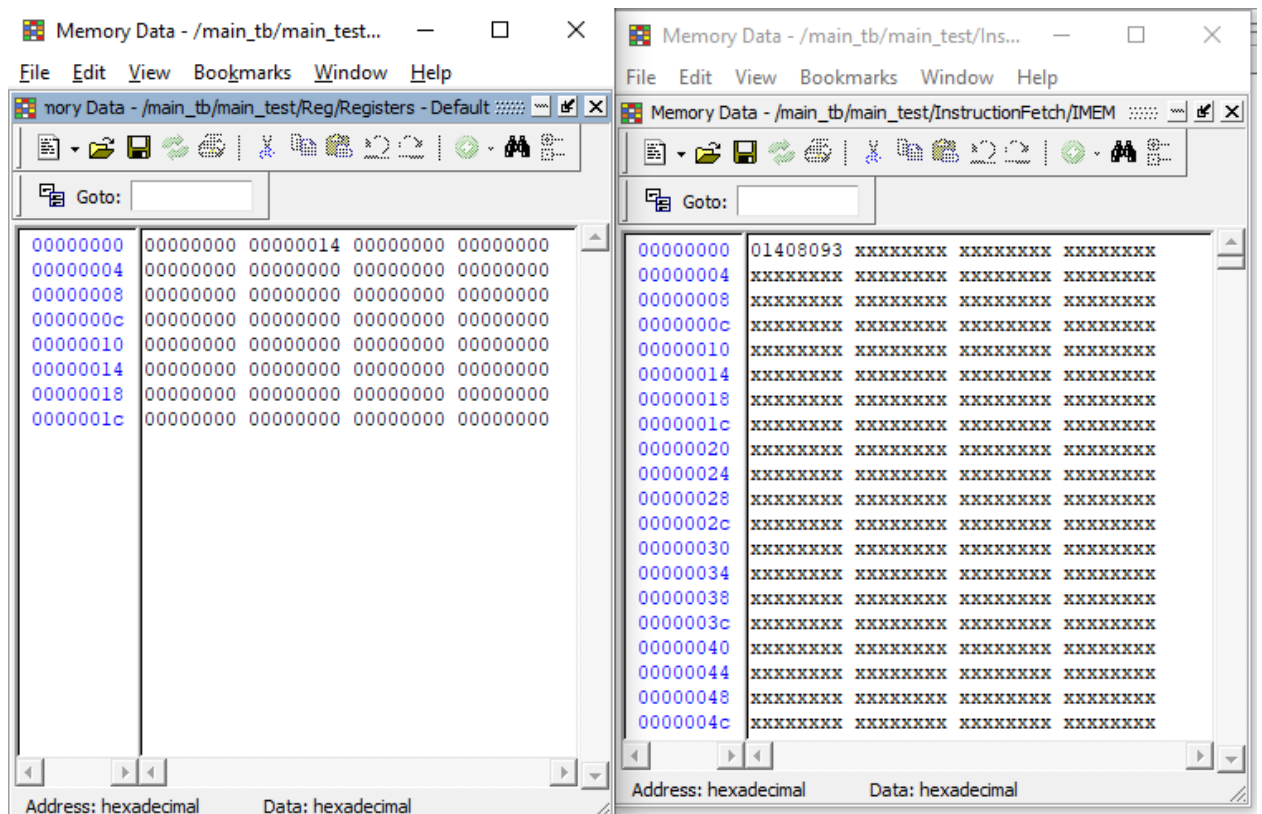
Hình 21: Câu lệnh thử nghiệm.

Đây là lệnh cộng vào thanh ghi x1 giá trị 20.



Hình 22: Dạng sóng.

Nhìn vào dạng sóng ta có thể thấy được rằng CPU đã thực hiện lệnh cộng trên cho một chu kỳ.



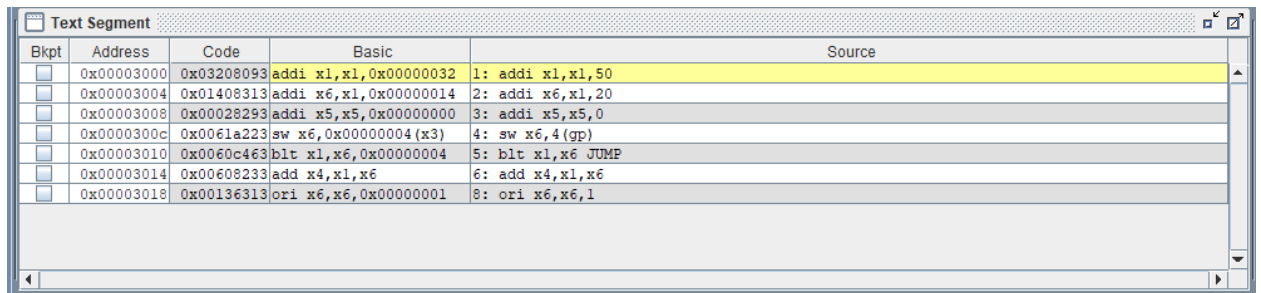
Hình 23: Các bộ nhớ.

Bộ nhớ chương trình đã load opcode của lệnh, thanh ghi register đã hiển thị giá trị thanh ghi x1 = 20 (0x14).

⇒ Lệnh thực hiện đúng với thực tế.

Thử nghiệm với tất cả những lệnh còn lại cũng đã cho ra kết quả đúng thực tế.

- Chương trình thử nghiệm.



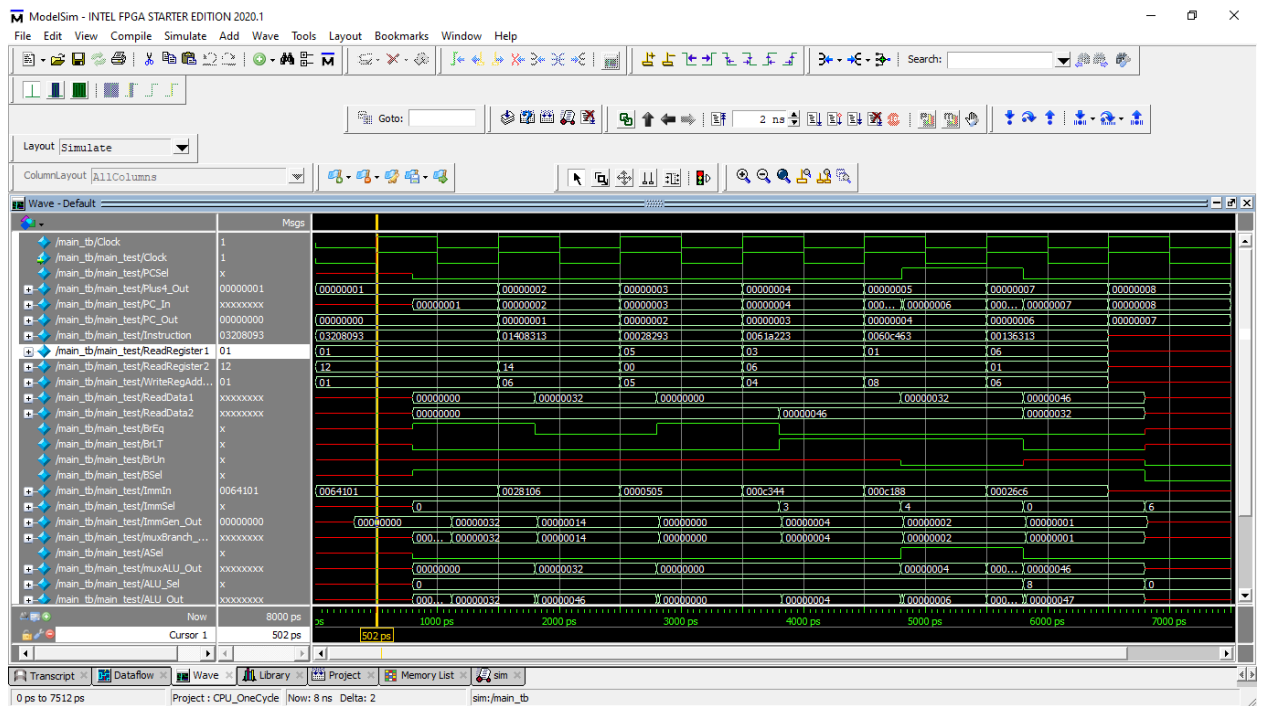
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00003000	0x03208093	addi x1,x1,0x00000032	1: addi x1,x1,50
<input type="checkbox"/>	0x00003004	0x01408313	addi x6,x1,0x00000014	2: addi x6,x1,20
<input type="checkbox"/>	0x00003008	0x00028293	addi x5,x5,0x00000000	3: addi x5,x5,0
<input type="checkbox"/>	0x0000300c	0x0061a223	sw x6,0x00000004(x3)	4: sw x6,4(gp)
<input type="checkbox"/>	0x00003010	0x0060c463	blt x1,x6,0x00000004	5: blt x1,x6 JUMP
<input type="checkbox"/>	0x00003014	0x00608233	add x4,x1,x6	6: add x4,x1,x6
<input type="checkbox"/>	0x00003018	0x00136313	ori x6,x6,0x00000001	8: ori x6,x6,1

Hình 24: Chương trình thử nghiệm.

Chương trình thực hiện tuần tự như sau:

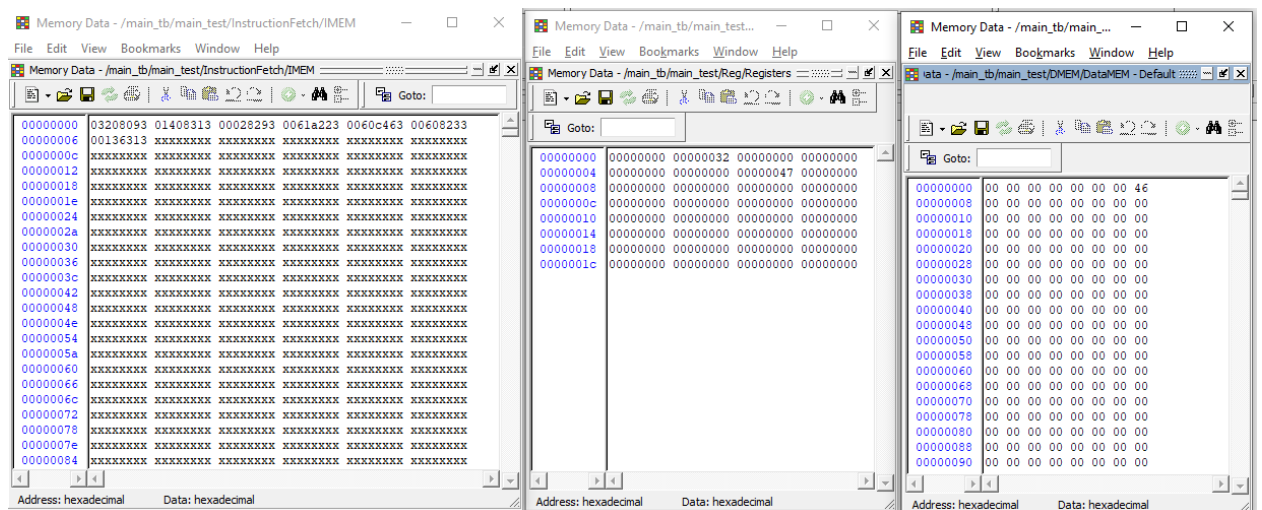
Ghi vào thanh ghi 1 giá trị 50. Ghi vào thanh ghi x6 tổng của giá trị trong thanh ghi x1 với giá trị 20. Sau lệnh này giá trị của thanh ghi x6 là 70. Sau đó lưu giá trị 0 vào thanh ghi x5.

Tiếp theo lưu giá trị trong thanh ghi x6 là 70 (0x46) vào ô nhớ có địa chỉ là 0x4. Tiếp theo thực hiện lệnh so sánh x1 và x6. Vì giá trị của thanh ghi x1 là 50 nhỏ hơn thanh ghi x6 là 70, do đó chương trình nhảy tới nhãn JUMP. Vì vậy câu lệnh add x4,x1,x6 không được thực hiện. Câu lệnh tiếp theo được thực hiện là ori giá trị trong thanh ghi x6 và 1 sau đó kết quả sẽ được lưu vào thanh ghi x6. Kết thúc chương trình: Giá trị trong thanh ghi x1 = 50, x5 = 0, x6 = 47. Giá trị trong ô nhớ có địa chỉ 0x4 là 70 (0x46).



Hình 25: Dạng sóng.

Nhìn vào dạng sóng ta thấy rằng chương trình đã thực hiện các lệnh liên tiếp nhau và mỗi lệnh được thực hiện trong một chu kỳ duy nhất.



Hình 26: Các bộ nhớ.

Các bộ nhớ IMEM, DMEM, Register cũng đã cho ra kết quả đúng nội dung của code assembly.

⇒ Chương trình chạy cho ra kết quả đúng với thực tế.

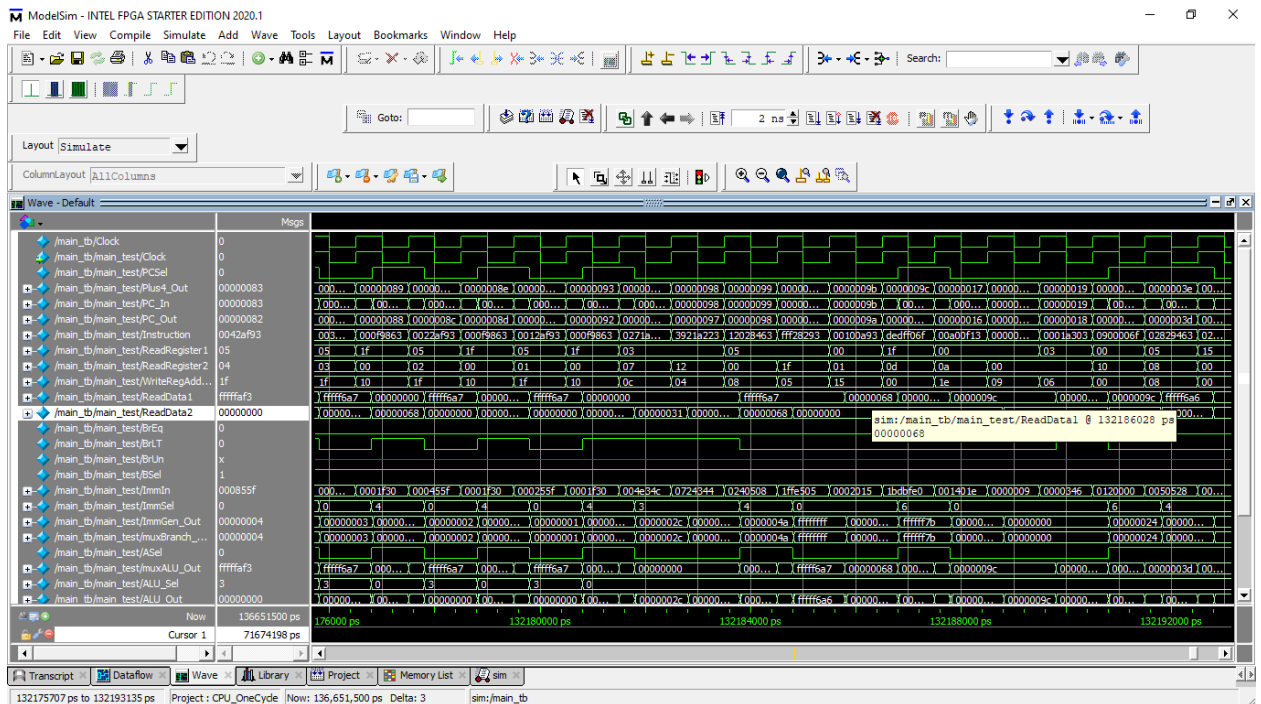
- Thử nghiệm với chương trình được yêu cầu:

Text Segment				
Bkpt	Address	Code	Basic	Source
	0x00003000	0x01700293	addi x5,x0,0x00000017	51: li t0, 23
	0x00003004	0x0051a023	sw x5,0x00000000(x3)	52: sw t0, 0(gp)
	0x00003008	0x03000293	addi x5,x0,0x00000030	53: li t0, 48
	0x0000300c	0x0051a223	sw x5,0x00000004(x3)	54: sw t0, 4(gp)
	0x00003010	0x00900293	addi x5,x0,0x00000009	55: li t0, 9
	0x00003014	0x0051a423	sw x5,0x00000008(x3)	56: sw t0, 8(gp)
	0x00003018	0x01100293	addi x5,x0,0x00000011	57: li t0, 17
	0x0000301c	0x0051a623	sw x5,0x0000000c(x3)	58: sw t0, 12(gp)
	0x00003020	0x00a00293	addi x5,x0,0x0000000a	59: li t0, 10
	0x00003024	0x0051a823	sw x5,0x00000010(x3)	60: sw t0, 16(gp)

Hình 27: Chương trình test.

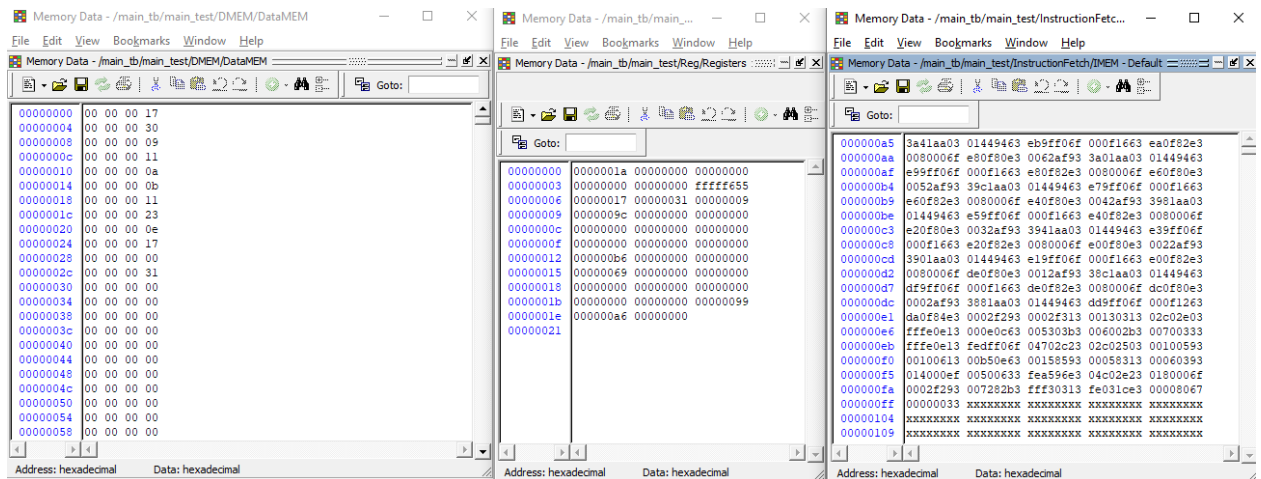
Đây là một đoạn chương trình mẫu với hơn 250 câu lệnh được giáo viên cung cấp.

Tuy nhiên nhóm thực sự chưa biết chương trình này kết quả cuối cùng sẽ như thế nào vì khi chạy thử trên phần mềm compile đã gặp một vòng lặp rất dài.



Hình 28: Dạng sóng.

Dạng sóng khi chạy thử CPU của nhóm thiết kế cũng gặp tình trạng vòng lặp rất dài.



Hình 29: Các bộ nhớ.

Bộ nhớ IMEM đã load hoàn toàn tất cả các câu lệnh sau khi được compile. Vì không biết được kết quả cuối cùng của chương trình nên nhóm không thể đưa ra các kết quả so sánh với hai bộ DMEM và Register.

6. Kết luận

Nhóm đã thiết kế thành công CPU RISC-V 32 bit với kết quả chạy đúng với thực tế.

Tuy nhiên còn một số câu lệnh nhóm chưa đưa vào thiết kế, đó là các câu lệnh thực thi với dữ liệu byte và half word (lb, lhu, sb, sbu,...).