LAB 6: THIẾT KẾ BỘ XỬ LÝ MIPS ĐƠN GIẢN

Trần Minh Tiến 23521586
Trần Hoàng Thông 23521529

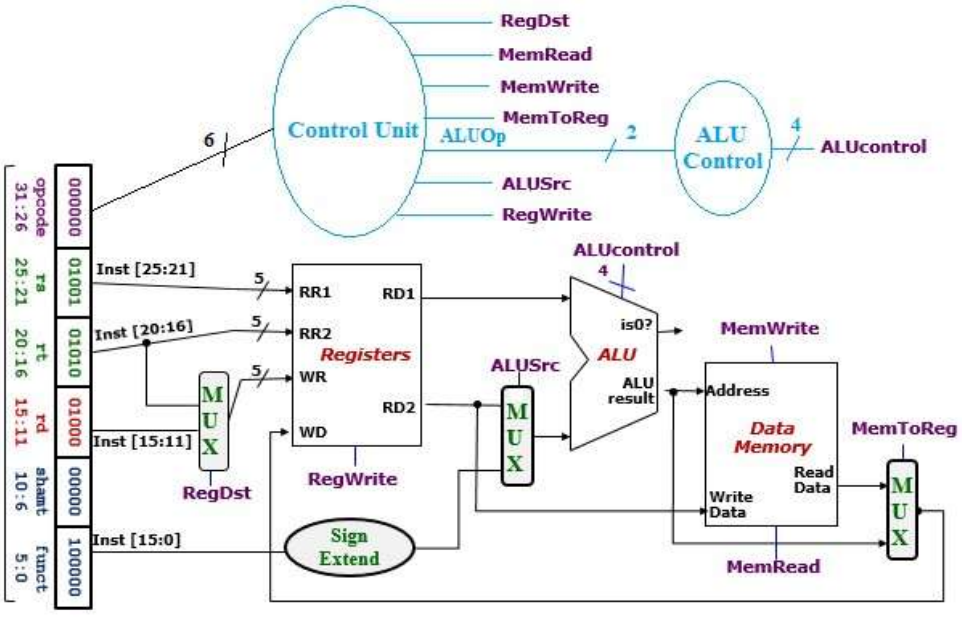
1.1 Mục tiêu

Sinh viên ôn tập kiến thức về tập lệnh và Datapath của MIPS. Thiết kế Datapath và Control Unit cho bộ xử lý đơn chu kỳ theo kiến trúc MIPS.

1.2 Nội dung thực hành

Sinh viên dựa vào lý thuyết về DATAPATH của kiến trúc MIPS đã học trong môn Kiến trúc máy tính như trên Hình 6-1 và sử dụng lại các module đã thiết kế trong các lab trước để thiết kế Datapath và Control Unit thực hiện được một số lệnh cơ bản của MIPS, như:

- add $1, $2, $3
- addi $4, $5, -23
- slt $3, $2, $1
- lw $1, 16($2)
- sw $1, 0($2)



Hình 6-1 Data path và Control Unit theo kiến trúc MIPS

Thực hành: Thiết kế Luận lý Số (CE118)

1.3 Sinh viên chuẩn bị

1. Kiểm tra lại các module Register File, ALU đã thiết kế trong các bài Lab trước.
2. Khối DataMem có thể thiết kế tương tự khối Register File hoặc SRAM để phục vụ cho nội dung bài lab.
3. Tìm hiểu lại kiến trúc và cách thức hoạt động của Datapath và Control unit trong kiến trúc MIPS.
4. Lập bảng sự thật cho tín hiệu điều khiển:

Opcode	Lệnh	ALUOp [1:0]	ALUcontrol [3:0]	RegDst	MemRead	memWrite	MemToReg	ALUSrc	RegWrite
000001	add	10	0010						
000010	sw	00	0010						
000100	lw	00	0010						

1.4 Hướng dẫn thực hành

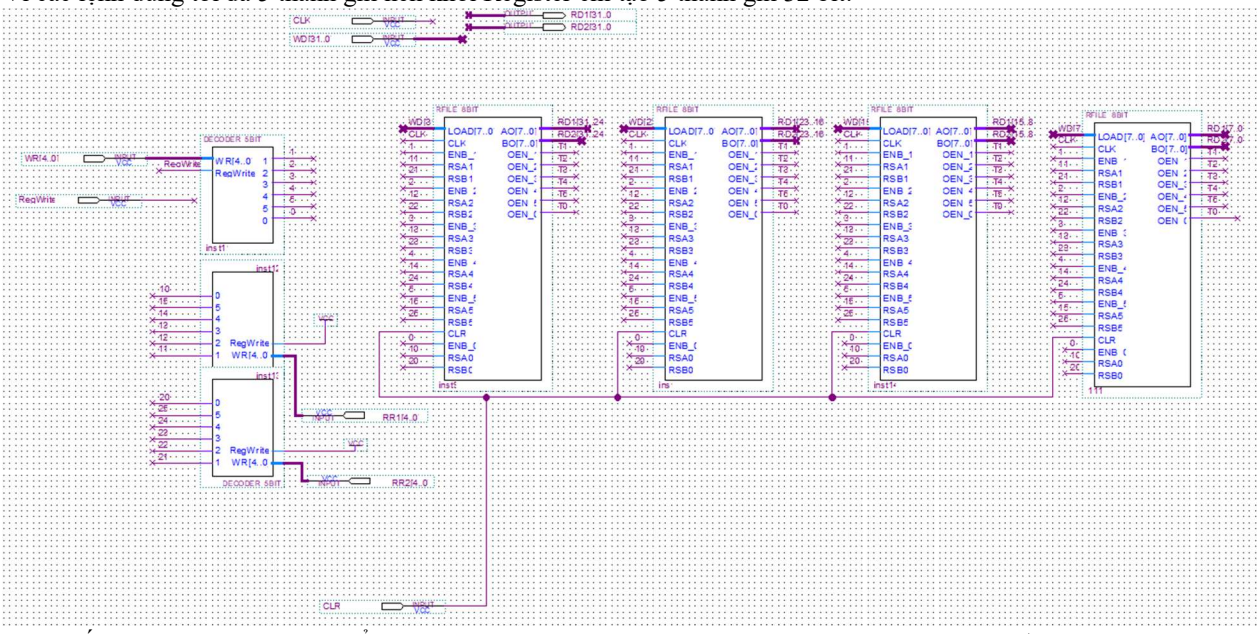
1. Tạo một project mới, đặt tên: E/CE118_lab/lab6_MSSV
2. Kết nối các thành phần đơn lẻ để tạo thành DATAPATH hoàn chỉnh trên phần mềm Quartus II
3. Thiết kế Control Unit trên phần mềm Quartus II
4. Kết nối Datapath và Control Unit, kiểm tra thực thi các lệnh MIPS ở trên (SV có thể thực hiện thêm các lệnh khác). SV thực hiện từng lệnh riêng lẽ hoặc có thể chạy một chuỗi lệnh gồm nhiều lệnh
5. Kiểm tra mỗi lệnh thực hiện trong một chu kỳ clock và giá trị các thanh ghi sau khi thực hiện mỗi câu lệnh.
6. Hiện thực thiết kế và kiểm tra kết quả mô phỏng.

Bảng sự thật cho tín hiệu điều khiển:

Opcode	Lệnh	ALUOp [1:0]	Function	ALUcontrol [3:0]	RegDst	MemRead	MemWrite	MemToReg	ALUSrc	RegWrite	Opcode decimal
000000	add (R)	10	100000	0010	1	0	0	0	0	1	0
101011	sw (I)	00	x	0010	X	0	1	X	1	0	43
100011	lw (I)	00	x	0010	0	1	0	1	1	1	35
001000	addi (I)	00	x	0010	0	0	0	0	1	1	8
000000	slt (R)	10	101010	0011	1	0	0	0	0	1	0

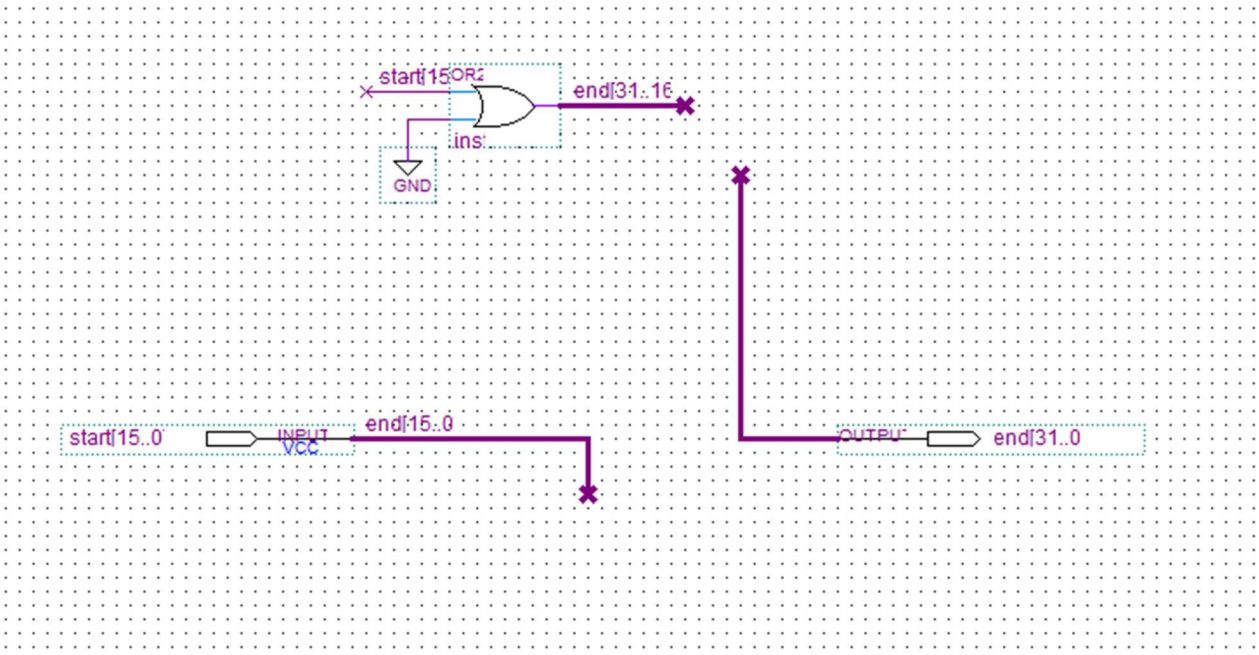
Khối Register:

Vì các lệnh dùng tối đa 5 thanh ghi nên khối Register chỉ tạo 5 thanh ghi 32 bit:

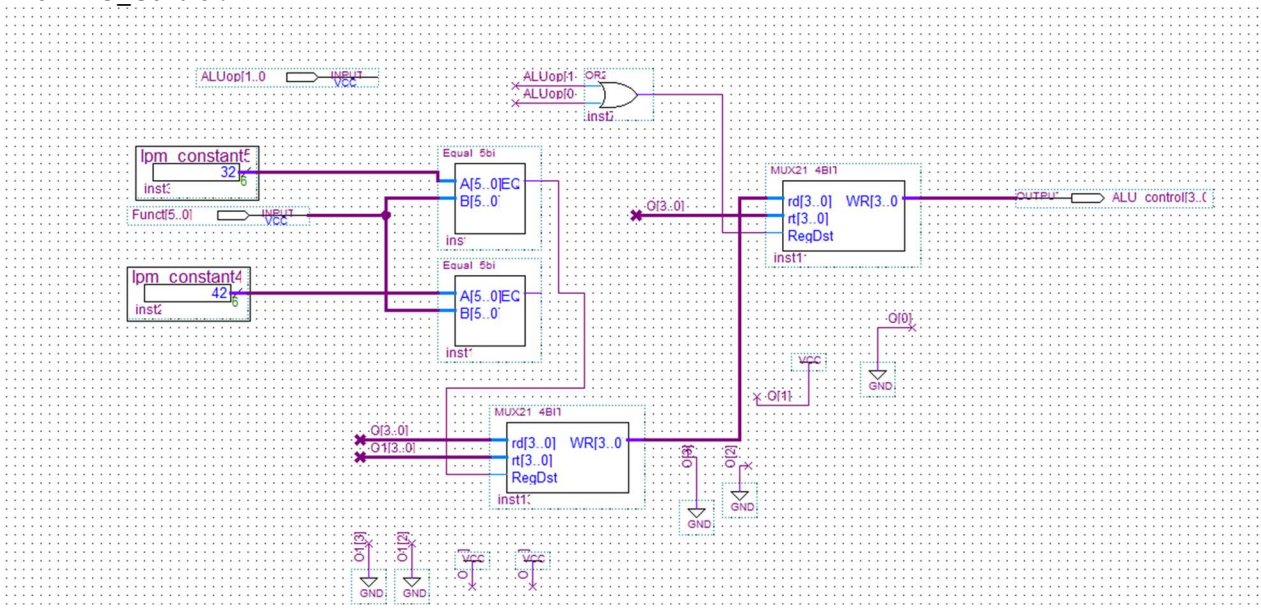


Được nối từ 4 register file 8bits để tạo thành register 32bit, decoder chỉ giải mã 0->5, mặc định $0 sẽ lưu giá trị là 0 và không thể thay đổi. Khối decoder của RD1 và RD2 có RegWrite được nối VCC để đảm bảo luôn đọc được giá trị tại thanh ghi.

Khối sign-extend được thực hiện để mở rộng bit dấu:



Khối ALU_Control:

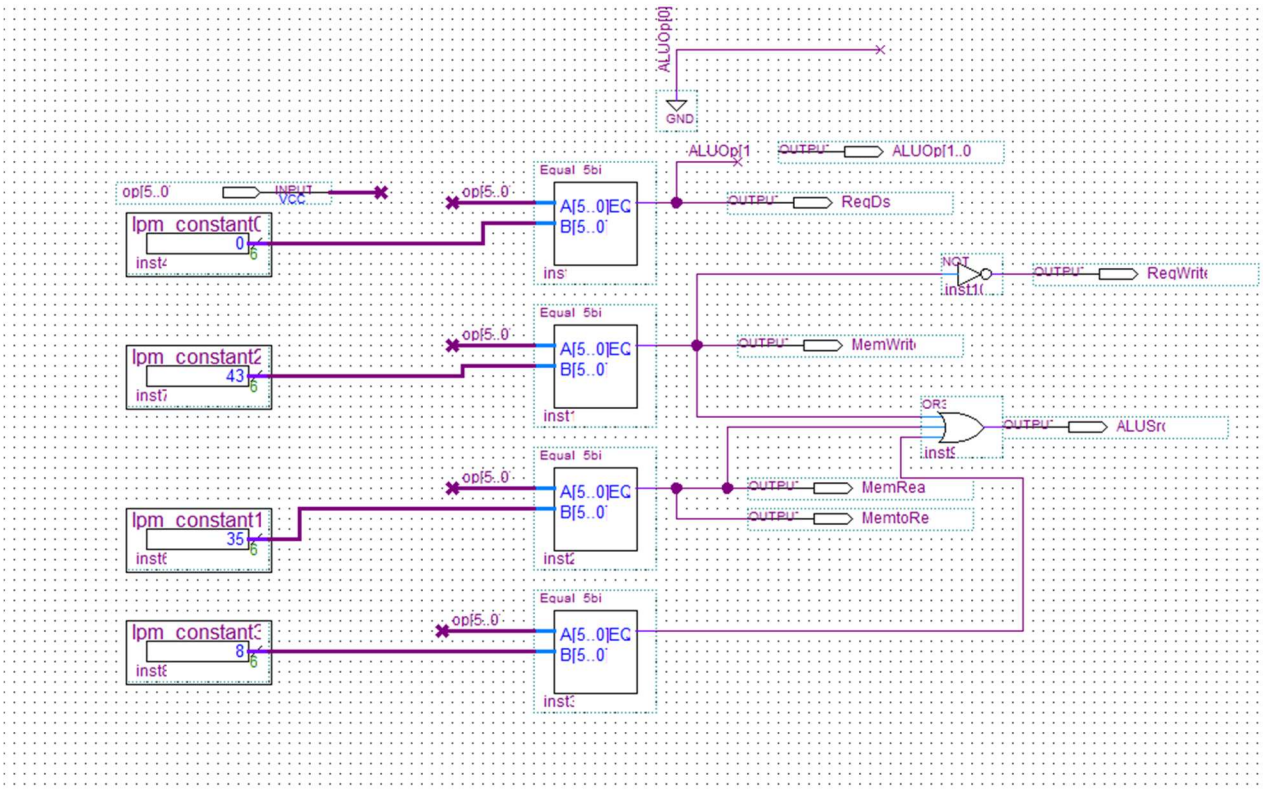


-Vì chỉ có hai dạng lệnh là I type và R type nên không xử lý lệnh branch, mặc định khi ALU_Control nhận ALU_op là 00 của I-type thì sẽ luôn in ra 4bit ALU_control là 0010 để hiện thực phép cộng.

- Khi xử lý lệnh R-type vì ở đây chỉ có 2 lệnh add và slt là R-type nên sẽ dựa vào so sánh 6 bit function của 2 lệnh với giá trị được định sẵn ở dạng decimal để chọn tính hiệu ALU_control đầu ra kết hợp với ALUOp là 10 của R-type để chọn.

ALUOp	Function	ALU_Control
00	x	0010
10	100000	0010
	101010	0011

Khối Control_Unit:



Bảng tính hiệu điều khiển của khối Control_Unit:

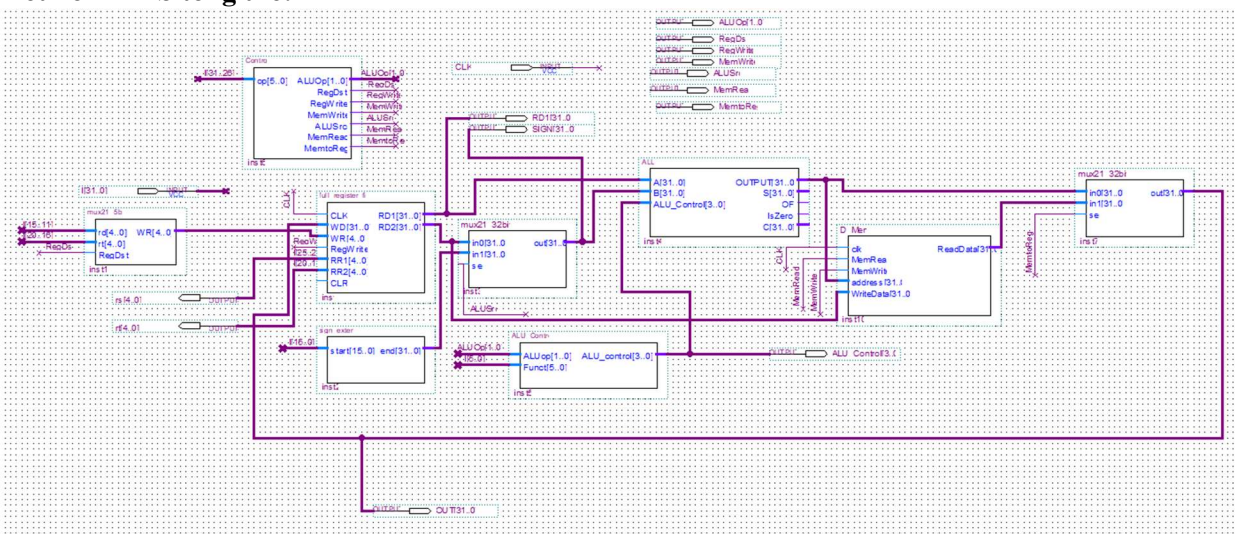
Opcode	Lệnh	ALUOp [1:0]	RegDst	MemRead	MemWrite	MemToReg	ALUSrc	RegWrite	Opcode decimal
000000	add (R)	10	1	0	0	0	0	1	0
101011	sw (I)	00	X	0	1	X	1	0	43
100011	lw (I)	00	0	1	0	1	1	1	35
001000	addi (I)	00	0	0	0	0	1	1	8
000000	slt (R)	10	1	0	0	0	0	1	0

Vì các opcode của các lệnh là khác nhau nên sẽ thực hiện 1 bộ so sánh Equal để so sánh giá trị của 6bit opcode đầu vào với giá trị Decimal được định sẵn, nếu kết quả so sánh bằng nhau thì đầu ra của khối Equal sẽ là 1.

- -RegDst= add+ slt= add
- -MemRead = lw
- -MemWrite= sw
- -MemToReg= lw
- -ALUSrc= sw + lw + addi
- -RegWrite= sw’

ALUOp chỉ có ALUop[0]= 0 nên mặc đinh nối GND cho tín hiệu của ALUOp[0] và ALUOp[1]=add

Kết nối MIPS tổng thể:



Các giá trị Output kiểm tra:

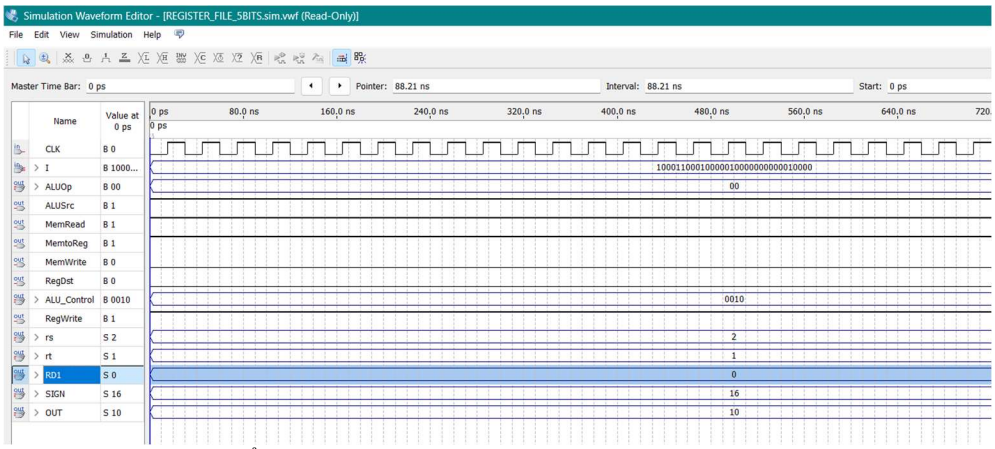
- ALUOp[1..0] : 2bit từ control vào ALUControl
- ALUSrc : bit chọn đầu vào cho mux2-32bit vào ALU của RR2
- MemRead : tín hiệu đọc D-Mem
- MemtoReg: tín hiệu chọn đầu ra từ ALU hoặc Dmem cho WD
- MemWrite : ghi vào Dmem
- RegDst: chọn rt hoặc rd dành cho phân biệt R-type và I-type
- RegWrite: cho phép ghi vào Register file
- ALU_Control: tín hiệu điều khiển khối ALU
- Rs : đầu vào RR1
- Rt: đầu vào RR2
- RD1[31..0]: giá trị được đọc ra từ RR1 vào ALU
- SIGN[31..0]: giá trị thứ 2 và ALU có thể là từ khối sign-extended hoặc RR2 qua cổng MUX
- Out[31..0]: giá trị cuối cùng quay lại khi cần ghi lại vào Register file

Kết quả các lệnh và mô phỏng:

✞ lw $1, 16($2)

Hexa: 8C410010

Giả định giá trị lưu tại Word có địa chỉ là 4 sẽ là 10:
Mã binary: 1000 1100 0100 0001 0000 0000 0001 0000

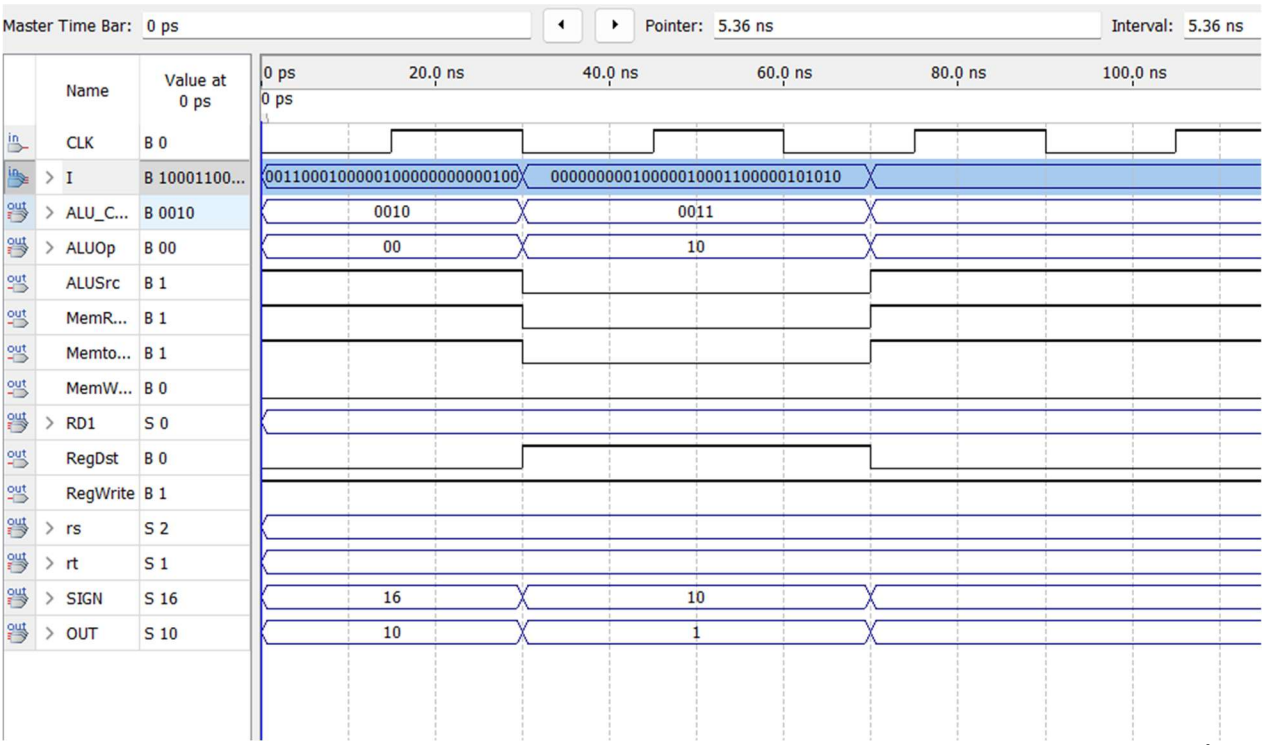


$1 sẽ có giá trị là **10** hiển thị ở OUT sau khi thực thi lệnh.

- slt $3, $2, $1 :

  mã binary: 0000 0000 0100 0001 0001 1000 0010 1010

  hexa:0x0041182A



Vì $2 chưa được nạp giá trị nên sẽ lưu giá trị là 0 nên bé hơn $1 đang lưu giá trị là 10 nên Out là 1 sẽ được lưu về cho $3

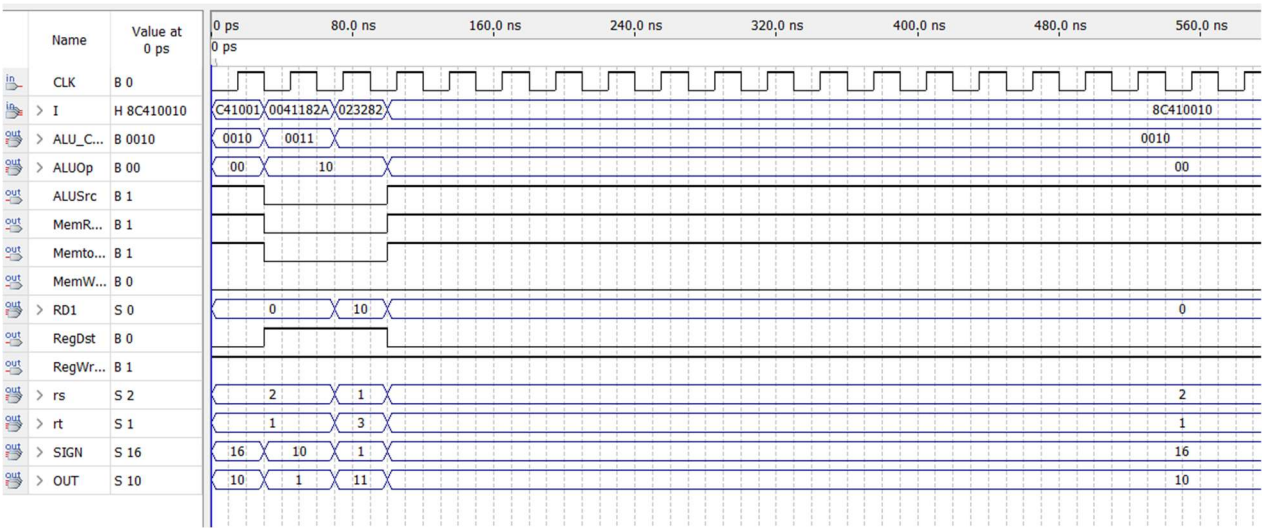
$3 sẽ có giá trị là **1** sau khi thực thi lệnh.

- add $1, $2, $3

sẽ chỉnh sửa lại thành add $5,$1,$3 để kiểm tra lưu giá trị của $1 và $3:

add $5, $1, $3 mã binary: 0000 0000 0010 0011 0010 1000 0010 0000

hexa: 0x00232820

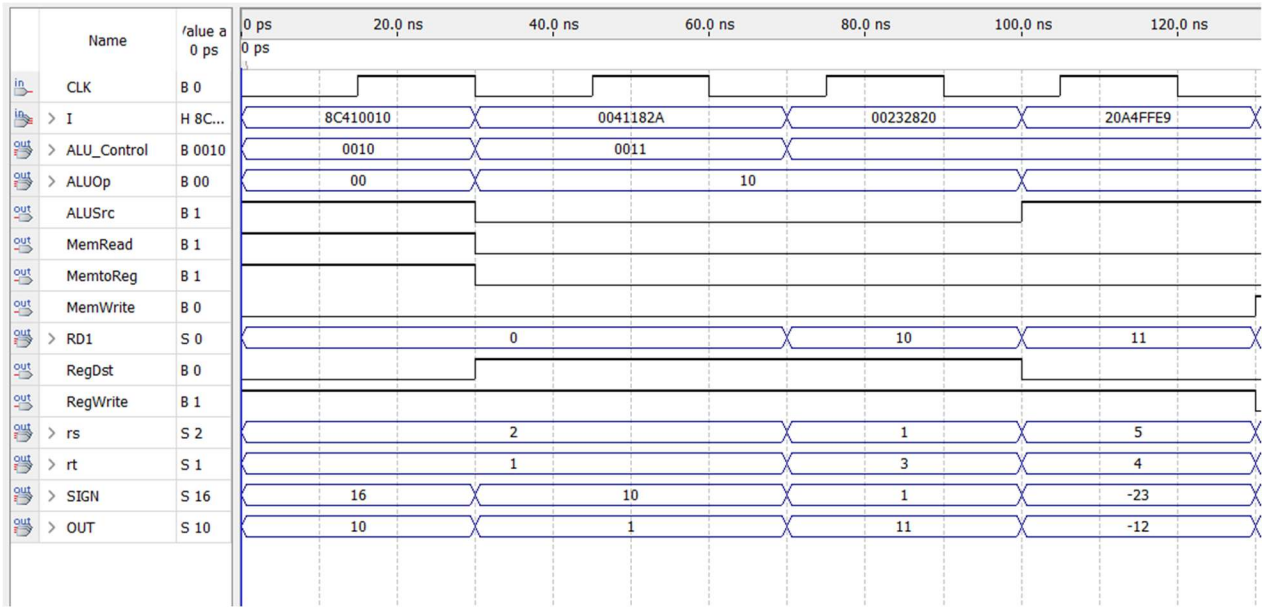


Giá trị của **$5** là **11** hiện ở OUT sau khi thực thi lệnh

- addi $4, $5, -23

  addi $4, $5, -23 mã binary: 0010 0000 1010 0100 1111 1111 1110 1001

  hexa: 0x20A4FFE9

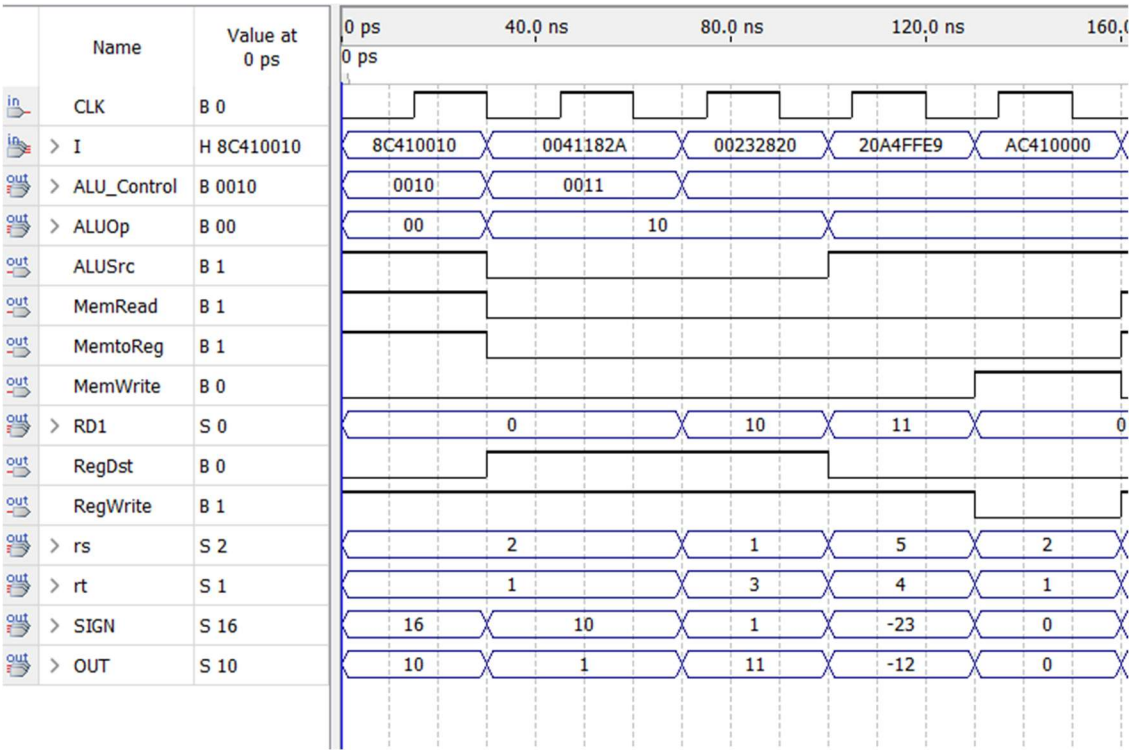


$4 lưu giá trị là **-12** sau khi thực hiện lệnh

- sw $1, 0($2)

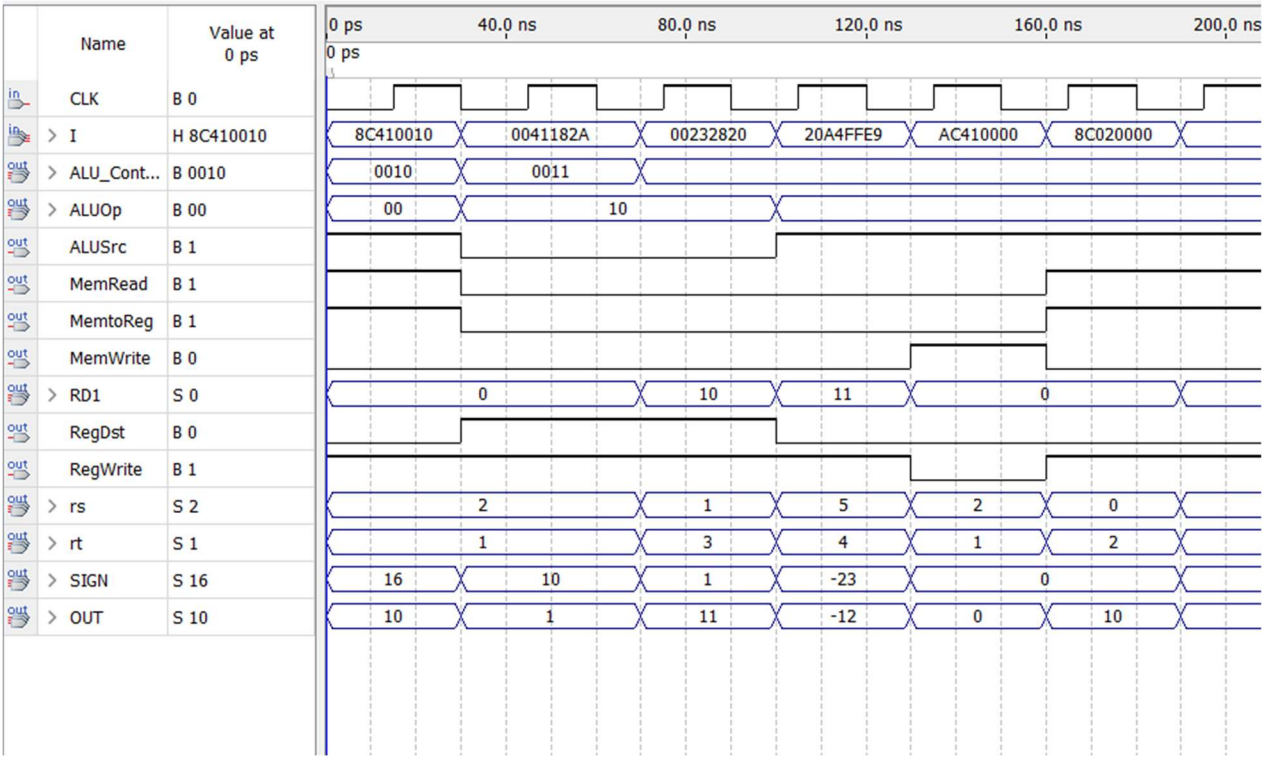
  mã binary: 1010 1100 0100 0001 0000 0000 0000 0000

  Hex: 0xAC410000



Kiểm tra word có địa chỉ là 0 thì có lưu giá trị của **$1=10** vào hay không bằng lệnh :

lw $2, 0($0): 1000 1100 0000 0010 0000 0000 0000 0000
Hex: 0x8C020000



Ta thấy **OUT= 10** là **$2** đã lấy giá trị tại word có địa chỉ tại 0x0 là 10. Vậy sw thành công.