ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH LAB4

THIẾT KẾ SINGLE CYCLE RISC-V PROCESSOR

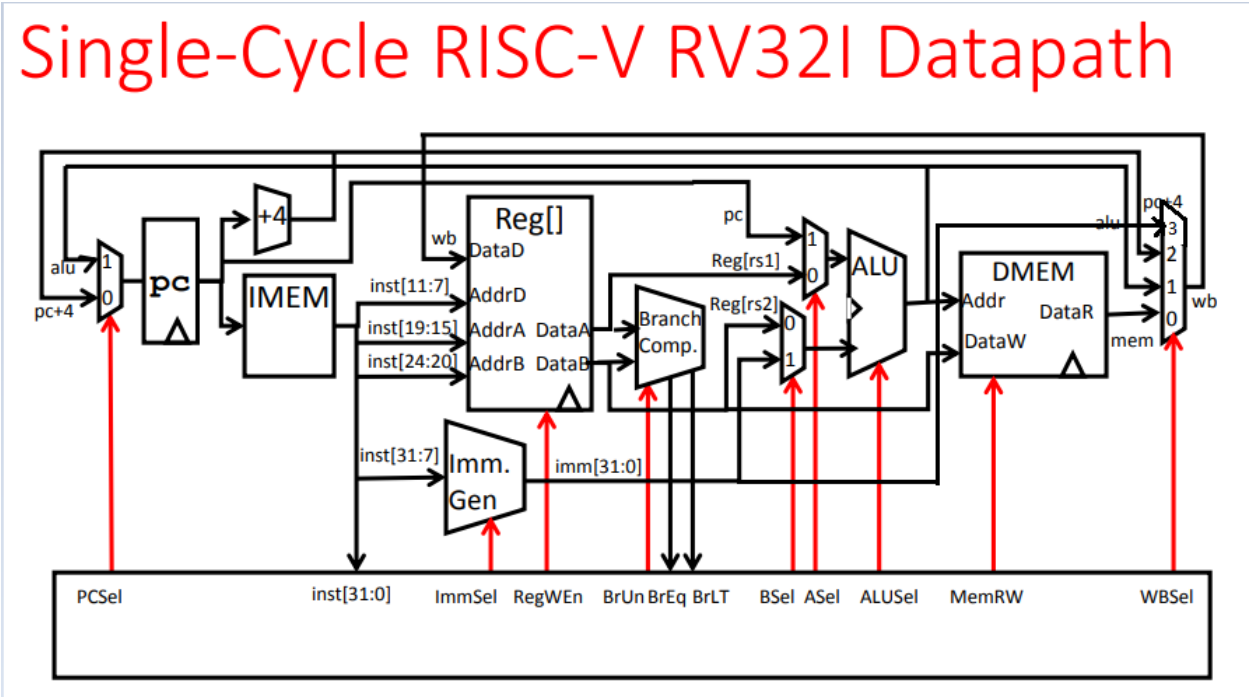
Họ và tên: Phan Quốc Linh

MSSV: 18520993

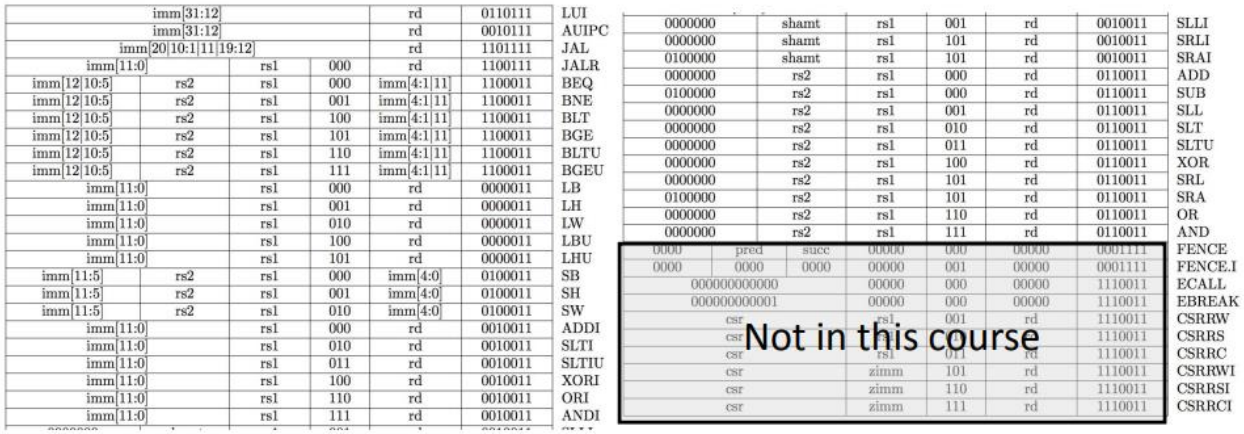
Lớp: CE409.M11

MỤC LỤC

1. TỔNG QUAN THIẾT KẾ



Hình 1: Kiến trúc của Processor.



Hình 2: Số loại lệnh có thể thực hiện được dựa theo kiến trúc ở hình 1.

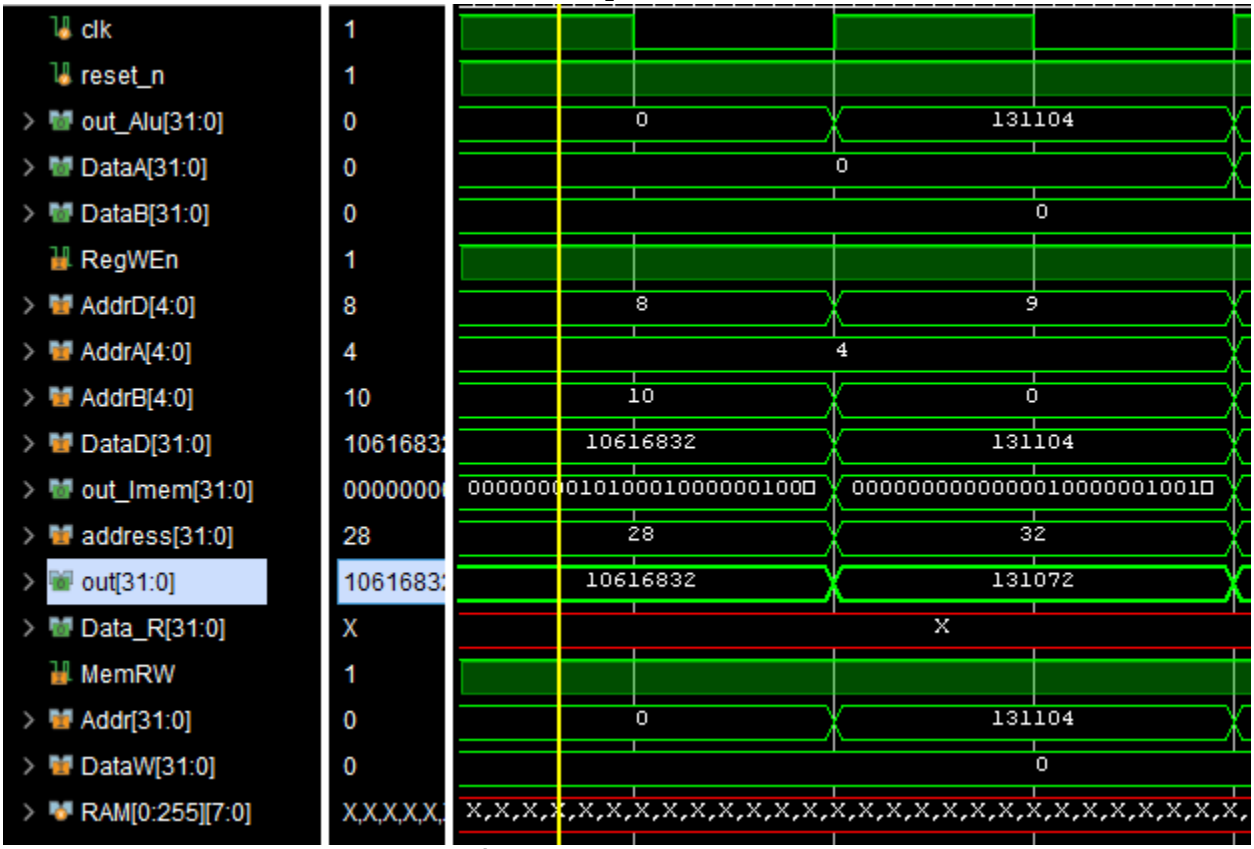
2. GIẢI THÍCH CÁC LOẠI CÂU LỆNH VÀ KẾT QUẢ CHẠY MÔ PHỎNG

Tập lệnh của em được lưu trong file Instruction.txt để đưa vào Imem. Trong file text.txt em có giải thích các câu lệnh của mình (dịch từ các bit 0 1 sang mã giả).

*** Lệnh LUI và AUIPC thuộc loại U-type.** Lệnh LUI sẽ lưu U-imm (imm[31:12]) vào 20 bit cao nhất của thanh ghi rd, 12 bit thấp tiếp theo sẽ bằng 0. Còn lệnh AUIPC sẽ cộng offset của mình với 20 bit cao là U-

imm(imm[31:12]) và 12 bit 0 với giá trị pc hiện tại để lưu vào rd. Dưới đây là kết quả chạy 2 lệnh trong tập lệnh Instruction:

00000000101000100000_01000_0110111 // pc=28// LUI r8, 10616832
00000000000000100000_01001_0010111 // pc=32// AUIP r9, 131072



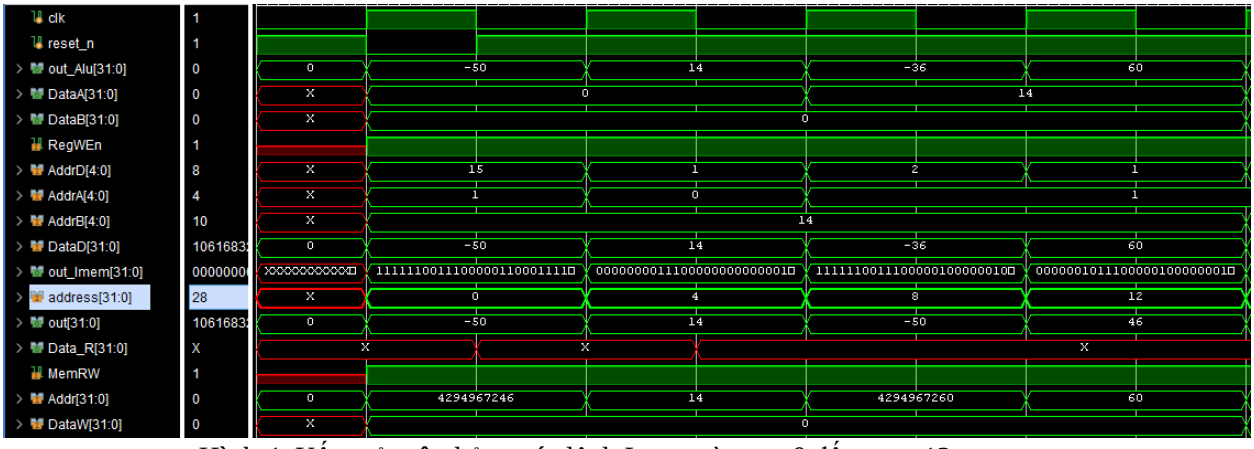
Hình 3: Kết quả lệnh LUI và AUIPC.

Tại pc = address[31:0] = 28 giá trị DataD lưu và thanh ghi r8 có giá trị bằng offset = 1061832. Còn tại pc = 32, giá trị DataD lưa vào r9 có giá trị bằng offset = 131072 + 32.

* Các lệnh I-type.

Theo kiến trúc Datapath trong hình 1 thì có thể tính toán hết các lệnh I-type. Các lệnh I-type có trong Instruction:

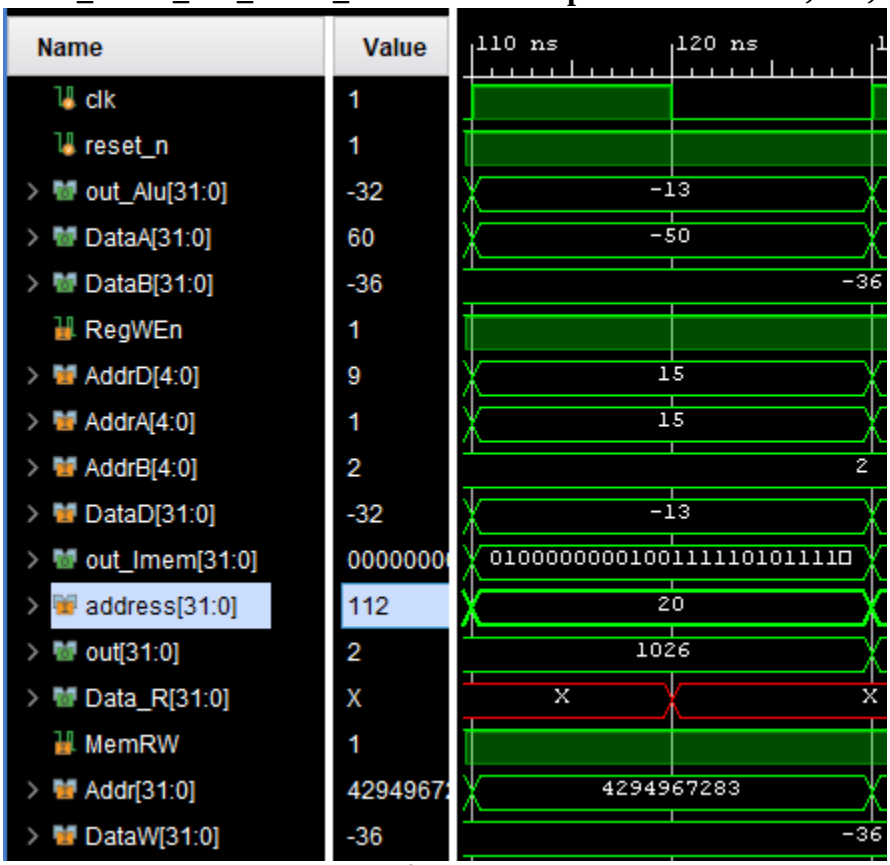
111111001110_00001_100_01111_0010011 // pc=0// XORI r15, r0, -50
000000001110_00000_000_00001_0010011 // pc=4// ADDI r1, r0, 14
111111001110_00001_000_00010_0010011 // pc=8// ADDI r2, r1, -50
000000101110_00001_000_00001_0010011 // pc=12// ADDI r1, r1, 46



Hình 4: Kết quả mô phỏng các lệnh I-type từ pc = 0 đến pc = 12.

Instruction:

0100000_00010_01111_101_01111_0010011 // pc=20// SRAI r15, r15, 2



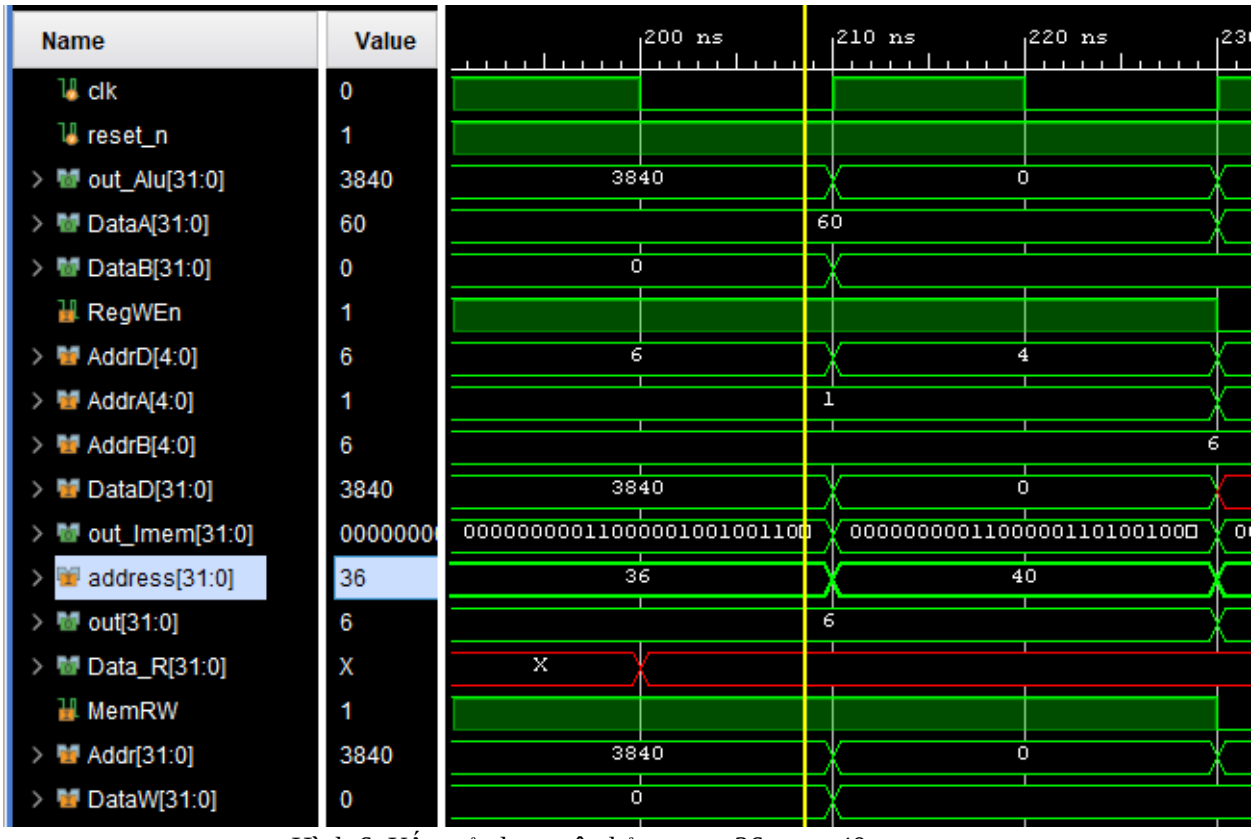
Hình 5: Kết quả lệnh SRAI

Lệnh SRAI nghĩa là dịch phải số học ở đây r15 = DataA = -50, dịch phải số học cho 2 nên DataD = -13.

Instruction:

000000000110_00001_001_00110_0010011 // pc=36// SLLI r6, r1, 6

000000000110_00001_101_00100_0010011 // pc=40// SRLI r4, r1, 6



Hình 6: Kết quả chạy mô phỏng pc = 36, pc = 40

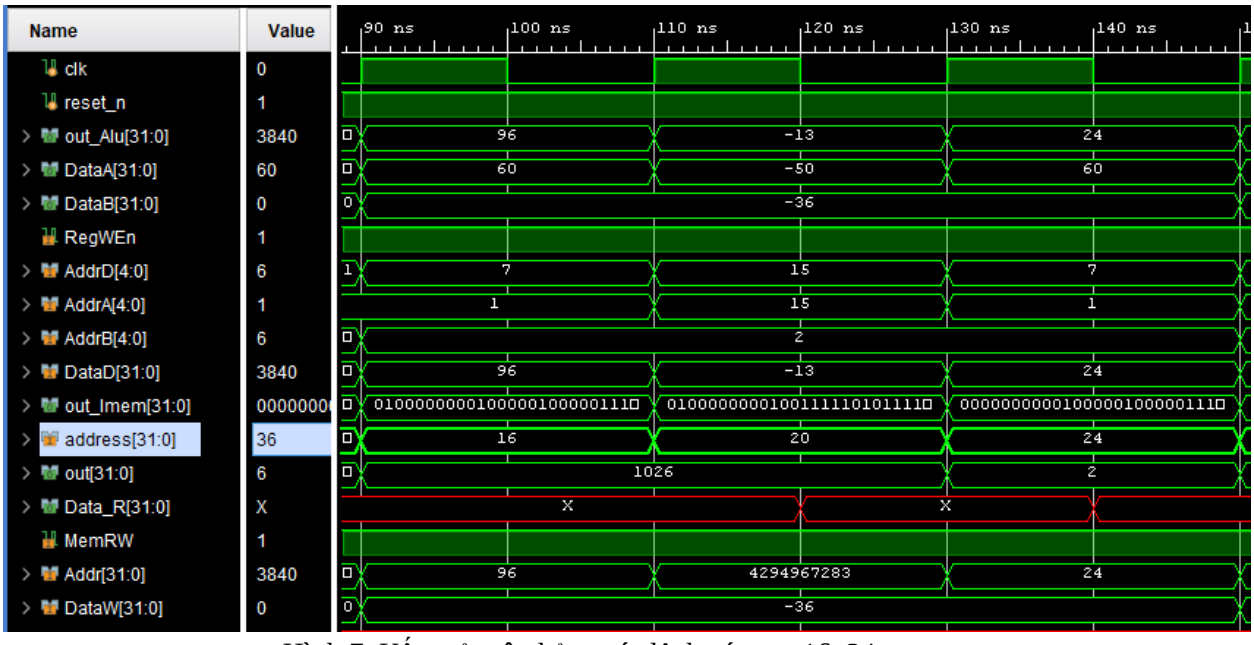
Lệnh SLLI nghĩa là dịch trái logic ở đây r1 = DataA = 60, dịch trái 6 nên DataD = 60*2^6 = 3840. Còn lệnh SRLI là dịch phải logic. 60 = ‘b0011100, nên khi dịch phải cho 6 = 0.

*** Các lệnh R-type**

Theo kiến trúc Datapath trong hình 1 thì có thể tính toán hết các lệnh R-type. Các lệnh R-type có trong Instruction:

0100000_00010_00001_000_00111_0110011 // pc=16// SUB r7. r1. r2

0000000_00010_00001_000_00111_0110011 // pc=24// ADD r7, r1, r2



Hình 7: Kết quả mô phỏng các lệnh có pc = 16, 24

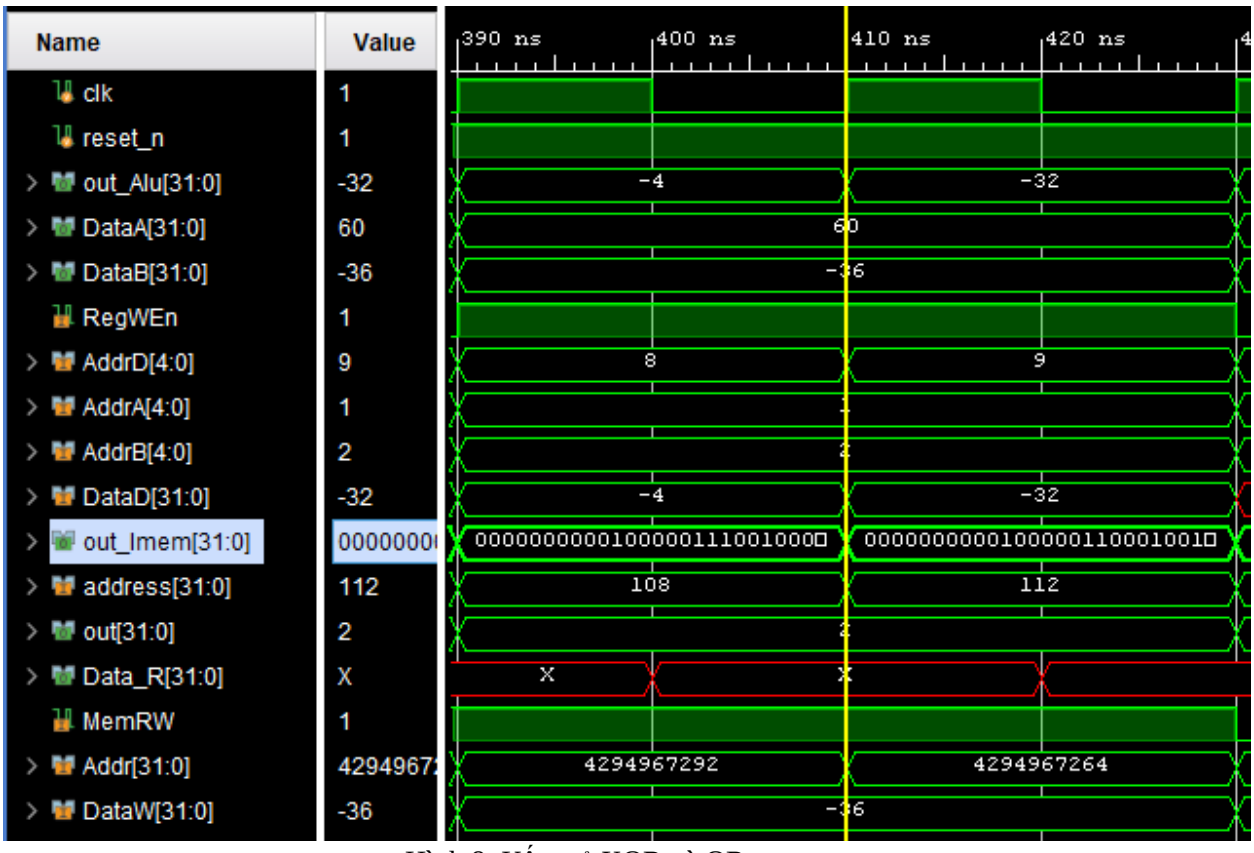
Tại pc = 16 là lênh SUB, lấy r1 trừ r2, tương đương với DataA = 60, DataB = -36, nên khi trừ thì out_Alu = 96.

Tại pc = 24 là lệnh ADD, lấy r1 cộng r2, tương đương với lấy 60 + -36 = 24.

Instruction:

0000000_00010_00001_110_01000_0110011 // pc=108// OR r8, r1, r2

0000000_00010_00001_100_01001_0110011 // pc=112// XOR r9, r1, r2

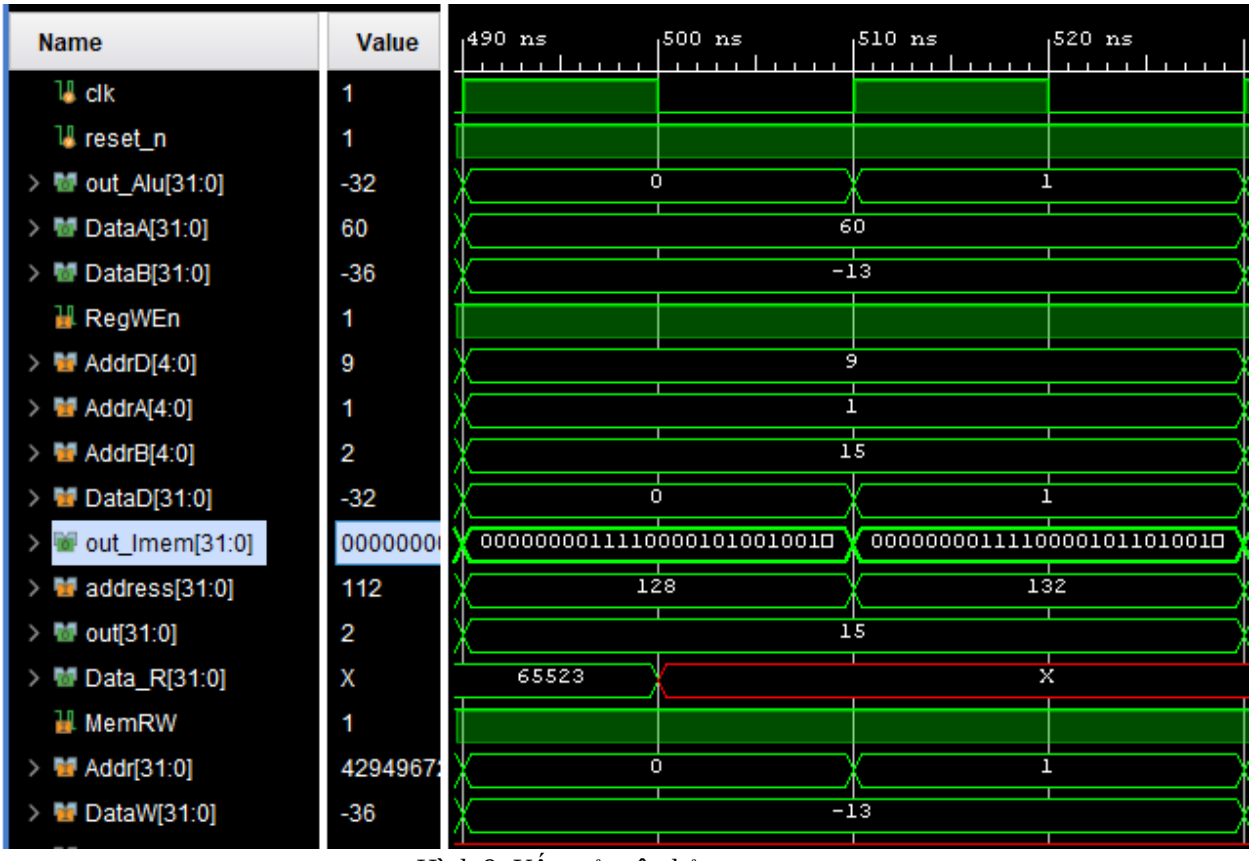


Hình 8: Kết quả XOR và OR.

Instruction:

0000000_01111_00001_010_01001_0110011 // pc=128// SLT r9, r1, r15

0000000_01111_00001_011_01001_0110011 // pc=132// SLTU r9, r1, r15



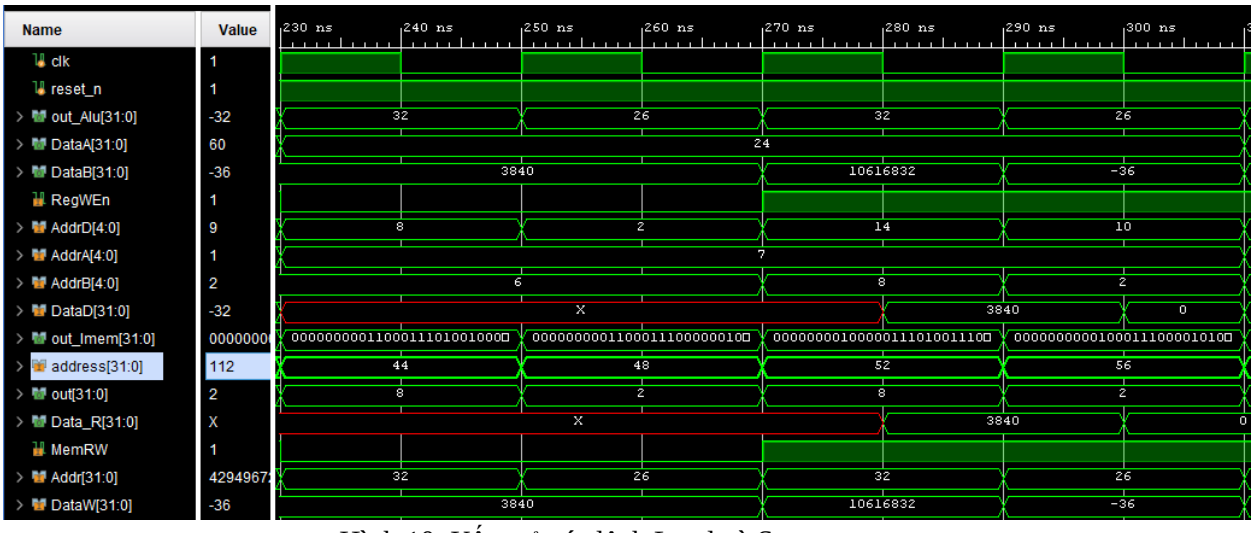
Hình 9: Kết quả mô phỏng.

Lệnh SLT là lệnh so sánh signed bit, nếu r1 < r15 thì xuất ra 1 còn không thì xuất ra 0. Ở đây r1 = 60, r15 = -13 nên r1 > r15 nên out_Alu = 0.
Lệnh SLTU là lệnh so sánh unsigned bit, nên r1 < r15 nên out_Alu = 1.

*Lệnh Store và Load

Lệnh Store là lệnh S-type. Còn các lệnh Load thuộc lọa I-type.
Các lệnh Store và Load trong tập Instruction:

0000000_00110_00111_010_01000_0100011 // pc=44// SW r6, 8(r7)
0000000_00110_00111_000_00010_0100011 // pc=48// SB r6, 2(r7)
000000001000_00111_010_01110_0000011 // pc=52// LW r14, 8(r7)
000000000010_00111_000_01010_0000011 // pc=56// LB r10, 2(r7)



Hình 10: Kết quả các lệnh Load và Store

Tại pc = 44, lênh SW ta store thanh ghi r6 = 3840 vào địa chỉ bắt đầu là 8(r7) = 8 + r7 = 8 + 24 = 32. Vì Dmem được lưu dưới dạng 1 byte, nên khi store 1 word thì ta cần 4 byte, nên thanh ghi r6 được lưu vào 4 vị trí lần lượt là 32, 33, 34, 35.

Tại pc = 48, lệnh SB ta store 8 bit cuối của thanh ghi r6 = 8'b00000000 vào địa chỉ 8(r7) = 2 + r7 = 2 + 24 = 26.

Tại pc = 52, lệnh LW ta load giá trị 1 word tại địa chỉ bắt đầu là 32 vào thanh ghi r14, nên DataD = 3840.

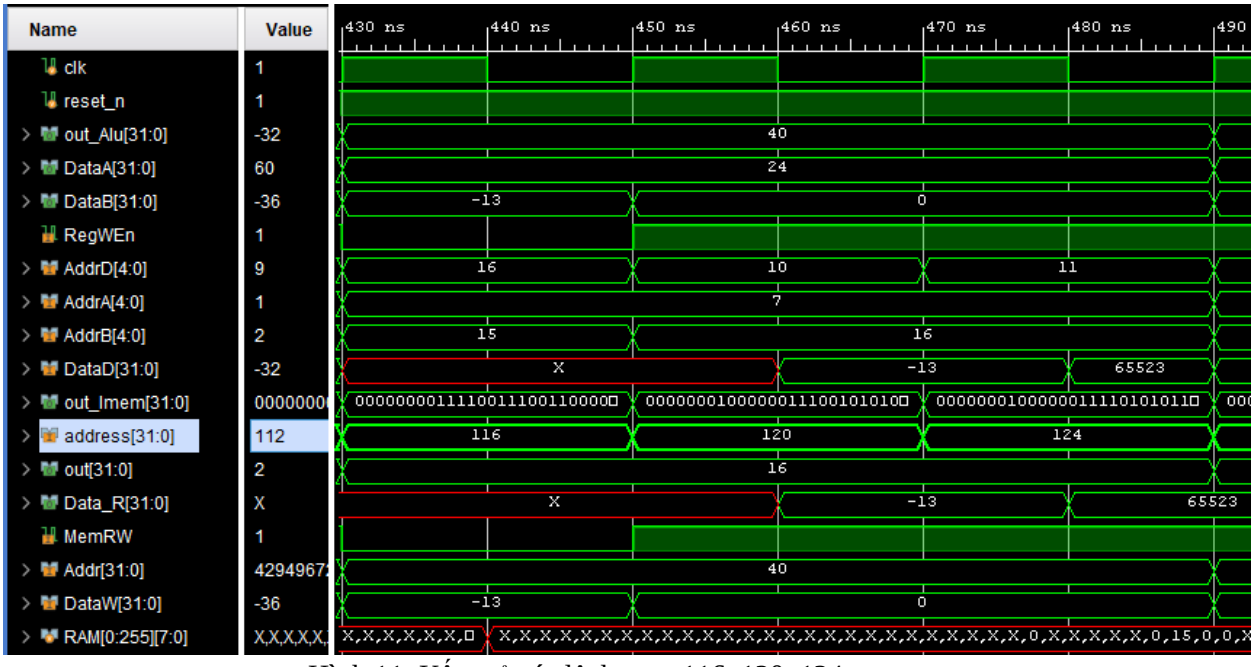
Tại pc = 56, lệnh LB ta load giá trị 1 byte tại địa chỉ là 26 vào thanh ghi r10, nên DataD = 0.

Instruction:

0000000_01111_00111_001_10000_0100011 // pc=116// SH r15, 16(r7)

000000010000_00111_001_01010_0000011 // pc=120// LH r10, 16(r7)

000000010000_00111_101_01011_0000011 // pc=124// LHU r11, 16(r7)



Hình 11: Kết quả các lệnh pc = 116, 120, 124.

Tại pc = 116, lệnh SH ta store 1/2 word = 2 byte cuối của thanh ghi r15 = -13 vào địa chỉ bắt đầu là 16(r7) = 16+ r7 = 15 + 24 = 40. Vì Dmem được lưu dưới dạng 1 byte, nên khi store 2 byte thì ta cần lưu vào 2 vị trí lần lượt là 40, 41.

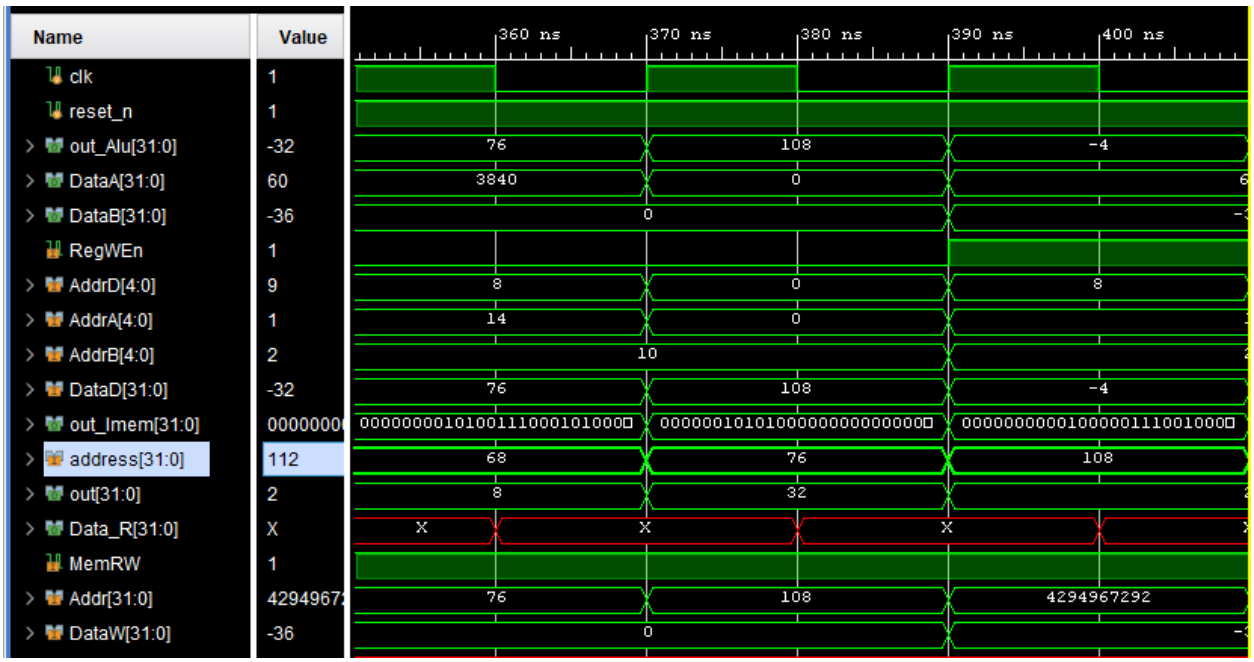
Tại pc = 120, lệnh LH ta load giá trị 1/2 word = 2 byte tại địa chỉ bắt đầu là 40 trong Dmem vào thanh ghi r10 với 2 byte đầu lấy từ Dmem và 2 byte cao được signed extend, nên DataD = -13.

Tại pc = 120, lệnh LHU ta load giá trị 1/2 word = 2 byte tại địa chỉ bắt đầu là 40 trong Dmem vào thanh ghi r10 với 2 byte đầu lấy từ Dmem và 2 byte cao là 16'b0, nên DataD = 65523.

* Các lệnh B-type (các lệnh nhảy)

Instruction:

0000000_01010_01110_001_01000_1100011 // pc=68// BNE r14, r10, 8
111111001110_00001_100_01111_0010011 // pc=72// XORI r15, r0, -50
0000001_01010_00000_000_00000_1100011 // pc=76// BEQ r0, r10, 32



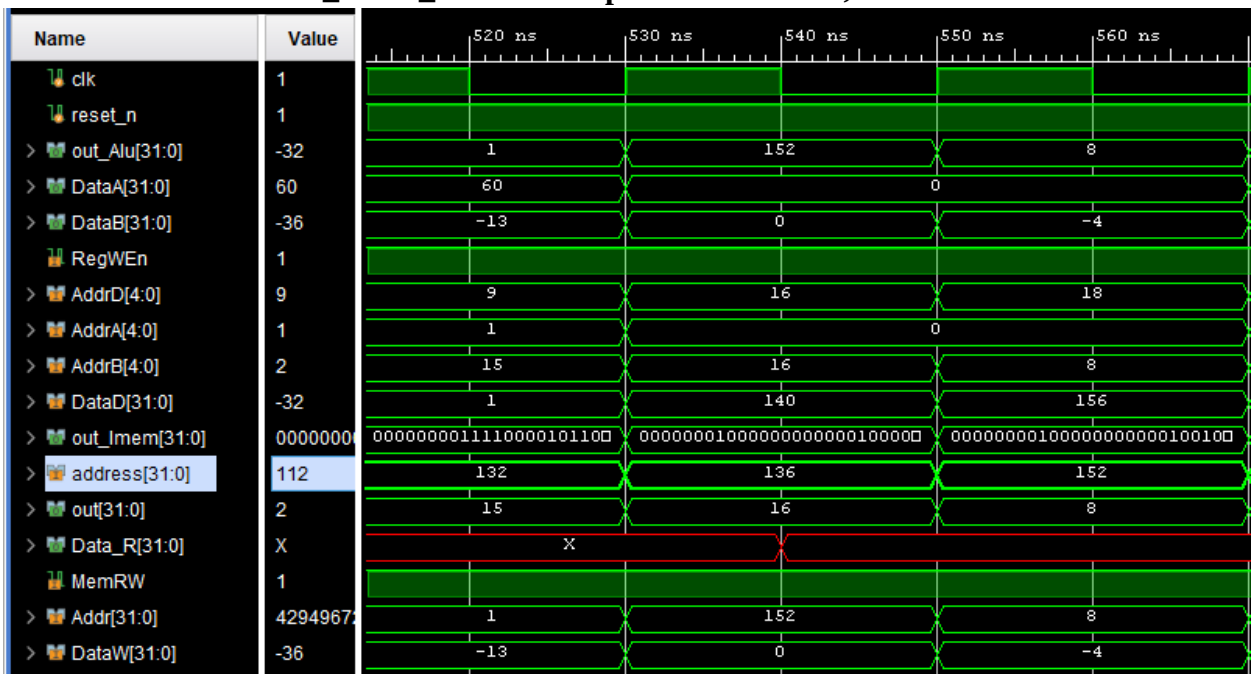
Hình 12: Các lệnh nhảy BEQ và BNE.

Tại pc = 68, lệnh BNE so sánh giá trị 2 thanh ghi r14 = 3840, r10 = 0, 2 thanh ghi có giá trị khác nhau nên pc_next = pc + offset = 68 + 8 = 76.
Tại pc = 76, lệnh BNE so sánh giá trị 2 thanh ghi r0 = 0, r10 = 0, 2 thanh ghi có giá trị bằng nhau nên pc_next = pc + offset = 76 + 32 = 108.

* Lệnh JAL (J-type)

Instruction:

00000001000000000000_10000_1101111 // pc=136// JAL r16, 16



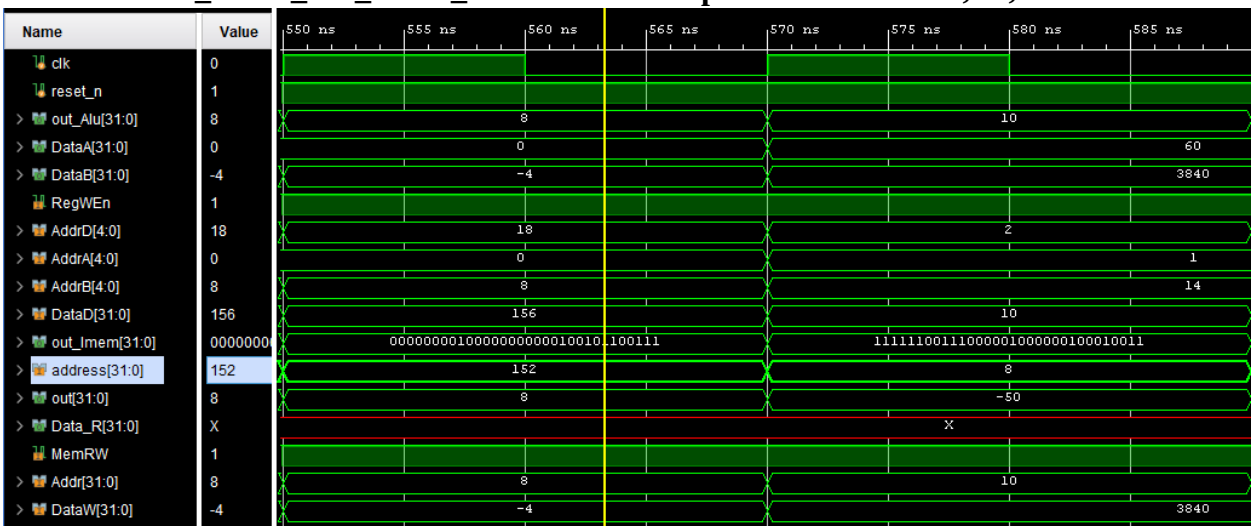
Hình 13: Mô phỏng lệnh JAL

Tại pc = 136, lệnh JAL sẽ lấy kết quả pc + 4 = 140 lưu vào thanh ghi r16. Và pc_next = pc + offset = 136 + 16 = 152.

* Lệnh JALR (thuộc I-type)

Instruction:

000000001000_00000_000_10010_1100111 // pc=152// JALR r18, r0, 8



Hình 14: Mô phỏng lệnh JALR.

Tại pc = 152, lệnh JALR lấy giá trị pc + 4 = 152 + 4 = 156 lưu vào r18. Và pc_next = pc + r0 = 8 + 0 = 8. Nên lệnh tiếp theo thực hiện có giá trị pc = 8.

----- HẾT -----