**Họ và tên: Phan Quốc Linh**

**MSSV: 18520993**

**Lớp: CE409.M11**

1



ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

**BÁO CÁO THỰC HÀNH LAB5**

**THIẾT KẾ SINGLE CYCLE RISC-V PROCESSOR**

**MỤC LỤC**

[**1.** **TỔNG QUAN THIẾT KẾ** 3](#_Toc87739572)

[**2.** **GIẢI THÍCH CÁC LOẠI CÂU LỆNH VÀ KẾT QUẢ CHẠY MÔ PHỎNG** 6](#_Toc87739573)

[**\* Lệnh LUI và AUIPC thuộc loại U-type.** 6](#_Toc87739574)

[**\* Các lệnh I-type.** 7](#_Toc87739575)

[**\* Các lệnh R-type** 8](#_Toc87739576)

[**\*Lệnh Store và Load** 10](#_Toc87739577)

[**\* Các lệnh B-type (các lệnh nhảy)** 12](#_Toc87739578)

[**\* Lệnh JAL (J-type)** 13](#_Toc87739579)

[**\* Lệnh JALR (thuộc I-type)** 14](#_Toc87739580)

# **TỔNG QUAN THIẾT KẾ**

Diagram, schematic

Description automatically generated

Hình 1: Kiến trúc của Processor.

Table

Description automatically generated

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

**\* Khối Imem:**

- Sẽ có 1 con rom để lưu các Instruction được đọc từ file Instruction.txt. Sau khi có input pc, thì giá trị của pc << 2 để lấy Instruction.

**\* Khối Reg[]**

- Là 1 register file dùng để lưu các giá trị của các thanh ghi từ r0 = 0, đến r31.

**\*Khối ImmGen**



- Là khối mở rộng bit cho các lệnh dùng đến offset.

- ImmSel = 0 => dùng để mở rộng bit cho các lệnh S-type.

- ImmSel = 1 => dùng để mở rộng bit cho các lệnh I-type.

- ImmSel = 2 => dùng để mở rộng bit cho các lệnh B-type.

- ImmSel = 3 => dùng để mở rộng bit cho lệnh JAL (J-type).

- ImmSel = 4 => dùng để mở rộng bit cho các lệnh U-type (LUI, AUIPC).

**\* Khối BranchComp:**

- Dùng để xử lý cho các lệnh nhảy B-type.

Text

Description automatically generated

**\* Khối ALU:**



- Khối ALU dùng để thực hiện các phép toán ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND được điều khiển bằng chân ALUSel.

**\* Khối Dmem:**



- MemRW dùng để điều khiển Read và Write. MemRW = 1 là Read, MemRW = 0 là Write.

- Tín hiệu funct3[2:0] dùng để điều khiển xem là lệnh này thuộc loại nào trong các lệnh LB, LH, LW, LBU, LHU khi mà MemRW = 1 (ở chế độ Read). Nếu MemRW = 0, thì funct3 dùng để điều khiển các lệnh SB, SH, SW.

**\* Khối Controller**



- Nhận input là 32bit Instruction, từ đó điều khiển các tín hiệu ouput để xử lý tính toán trong Datapath. Hai tín hiệu BrEq và BrLt dùng để điều khiển các lệnh nhảy thuộc B-type.

# **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**

Graphical user interface, timeline

Description automatically generated

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**

A screenshot of a computer

Description automatically generated with medium confidence

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**

Graphical user interface

Description automatically generated

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**

A picture containing graphical user interface

Description automatically generated

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**

Timeline

Description automatically generated

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**

A picture containing timeline

Description automatically generated

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**

Timeline

Description automatically generated

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)**

Schematic

Description automatically generated with low confidence

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)**

A picture containing timeline

Description automatically generated

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

Tại pc = 116, lênh 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**

Timeline

Description automatically generated

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**

A picture containing timeline

Description automatically generated

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**

A picture containing schematic

Description automatically generated

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 -----**