ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO
ĐỒ ÁN THIẾT KẾ LUẬN LÝ (CO3091)

Thiết kế RISC CPU đơn giản

Giảng viên hướng dẫn: KS. Huỳnh Phúc Nghị

Nhóm 08 - HK241

Báo cáo cuối kỳ - Ngày nộp: 21/12/2024

STT	Họ và tên	MSSV	Ghi chú
1	**Lâm Nữ Uyển Nhi**	**2212429**	**Nhóm trưởng**
2	Hồ Đăng Khoa	2211588	
3	Võ Huy Quang	2212761	
4	Võ Minh Quân	2212826	
5	Lê Phan Bảo Như	2212466	

Thành phố Hồ Chí Minh, tháng 12 năm 2024

Mục lục

1 Giới thiệu

Mô tả: Bao gồm các nội dung về giới thiệu đề tài, công cụ sử dụng, thiết bị sử dụng, các chức năng của sản phẩm.

Bộ xử lý trung tâm (CPU) là thành phần cốt lõi của mọi hệ thống máy tính, chịu trách nhiệm thực thi các lệnh và điều khiển hoạt động của toàn hệ thống. Trong lĩnh vực thiết kế vi xử lý, kiến trúc RISC (Reduced Instruction Set Computer) nổi bật với tập lệnh đơn giản, ngắn gọn, giúp tăng hiệu suất và tiết kiệm năng lượng. Đề tài "Thiết kế RISC CPU đơn giản" được thực hiện với mục tiêu nghiên cứu, tìm hiểu và xây dựng một bộ xử lý RISC cơ bản, từ đó làm nền tảng cho việc nghiên cứu chuyên sâu hơn về kiến trúc máy tính và thiết kế hệ thống nhúng.

Để hiện thực hóa mục tiêu này, đề tài sử dụng ngôn ngữ mô tả phần cứng HDL để mô tả cấu trúc và hoạt động của RISC CPU. HDL là ngôn ngữ phổ biến trong thiết kế vi mạch số, cho phép mô phỏng và kiểm tra thiết kế trước khi triển khai trên phần cứng thực tế. Bên cạnh đó, các công cụ thiết kế và mô phỏng FPGA như Xilinx Vivado sẽ được sử dụng để tổng hợp mã Verilog, phân tích, và kiểm tra chức năng của CPU. Kit phát triển FPGA Arty-Z7 sẽ là nền tảng phần cứng để triển khai và đánh giá hiệu năng của RISC CPU.

CPU RISC được thiết kế trong đề tài này sẽ bao gồm các thành phần chính như Instruction Memory (IM), Program Counter (PC), Data Memory (DM), Control Unit (CU), Accumulator, Arithmetic Logic Unit (ALU). CPU sẽ thực hiện các chức năng cơ bản như nạp và giải mã lệnh từ IM, thực thi các phép toán số học và logic với ALU, lưu trữ dữ liệu trong DM, điều khiển luồng chương trình bởi CU và giao tiếp với Uart (nằm ngoài CPU) để nạp lệnh. Phạm vi hoạt động của CPU bao gồm tập lệnh đơn giản với các lệnh số học, logic, di chuyển dữ liệu và điều khiển luồng, hoạt động trên dữ liệu 8-bit và không gian địa chỉ 5-bit.

Đề tài "Thiết kế RISC CPU đơn giản" không chỉ giúp người thực hiện nắm vững kiến thức về kiến trúc RISC mà còn cung cấp kinh nghiệm thực tế trong việc sử dụng ngôn ngữ HDL, và công cụ thiết kế FPGA.

2 Danh sách công việc và tiến độ

Những việc cần làm:

- Xác định yêu cầu đề bài và đặc tả.
- Xác định tập lệnh của CPU.
- Xác định kỹ thuật thiết kế CPU.
- Thiết kế CPU.
- Tạo testcase kiểm thử tính đúng đắn của kiến trúc so với đặc tả.
- Giải quyết vấn đề hazard (nếu có).
- Tiến hành dùng verilog mô tả CPU.
- Chạy mô phỏng và kiểm thử theo testcase đã tạo.
- Tổng hợp luận lý.
- Nạp xuống FPGA và kiểm thử quá trình chạy.

Hiện tại, nhóm đã hoàn thành tất cả công việc trên.

3 Cơ sở lý thuyết

Mô tả: Các kiến thức nền cần có để có thể phân tích được đề, lên ý tưởng và hiện thực ý tưởng đó

3.1 Đại số Boolean và cổng luận lý

Bảng chân trị của các cổng logic cơ bản:

Cổng AND

Hình 1: Hình ảnh cổng AND

B	A	x = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Hình 2: Bảng chân trị cổng AND

Cổng OR

Hình 3: Hình ảnh cổng OR

B	A	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Hình 4: Bảng chân trị cổng OR

Cổng NOT:

Hình 5: Hình ảnh cổng NOT

A	x= $\overline{A}$
0	1
1	0

Hình 6: Bảng chân trị cổng NOT

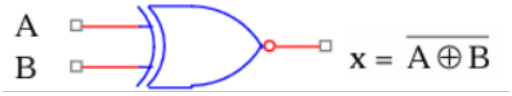
Cổng XOR:

Hình 7: Hình ảnh cổng XOR

A (Input 1)	B (Input 2)	X = A'B + AB'
0	0	0
0	1	1
1	0	1
1	1	0

Hình 8: Bảng chân trị cổng XOR

Cổng XNOR:

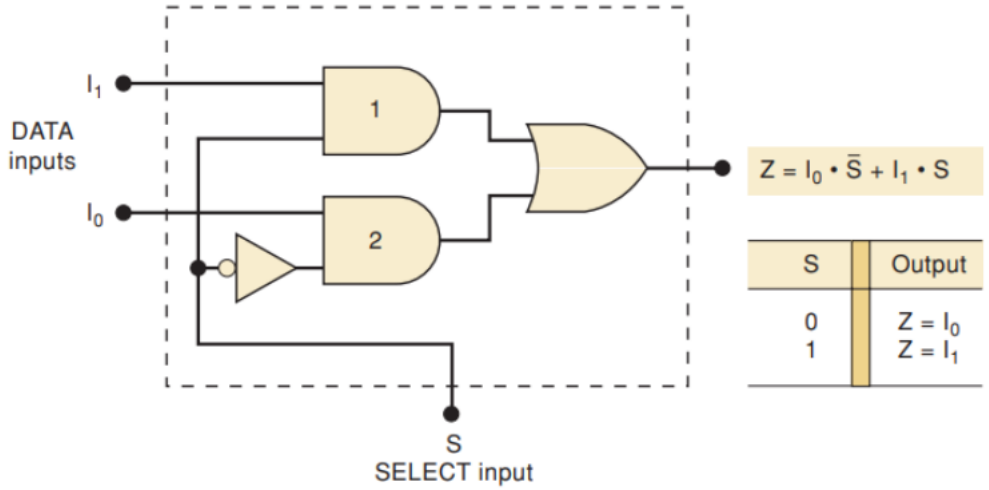


Hình 9: Hình ảnh cổng XNOR

Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Hình 10: Bảng chân trị cổng XNOR

Cổng MUX:



Hình 11: Hình ảnh cổng MUX

3.2 Flip-flop

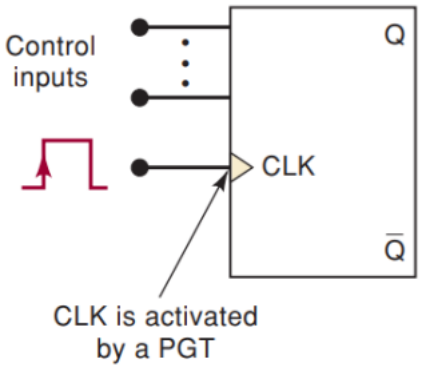
3.2.1 Clocked Flip-flop

- FF được đồng bộ hóa (Clocked FFs) thường có một ngõ vào xung clock thường được ký hiệu là CLK, CK hoặc CP. Chúng ta thường sẽ sử dụng CLK, như được hiển thị trong Hình dưới. Trong hầu hết các FF được đồng bộ hóa, ngõ vào CLK được kích hoạt theo cạnh (edge-triggered), có nghĩa là nó được kích hoạt bởi một sự chuyển đổi tín hiệu; điều này được biểu thị bằng sự hiện diện của một hình tam giác nhỏ trên ngõ vào CLK. Điều này trái ngược với các latch, là loại kích hoạt theo mức (level-triggered).

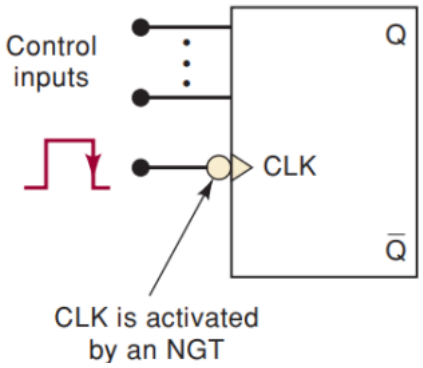
  Hình 12 là một FF với một hình tam giác nhỏ trên ngõ vào CLK của nó để chỉ ra rằng ngõ vào này chỉ được kích hoạt khi xảy ra chuyển đổi tăng (PGT - positive-going transition); không có phần nào khác của xung đầu vào sẽ có tác dụng đối với ngõ vào CLK. Trong Hình 13, ký hiệu FF có một vòng tròn cũng như một hình tam giác trên ngõ vào CLK của nó. Điều này biểu thị rằng ngõ vào CLK chỉ được kích hoạt khi xảy ra chuyển đổi giảm (NGT - negative-going transition); không có phần nào khác của xung đầu vào sẽ có tác dụng đối với ngõ vào CLK.

- FF được đồng bộ hóa cũng có một hoặc nhiều ngõ vào điều khiển có thể có nhiều tên khác nhau tùy thuộc vào hoạt động của chúng. Các ngõ vào điều khiển sẽ không có tác dụng đối với Q cho đến khi xảy ra chuyển đổi xung clock tích cực. Nói cách khác, tác dụng của chúng được đồng bộ hóa với tín hiệu được áp dụng cho CLK. Vì lý do này, chúng được gọi là ngõ vào điều khiển đồng bộ.

  Ví dụ: các ngõ vào điều khiển của FF trong Hình a sẽ không có tác dụng đối với Q cho đến khi xảy ra PGT của tín hiệu xung clock. Tương tự như vậy, các ngõ vào điều khiển trong Hình b sẽ không có tác dụng cho đến khi xảy ra NGT của tín hiệu xung clock.Các loại flip-flop phổ biến (ví dụ: D flip-flop, ...) và bảng chân trị của chúng.

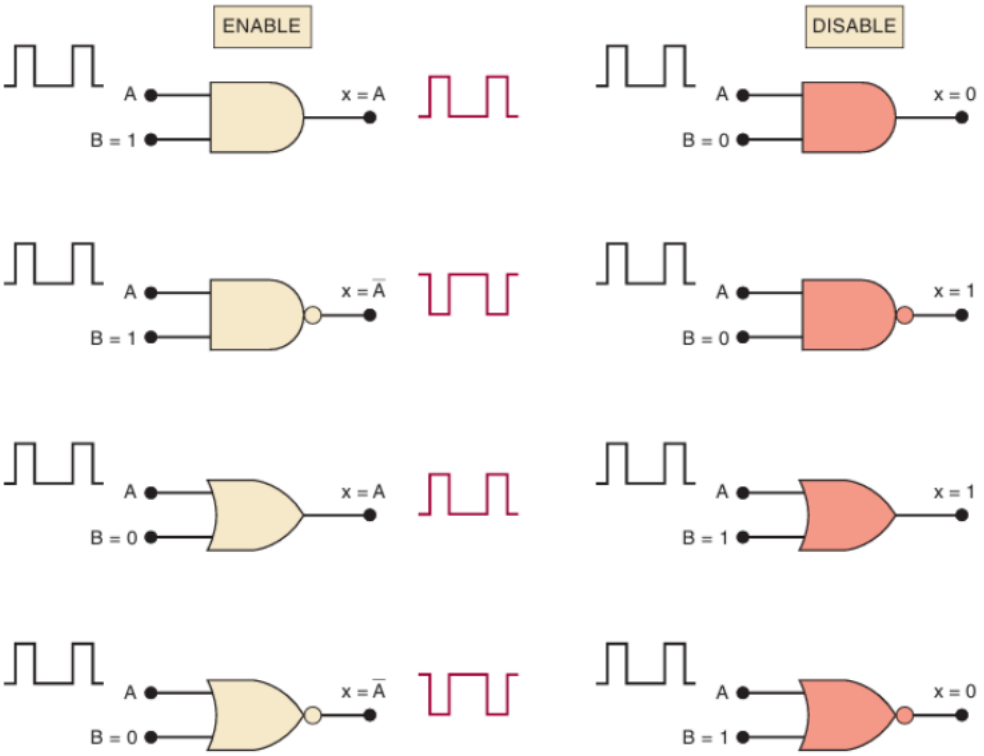


Hình 12: Position-going transition



Hình 13: Negative-going transition

3.2.2 Enable/Disable Circuits



Hình 14: Hình ảnh về các mạch Enable/Disable

Hình trên minh họa cách mỗi cổng logic cơ bản (AND, OR, NAND, NOR) có thể được sử dụng để cho phép hoặc chặn tín hiệu đầu vào đi qua đầu ra, được điều khiển bởi mức logic ở đầu vào điều khiển B.

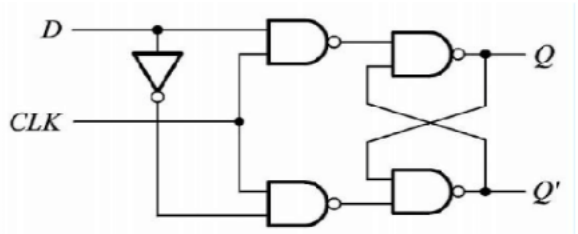
- Cổng AND
  - Khi B = 1 (cho phép), đầu ra (x) sẽ đi theo tín hiệu đầu vào A.
  - Khi B = 0 (không cho phép), đầu ra (x) sẽ luôn ở mức thấp (0).

- Cổng OR
  - Khi B = 1 (cho phép), đầu ra (x) sẽ đi theo tín hiệu đầu vào A.
  - Khi B = 0 (không cho phép), đầu ra (x) sẽ luôn ở mức cao (1).

- Cổng NAND:
  - Khi B = 0 (cho phép), đầu ra (x) sẽ là nghịch đảo của tín hiệu đầu vào A.
  - Khi B = 1 (không cho phép), đầu ra (x) sẽ luôn ở mức cao (1).

- Cổng NOR:
  - Khi B = 0 (cho phép), đầu ra (x) sẽ là nghịch đảo của tín hiệu đầu vào A.
  - Khi B = 1 (không cho phép), đầu ra (x) sẽ luôn ở mức thấp (0).

3.2.3 D FLip-flop



Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Hình 15: Hình ảnh và Bảng chân trị của D Flip-flop

3.3 Khối cộng

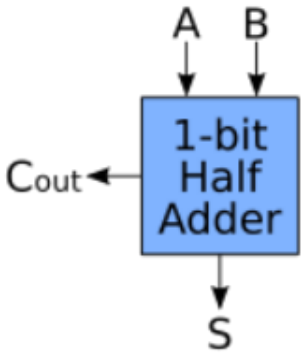
Bộ cộng, hay bộ cộng số, là một mạch kỹ thuật số thực hiện phép cộng các số. Trong nhiều máy tính và các loại bộ xử lý khác, bộ cộng được sử dụng trong các đơn vị logic số học (ALU). Chúng cũng được sử dụng trong các phần khác của bộ xử lý, nơi chúng được sử dụng để tính toán địa chỉ, chỉ mục bảng, toán tử tăng và giảm và các phép toán tương tự.

Mặc dù bộ cộng có thể được xây dựng cho nhiều dạng biểu diễn số, chẳng hạn như mã nhị phân thập phân hoặc excess-3, nhưng các bộ cộng phổ biến nhất hoạt động

trên số nhị phân. Trong trường hợp sử dụng bù hai hoặc bù một để biểu diễn số âm, việc sửa đổi bộ cộng thành bộ cộng-trừ là rất đơn giản. Các biểu diễn số có dấu khác yêu cầu nhiều logic hơn xung quanh bộ cộng cơ bản.

Các mạch cộng phổ biến (Ví dụ: mạch cộng nửa, mạch cộng đầy đủ):

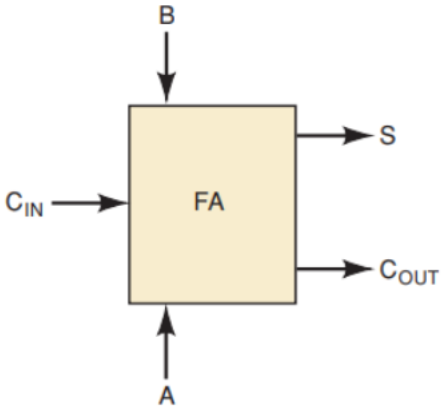
- **Bộ cộng nửa (Half Adder):** Bộ cộng nửa cộng hai bit đầu vào và tạo ra một số nhớ và một tổng, là hai đầu ra của bộ cộng nửa. Các biến đầu vào của bộ cộng nửa được gọi là các bit số bị cộng và số cộng. Các biến đầu ra là tổng và số nhớ.



Hình 16: Hình ảnh về Bộ cộng nửa

- **Bộ cộng đầy đủ (Full adder):** Một bộ cộng đầy đủ cộng các số nhị phân và tính đến các giá trị được mang vào cũng như ra. Một bộ cộng đầy đủ một bit cộng ba số một bit, thường được viết là A, B và Cin; A và B là các toán hạng và Cin là một bit được mang vào từ giai đoạn ít quan trọng trước đó. Mạch tạo ra đầu ra hai bit. Số nhớ ra và tổng thường được biểu diễn bằng các tín hiệu Cout và S, trong đó tổng bằng 2Cout + S. Bộ cộng đầy đủ thường là một thành phần trong một chuỗi các bộ cộng, cộng 8, 16, 32, v.v. số nhị phân bit.

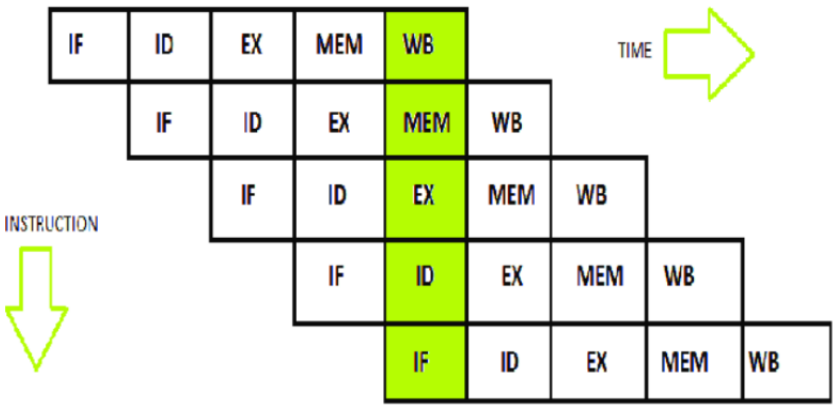
Augend bit input	Addend bit input	Carry bit input		Sum bit output	Carry bit output
A	B	C_{IN}		S	C_{OUT}
0	0	0		0	0
0	0	1		1	0
0	1	0		1	0
0	1	1		0	1
1	0	0		1	0
1	0	1		0	1
1	1	0		0	1
1	1	1		1	1



Hình 17: Hình ảnh về Bộ cộng đầy đủ

3.4 Kỹ thuật pipeline

Pipeline là một kỹ thuật quan trọng trong thiết kế máy tính, nhằm tăng tốc độ xử lý bằng cách chia nhỏ các lệnh thành các giai đoạn nhỏ và thực hiện chúng đồng thời. Hãy tưởng tượng một dây chuyền lắp ráp, mỗi công nhân thực hiện một công đoạn nhỏ, và nhiều sản phẩm được lắp ráp đồng thời ở các công đoạn khác nhau. Pipeline trong máy tính cũng tương tự như vậy.



Hình 18: Ví dụ cụ thể của 1 CPU dùng pipeline có 5 giai đoạn

Các giai đoạn thực hiện của một CPU sử dụng kiến trúc pipeline:

- IF (Instruction Fetch): Lấy lệnh từ bộ nhớ.
- ID (Instruction Decode): Giải mã lệnh để xác định thao tác cần thực hiện.
- EX (Execute): Thực hiện thao tác, ví dụ như tính toán số học hoặc truy cập bộ nhớ.
- MEM (Memory Access): Truy cập bộ nhớ để đọc hoặc ghi dữ liệu.
- WB (Write Back): Ghi kết quả vào thanh ghi.

Ưu điểm và nhược điểm của pipeline:

- **Ưu điểm:**
  - Pipelining cho phép thực thi đồng thời nhiều lệnh, dẫn đến việc tăng đáng kể số lượng lệnh được hoàn thành mỗi giây (IPS - Instructions Per Second).
  - Bằng cách chia nhỏ việc thực thi lệnh thành các giai đoạn nhỏ hơn, pipelining cho phép mỗi giai đoạn được hoàn thành nhanh hơn, dẫn đến việc giảm thời gian thực thi lệnh trung bình.
  - Pipelining cho phép các thành phần khác nhau của bộ xử lý được sử dụng đồng thời, tránh lãng phí tài nguyên và tăng hiệu quả sử dụng.
- **Nhược điểm:**
  - Mặc dù thời gian thực thi lệnh trung bình giảm, nhưng việc thêm các giai đoạn pipeline có thể làm tăng độ trễ (latency) của từng lệnh riêng lẻ.
  - Khi nhiều lệnh được thực thi đồng thời, có khả năng xảy ra xung đột tài nguyên, chẳng hạn như hai lệnh cố gắng truy cập cùng một vị trí bộ nhớ.
  - Việc thiết kế pipeline phức tạp và đòi hỏi phải xem xét kỹ lưỡng các vấn đề về đồng bộ và điều khiển luồng lệnh.

Các vấn đề thường gặp trong thiết kế pipeline (ví dụ: data hazard, control hazard):

- Data hazard: Xung đột dữ liệu xảy ra khi pipeline phải bị dừng lại vì một bước phải chờ bước khác hoàn thành. Giả sử bạn tìm thấy một chiếc tất ở trạm gấp mà không có chiếc nào khớp. Một chiến lược khả thi là chạy xuống phòng của bạn và tìm kiếm trong tủ quần áo để xem liệu bạn có thể tìm thấy chiếc tất phù hợp

hay không. Rõ ràng, trong khi bạn đang tìm kiếm, các mẻ quần áo đã sấy xong và sẵn sàng để gấp cũng như những mẻ đã giặt xong và sẵn sàng để sấy phải chờ đợi.

- Control hazarad: Cũng được gọi là branch hazard. Khi một lệnh không thể thực thi đúng trong chu kỳ xung nhịp của pipeline vì lệnh đã nạp không phải là lệnh cần thiết; nghĩa là, luồng địa chỉ lệnh không phải như pipeline mong đợi.

4 Thiết kế

Mô tả: Bao gồm sơ đồ khối, các khối chức năng, trình bày chức năng của các khối.

Các khối chức năng và đặc tả của chúng được thể hiện như sau:

4.1 Instruction Memory

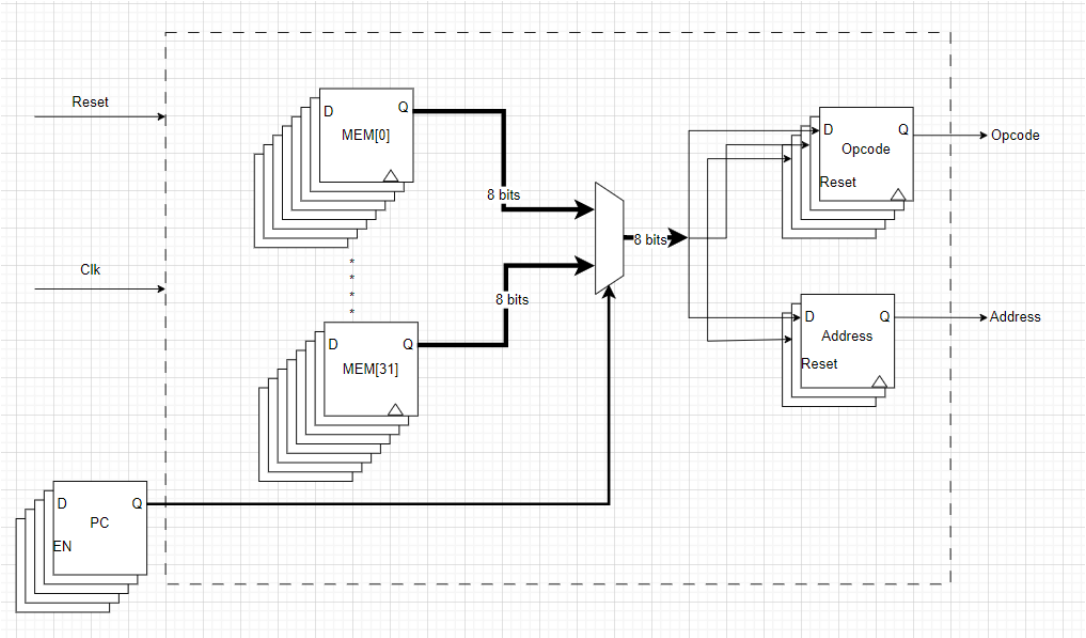
Input: Clk, Reset, PC

Output: Giá trị Opcode và Address lưu trữ trong thanh ghi

Luồng hoạt động:

- Khai báo một tập hợp 32 lệnh (mỗi lệnh 8 bit)
- Nhận địa chỉ PC (Program Counter) để xác định lệnh cần đọc.
- Tách lệnh tương ứng thành Mã lệnh (Opcode - 3bit) và Địa chỉ lệnh (Adddress - 5bit)
- Xuất các giá trị Opcode và Address một cách đồng bộ với tín hiệu đồng hồ Clk.
- Tín hiệu Reset bất đồng bộ đặt Opcode và Address về 0.
- Program_counter, Opcode, Address thay đổi theo cạnh lên của Clock
- Nạp lệnh thông qua module giao tiếp UART.

Sơ đồ khối:



Hình 19: Kiến trúc của khối Instruction Memory

4.2 Control Unit

Input: Clk, Reset, En, thanh ghi 3 bit Opcode

Output: Giá trị En_write_mem En_write_reg, ALU_OP (3-bit) lưu trữ trong thanh ghi.

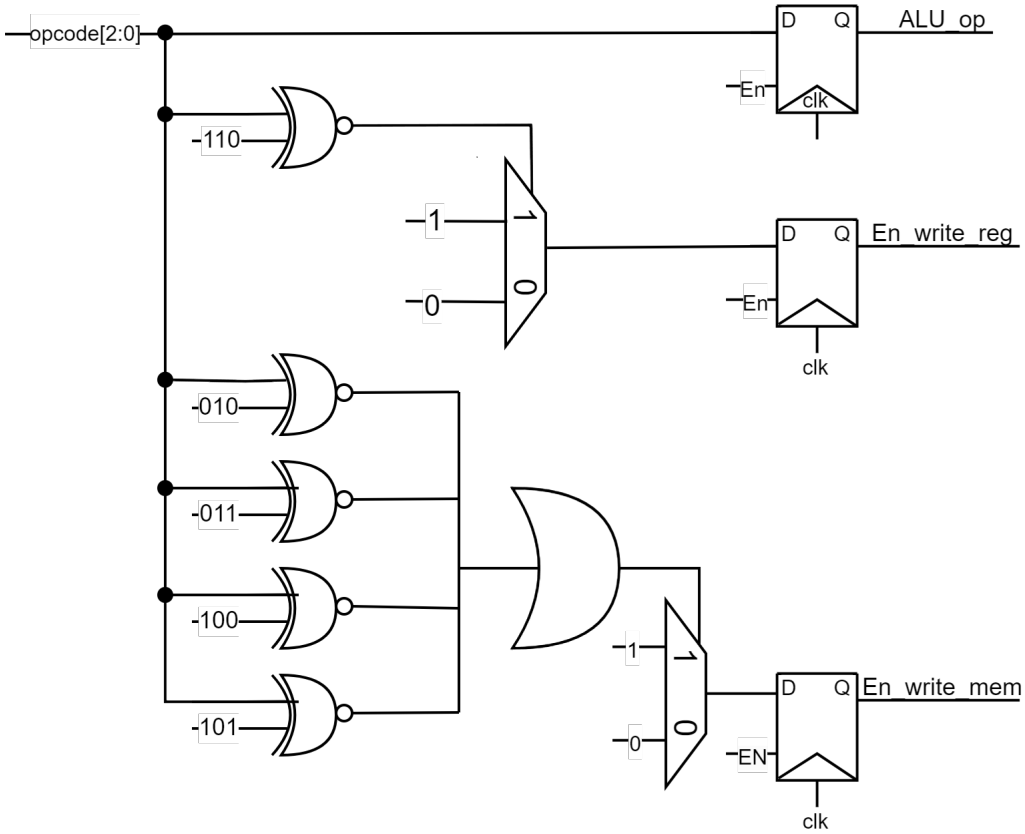
Luồng hoạt động:

- Khi Reset = 1, các giá trị đầu ra được đặt lại.

- Khi Reset = 0 và En = 1, En_write_reg mang giá trị là 1 nếu là lệnh ghi vào Accumulator và ngược lại. Nếu là lệnh ghi vào Data memory thì En_write_mem mang giá trị là 1 và ngược lại. ALUop mang giá trị bằng opcode nhận được trong chu kỳ đó.

- Các tín hiệu đầu ra thay đổi chỉ tại cạnh lên của xung đồng hồ (posedge Clk).

Sơ đồ khối:



Hình 20: Kiến trúc của Control Unit

4.3 Data Memory

Input: Clock (nhận xung clock vào khối), Reset (reset khối memory - active HIGH), Enable (cho phép ghi data vào trong memory (active HIGH)), Address ((5 bit): trỏ đến địa chỉ trong memory cần đọc/ghi data), Data in (8 bit) (nhận data dùng để ghi vào memory)

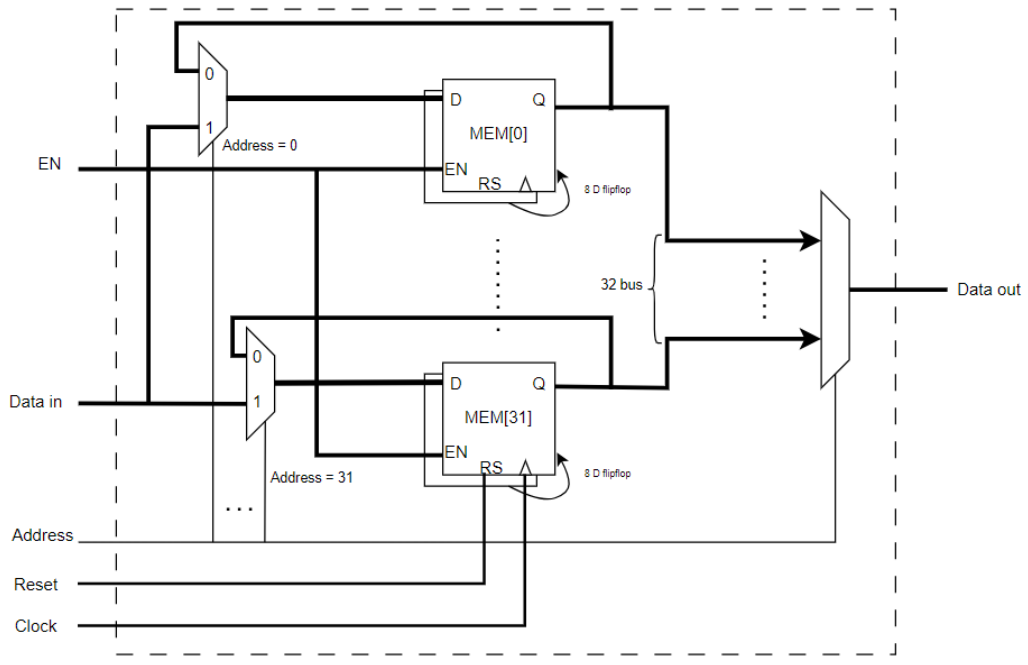
Output: Data out (8 bit) (lấy data đã đọc từ memory)

Kích thước bộ nhớ: có không gian là 32 (5 bit) và đơn vị nhỏ nhất là byte (8 bit), nên kích thước là 32 byte hoặc 256 bit.

Luồng hoạt động:

- Khởi tạo: Ban đầu có thể thiết lập các giá trị ban đầu trong bộ nhớ ở các vị trí mong muốn hoặc có thể để mặc định các register là 0x00.
- Đọc data:
  - o Data out luôn lấy được data từ địa chỉ mà address trỏ vào.
  - o Khi Address thay đổi, data out thay đổi theo ngay lập tức.
- Ghi data: Ghi giá trị từ Data in được ghi vào vị trí mà Address trỏ tới trong bộ nhớ khi có tín hiệu Enable (cho phép ghi) và cạnh lên của clock.

Sơ đồ khối:



Hình 21: Kiến trúc của Control Unit

4.4 ALU

Input: inA (đầu vào thứ nhất để tính toán (bus 8 bit)), inB (đầu vào thứ hai để tính toán (bus 8 bit)), ALU_OP (lệnh tính toán (bus 3 bit))

Output: result (kết quả tính toán (bus 8 bit)), SKZ_cmp (kết quả is_zero phục vụ bộ so sánh sớm)

Cơ chế hoạt động:

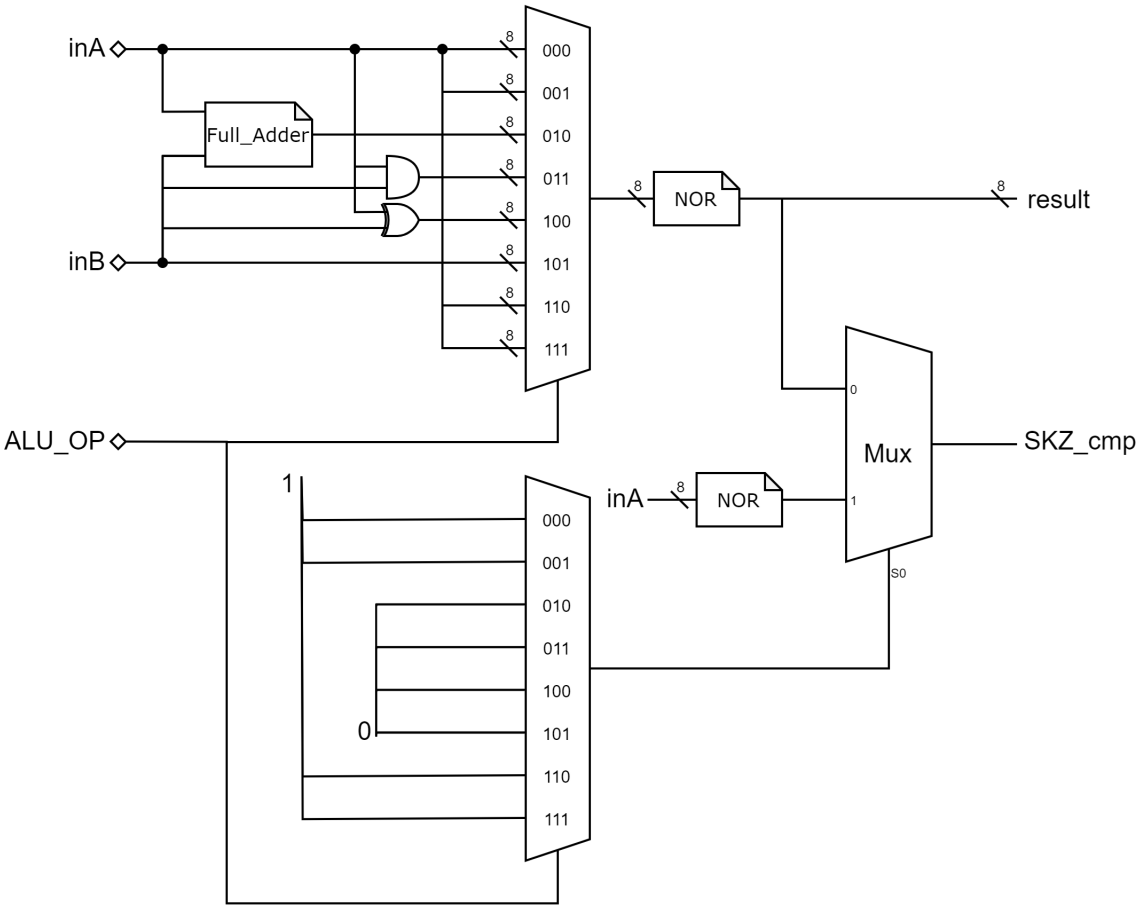
Với inA được nối với output của thanh ghi Accumulator và inB nối với output của bộ nhớ Data, bộ ALU hoạt động theo cơ chế ở trong hình 22.

Opcode	Mã	Hoạt động
HLT	000	Dừng hoạt động chương trình
SKZ	001	Trước tiên sẽ kiểm tra kết quả của ALU có bằng 0 hay không, nếu bằng 0 thì sẽ bỏ qua câu lệnh tiếp theo, ngược lại sẽ tiếp tục thực thi như bình thường
ADD	010	Cộng giá trị trong Accumulator vào giá trị bộ nhớ địa chỉ trong câu lệnh và kết quả được trả về Accumulator.
AND	011	Thực hiện AND giá trị trong Accumulator và giá trị bộ nhớ địa chỉ trong câu lệnh và kết quả được trả về Accumulator.
XOR	100	Thực hiện XOR giá trị trong Accumulator và giá trị bộ nhớ địa chỉ trong câu lệnh và kết quả được trả về Accumulator.
LDA	101	Thực hiện đọc giá trị từ địa chỉ trong câu lệnh và đưa vào Accumulator.
STO	110	Thực hiện ghi dữ liệu của Accumulator vào địa chỉ trong câu lệnh.
JMP	111	Lệnh nhảy không điều kiện, nhảy đến địa chỉ đích trong câu lệnh và tiếp tục thực hiện chương trình

Hình 22: Cơ chế hoạt động của ALU

Với các lệnh không làm thay đổi giá trị của Accumulator như HLT, SKZ, STO và JMP, is_zero bất đồng bộ nhằm cho biết input inA có bằng 0 hay không. Với các lệnh làm thay đổi Accumulator, nhằm đảm bảo kết quả so sánh sớm là kết quả của câu lệnh trước nó, is_zero bất đồng bộ gán bằng giá trị ALU tại thời điểm hiện tại.

Sơ đồ khối:

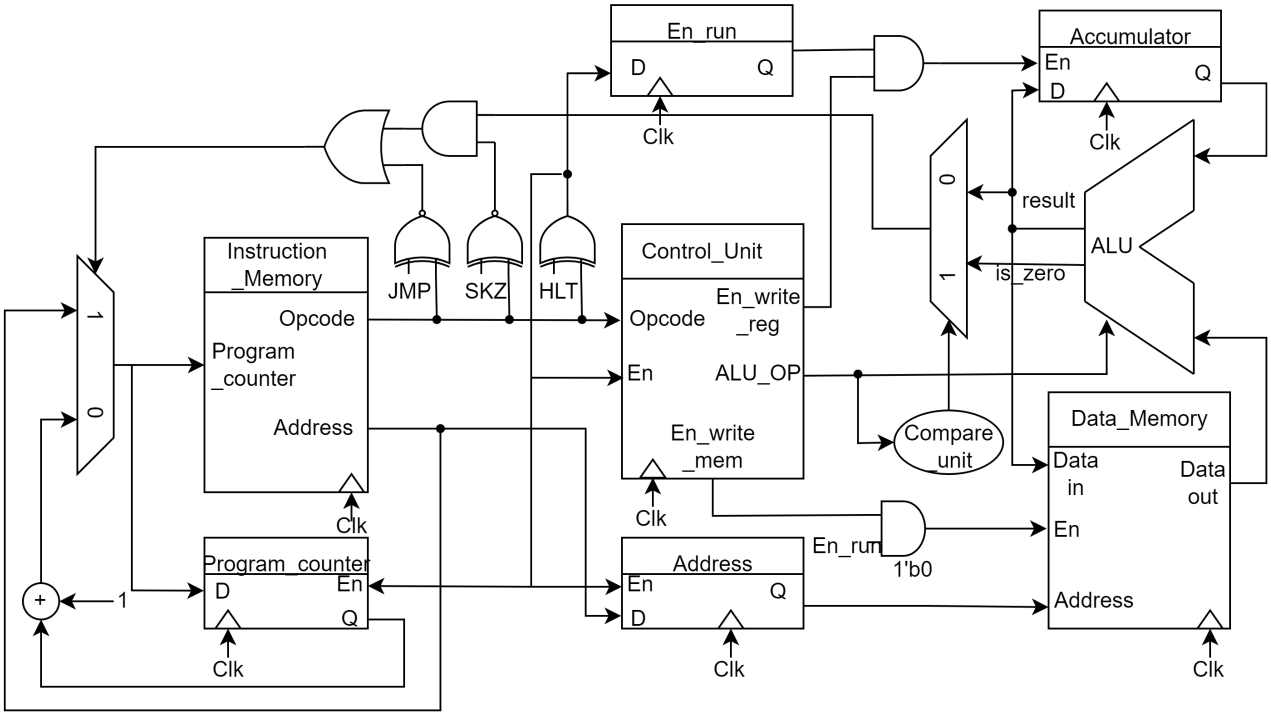


Hình 23: Kiến trúc của khối ALU

5 Hiện thực

Mô tả: Bao gồm cách thức hiện thực thiết kế (Sơ đồ khối, cấu trúc mạch, mô hình thiết kế, v.v.). Có thể vẽ các flow-chart thể hiện luồng xử lý của hệ thống.

5.1 Sơ đồ khối tổng của kiến trúc



Hình 24: Sơ đồ khối

5.2 Luồng hoạt động của hệ thống

Cơ chế pipeline:

- Một lệnh thực thi tối đa 3 chu kì xung clock
- Không có stall khi thực thi lệnh

Luồng hoạt động:

Trong kiến trúc hiện tại của nhóm, các lệnh và data sẽ được khởi tạo cứng trong bộ nhớ.

Khi có Reset, PC trỏ đến vị trí có địa chỉ 0x00 trong khối Instruction memory.

Chu kỳ đầu tiên (Fetch instruction):

- Lấy lệnh từ Instruction memory tại vị trí PC đang trỏ vào, tách thành Opcode[7:5]

và Address[4:0] để làm input cho các khối sau

- Có các bộ so sánh sớm để thực thi 2 lệnh JMP và SKZ ngay ở chu kì đầu tiên mà không cần sự can thiệp của Control Unit

+ Với lệnh JMP, bộ so sánh sớm sẽ lấy giá trị của Address[4:0] làm địa chỉ thay vì PC, đồng thời lưu giá trị vào thanh ghi PC.

+ Với lệnh SKZ, bộ so sánh sớm kiểm tra giá trị của kết quả is_zero và tăng PC lên hai đơn vị nếu is_zero bằng 1.

- Nếu Opcode[7:5] có giá trị bằng lệnh HLT, bộ so sánh sớm gán giá trị cho thanh ghi En_run bằng 0.

Chu kỳ thứ 2 (Execute):

- Control Unit sẽ nhận Opcode[7:5] đầu vào, xuất ra các tín hiệu điều khiển các khối liên quan để thực hiện tính toán cho lệnh.

- Data memory nhận giá trị Address[4:0] và thay đổi con trỏ địa chỉ, thay đổi giá trị output bằng giá trị mem ở địa chỉ mới.

- ALU luôn hoạt động:

o Nhận tín hiệu điều khiển control unit, thay đổi hành vi.

o Nhận data từ Memory và Accumulator bất đồng bộ để tính toán

o Kết quả ALU dao động theo các input ở giai đoạn đầu chu kì và tính toán được hoàn thành trong chu kì

- Nếu En_run bằng 0, chân Enable của mọi thanh ghi và bộ nhớ tích cực mức thấp, dừng quá trình hoạt động của các thanh ghi và bộ nhớ đó.

Chu kỳ thứ 3 (Write):

Dựa vào tín hiệu điều khiển của Control unit, khi có cạnh lên của clock, các kết quả tính toán từ ALU sẽ được ghi lại vào Memory và Accumulator tùy theo lệnh.

Hoạt động theo các câu lệnh:

<table>
<tr><th>Lệnh</th><th>Chu kì 1</th><th>Chu kì 2</th><th>Chu kì 3</th></tr>
<tr><td>ADD/AND/XOR</td><td rowspan="3">Lấy lệnh: tách thành Opcode và Address</td><td rowspan="3">Control unit:
- Điều khiển ALU tính toán phù hợp
- Cho phép ghi vào Accumu
Data Mem:
- Thay đổi data out theo địa chỉ nhận vào
ALU:
- Tính toán theo các input và tín hiệu điều khiển</td><td rowspan="3">Ghi data vào memory hoặc Accumulator tùy theo tín hiệu điều khiển từ Control unit</td></tr>
<tr><td>LDA</td></tr>
<tr><td>STO</td></tr>
<tr><td>SKZ</td><td>Lấy lệnh: tách thành Opcode và Address
So sánh sớm:
- Sử dụng kết quả ALU tính toán từ lệnh trước (đang trong chu kì 2)
- Tăng PC nếu thỏa điều kiện</td><td>NOPE</td><td>NOPE</td></tr>
<tr><td>JUMP</td><td>Lấy lệnh: tách thành Opcode và Address
So sánh sớm:
- Thay đổi giá trị PC hiện tại thành Address</td><td>NOPE</td><td>NOPE</td></tr>
</table>

6 Kết luận

Mô tả: Bao gồm kết của nhóm, hướng phát triển trong tương lai, những khó khăn gặp phải và bảng phân chia công việc.

Nhóm đã thực hiện đồ án môn Thiết kế luận lý với đề tài **Thiết kế RISC CPU đơn giản**. Nhóm có cơ hội hiện thực một kiến trúc RISC CPU đơn giản có các tính năng cơ bản như lưu trữ giá trị và thực hiện các phép tính. Bằng cách sử dụng ngôn ngữ HDL và công cụ thiết kế FPGA cùng các kiến thức nền đã được học trong môn Hệ thống số, Thiết kế luận lý với HDL và Kiến trúc máy tính, nhóm đã xây dựng được một môi trường thử nghiệm để hiểu rõ hơn về cách làm việc và hoạt động của một kiến trúc CPU thực tế.

Trong quá trình thực hiện đề tài "Thiết kế RISC CPU đơn giản", nhóm gặp phải nhiều khó khăn liên quan đến việc lựa chọn kiến trúc phù hợp và giải quyết các vấn đề kỹ thuật. Khi áp dụng kỹ thuật pipeline, kiến trúc của nhóm gặp phải xung đột về tín hiệu và dữ liệu giữa các khối chức năng, và việc giải quyết vấn đề này một trong những thách thức lớn nhất đối với nhóm. Nhóm đã giải quyết bằng cách chia nhỏ chu kỳ CPU, thêm thanh ghi tạm và điều chỉnh hoạt động bộ nhớ. Ngoài ra, các vấn đề hazard như data hazard với STO/LDA và control hazard với SKZ, JMP cũng phát sinh, và được nhóm xử lý bằng cách điều chỉnh kiến trúc bộ nhớ và dự đoán lệnh sớm. Để đảm bảo thiết kế không gặp hiện tượng stall, nhóm đã sử dụng các bộ so sánh sớm, mux, và thanh ghi tạm. Cuối cùng, nhóm sử dụng công cụ vẽ đồ thị tín hiệu để kiểm tra và đảm bảo tính đúng đắn của kiến trúc thiết kế.

Ngoài ra, Khi thực hiện đề tài "Thiết kế RISC CPU đơn giản", nhóm không chỉ đối mặt với các khó khăn kỹ thuật mà còn gặp trở ngại trong việc sắp xếp thời gian và quản lý khối lượng công việc. Với lịch học tập và hoạt động cá nhân dày đặc, việc đồng bộ thời gian giữa các thành viên để cùng nghiên cứu, thảo luận và thực hiện dự án trở thành một thách thức lớn. Bên cạnh đó, để có thể áp dụng vào thực tiễn đòi hỏi mỗi thành viên đều phải nghiên cứu sâu các kiến thức liên quan đến việc thiết kế CPU, đặc biệt là các khái niệm liên quan đến pipeline, hazard và tối ưu hóa hiệu năng. Để vượt qua những khó khăn này, nhóm đã phân chia công việc rõ ràng, tận dụng tối đa thời gian rảnh của từng thành viên và duy trì việc liên lạc thường xuyên.

Đồng thời, nhóm chia nhỏ các vấn đề để giải quyết từng bước, phối hợp chặt chẽ giữa việc học và thực hành để cân bằng giữa lý thuyết và ứng dụng.

Nhóm thấy khá đáng tiếc khi nhóm không có đủ thời gian để tiến hành hiện thực và kiểm thử việc dùng UART để nạp lệnh vào mạch. Nhóm thấy khá đáng tiếc khi không có đủ thời gian để hiện thực và kiểm thử việc sử dụng UART để nạp lệnh vào mạch. Trong tương lai, nhóm dự định sẽ bổ sung thêm mạch nạp UART để tăng tính linh hoạt và tiện lợi trong việc nạp lệnh. Đồng thời, nhóm cũng lên kế hoạch mở rộng kiến trúc từ 16-bit hiện tại sang 32-bit nhằm cải thiện hiệu năng và khả năng xử lý, tạo tiền đề cho các ứng dụng phức tạp hơn. Những bước phát triển này không chỉ giúp hoàn thiện dự án mà còn mở ra cơ hội nghiên cứu và áp dụng sâu hơn vào lĩnh vực thiết kế vi mạch.

Tóm lại, sau khi thực hiện đồ án, nhóm đã có thêm những kiến thức cũng như cách hiện thực một kiến trúc RISC CPU đơn giản. Tuy gặp phải những khó khăn nhất định, nhóm đã vượt qua và có cho mình những kinh nghiệm để phát triển trong tương lai.

Nhóm đã phân chia công việc cho các thành viên như sau:

STT	Họ và tên	MSSV	Nhiệm vụ	Đóng góp
1	Lâm Nữ Uyển Nhi	2212429	- Hoàn thành việc thiết kế kiến trúc RISC - Tổng hợp luận lý - Giải quyết vấn đề hazard - Nạp xuống FPGA và kiểm thử quá trình chạy.	100%
2	Hồ Đăng Khoa	2211588	- Thiết kế kiến trúc khối ALU - Tạo testcase kiểm thử tính đúng đắn của kiến trúc tổng so với đặc tả - Giải quyết vấn đề hazard - Nạp xuống FPGA và kiểm thử quá trình chạy.	100%
3	Võ Huy Quang	2212761	- Thiết kế kiến trúc khối Data Memory - Tạo testcase kiểm thử tính đúng đắn của kiến trúc tổng so với đặc tả - Giải quyết vấn đề hazard	100%
4	Võ Minh Quân	2212826	- Thiết kế kiến trúc khối Intrustion Memory - Làm báo cáo	100%
5	Lê Phan Bảo Như	2212466	- Thiết kế kiến trúc khối Control Unit - Làm báo cáo	100%