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


# THIẾT KẾ VI MẠCH (CO3098)

**Assignment**

*"Thiết kế RISC CPU đơn giản "*

**Giáo viên hướng dẫn:** Nguyễn Thành Lộc
**Lớp:** L01, Nhóm 3

| **Sinh viên thực hiện** | MSSV |
|---|---|
| Phạm Việt Hùng | 211349 |
| Nguyễn Nhật Khang | 211454 |
| Hoàng Xuân Hùng | 211326 |

**Ho Chi Minh City, 2025**

# Mục lục

# Danh sách bảng

# 1 Giới thiệu

## 1.1 Giới thiệu đề tài

RISC (Reduced Instructions Set Computer) là một phương pháp thiết kế bộ xử lý hiện nay. Trong đề tài này, chúng ta sẽ thiết kế RISC CPU đơn giản với 3-bit opcode và và 5-bit toán hạng. Tức là có 8 loại câu lệnh và 32 không gian địa chỉ. Bộ xử lý hoạt động dựa trên tín hiệu clock và reset. Chương trình sẽ dừng lại khi có tín hiệu HALT.

## 1.2 Công cụ sử dụng

- Ngôn ngữ mô tả phần cứng: Verilog HDL
- Công cụ thiết kế: Cadence

## 1.3 Chức năng sản phẩm

Về thiết kế hệ thống cần có những khối chức năng như sau:

- Program Counter: chức năng lưu trữ thanh ghi địa chỉ của chương trình (program address)
- Address Mux: nhằm lựa chọn giữa địa chỉ chương trình hoặc địa chỉ của câu lệnh (instruction)
- Memory: lưu trữ và cung cấp dữ liệu cho chương trình
- Instruction Register: xử lý dữ liệu instruction
- Accumulator Register: xử lý dữ liệu từ ALU
- ALU: xử lý dữ liệu từ Memory, Accumulator và opcode của Instruction,
- Mỗi khối chức năng đều cần có testbench tương ứng.

Về chức năng hệ thống:

- Nạp lệnh từ Memory
- Giải mã lệnh
- Lấy dữ liệu toán hạng từ Memory nếu cần
- Thực thi câu lệnh, xử lý các phép toán nếu cần
- Lưu trữ kết quả trở lại Memory hoặc Accumulator. Quá trình này sẽ lặp lại cho tới khi có câu lệnh kết thúc chương trình

# 2 Lý thuyết sơ lược về sản phẩm và những tính năng được ứng dụng trong thực tế

## 2.1 Lý thuyết sơ lược về sản phẩm

Kiến trúc **RISC** (Reduced Instruction Set Computer) là một phương pháp thiết kế bộ xử lý hiện đại, trong đó tập lệnh được tối giản để mỗi lệnh có thể được thực hiện trong một chu kỳ đồng hồ hoặc một số ít chu kỳ. Với mục tiêu đơn giản hóa phần cứng và tăng tốc độ xử lý, kiến trúc RISC đã trở thành nền tảng cho nhiều hệ thống nhúng, vi xử lý và bộ vi điều khiển trong thực tế.
Trong đề tài này, nhóm chúng em thiết kế một CPU đơn giản theo kiến trúc RISC với các đặc điểm cơ bản sau:

- Bộ xử lý sử dụng **3-bit opcode** cho phép định nghĩa tối đa **8 loại câu lệnh khác nhau**, và **5-bit operand** cho phép truy cập tới **32 địa chỉ bộ nhớ**.
- Các dữ liệu được xử lý trong hệ thống có độ rộng 8-bit.
- Bộ xử lý hoạt động dựa trên tín hiệuclock và có thểfreset hệ thống về trạng thái khởi tạo ban đầu.
- Hệ thống sẽ kết thúc chương trình khi nhận lệnh đặc biệt **HALT**.

Các thành phần chức năng chính trong hệ thống CPU bao gồm:

- **Program Counter (PC)**: giữ địa chỉ lệnh hiện tại, tự động tăng sau mỗi lệnh hoặc nạp địa chỉ mới khi thực hiện lệnh nhảy.
- **Address Mux**: bộ chọn địa chỉ, cho phép CPU lựa chọn giữa địa chỉ chương trình và địa chỉ toán hạng.
- **Instruction Register (IR)**: lưu trữ lệnh được nạp từ bộ nhớ và chuẩn bị cho quá trình giải mã.
- **Accumulator Register (AC)**: thanh ghi trung gian để lưu trữ các giá trị kết quả sau các phép toán.
- **Arithmetic Logic Unit (ALU)**: thực hiện các phép toán số học và logic dựa trên opcode của lệnh.
- **Memory**: bộ nhớ dùng để lưu trữ chương trình (instruction) và dữ liệu (data).
- **Controller**: khối điều khiển toàn bộ hoạt động CPU, bao gồm nạp lệnh, giải mã, điều phối các tín hiệu điều khiển như đọc/ghi dữ liệu và thực thi lệnh

Chu trình hoạt động của CPU được tổ chức tuần tự qua các pha: nạp lệnh, giải mã lệnh, thực thi lệnh, cập nhật kết quả và tiếp tục lặp lại cho tới khi gặp lệnh HALT.
Với cấu trúc này, hệ thống CPU đơn giản nhưng đầy đủ các bước cơ bản của một bộ xử lý thực thụ, đồng thời giúp sinh viên làm quen với quy trình thiết kế phần cứng số theo mô hình phân cấp (hierarchical modeling).

## 2.2 Những tính năng được ứng dụng cho sản phẩm trong thực tế

CPU RISC đơn giản được thiết kế trong đề tài mang lại nhiều giá trị ứng dụng thực tế trong các lĩnh vực kỹ thuật và công nghiệp.
Trước hết, về **giáo dục và đào tạo**, sản phẩm này là một công cụ rất tốt để sinh viên ngành kỹ thuật máy tính có thể trực tiếp trải nghiệm quá trình thiết kế, mô phỏng và vận hành một bộ xử lý hoàn chỉnh. Qua đó, sinh viên có thể củng cố kiến thức về thiết kế vi mạch số, nắm vững nguyên lý hoạt động của CPU, cũng như thực hành các kỹ năng mô hình hóa bằng Verilog HDL.

Về **ứng dụng công nghiệp**, bộ xử lý kiểu này có thể được tích hợp trong các hệ thống nhúng đơn giản như thiết bị IoT (Internet of Things), thiết bị đo lường, cảm biến thông minh hoặc các bộ điều khiển tín hiệu tự động. Với cấu trúc gọn nhẹ, tập lệnh tối giản và khả năng vận hành nhanh, sản phẩm

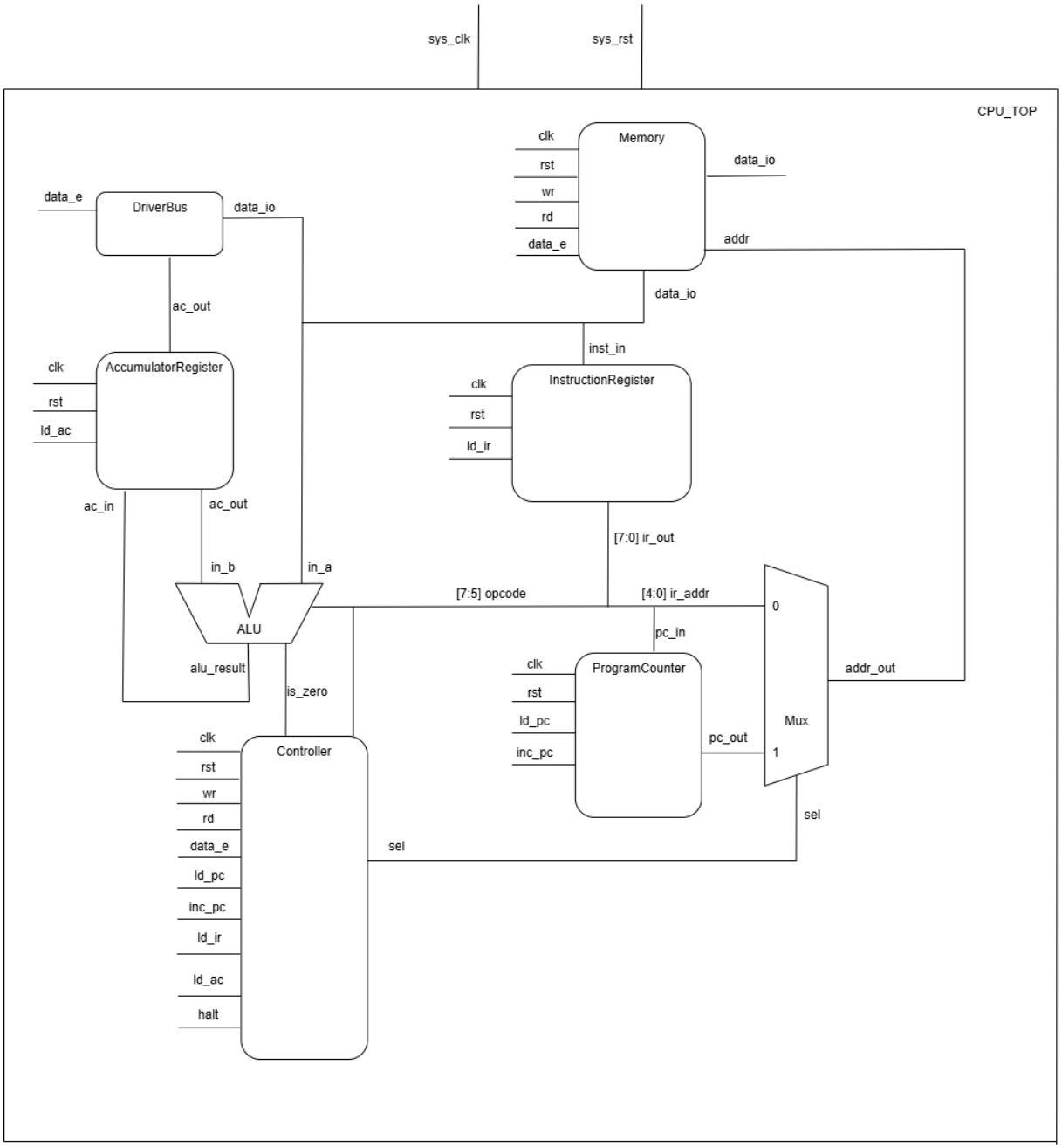
phù hợp với các ứng dụng yêu cầu tính hiệu quả năng lượng cao, chi phí thấp và độ tin cậy cao.

Ngoài ra, việc thiết kế theo phương pháp **modular design** (thiết kế module hóa) giúp CPU dễ dàng mở rộng hoặc nâng cấp. Ví dụ: có thể mở rộng thêm các lệnh mới trong tập lệnh, tăng dung lượng bộ nhớ, hoặc cải tiến ALU để hỗ trợ các phép toán phức tạp hơn như nhân (MUL), chia (DIV), hoặc các phép toán số thực (floating point).
Bên cạnh đó, sản phẩm còn giúp sinh viên thực hành các kỹ thuật chuyên sâu như **quản lý trạng thái hệ thống (State Machine)**, **xử lý hazard**, và thiết kế hệ thống **đồng bộ hóa tín hiệu** trong môi trường có nhiều thành phần giao tiếp với nhau.

Tóm lại, thiết kế CPU RISC đơn giản không chỉ mang lại giá trị học thuật mà còn mở ra nhiều hướng ứng dụng thực tiễn, là tiền đề cho việc xây dựng các hệ thống vi xử lý phức tạp hơn trong tương lai.

# 3 Thiết kế

## 3.1 Sơ đồ khối tổng thể




**Hình 1:** *Block Diagram CPU TOP*

Bao gồm:

- Program Counter (PC)
- Address Mux
- Memory
- Instruction Register (IR)

- Accumulator (AC)
- Arithmetic Logic Unit (ALU)
- Controller

## 3.2 Mô tả các khối chức năng

1. **Program Counter**:
   - Counter là bộ đếm quan trọng dùng để đếm câu lệnh của chương trình. Ngoài ra còn có thể dùng đếm các trạng thái của chương trình.
   - Counter phải hoạt động khi có xung lên của clk.
   - Reset kích hoạt mức cao, bộ đếm trở về 0.
   - Counter với độ rộng số đếm là 5.
   - Counter có chức năng load một số bất kì vào bộ đếm. Nếu không, bộ đếm sẽ hoạt động bình thường.
2. **Address Mux**:
   - Khối Address Mux với chức năng của Mux sẽ chọn giữa địa chỉ lệnh trong giai đoạn nạp lệnh và địa chỉa toán hạng trong giai đoạn thực thi lệnh.
   - Mux sẽ có độ rộng mặc định là 5.
   - Độ rộng cần sử dụng parameter để vẫn thay đổi được nếu cần.
3. **Memory**:
   - Memory sẽ lưu trữ instruction và data.
   - Memory cần được thiết kế tách riêng chức năng đọc/ghi bằng cách sử dụng Single bidirectional data port. Không được đọc và ghi cùng lúc.
   - 5-bit địa chỉ và 8-bit data.
   - 1-bit tín hiệu cho phép đọc/ghi.
   - Memory phải hoạt động khi có xung lên của clk.
4. **Register**:
   - Tín hiệu đầu vào có độ rộng 8 bits.
   - Tín hiệu rst đồng bộ và kích hoạt mức cao.
   - Register phải hoạt động khi có xung lên của clk.
   - Khi có tín hiệu load, giá trị đầu vào sẽ chuyển đến đầu ra.
   - Ngược lại giá trị đầu ra sẽ không đổi.
   - Instruction Register: Giữ lệnh đang thực thi.
   - Accumulator: Thanh ghi tạm lưu kết quả phép toán.
5. **ALU**:
   - ALU thực thi những phép toán số học. Phép tính được thực thi sẽ phụ thuộc vào toán tử của câu lệnh.
   - ALUthực thi 8 phép toán trên số hạng 8bit (inA và inB). Kết quả sẽ cho ra 8-bit output và 1bit is_zero.
   - is_zero bất đồng bộ nhằm cho biết input inA có bằng 0 hay không.
   - Đầu vào opcode 3bit sẽ quyết định phép toán nào được sử dụng như mô tả trong bảng 4.
6. **Controller**:
   - Controller uảnlý những tín hiệu điều khiển của CPU.Bao gồm nạp và thực thi lệnh.

- Controller phải hoạt động khi có xung lên của clk.
- Tín hiệu rst đồng bộ và kích hoạt mức cao.
- Tín hiệu đầu vào opcode 3bit tương ứng với ALU.
- Controller có 9 output như bảng 5.
- Controller có 8 trạng thái hoạt động liên tục trong 8 chu kỳ clk theo thứ tự: INST_ADDR, INST_FETCH, INST_LOAD,IDLE, OP_ADDR, OP_FETCH, ALU_OP, STORE. Trạng thái reset là INST_ADDR.
- Output của Controller dựa theo trạng thái và opcode như bảng 6.

7. **Driver Bus**:
   - Thực hiện ghi dữ liệu của Accumulator vào địa chỉ trong câu lệnh(STO).

# 4 Hiện thực

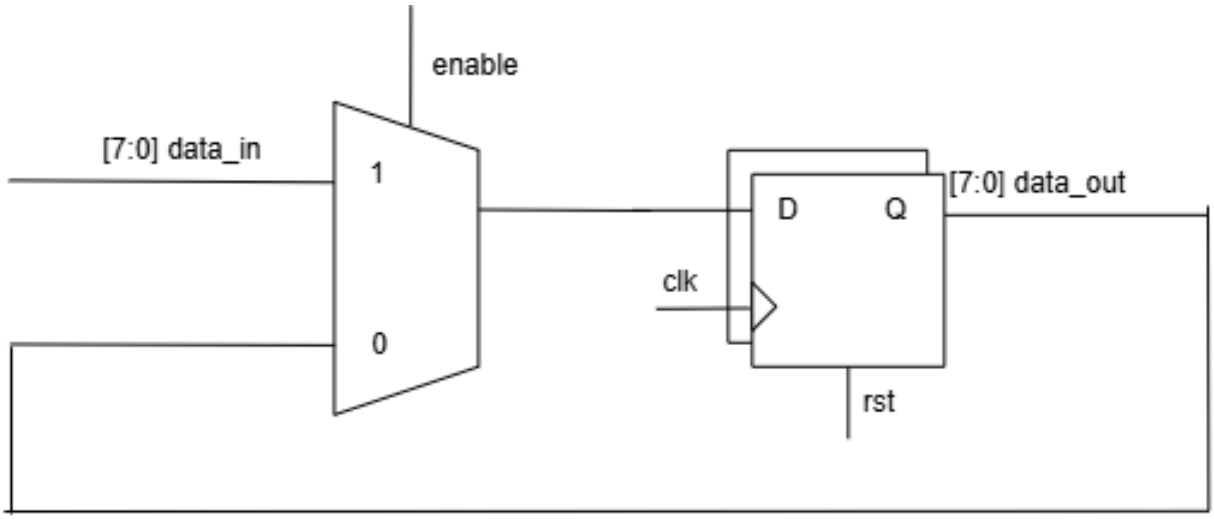
## 4.1 Accumulator

### 4.1.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **rst** | 1 | Input | Synchronous active high reset |
| **clk** | 1 | Input | Clock signal |
| **ld_ac** | 1 | Input | Load accumulator |
| **ac_in** | 8 | Input | Data in |
| **ac_out** | 8 | Output | Data out |

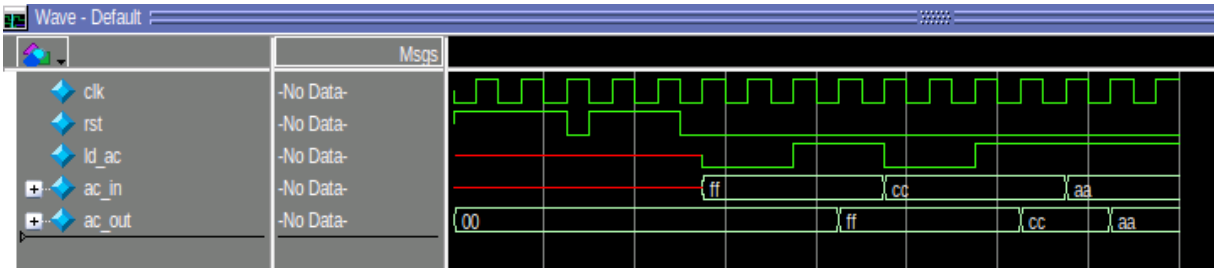
**Bảng 1:** *Accumulator IO port list*

### 4.1.2 Logic Diagram




**Hình 2:** *Logic Diagram Accumulator*

### 4.1.3 Waveform




**Hình 3:** *Waveform Accumulator*

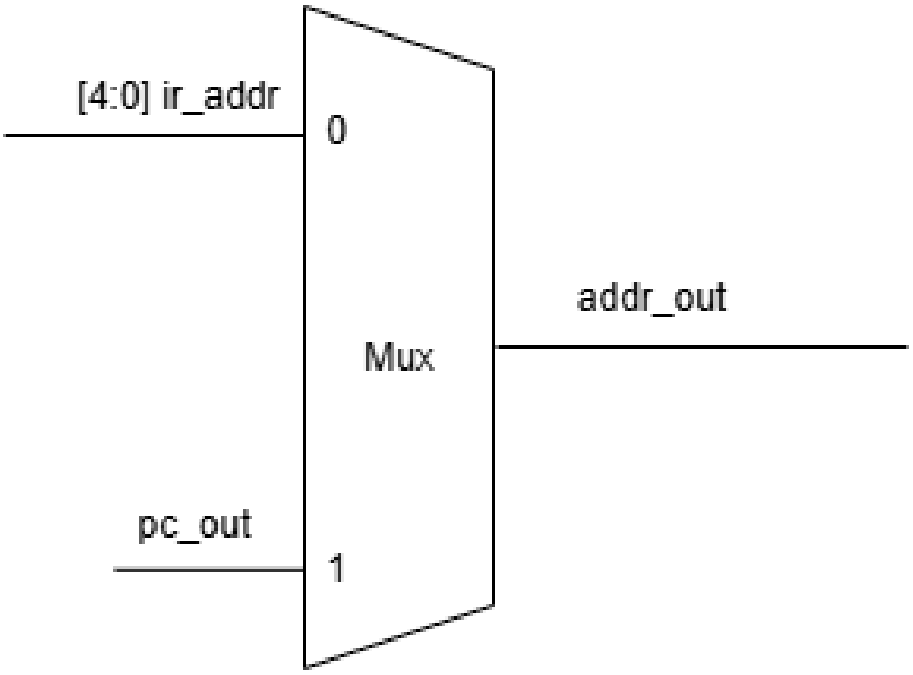
## 4.2 Address Mux

### 4.2.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **sel** | 1 | Input | sel |
| **pc_addr** | 5 | Input | PC address |
| **operand_addr** | 5 | Input | Operand address |
| **addr_out** | 5 | Output | Output address mux |

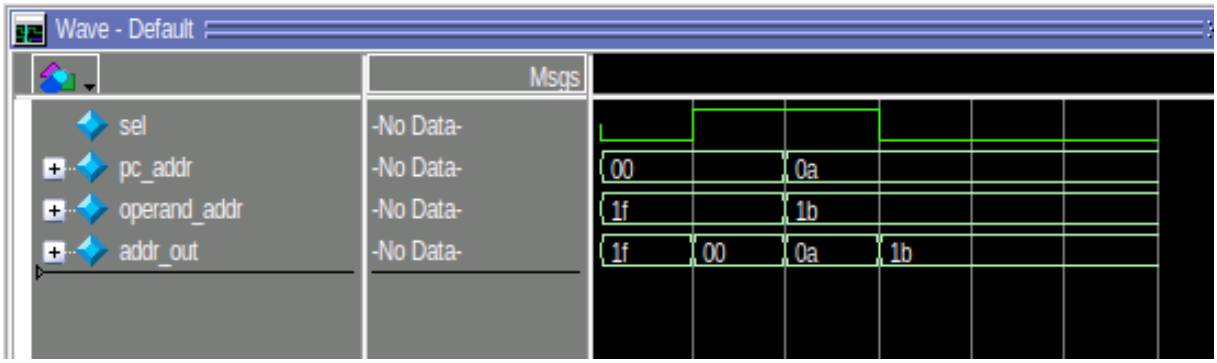
**Bảng 2:** *Address Mux IO port list*

### 4.2.2 Logic Diagram




**Hình 4:** *Logic Diagram Address Mux*

### 4.2.3 Waveform




**Hình 5:** *Waveform Accumulator*

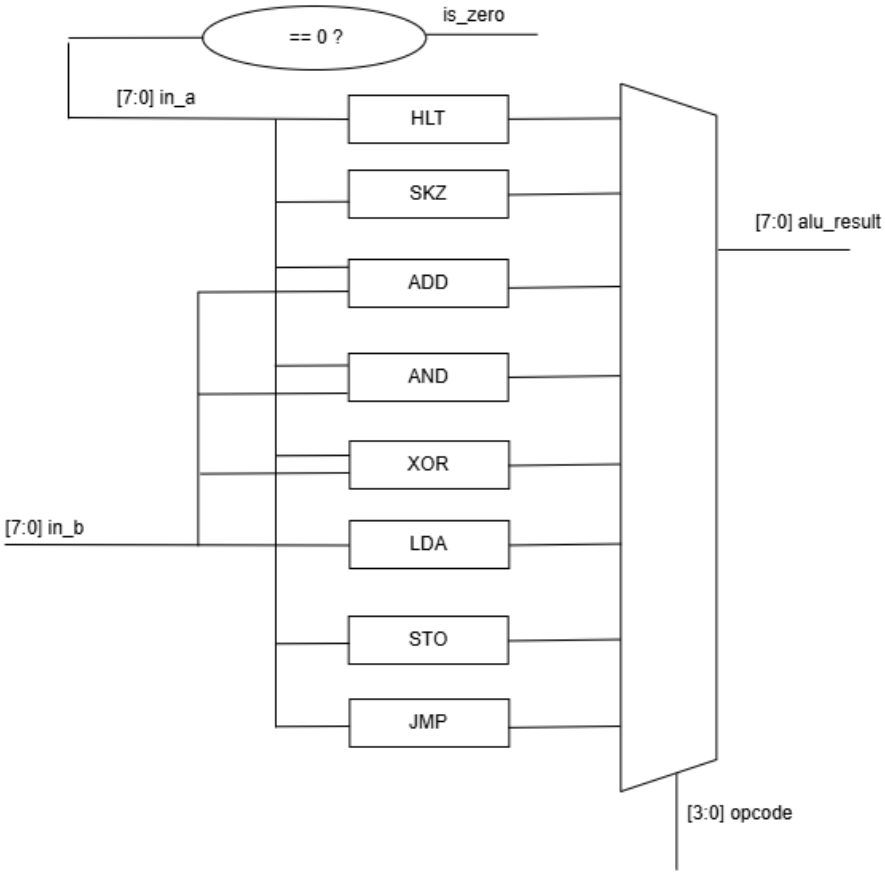
## 4.3 ALU

### 4.3.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **in_a** | 8 | Input | Operand a |
| **in_b** | 8 | Input | Operand b |
| **opcode** | 3 | Input | Indicate which operation the CPU should execute |
| **is_zero** | 1 | Output | Indicate is input in_a equal to zero |
| **result** | 8 | Output | Result of module |

**Bảng 3:** *ALU IO port list*

### 4.3.2 Logic Diagram

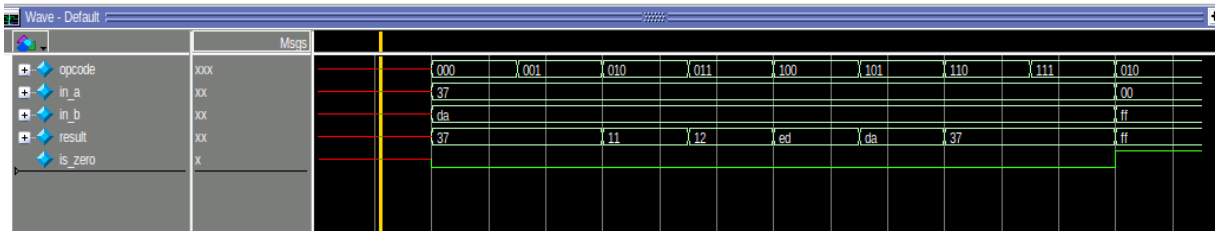



**Hình 6:** *Logic Diagram ALU*

| Opcode | Mã | Hoạt động | Output |
|---|---|---|---|
| HLT | 000 | Dừng hoạt động chương trình | inA |
| 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 | inA |
| ADD | 010 | Cộng giá trị trong Accumulator vào giá trị bộ nhớ địa chỉ trong câu lệnh và kết quả được trả về Accumulator. | inA + inB |
| 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. | inA and inB |
| 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. | inA xor inB |
| LDA | 101 | Thực hiện đọc giá trị từ địa chỉ trong câu lệnh và đưa vào Accumulator. | inB |
| STO | 110 | Thực hiện ghi dữ liệu của Accumulator vào địa chỉ trong câu lệnh. | inA |
| JMP | 111 | Lệnh nhảy không điều kiện, nhảy đến địa chỉ định trong câu lệnh và tiếp tục thực hiện chương trình. | inA |

**Bảng 4:** *ALU Opcode*

### 4.3.3 Waveform




**Hình 7:** *Waveform ALU*

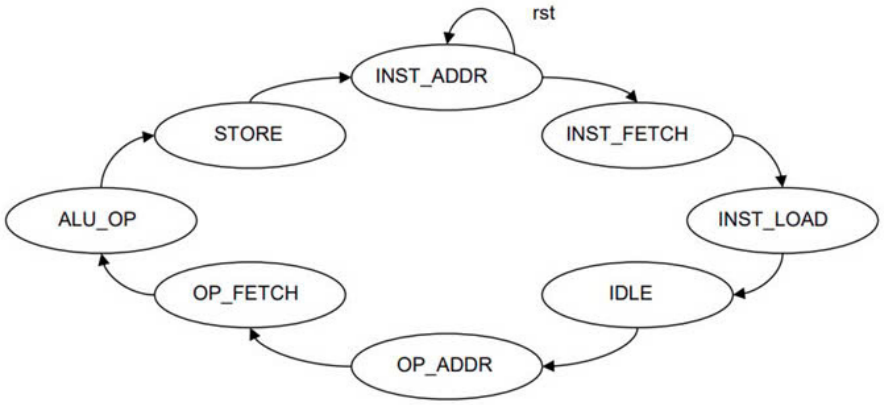
## 4.4 Controller

### 4.4.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **clk** | 1 | Input | Clock signal |
| **rst** | 1 | Input | Synchronous active high reset |
| **opcode** | 3 | Input | Indicate which operation the CPU should execute |
| **is_zero** | 1 | Input | Indicate is input in_a equal to zero |
| **sel** | 1 | Output | Select |
| **rd** | 1 | Output | Memory read |
| **ld_ir** | 1 | Output | Load instruction register |
| **halt** | 1 | Output | Halt |
| **inc_pc** | 1 | Output | Increment program counter |
| **ld_ac** | 1 | Output | Load accumulator |
| **ld_pc** | 1 | Output | Load program counter |
| **wr** | 1 | Output | Memory write |
| **data_e** | 1 | Output | Data enable |

**Bảng 5:** *Controller IO port list*

### 4.4.2 Logic Diagram

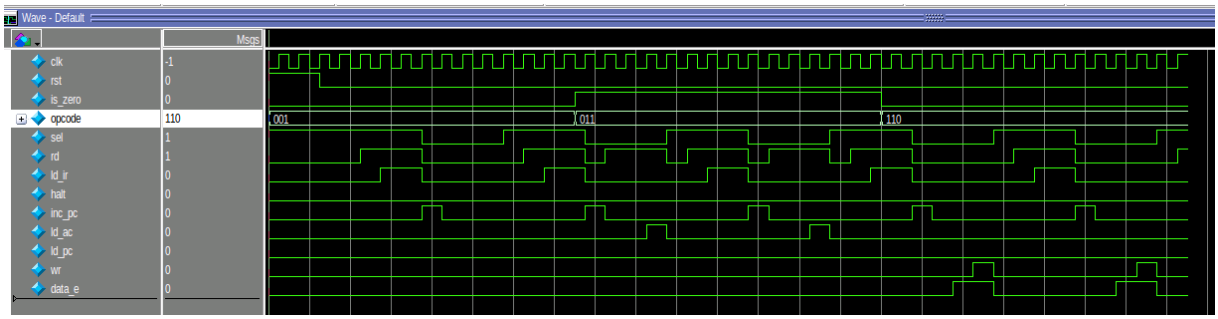



**Hình 8:** *Finite State Machine of Controller*

| Outputs | Phase | | | | | | | | Notes |
|---|---|---|---|---|---|---|---|---|---|
| | INST_ADDR | INST_FETCH | INST_LOAD | IDLE | OP_ADDR | OP_FETCH | ALU_OP | STORE | |
| sel | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| rd | 0 | 1 | 1 | 1 | 0 | ALUOP | ALUOP | ALUOP | |
| ld_ir | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | ALU OP = 1 if opcode is ADD, AND, XOR or LDA |
| halt | 0 | 0 | 0 | 0 | HALT | 0 | 0 | 0 | |
| inc_pc | 0 | 0 | 0 | 0 | 1 | 0 | SKZ && zero | 0 | |
| ld_ac | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ALUOP | |
| ld_pc | 0 | 0 | 0 | 0 | 0 | JMP | JMP | 0 | |
| wr | 0 | 0 | 0 | 0 | 0 | 0 | STO | STO | |
| data_e | 0 | 0 | 0 | 0 | 0 | 0 | STO | STO | |

**Bảng 6:** *Output controller*

### 4.4.3 Waveform




**Hình 9:** *Waveform Controller*

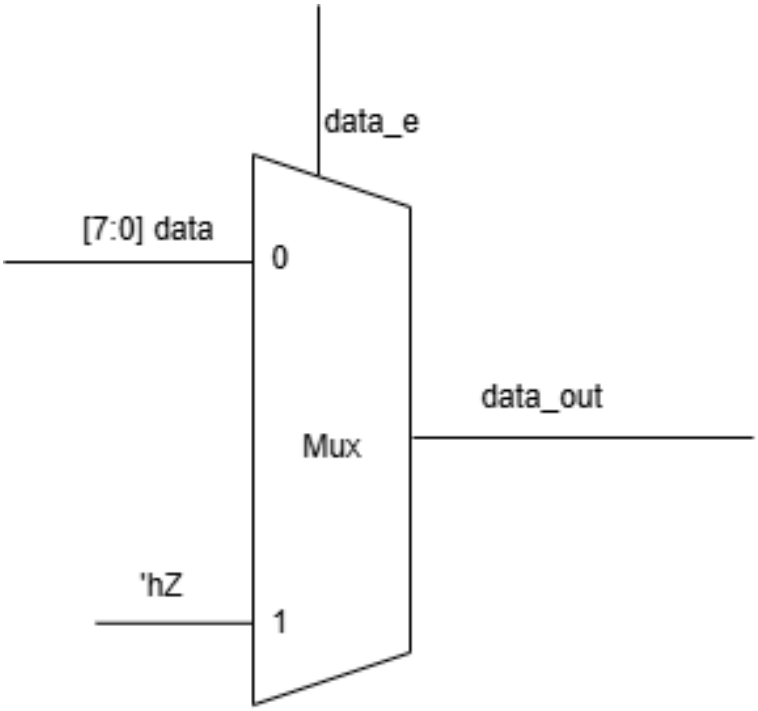
## 4.5 Driver Bus

### 4.5.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **data_e** | 1 | Input | Data enable |
| **data** | 8 | Input | Data input |
| **data_out** | 8 | Output | Data output |

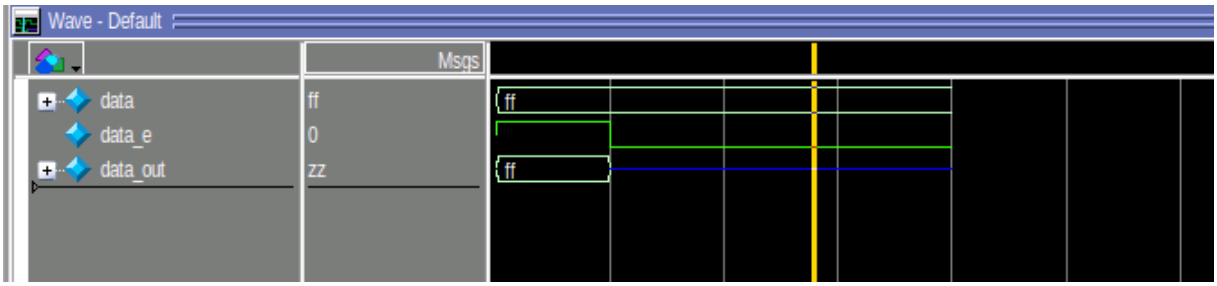
**Bảng 7:** *Driver Bus IO port list*

### 4.5.2 Logic Diagram




**Hình 10:** *Logic Diagram Driver Bus*

### 4.5.3 Waveform




**Hình 11:** *Waveform Driver Bus*

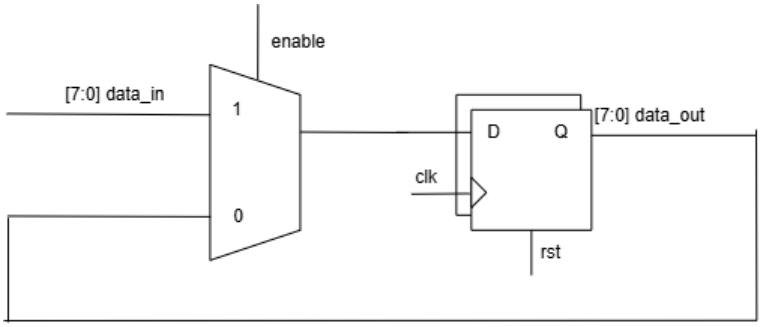
## 4.6 Instruction Register

### 4.6.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **clk** | 1 | Input | Clock signal |
| **rst** | 1 | Input | Synchronous active high reset |
| **ld_ir** | 1 | Input | load instruction register |
| **opcode** | 3 | Output | Indicate which operation the CPU should execute |
| **operand** | 5 | Output | Operand address |

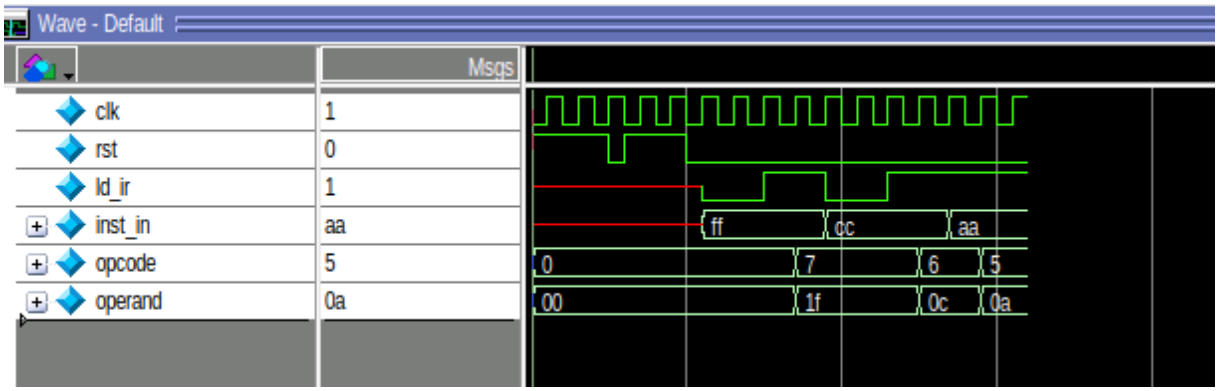
**Bảng 8:** *Instruction Register IO port list*

### 4.6.2 Logic Diagram




**Hình 12:** *Logic Diagram Instruction Register*

### 4.6.3 Waveform




**Hình 13:** *Waveform Instruction Register*

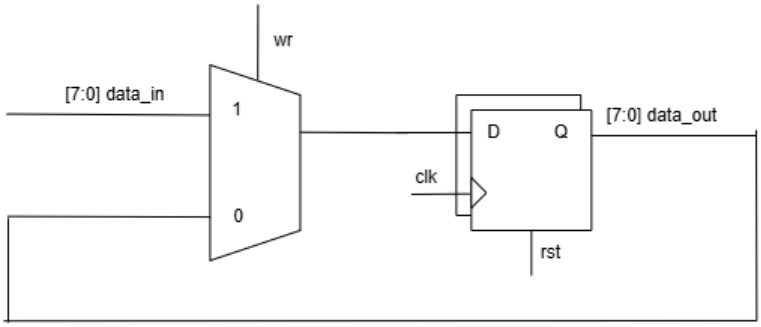
## 4.7 Memory

### 4.7.1 IO port list

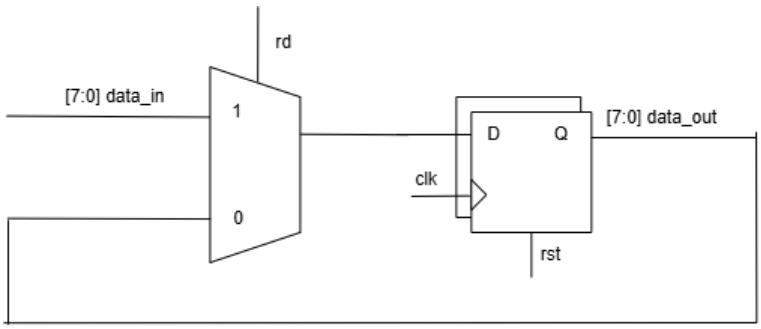
| Signal name | Width | Direction | Description |
|---|---|---|---|
| **clk** | 1 | Input | Clock signal |
| **wr** | 1 | Input | write enable |
| **rd** | 1 | Input | read enable |
| **addr** | 5 | Input | Address of memory |
| **data_io** | 8 | Inout | Single bidirectional data port |

**Bảng 9:** *Memory IO port list*

### 4.7.2 Logic Diagram

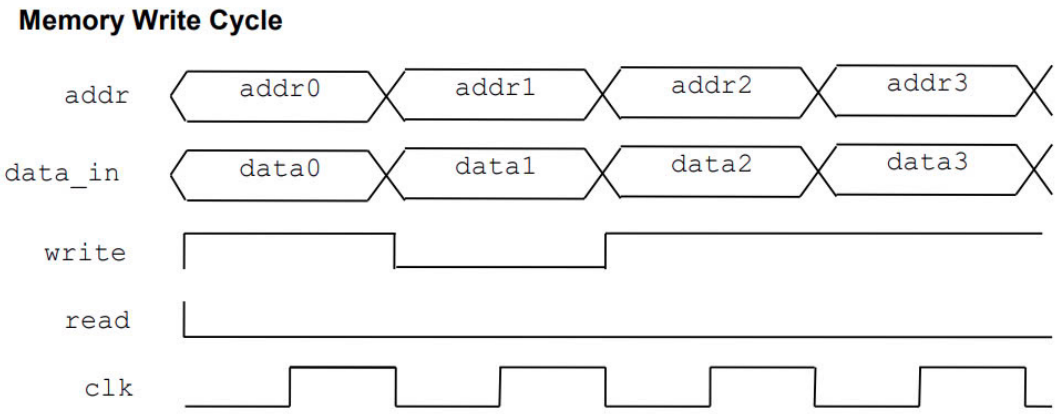



**Hình 14:** *Write tranfer*

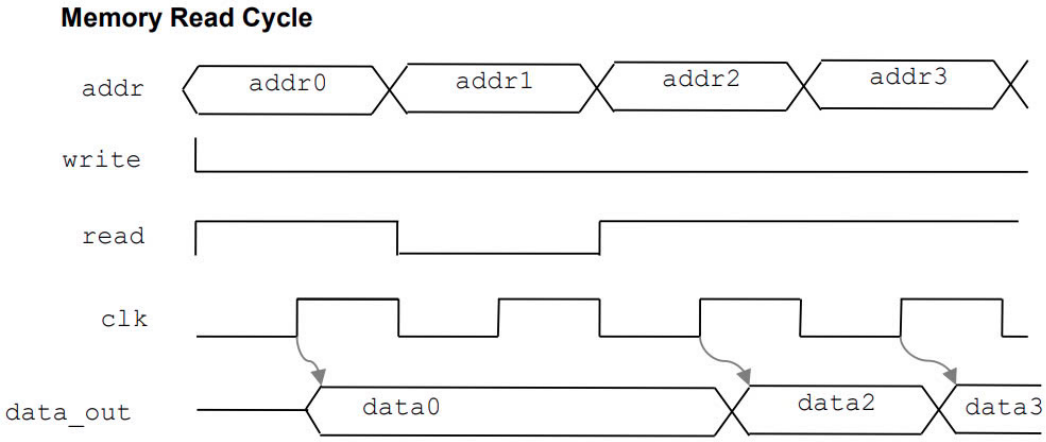



**Hình 15:** *Read tranfer*

### 4.7.3 Waveform




**Hình 16:** *Waveform Write tranfer*




**Hình 17:** *Waveform read tranfer*

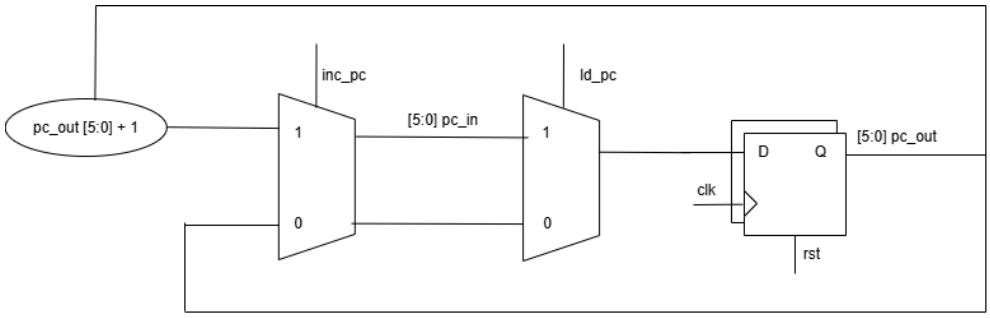
## 4.8 Program Counter

### 4.8.1 IO port list

| Signal name | Width | Direction | Description |
|---|---|---|---|
| **clk** | 1 | Input | Clock signal |
| **rst** | 1 | Input | Synchronous active high reset |
| **ld_pc** | 1 | Input | load program counter |
| **inc_pc** | 1 | Intput | increment program counter |
| **pc_in** | 5 | Input | Branch address |
| **pc_out** | 5 | Output | indicate instruction which CPU execute |

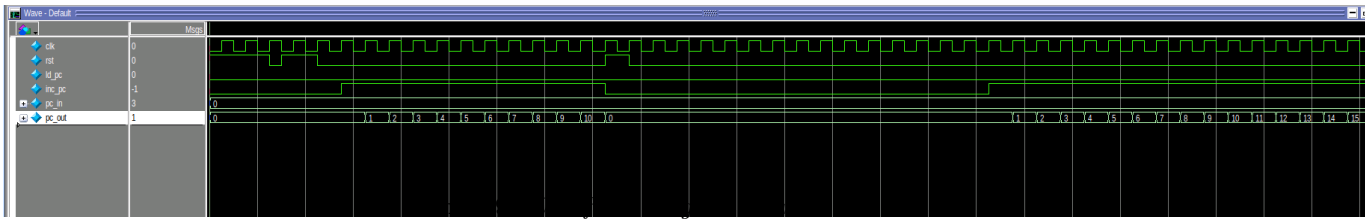
**Bảng 10:** *Program Counter IO port list*

### 4.8.2 Logic Diagram



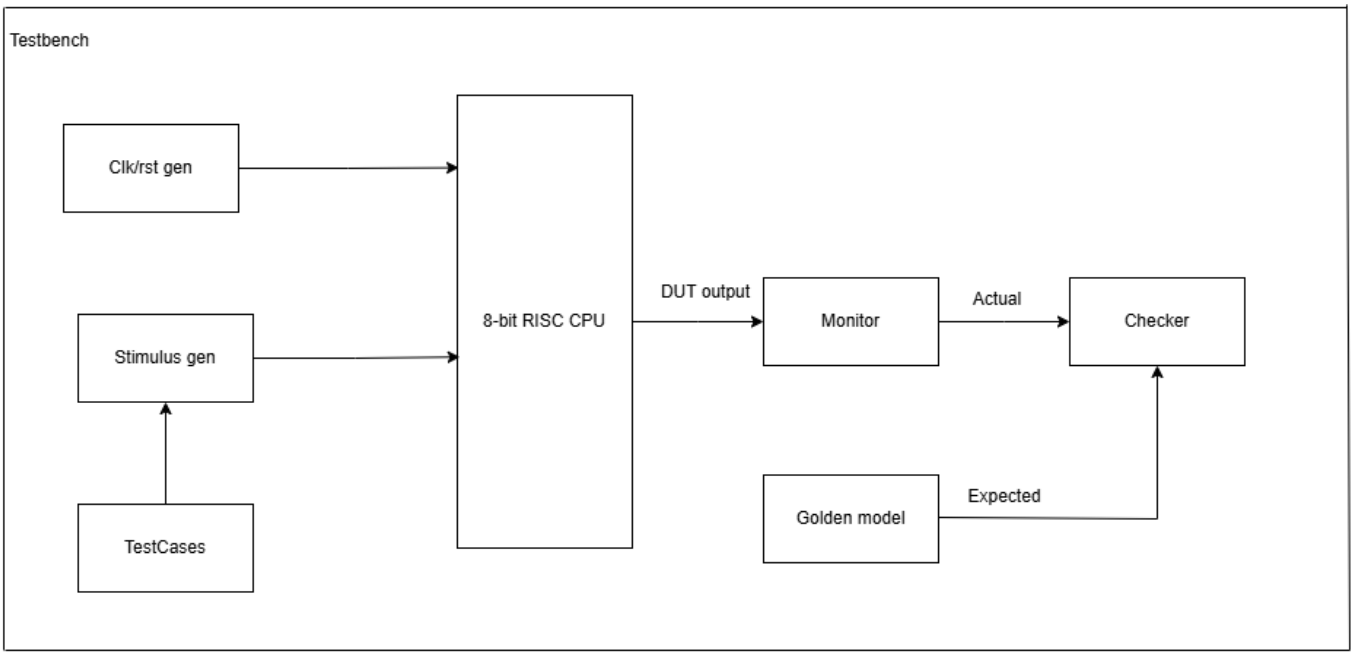

**Hình 18:** *Logic Diagram Program Counter*

### 4.8.3 Waveform



# 5 Kiểm thử và đánh giá

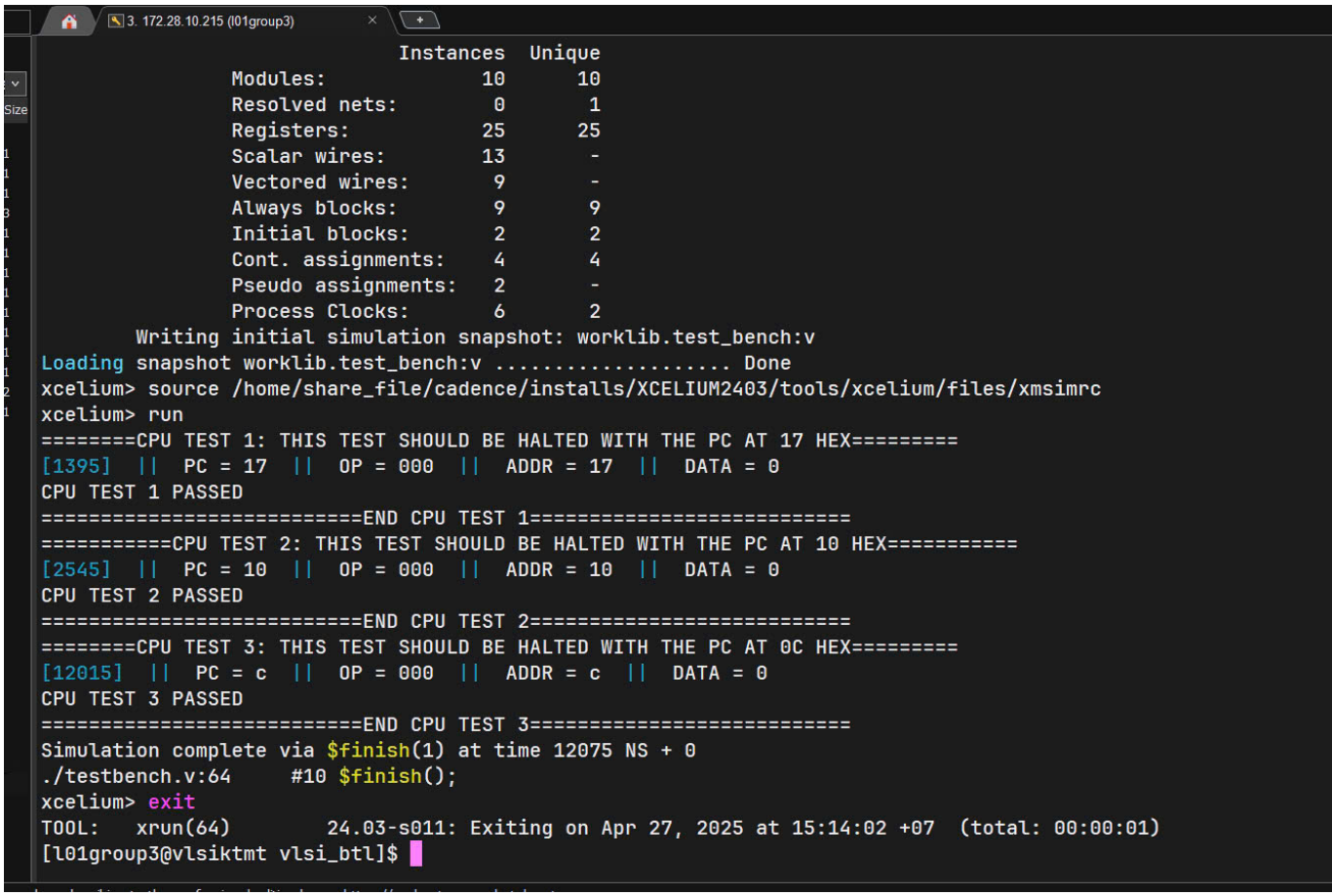
## 5.1 Môi trường kiểm thử - Verification Enviroment




**Hình 20:** *Verification Enviroment*

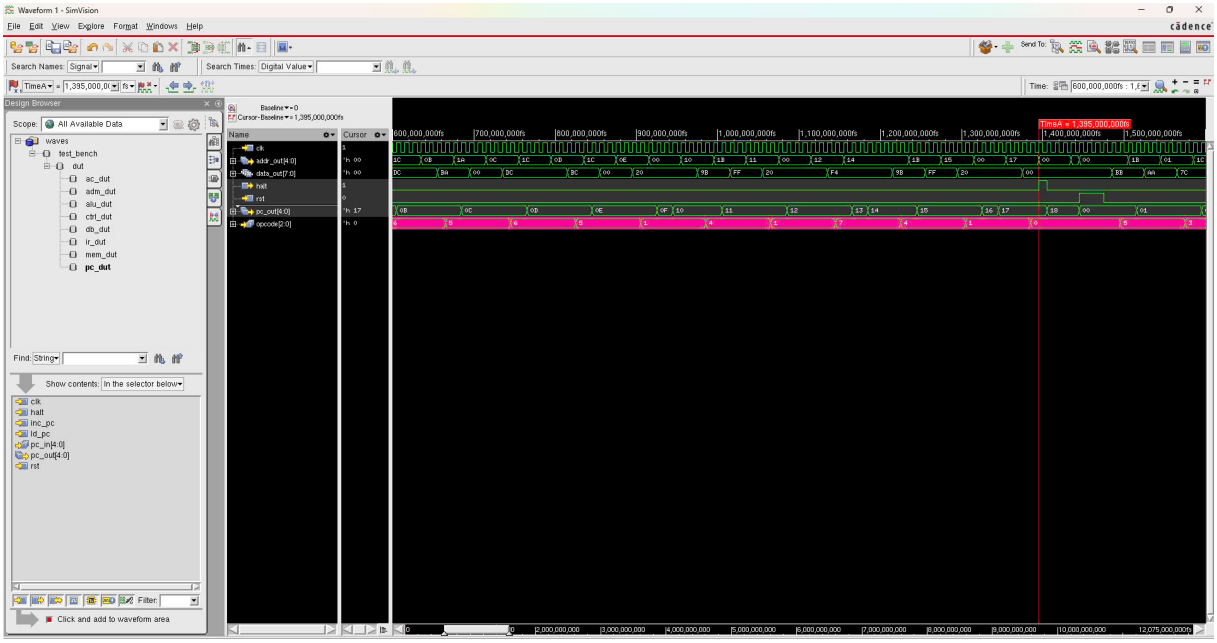
## 5.2 Kết quả mô phỏng

### 5.2.1 Kết quả hiển thị trên terminal của 3 testcase mẫu



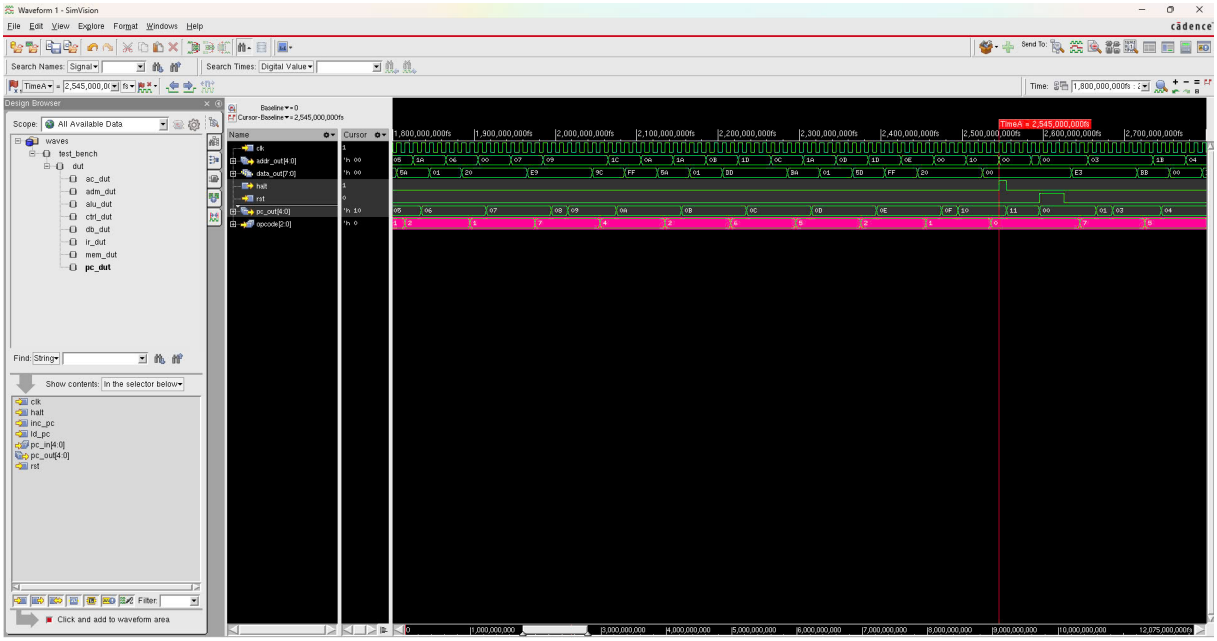

**Hình 21:** *Kết quả hiển thị trên terminal*

### 5.2.2 Waveform của testcase 1



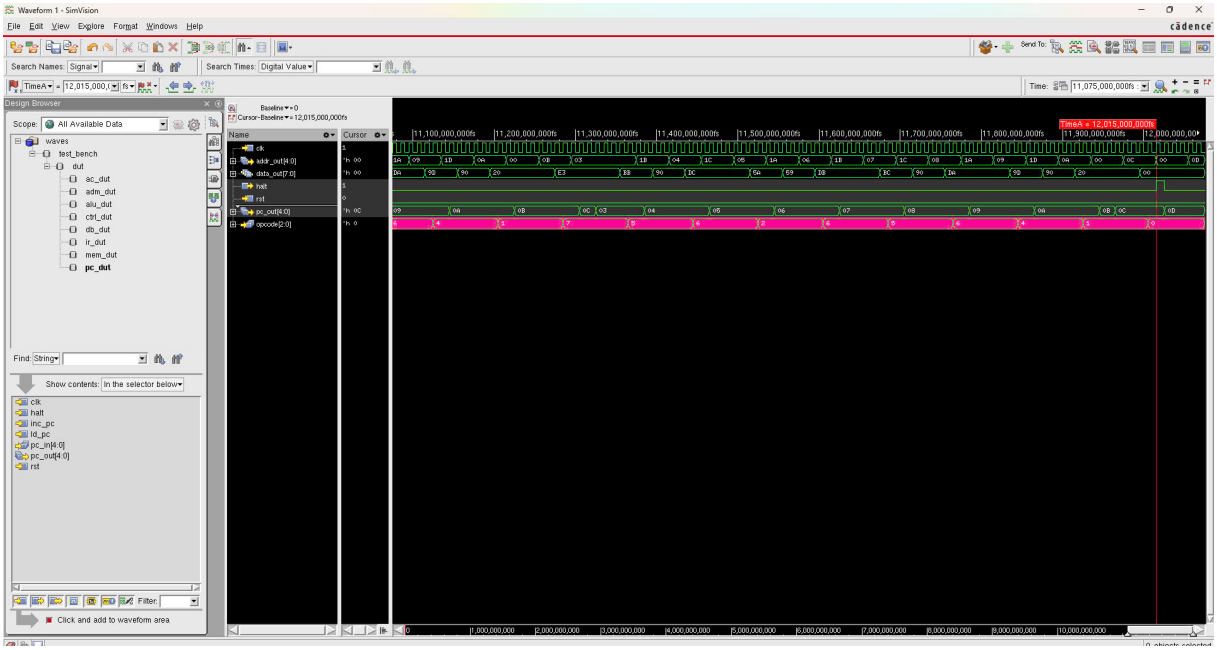
**Hình 22:** *Waveform của testcase 1*

### 5.2.3 Waveform của testcase 2



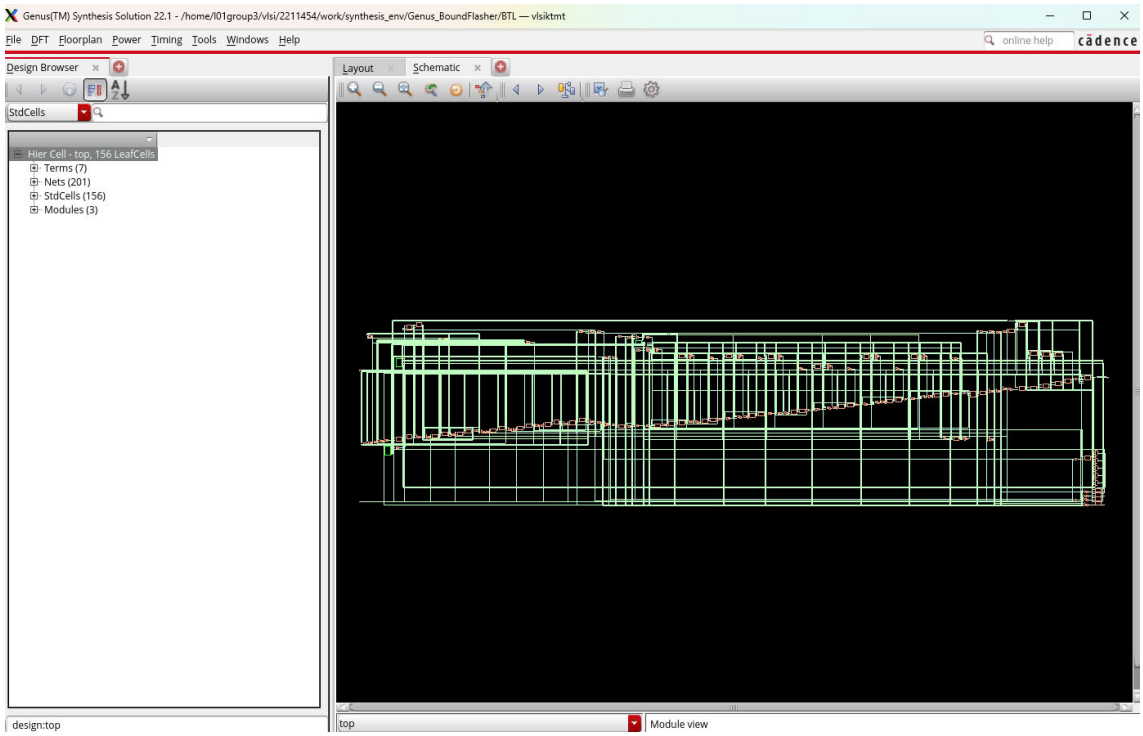
**Hình 23:** *Waveform của testcase 2*

### 5.2.4 Waveform của testcase 3



**Hình 24:** *Waveform của testcase 3*

### 5.2.5 Schematic




**Hình 25:** *Schematic CPU 8bit*

# 6 Kết luận

Trong đề tài "Thiết kế RISC CPU đơn giản", nhóm đã xây dựng và hiện thực hóa một bộ xử lý với kiến trúc tối giản nhưng đầy đủ các thành phần thiết yếu của một hệ thống vi xử lý. Thông qua việc thiết kế các khối chức năng như Program Counter, ALU, Instruction Register, Memory, và Controller, nhóm không chỉ tái hiện được hoạt động của một CPU thực tế mà còn hiểu sâu hơn về cơ chế phối hợp giữa các khối phần cứng trong một hệ thống số.

Điểm nổi bật của đề tài nằm ở việc hiện thực hóa toàn bộ CPU theo kiến trúc RISC với độ rộng dữ liệu 8-bit, sử dụng tập lệnh 3-bit opcode và 5-bit toán hạng. Hệ thống được xây dựng theo hướng module hóa, cho phép kiểm thử từng khối riêng biệt và dễ dàng mở rộng trong tương lai. Các kết quả mô phỏng cho thấy CPU hoạt động chính xác qua từng pha: nạp lệnh, giải mã, thực thi và lưu kết quả, từ đó phản ánh sự thành công trong thiết kế điều khiển và xử lý dữ liệu.

Bên cạnh đó, nhóm cũng thiết kế môi trường kiểm thử riêng biệt, từ đó đánh giá được chức năng của từng khối và toàn bộ hệ thống một cách trực quan thông qua waveform và kết quả hiển thị trên terminal. Đây là bước thực hành quan trọng giúp nhóm củng cố kỹ năng phân tích tín hiệu, xử lý trạng thái và xác minh hoạt động hệ thống trong môi trường thực nghiệm.

Từ một thiết kế ban đầu mang tính học thuật, sản phẩm CPU này có tiềm năng ứng dụng trong các hệ thống nhúng nhỏ, nơi yêu cầu về chi phí, hiệu năng và độ tin cậy được đặt lên hàng đầu. Đề tài cũng tạo nền tảng cho những hướng mở rộng như phát triển pipeline, bổ sung tập lệnh nâng cao hoặc tích hợp ngoại vi giao tiếp trong các hệ thống thực tế.

Tóm lại, đề tài không chỉ giúp nhóm hiểu rõ hơn về kiến trúc và hoạt động của một bộ vi xử lý, mà còn mang lại trải nghiệm thiết kế vi mạch toàn diện – từ lý thuyết đến mô phỏng và kiểm thử – góp phần nâng cao năng lực thực hành trong lĩnh vực kỹ thuật máy tính.