**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ
**BỘ MÔN ĐIỆN TỬ**

—————o0o—————


# PROJECT COMPUTER ARCHITECTURE

# MILESTONE III

# DESIGN OF A PIPELINED PROCESSORS

**Supervisor: Dr. Tran Hoang Linh**
**TA. Hai Cao**
**LỚP L01 - NHÓM 31**

| STT | Họ và tên | MSSV |
|---|---|---|
| 1 | Nguyễn Ngọc Kiều Duyên | 2113053 |
| 2 | Huỳnh Thịnh Phát | 2114369 |
| 3 | Lê Quốc Thái | 2114746 |

**Tp. Hồ Chí Minh, 15/12/2024**

# Contents

**List of Tables** **2**

**List of Figure** **3**

**1 Introduction** **4**

**2 Design Strategy** **5**

2.1 Non-forwarding . . . . . 5

2.2 Forwarding . . . . . 6

2.3 Branch Prediction . . . . . 7

2.3.1 Branch Predictor . . . . . 7

2.3.2 2-bit Saturating Counter . . . . . 7

2.4 Comparison of Three Theoretical Designs . . . . . 8

**3 Verification Strategy** **9**

3.1 Advanced Design . . . . . 9

3.2 Quartus Results . . . . . 9

3.3 Modelsim Results . . . . . 11

3.4 Results on kit DE2 . . . . . 11

3.5 Results on server . . . . . 12

3.5.1 Single Cycle . . . . . 13

3.5.2 Non-forwarding . . . . . 14

3.5.3 Forwarding . . . . . 15

3.5.4 Branch Prediction . . . . . 16

**4 Evaluation** **17**

**5 Conclusion** **18**

# List of Tables

Table 1 Comparison of Three Pipeline Designs . . . . . 8
Table 2 Comparison of pipeline performance metrics . . . . . 17

# List of Figures

Figure 1 Non-Forwarding Diagram . . . . . 5
Figure 2 Forwarding Diagram . . . . . 6
Figure 3 Two Bit Prediction Diagram . . . . . 7
Figure 4 Quartus Result of Single Cycle . . . . . 9
Figure 5 Quartus Result of Non-Forwarding . . . . . 10
Figure 6 Quartus Result of Forwarding . . . . . 10
Figure 7 Quartus Result of 2bit Branch Prediction . . . . . 10
Figure 8 Modelsim Result . . . . . 11
Figure 9 Results on kit DE2 . . . . . 11
Figure 10 Test Single Cycle on server . . . . . 13
Figure 11 Test Non-forwarding on server . . . . . 14
Figure 12 Test Forwarding on server . . . . . 15
Figure 13 Test Branch Prediction on server . . . . . 16

**MILESTONE III - COMPUTER ARCHITECTURE**

# 1 Introduction

Bộ xử lý đa giai đoạn (*pipelined processor*) là một cải tiến so với bộ xử lý đơn chu kỳ (*single-cycle processor*). Trong khi bộ xử lý đơn chu kỳ thực hiện mỗi lệnh trong một chu kỳ đồng hồ, dẫn đến hiệu suất hạn chế do độ trễ lệnh kéo dài, bộ xử lý đa giai đoạn chia quá trình thực thi lệnh thành nhiều giai đoạn (như lấy lệnh, giải mã, thực thi, truy cập bộ nhớ và ghi kết quả). Điều này cho phép xử lý đồng thời nhiều lệnh, tăng hiệu suất và tận dụng tài nguyên phần cứng hiệu quả hơn.

Trong bộ xử lý đa giai đoạn, các giai đoạn hoạt động đồng bộ, giúp duy trì luồng lệnh liên tục, cải thiện hiệu năng so với bộ xử lý đơn chu kỳ. Tuy nhiên, thiết kế này cũng gặp các thách thức như:

- **Mâu thuẫn dữ liệu (*data hazards*):** Do phụ thuộc giữa các lệnh.
- **Mâu thuẫn điều khiển (*control hazards*):** Do lệnh nhảy thay đổi luồng chương trình.
- **Mâu thuẫn cấu trúc (*structural hazards*):** Do tranh chấp tài nguyên phần cứng.

Để khắc phục, các kỹ thuật như *forwarding*, *branch prediction* và *hazard detection logic* được áp dụng, giúp giảm tác động của mâu thuẫn và duy trì hiệu suất cao.

**Tóm lại**, bộ xử lý đa giai đoạn cải thiện đáng kể hiệu suất và sử dụng tài nguyên hiệu quả hơn so với bộ xử lý đơn chu kỳ, mặc dù phải đối mặt với các thách thức cần được xử lý bằng các kỹ thuật tối ưu hóa phù hợp.

# 2 Design Strategy

## 2.1 Non-forwarding

Non-forwarding – một kiểu thiết kế đơn giản trong kiến trúc vi xử lý (CPU). Đây là kiến trúc không sử dụng cơ chế chuyển tiếp dữ liệu (data forwarding). Kiểu thiết kế này giúp giảm độ phức tạp của phần cứng nhưng có thể làm chậm hiệu suất do phải tạm dừng (stall) pipeline trong trường hợp xảy ra xung đột dữ liệu (data hazard).

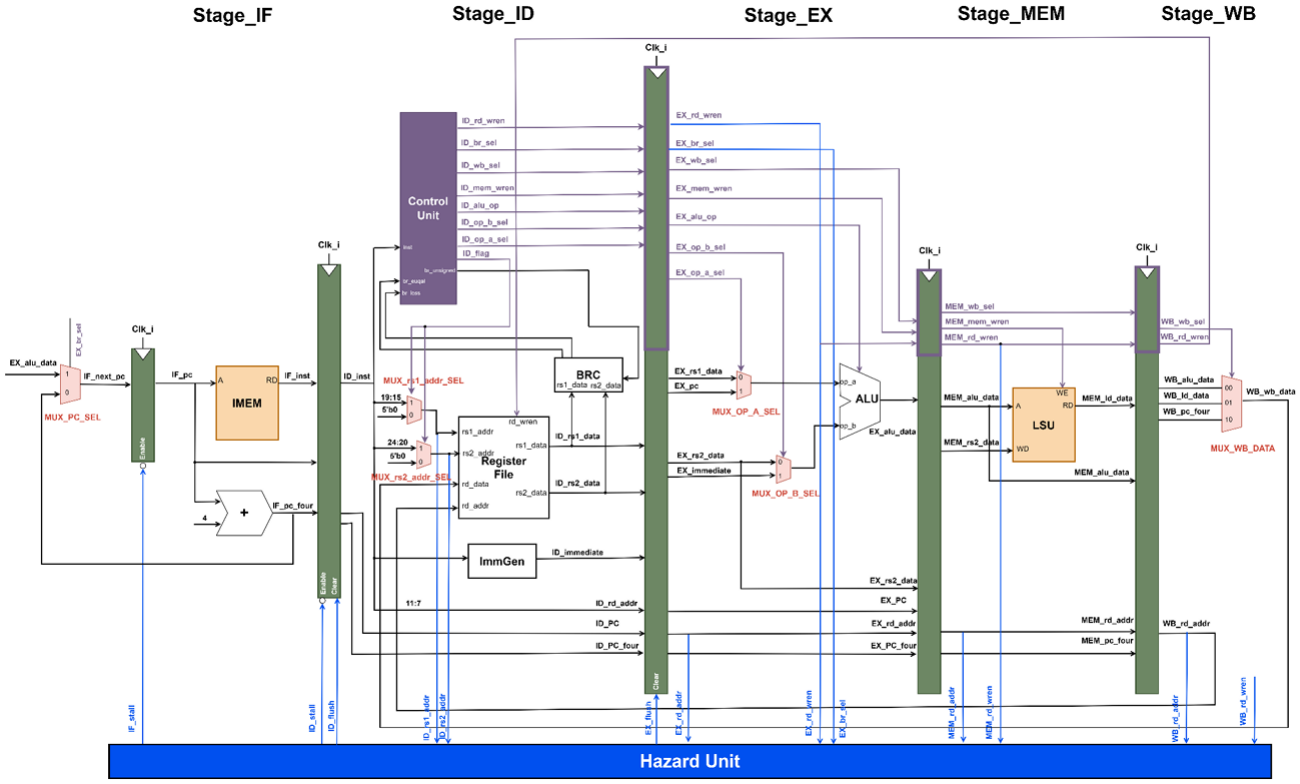


Figure 1: Non-Forwarding Diagram

Pipeline được chia thành 5 tầng giai đoạn chính như sau:

1. **IF (Instruction Fetch - Lấy lệnh)**: Truy xuất lệnh từ bộ nhớ.
2. **ID (Instruction Decode - Giải mã lệnh)**: Giải mã lệnh và truy xuất dữ liệu từ thanh ghi.
3. **EX (Execute - Thực thi)**: Thực hiện các phép toán số học, logic.
4. **MEM (Memory Access - Truy cập bộ nhớ)**: Đọc hoặc ghi dữ liệu vào bộ nhớ.
5. **WB (Write Back - Ghi lại)**: Ghi kết quả vào thanh ghi đích.

Bổ sung thêm HazardUnit để phát hiện các loại Hazard và bổ sung các tín hiệu cho các Pipeline_register (Stall, Flush)

## 2.2 Forwarding

Forwarding – một phương pháp được sử dụng để giảm thiểu xung đột dữ liệu (data hazard) trong quá trình xử lý lệnh của CPU. Khác với kiến trúc Non-forwarding, dữ liệu trong sơ đồ này được chuyển tiếp trực tiếp giữa các giai đoạn của pipeline mà không cần phải đợi đến khi dữ liệu được ghi vào thanh ghi. Điều này giúp cải thiện hiệu năng của pipeline và giảm số lần tạm dừng (stall).

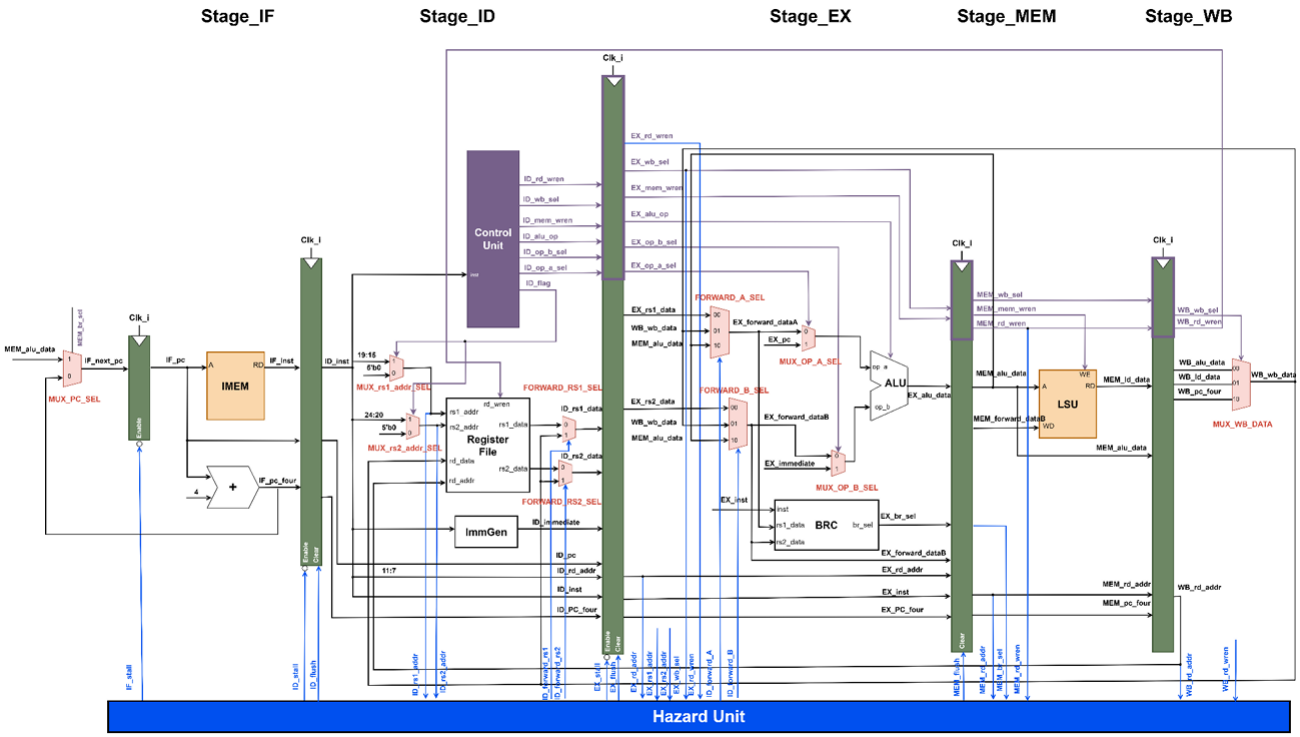


Figure 2: Forwarding Diagram

Pipeline với cơ chế **Forwarding** cũng chia thành 5 giai đoạn chính giống như **Non-Forwarding**.

Sự khác biệt chính nằm ở việc sử dụng các khối **Forwarding** để truyền dữ liệu giữa các giai đoạn khi cần thiết.

- Thực hiện di chuyển bộ **branch comparison** qua tầng **EX** nhằm phục vụ cho việc **forward data** từ tầng **MEM** và **WB** trở về.
- Thêm 2 bộ **multiplexer** (MUX) lựa chọn 1 trong 3 tín hiệu:
  - Tín hiệu **forward** từ **stage_MEM** trở về **MEM_alu_data**.
  - Tín hiệu **forward** từ **stage_WB** trở về **WB_wb_data**.
  - Tín hiệu **rsx_data** lấy từ **pipeline_register ID_EX**.
- Với tín hiệu lựa chọn được lấy từ bộ **Hazard Unit**: **ID_Forward_A**, **ID_Forward_B**.
- Bên cạnh đó, thực hiện thêm 2 bộ **MUX** ở cuối tầng **ID** nhằm **Forward rsx_data** từ tầng **WB** trở về trong trường hợp *inst* ở tầng **ID** cần kết quả ở tầng **WB**.

## 2.3 Branch Prediction

Khối Branch Prediction được thiết kế để cải thiện hiệu suất của pipeline bằng cách dự đoán trước hướng nhánh (branch direction) và địa chỉ mục tiêu của các lệnh nhánh (branch instructions). Việc này giúp giảm thiểu các chu kỳ bị tạm dừng (stall) do pipeline phải chờ xác định kết quả của lệnh nhánh.

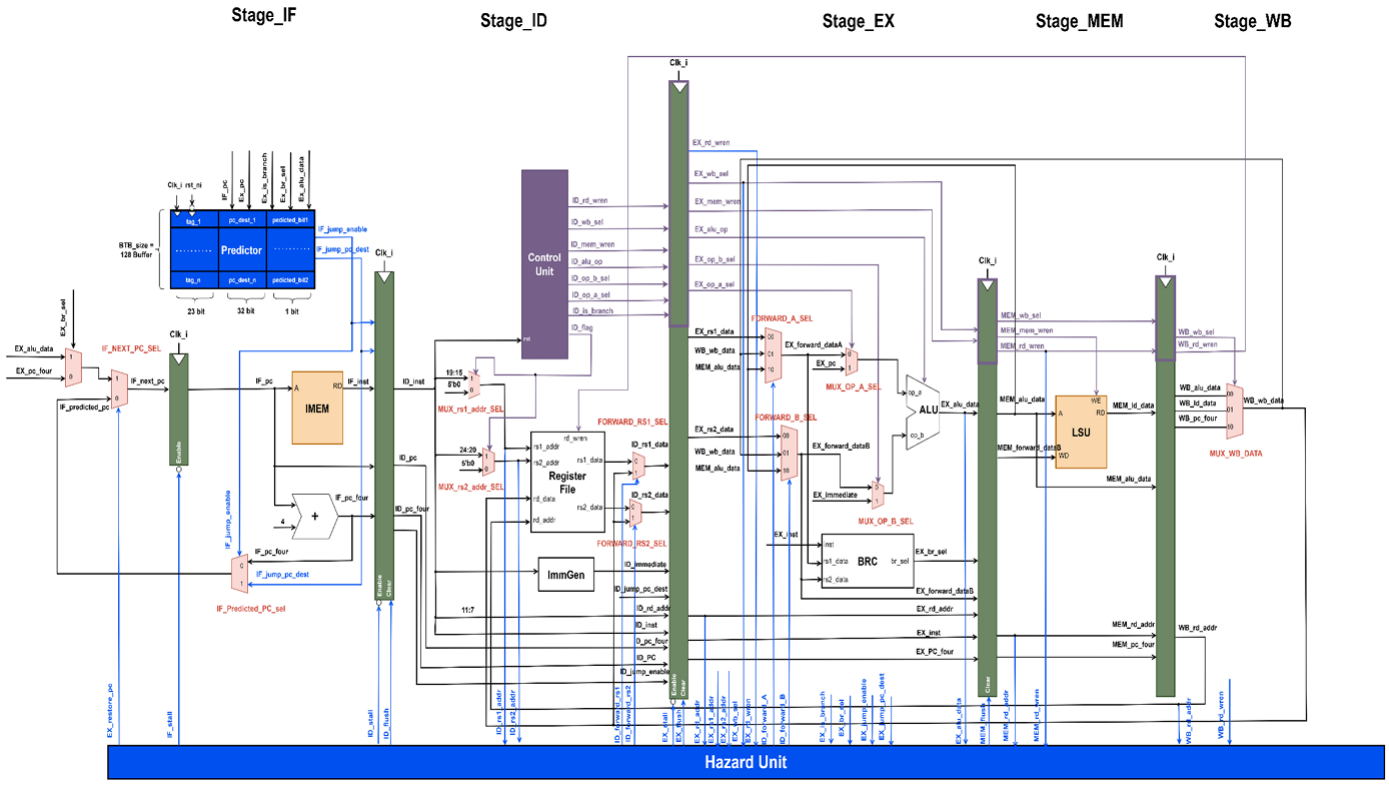


Figure 3: Two Bit Prediction Diagram

Khối **Branch Prediction** 2-bit sử dụng một bộ dự đoán (**Predictor**) để đưa ra quyết định:

- **Branch Taken**: Nếu dự đoán rằng lệnh nhánh sẽ được thực hiện, pipeline sẽ nhảy đến địa chỉ mục tiêu (branch target address).
- **Branch Not Taken**: Nếu dự đoán rằng lệnh nhánh không được thực hiện, pipeline tiếp tục thực thi tuần tự (PC + 4).

Dự đoán dựa trên lịch sử 2-bit của các lệnh nhánh trước đó để giảm thiểu lỗi dự đoán sai.

### 2.3.1 Branch Predictor

Là một bộ nhớ nhỏ chứa các thông tin dự đoán cho các lệnh nhánh trước đó, được tổ chức dưới dạng bảng (**Branch Target Buffer - BTB**)

### 2.3.2 2-bit Saturating Counter

- Mỗi lệnh nhánh được gán 2 bit dự đoán.
- Các trạng thái 2-bit:
  - **00 (Strongly Not Taken)**: Dự đoán mạnh là không nhánh.

  - **01 (Weakly Not Taken)**: Dự đoán yếu là không nhánh.
  - **10 (Weakly Taken)**: Dự đoán yếu là nhánh.
  - **11 (Strongly Taken)**: Dự đoán mạnh là nhánh.
- Cơ chế: Chuyển đổi trạng thái giữa các bit dựa trên kết quả của các lệnh nhánh trước đó.

## 2.4 Comparison of Three Theoretical Designs

| **Tiêu chí** | **Non-Forwarding Pipeline** | **Forwarding Pipeline** | **Branch Prediction Pipeline** |
|---|---|---|---|
| 1. Chức năng chính | Xử lý lệnh tuần tự, không có cơ chế chuyển tiếp dữ liệu giữa các giai đoạn. | Chuyển tiếp dữ liệu giữa các giai đoạn để giảm xung đột dữ liệu (data hazard). | Dự đoán nhánh để giảm thiểu các chu kỳ bị mất do lệnh rẽ nhánh. |
| 2. Xử lý xung đột dữ liệu | Tạm dừng pipeline (stall) hoặc chèn lệnh NOP khi có xung đột dữ liệu. | Sử dụng các khối forwarding để chuyển dữ liệu trực tiếp giữa các giai đoạn. | Không xử lý trực tiếp xung đột dữ liệu mà dự đoán trước hướng nhánh. |
| 3. Cơ chế chính | - Pipeline bị dừng chờ dữ liệu được ghi vào thanh ghi. | - Dữ liệu được chuyển tiếp từ các giai đoạn MEM hoặc WB sang EX hoặc ID. | - Sử dụng lịch sử nhánh (2-bit predictor) để dự đoán hướng nhánh. |
| 4. Phần cứng bổ sung | Không yêu cầu phần cứng đặc biệt. | Yêu cầu các khối Forwarding A/B và bộ Hazard Unit để kiểm tra xung đột. | Yêu cầu Branch Target Buffer (BTB) để lưu thông tin dự đoán và địa chỉ mục tiêu. |
| 5. Hiệu suất | Thấp nhất trong 3 khối do pipeline thường xuyên bị tạm dừng. | Cao hơn nhờ giảm số chu kỳ bị mất do xung đột dữ liệu. | Cao nhất khi dự đoán đúng, nhưng giảm hiệu suất khi dự đoán sai. |
| 6. Xử lý lệnh nhánh | Không có cơ chế đặc biệt, phải chờ tính toán hoàn tất. | Không có cơ chế đặc biệt, xử lý lệnh nhánh tuần tự. | Tối ưu hóa với dự đoán hướng nhánh (Taken hoặc Not Taken). |
| 7. Ưu điểm | - Đơn giản, dễ triển khai. | - Giảm đáng kể số chu kỳ stall. | - Giảm số chu kỳ bị mất do lệnh nhánh. |
| 8. Nhược điểm | - Hiệu suất thấp, phụ thuộc vào số lượng xung đột dữ liệu. | - Yêu cầu phần cứng phức tạp hơn (MUX, Hazard Unit, Forwarding logic). | - Cần phần cứng dự đoán và có khả năng sai gây mất hiệu suất. |

Table 1: Comparison of Three Pipeline Designs

# 3 Verification Strategy

Nhóm sẽ tiến hành viết testbench cho phần Advanced Design để test trên Modelsim.

## 3.1 Advanced Design

- Chương trình ứng dụng được lập trình bằng Assembly trên RISC-V và xuất ra mã hex (file "`instruction_memory.txt`") bằng web "https://venus.cs61c.org"
- Chương trình có các chứng năng:
  - Đọc dữ liệu từ công tắc (switch): Lấy giá trị từ công tắc và lưu vào thanh ghi.
  - Chuyển đổi dữ liệu nhị phân sang BCD: Chuyển giá trị nhị phân lấy từ công tắc thành mã BCD để dễ hiển thị từng chữ số.
  - Chuyển đổi BCD sang mã 7 đoạn: Chuyển đổi từng chữ số BCD thành mã để hiển thị trên màn hình 7 đoạn.
  - Xuất dữ liệu ra LED và màn hình 7 đoạn: Hiển thị kết quả lên LED và màn hình 7 đoạn để người dùng có thể thấy trực tiếp.

## 3.2 Quartus Results

| Flow Summary | |
|---|---|
| Flow Status | Successful - Thu Nov 14 19:29:16 2024 |
| Quartus II 64-Bit Version | 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | singlecycle |
| Top-level Entity Name | singlecycle |
| Family | Cyclone II |
| Total logic elements | 15,835 / 33,216 ( 48 % ) |
| Total combinational functions | 9,080 / 33,216 ( 27 % ) |
| Dedicated logic registers | 9,888 / 33,216 ( 30 % ) |
| Total registers | 9888 |
| Total pins | 419 / 475 ( 88 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 483,840 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 70 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |
| Device | EP2C35F672C6 |
| Timing Models | Final |

Figure 4: Quartus Result of Single Cycle

| Flow Summary | |
|---|---|
| Flow Status | Successful - Sun Dec 15 17:01:04 2024 |
| Quartus II 64-Bit Version | 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | wrapper |
| Top-level Entity Name | wrapper |
| Family | Cyclone II |
| Device | EP2C35F672C6 |
| Timing Models | Final |
| Total logic elements | 14,495 / 33,216 ( 44 % ) |
| Total combinational functions | 9,113 / 33,216 ( 27 % ) |
| Dedicated logic registers | 10,254 / 33,216 ( 31 % ) |
| Total registers | 10254 |
| Total pins | 115 / 475 ( 24 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 483,840 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 70 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

Figure 5: Quartus Result of Non-Forwarding

| Flow Summary | |
|---|---|
| Flow Status | Successful - Fri Dec 13 00:56:08 2024 |
| Quartus II 64-Bit Version | 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | wrapper |
| Top-level Entity Name | wrapper |
| Family | Cyclone II |
| Device | EP2C35F672C6 |
| Timing Models | Final |
| Total logic elements | 14,133 / 33,216 ( 43 % ) |
| Total combinational functions | 9,261 / 33,216 ( 28 % ) |
| Dedicated logic registers | 10,196 / 33,216 ( 31 % ) |
| Total registers | 10196 |
| Total pins | 115 / 475 ( 24 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 483,840 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 70 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

Figure 6: Quartus Result of Forwarding

| Flow Summary | |
|---|---|
| Flow Status | Successful - Fri Dec 13 00:53:09 2024 |
| Quartus II 64-Bit Version | 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | wrapper |
| Top-level Entity Name | wrapper |
| Family | Cyclone II |
| Device | EP2C35F672C6 |
| Timing Models | Final |
| Total logic elements | 15,168 / 33,216 ( 46 % ) |
| Total combinational functions | 10,312 / 33,216 ( 31 % ) |
| Dedicated logic registers | 10,519 / 33,216 ( 32 % ) |
| Total registers | 10519 |
| Total pins | 115 / 475 ( 24 % ) |
| Total virtual pins | 0 |
| Total memory bits | 7,040 / 483,840 ( 1 % ) |
| Embedded Multiplier 9-bit elements | 0 / 70 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

Figure 7: Quartus Result of 2bit Branch Prediction

## 3.3 Modelsim Results

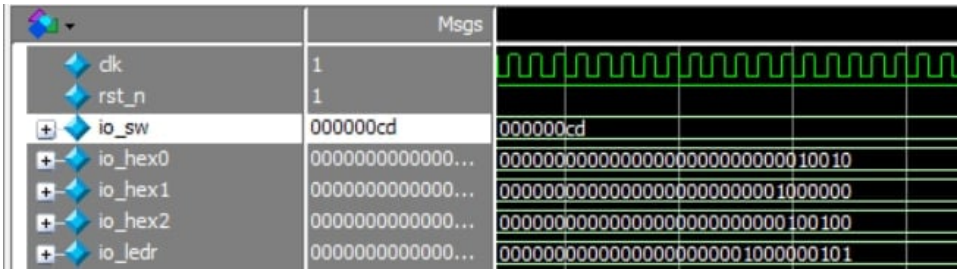


Figure 8: Modelsim Result

Ta gán giá trị Input Switch trong Modelsim là 0xCD (Hex), tương đương 205 (Decimal).

- **LED 7 đoạn (hex0, hex1, hex2):**
  Số "205" được tách thành các chữ số:
  - **hex2 (số 2)**
  - **hex1 (số 0)**
  - **hex0 (số 5)**
- **LED_RED:**
  Số "205" được mã hóa sang **BCD** (Binary-Coded Decimal):
  - Hàng trăm (2) → `0010`.
  - Hàng chục (0) → `0000`.
  - Hàng đơn vị (5) → `0101`.

  Ghép lại thành: `0010 0000 0101`.
- **Kết quả:**
  - LED 7 đoạn hiển thị trực tiếp "205".
  - LED_RED hiển thị mã BCD của "205".

## 3.4 Results on kit DE2

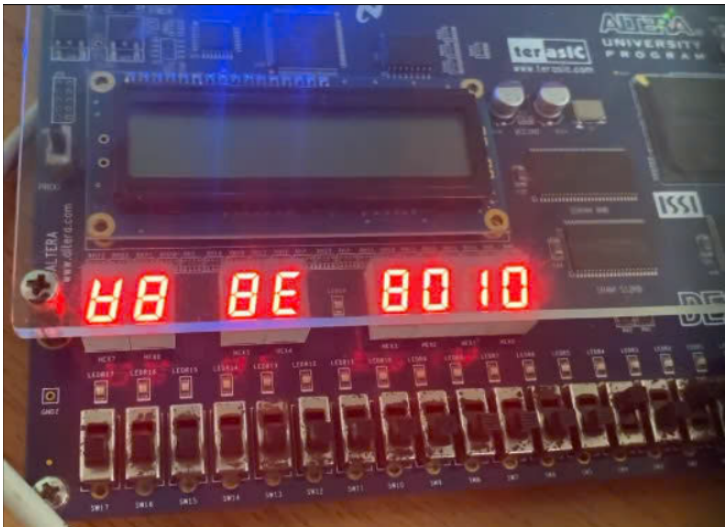


Figure 9: Results on kit DE2

## 3.5 Results on server

Ta thêm các lệnh Count_cycle tính Cycle ở từng khối để tính $IPC$.
Thêm Count_jump và Count_miss ở khối Branch Prediction để tính $P_{br_{miss}}$.

- **Cách kiểm tra Count_cycle**: theo dõi số chu kỳ clock mà hệ thống đã thực thi trong quá trình mô phỏng.
  Kiểm tra điều kiện `if (pc_WB == 32'h1680)`:
  - Điều kiện này kiểm tra xem giá trị của **PC (Program Counter)** tại giai đoạn **Writeback (WB)** có bằng `32'h1680` (địa chỉ PC cuối tại scoreboard) hay không.
  - Nếu không đạt điều kiện trên (tức là `pc_WB` không phải là `0x1680`), chương trình sẽ tăng giá trị `count_cycle` **lên 1** mỗi chu kỳ clock: `count_cycle <= count_cycle + 1`.

- **Cách kiểm tra Count_jump**: Đếm tổng số các lần nhảy (jump hoặc branch instruction) mà bộ xử lý đã thực hiện.
  - Biến `count_jump` được tăng lên mỗi khi một instruction là nhánh (branch instruction) được phát hiện trong chu kỳ `clk_i`.
  - Điều kiện kiểm tra là: nếu instruction tại `instr_EX_i` có mã opcode là nhánh (branch, `OPCODE_BRANCH`), hoặc là JAL (jump and link) hoặc JALR (jump and link register), thì `count_jump` sẽ được tăng lên 1.
  - `count_jump <= count_jump + 1;` — Điều này có nghĩa là mỗi khi có một nhánh, bộ đếm sẽ tăng thêm 1.

- **Cách kiểm tra Count_miss**: Đếm số lần mà bộ dự đoán nhánh (Branch Target Buffer - BTB) đã dự đoán sai.
  - `count_miss` tăng lên mỗi khi có một nhánh và việc dự đoán nhánh của BTB là sai.
  - Điều kiện kiểm tra là: nếu địa chỉ pc_EX_i (PC tại phase execute) không trùng với tag trong BTB tại vị trí chỉ mục (`index_W`), tức là dự đoán về nhánh đã sai, thì `count_miss` sẽ được tăng lên 1.
  - `if (pc_EX_i[31:9] != tag[pc_EX_i[8:2]])` — Điều kiện này kiểm tra xem phần tag (phần trên của địa chỉ `pc_EX_i`) có khớp với tag trong bảng dự đoán nhánh tại chỉ mục `index_W` hay không. Nếu không khớp, tức là dự đoán sai.
  - Sau khi kiểm tra, `count_miss <= count_miss + 1;` — Tăng biến đếm số lần dự đoán sai.

### 3.5.1 Single Cycle

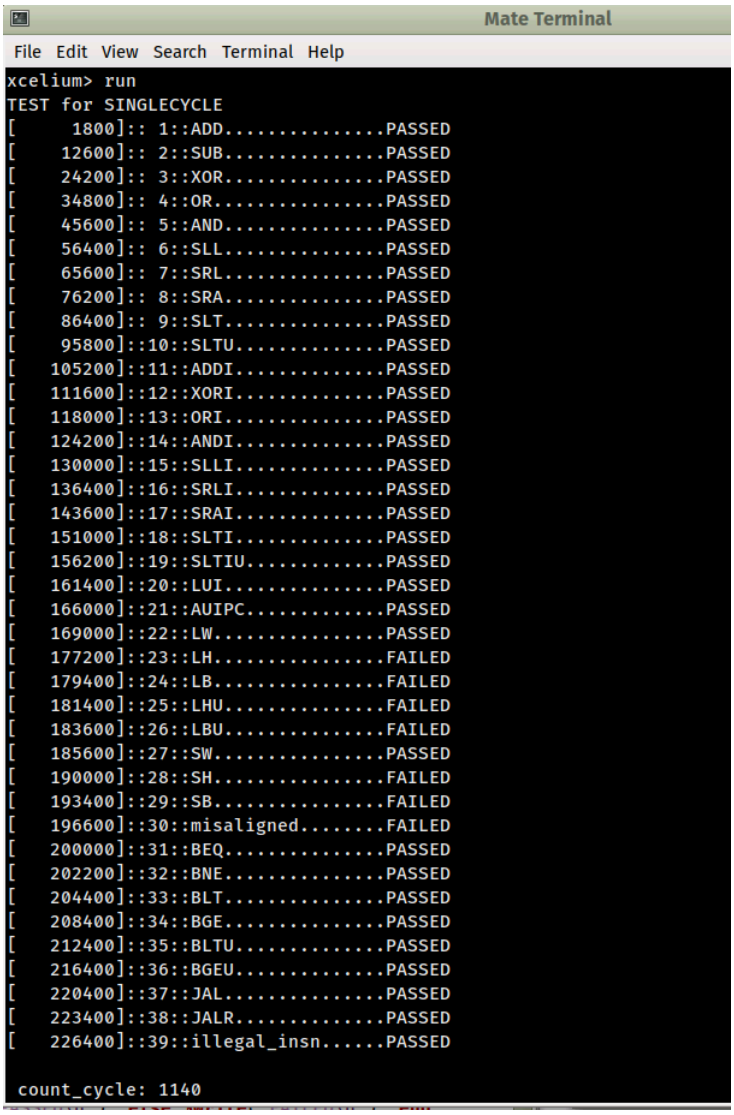


Figure 10: Test Single Cycle on server

### 3.5.2 Non-forwarding

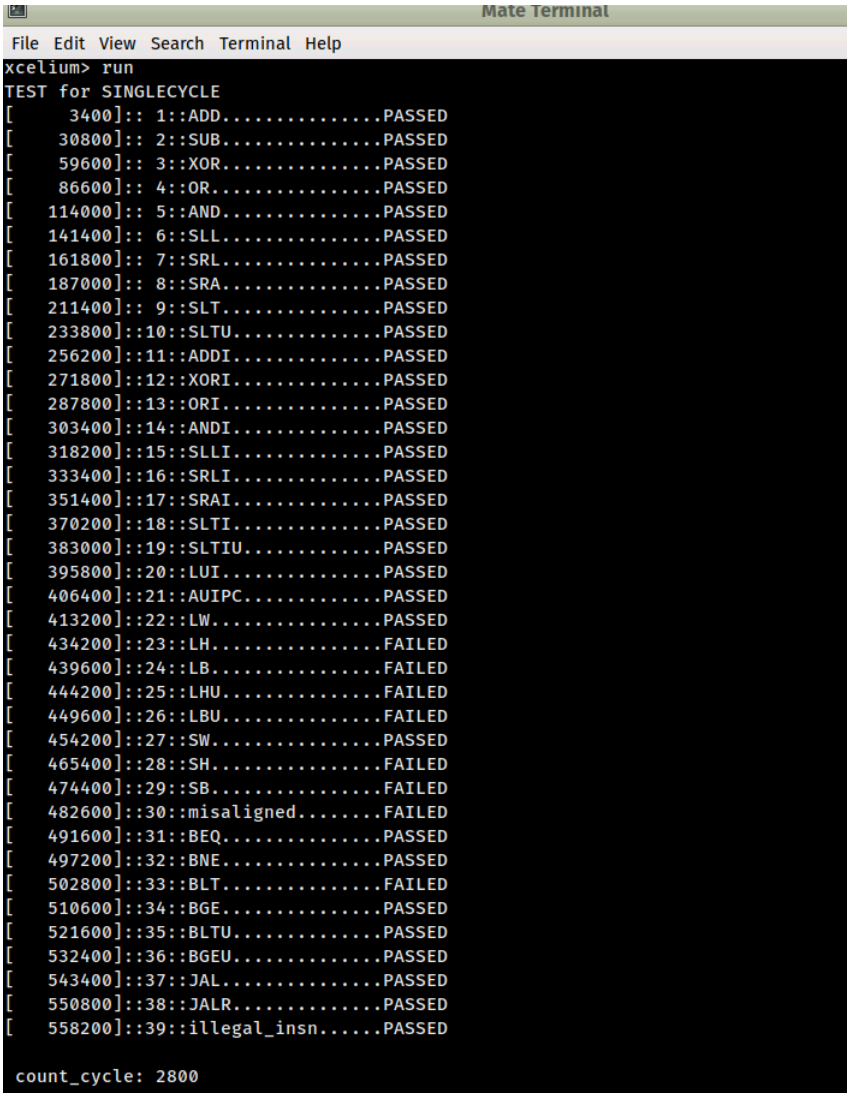


Figure 11: Test Non-forwarding on server

### 3.5.3 Forwarding

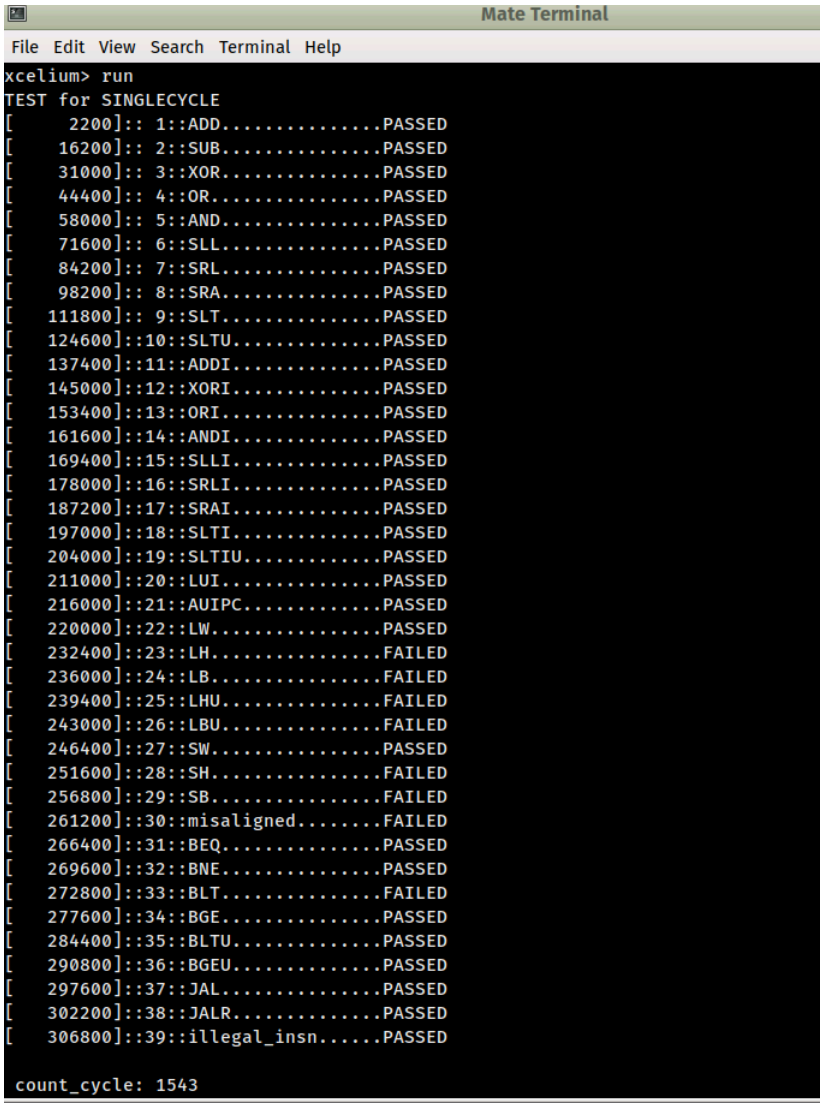


Figure 12: Test Forwarding on server

### 3.5.4 Branch Prediction

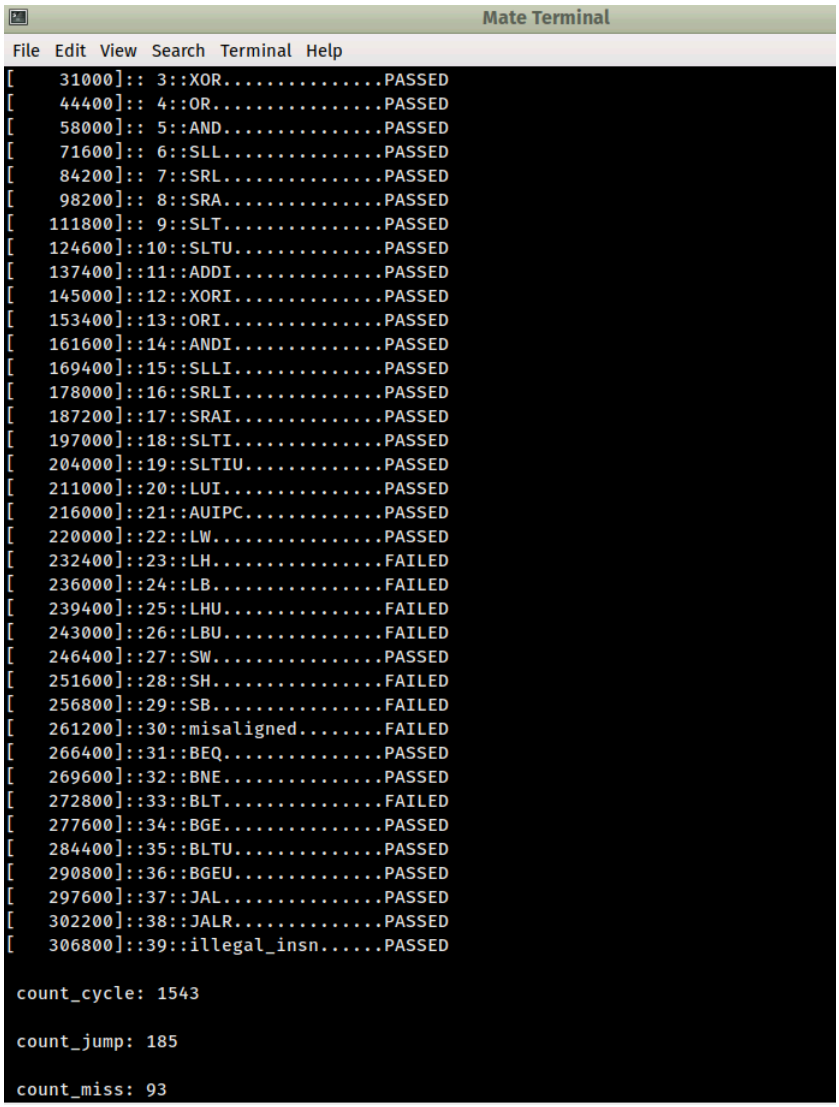


Figure 13: Test Branch Prediction on server

# 4 Evaluation

Ở trên ta đo được Cycle của Single Cycle là 1140 tương đương với lượng lệnh mà các khối pipeline chạy.

| **Pipeline** | **Cycle Count** | **Instruction Count** | **IPC** | $P_{br_{miss}}$ |
|---|---|---|---|---|
| Non-Forwarding | 2800 | 1140 | $\approx 0.407$ | N/A (No branch prediction) |
| Forwarding | 1543 | 1140 | $\approx 0.739$ | N/A (No branch prediction) |
| Branch Prediction | 1543 | 1140 | $\approx 0.739$ | $\frac{93}{185} \times 100 \approx 50.27\%$ |

Table 2: Comparison of pipeline performance metrics

Dựa vào bảng tóm tắt ta thấy:

- Hiệu năng xử lý (IPC)
  - **Non-Forwarding Pipeline (IPC ≈ 0.407):**
    - Đây là pipeline có hiệu suất thấp nhất do thời gian xử lý lâu hơn, số chu kỳ (Cycle Count) cao.
    - Lý do: Không có cơ chế forwarding, dẫn đến nhiều lần pipeline bị tạm dừng (stall) khi xảy ra xung đột dữ liệu.
  - **Forwarding Pipeline (IPC ≈ 0.739):**
    - Hiệu suất tăng đáng kể so với Non-Forwarding nhờ cơ chế chuyển tiếp dữ liệu (forwarding), giảm số chu kỳ bị mất do xung đột.
    - Kết quả cho thấy đây là cải tiến cần thiết cho hệ thống yêu cầu hiệu năng cao.
  - **Branch Prediction Pipeline (IPC ≈ 0.739):**
    - IPC tương tự Forwarding vì chu trình xử lý dữ liệu không thay đổi, nhưng có thêm cơ chế dự đoán nhánh.
    - Ưu điểm chính là giảm thiểu chu kỳ bị mất do lệnh nhánh, đặc biệt với các chương trình chứa nhiều nhánh.
- Tỷ lệ dự đoán nhánh sai ($P_{br_{miss}} \approx 50.27\%$)
  - **Branch Prediction Pipeline có tỷ lệ sai dự đoán nhánh ở mức trung bình (50.27%):** Điều này cho thấy mô hình dự đoán nhánh cần cải thiện, vì 1/2 số lần dự đoán là sai.
  - Sai dự đoán dẫn đến việc phải xóa (flush) pipeline và khởi động lại, làm giảm hiệu năng.

# 5 Conclusion

Bộ **Pipeline Processor** đã được thiết kế và triển khai thành công, đáp ứng được các yêu cầu về xử lý. Qua việc kiểm tra và mô phỏng, ta đã đảm bảo tính chính xác và hoạt động đúng của bộ pipeline. Hệ thống đã hoạt động một cách ổn định và đáp ứng được các yêu cầu xử lý.

- Non-Forwarding Pipeline có hiệu năng thấp nhất và chỉ phù hợp cho các hệ thống đơn giản, không yêu cầu hiệu suất cao.
- Forwarding Pipeline là bước cải tiến rõ rệt, giảm số lần pipeline bị stall, nâng cao IPC.
- Branch Prediction Pipeline là lựa chọn tối ưu hơn khi chương trình có nhiều lệnh nhánh, tuy nhiên nhóm cần cải phải thiện tỷ lệ dự đoán sai để phát huy tối đa tiềm năng.