**ĐẠI HỌC QUỐC GIA TP.HCM**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**🙝🟑🙟**

**A blue and white logo

Description automatically generated**

**BÁO CÁO**

Cấu trúc máy tính

**MILESTONE 3:**

**Pipelining**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Huỳnh Bảo Duy | 211078 |
| Trịnh Dương Quốc Hiếu | 211196 |

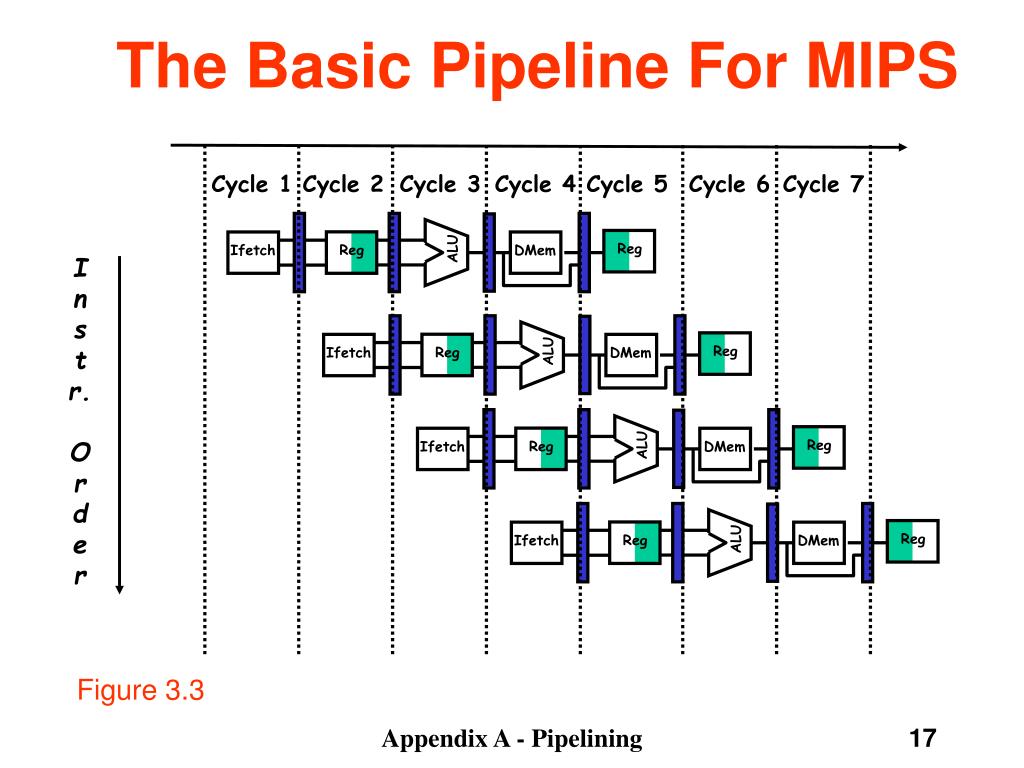
**TP.HCM, tháng 5 năm 2024**

1. **Giới thiệu**
   1. **Mô hình xử lý ống - pipelining**

Xử lý ống (pipelining) trong bộ xử lý là một trong những kỹ thuật hiện thực cho phép nhiều lệnh được thực thi một cách đồng thời và chồng lấn lên nhau. Kỹ thuật xử lý ống ngày nay được áp dụng không chỉ trong thiết kế và hiện thực các bộ xử lý mà còn ở rất nhiều lĩnh vực trong đời sống.

Mô hình xử lý ống thực hiện xử lý nhiều lệnh song song nhau, trong đó mỗi lệnh trong nhóm các lệnh đang thực thi lệnh song song sẽ được xử lý ở một giai đoạn khác nhau. Các giai đoạn xử lý một lệnh vẫn là năm như trong mô hình đơn giản, lần lượt theo trình tự là:

* **Giai đoạn đọc lệnh (IF):** đọc lệnh từ bộ nhớ lệnh.
* **Giai đoạn giải mã lệnh (ID):** giải mã lệnh và truy xuất các thanh ghi.
* **Giai đoạn thực thi lệnh (EX):** thực thi tác vụ của lệnh hoặc tính toán địa chỉ bộ nhớ.
* **Giai đoạn truy xuất bộ nhớ (MEM):** Truy xuất toán hạng bộ nhớ.
* **Giai đoạn cập nhật kết quả (WB):** cập nhật giá trị thanh ghi trong tập thanh ghi.



Có thể thấy rằng, so với mô hình tuần tự thì thời gian hoàn thành tất cả các tải của mô hình xử lý ống (pipeling) là nhanh hơn. Một cách tổng quát, khi số lượng tải đủ lớn và thời gian thực thi các công việc trong một quy trình là như nhau thì thời gian hoàn thành các tải của mô hình xử lý ống sẽ nhanh hơn mô hình tuần tự n lần ( hay còn gọi là độ tăng tốc – speed-up), với n là số lượng tầng của mô hình xử lý ống.

* 1. **Xây dựng đường dữ liệu**

A diagram of a machine

Description automatically generatedKhi áp dụng mô hình pipeling, lệnh và dữ liệu phải được giữ cục bộ và không đổi trong suốt chu kỳ để csac khối chức năng có thể tính toán dữ liệu chính xác của chu kỳ đó. Vì vậy, cần phải đặt các thanh ghi ( là các Flip-Flop) ở trước các giai đoạn để đảm bảo rằng dữ liệu ngõ ra của giai đoạn sẽ không bị ảnh hưởng bởi dữ liệu ngõ ra của giai đoạn ngay phía trước trong suốt chu kỳ. Đầu mỗi chu kỳ khi xuất hiện cạnh của xung nhịp thì các thanh ghi này sẽ cho phép dữ liệu ngõ ra của giai đoạn trước được đi ngang qua và làm đàu vào cho giai đoạn tiếp theo. Các thanh ghi này được gọi là các “thanh ghi giai đoạn” hay các “thanh ghi xử lý ống” (pipeline register). Tên của các thanh ghi là sự kết hợp của hai giai đoạn mà nó phân chia. Có tất cả 4 thanh ghi được sử dụng (kích thước các thanh ghi là khác nhau tùy thuộc vào độ rộng các đường dữ liệu) lần lượt là IF/ID, ID/EX, EX/MEM và MEM/WB.

* 1. **Các rủi ro trong pipelining**

Các phân tích và ví dụ cho mô hình xử lý ống ở trên đã cho thấy sức mạnh của việc thực thi các lệnh song song nhau theo mô hình pipelining. Tuy nhiên, khác với mô hình bộ xử lý đơn giản, mô hình pipelining tồn tại nhiều rủi ro (hazard) khác nhau. Rủi ro trong pinelining là tình trạng xuất hiện một sự kiện không mong muốn làm cản trở quá trình thực thi một cách bình thường của mô hình xử lý ống. Trong mô hình pinelining, tồn tại ba dạng rủi ro về cấu trúc (structural hazard), rủi ro về dữ liệu (data hazard) và rủi ro về điều khiển (control hazard).

* + 1. **Structural hazard**

Structural hazard xảy ra khi tài nguyên cần dùng để tính toán hay xử lý cho một giai đoạn nào đó chưa sẵn sàng để phục vụ cho giai đoạn đó. Cách giải quyết đó chính là thay đổi bộ reg file cho phép ghi ở xung clock cạnh xuống.

* + 1. **Data hazard**

Data hazard xảy ra khi quá trình pipelining không thể tiếp tục thực thi do một giai đoạn nào đó cần phải chờ giai đoạn khác hoàn thành và cung cấp dữ liệu để tiến hành xử lý.

Giả sử ta có hai trường hợp:

1. **add x2,x3,x4**

**sub x4,x2,x5**

**or x5,x2,x6**

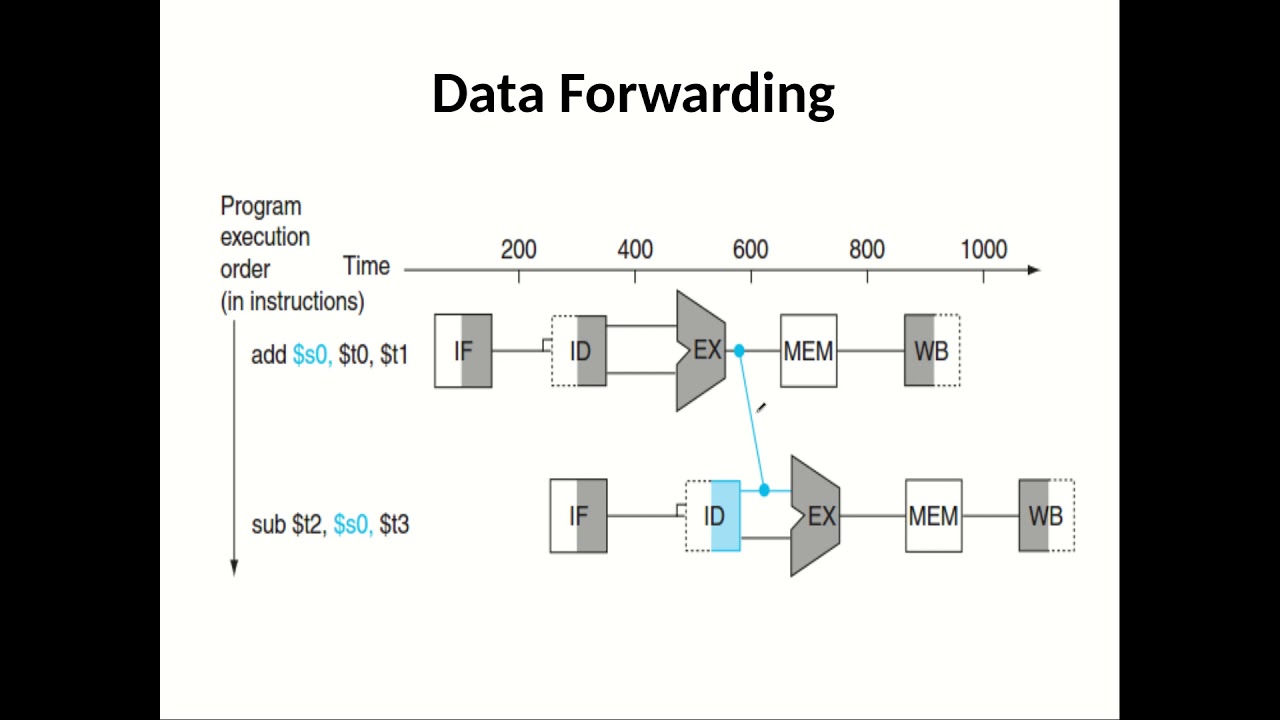
1. **lw x2,0(x31)**

**add x3,x2,x4**

Ở trường hợp a, dữ liệu được viết vào x2 ở WB stage ( cycle n), trong khi ở câu lệnh sub, dữ liệu thanh ghi x2 được đọc ở .

Cách giải quyết:

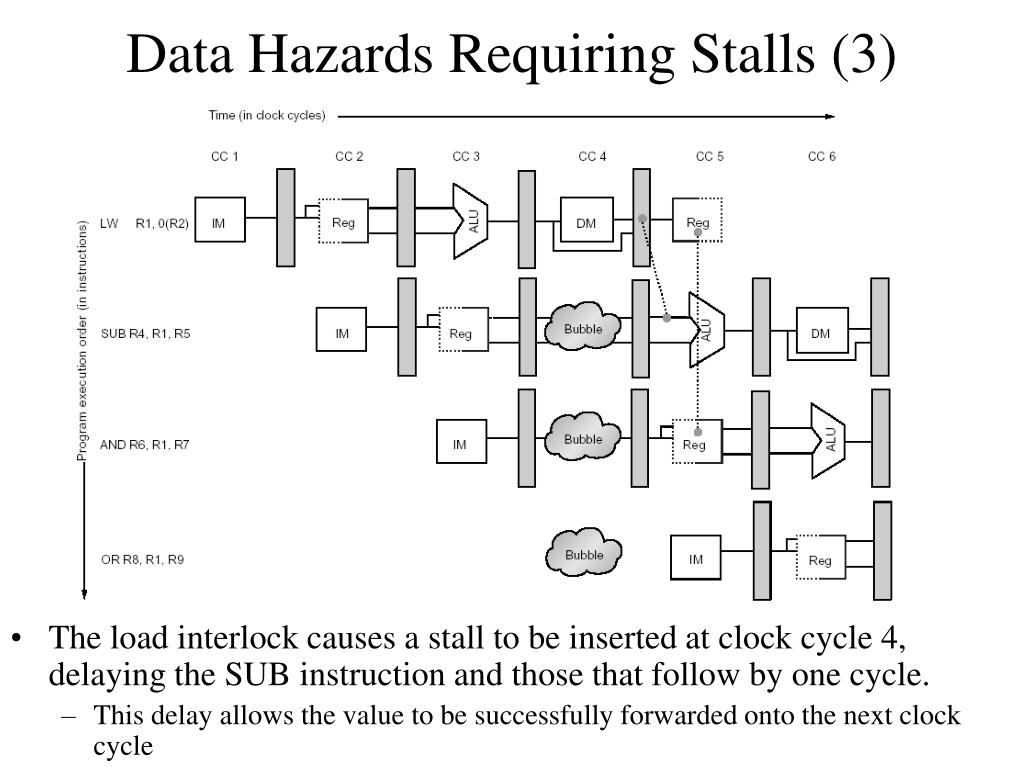
1. **Forwarding (bypass) đối với phép toán thông thường:**

Phương pháp Forwarding là phương pháp sử dụng dữ liệu ngay khi nó được sinh ra thay vì phải chờ cho đến khi dữ liệu được cập nhật vào tập thanh ghi rồi lại được đọc ra từ thanh ghi.

Trở lại với ví dụ trên, thay vì phải chờ cho đến hết một chu kỳ để cập nhật lại giá trị vào thanh ghi $s0 để tiếp tục tính toán thì ta sẽ lấy giá trị vừa được sinh ra sau giai đoạn EX để tính toán cho phép toán thứ hai trước khi câu lệnh hai thực hiện giai đoạn EX.

1. **Stall đối với load use data**

Xét trường hợp đoạn chương trình sau:



Theo ví dụ trên thì lệnh sub sẽ lấy giá trị R1 được lấy từ lệnh lw trước đó để tính toán. Tuy nhiên ta có thể thấy rằng dữ liệu sẽ được cập nhật vào thanh ghi R1 chỉ sẵn sàng ở cuối chu kỳ 4 tức trong giai đoạn MEM/WB trong khi lệnh sub sử dụng giá trị này ở đầu chu kỳ 4. Vì vậy ta dùng phương pháp Stall trong trường hợp này để giải quyết rủi ro.

Phương pháp Stall giải quyết được tất cả các rủi ro về dữ liệu bằng cách dừng sự thực thi của pipelining lại cho đến khi không còn rủi ro về dữ liệu. Nói cách khác, khi hai lệnh có quan hệ dữ liệu với nhau và có rủi ro về dữ liệu thì việc thực thi lệnh sử dụng dữ liệu sẽ bị hoãn lại (delay/stall) sao cho giai đoạn giải mã lệnh sử dụng dữ liêu trùng hoặc trễ hơn so với giai đoạn cập nhật kết quả của lệnh sinh ra dữ liệu nhằm bảo đảm rằng dữ liệu được truy suất trong lệnh sử dụng dữ liệu là dữ liệu chính xác.

* + 1. **Control hazard**

Control hazard xảy ra khi bộ xử lý không biết chắc chắn rằng lệnh nào sẽ được thực thi tiếp theo. Control hazard chỉ xảy ra khi bộ xử lý vừa đọc lệnh rẽ nhánh có điều kiện. Vì kết quả so sánh hai thanh ghi trong lệnh rẽ nhánh có điều kiện chỉ có sau giai đoạn EX, trong khi đó bộ xử lý phải đọc lệnh tiếp theo trong chu kỳ tiếp theo sau khi đã đọc rẽ nhánh có điều kiện. Tại thời điểm này, vì kết quả so sánh trong lệnh rẽ nhánh có điều kiện chữa sẵn sàng nên chưa thể xác định được lệnh sẽ được thực thi tiếp theo là lệnh ở địa chỉ đích của lệnh rẽ nhánh hay lệnh ngay sau lệnh rẽ nhánh.

Cách giải quyết:

* **Always taken module**: PC luôn nhảy cho đến khi phát hiện lỗi thì sửa lại địa chỉ PC
* **One bit prediction**: Từ mô hình always taken kết hợp với bộ máy trạng thái dự đoán 1 bit gia tăng xác suất dự đoán địa chỉ ở stage IF.
* **Two bit prediction**: thay thế bộ máy dự đoán trạng thái 1 bit thành 2 bit, gia tăng xác suất dự đoán địa chỉ ở stage IF. Cụ thể là ở các chương trình có câu lệnh LOOP Nest. Cả 3 module đều cho 2 cycle/penaty.

A diagram of a machine

Description automatically generatedNhư vậy ta sẽ có sơ đồ hoàn chỉnh:

1. Thiết kế
   1. Forwarding

Khối hazard unit có nhiệm vụ: Stall và flush các thanh ghi thông qua chân enable và clear của các thanh ghi bằng thuật toán:

1. Đối với lệnh thông thường:

if(regwrite\_MEM\_i & (rd\_MEM\_i != 0) & (rd\_MEM\_i == rs1\_EX\_i))

forwardA\_o = 2'b10;

else if (regwrite\_WB\_i & (rd\_WB\_i !=0 ) & (rd\_WB\_i == rs1\_EX\_i) & !(regwrite\_MEM\_i & (rd\_MEM\_i != 0) & (rd\_MEM\_i == rs1\_EX\_i)))

forwardA\_o = 2'b01;

else forwardA\_o = 2'b00;

Trong đó: chân forwardA\_o có nhiệm vụ cung cấp select signal cho bộ MUX\_3 chọn dữ liệu vào ngõ vào thứ nhất của ALU ở EX stage.

* forwardA\_o = 00: Chọn dữ liệu từ bộ reg file.
* forwardA\_o = 01: Chọn dữ liệu từ ALU\_out ở WB stage.
* forwardA\_o = 10: Chọn dữ liệu từ ALU\_out ở MEM stage.

Đối với ngõ vào thứ hai của ALU, ta làm tương tự.

1. Đối với load using.

if(((rs1\_D\_i == rd\_EX\_i) | (rs2\_D\_i == rd\_EX\_i)) & (rd\_EX\_i !=0) & wb\_sel\_EX\_i == 2'b00)

lwstall\_o = 1'b1;

else lwstall\_o = 1'b0;

Trong đó: lwstall\_o có nhiệm vụ cung cấp tín hiệu Stall cho PC và thanh ghi IF/ID và cung cấp tín hiệu flush cho thanh ghi ID/EX khi xảy ra hazard.

1. Control hazard

Hazard unit sẽ cho tín hiệu Stall khi gặp load use data hazard và các lệnh branch.

Thiết kế PC luôn update address +4 cho tới khi phát hiện lệnh nhảy từ câu lệnh nhảy gần nhất sẽ thay đổi (hoặc không) địa chỉ câu lệnh.

Hazard unit có tín hiệu đầu vào PC taken ở EX stage xác định lệnh có nhảy hay không từ đó đưa ra các output xóa hai câu lệnh data (NOP) qua chân flushE và flushD

Thiết kế thêm bộ cộng để tính địa chỉ nhảy ở trong EX stage. Kết hợp với các tín hiệu ngõ ra của tín hiệu hazard unit để có thể update địa chỉ nhảy đúng ngay sau khi phát hiện lệnh nhảy.

stallF\_o = lwstall\_o;

stallD\_o = lwstall\_o;

flushE\_o = lwstall\_o | PCsel;

flushD\_o = PCsel

A diagram of a machine

Description automatically generated

Sơ đồ khối của model Forwarding

Mô phỏng model khi nạp application code:

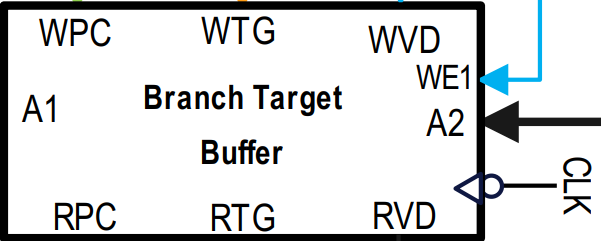
A screenshot of a computer

Description automatically generated

IPC của model = = 0.879

* 1. Dynamic prediction

Từ thiết kế forwarding, kết hợp thêm table BTB ở IF stage, BTB là 1 bảng gồm 1024 (10 bit ngõ vào) dòng, với mỗi dòng gồm 20 bit của tag, 1 bit valid và 32 bit địa chỉ nhảy tới.



**WPC**: Chân viết dữ liệu PC\_target

**WTG**: Chân viết dữ liệu tag (20 bit cao của PC)

**WVD**: Chân viết vào tín hiệu valid, được cấp từ tín hiệu BranchE từ khối ctrl\_unit ở EX stage

**WE1**: Chân enable , cho phép ghi vào **BTB** khi có xung clk cạnh xuống, WE1 và WVD đều lấy tín hiệu từ tín hiệu BranchE

**A1**: Địa chỉ viết dữ liệu

**A2**: địa chỉ đọc dữ liệu

**RPC:** Ngõ ra dữ liệu PC\_target

**RTG:** Ngõ ra dữ liệu Tag

**RVD:** Ngõ ra dữ liệu Valid

**Dự đoán địa chỉ PC**

Bảng BTB có vai trò lưu giữ lại tín hiệu cho phép biết được địa chỉ câu lệnh đó có phải là B-format instruction hay không. Từ đó ở IF stage có thể đưa ra dự đoán về địa chỉ tiếp theo đưa vào PC thông qua singal **Hit** , việc kiểm tra tính đúng đắn của dự đoán sẽ được xác định ở stage EX khi tín hiệu **PC\_taken** – kiểm tra tính đúng sai của lệnh Branch được xác định. Ta so sánh PC\_taken và Hit, nếu chúng khác nhau thì dự đoán ở stage IF sai và tiến hành stall 2 chu kỳ như mô hình forwarding và tiến hành sửa lại địa chỉ ở PC.

Việc kiểm tra tính đúng sai của dự đoán sẽ được thực hiện ở khối hazard\_detection\_unit và kết quả sẽ đưa ra chân Pcsel\_F và AddrFix\_F.

PcselF: dùng để chọn luồng dữ liệu địa chỉ vào PC, Pcsel\_F =1 , chọn ở luồng địa chỉ sửa, Pcsel\_F = 0, chọn ở luồng địa chỉ dự đoán

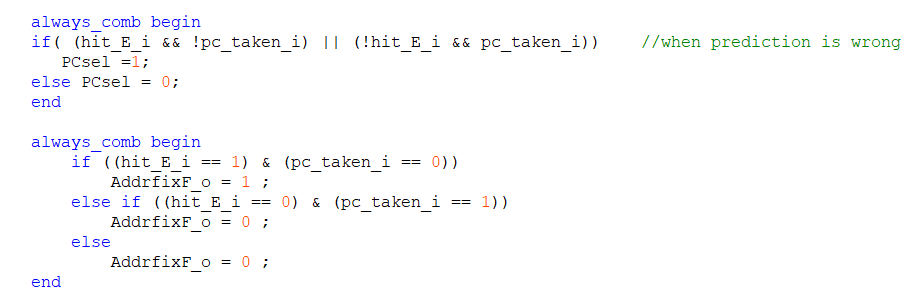
|  |  |  |  |
| --- | --- | --- | --- |
| HIt | Pc\_taken | Pc\_sel | AddrFix |
| 0 | 0 | 0 | x |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | x |

Bảng hành vi ở hazard\_detection\_unit

Pcsel\_F là chân sel cho mux 2 với 2 ngõ vào là địa chỉ dự đoán và địa chỉ sửa lại

AddrFix là chân sel cho mux 2 với 2 ngõ vào là địa chỉ sửa lại của 2 trường hợp, (đoán nhảy + không nhảy ) và (đoán không nhảy + nhảy).

Các chân Pcsel\_F và AddrFix được xác định theo thuật toán:



A diagram of a machine

Description automatically generated

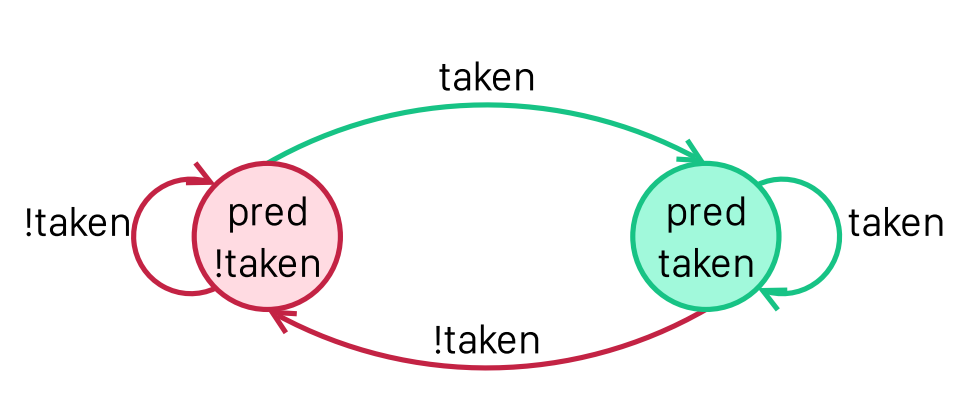
Sơ đồ khối khi có thêm BTB

Từ sơ đồ khối có BTB, ta phát triển khả năng dự đoán ở stage IF bởi bộ máy trạng thái 1 bit và 2 bit.

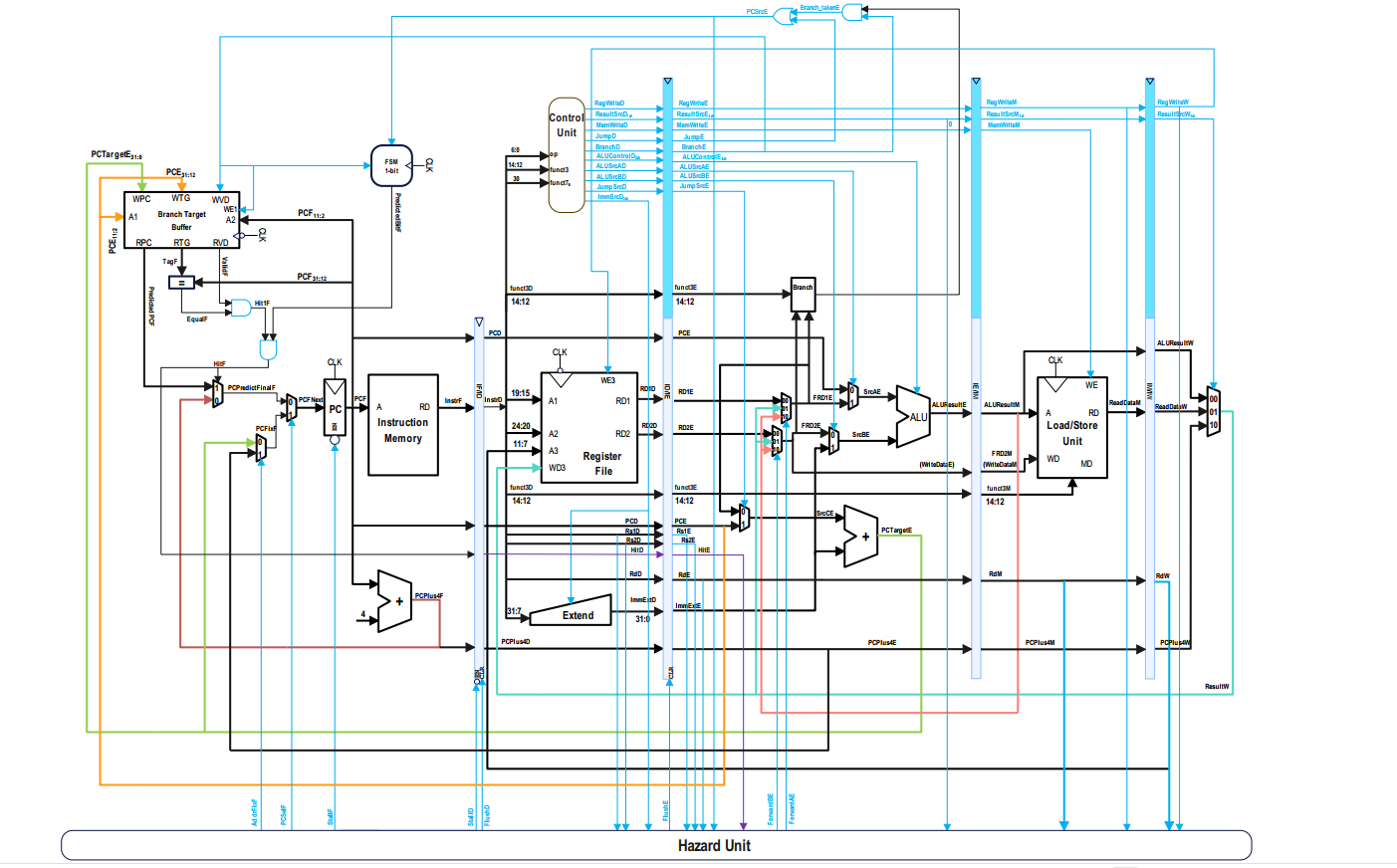
1. 1-bit predictor

Ở model này, ta thiết kế thêm bộ fsm 1bit lấy ngõ vào gồm 1 chân clk tích cực cao, 1 chân enable lấy từ tín hiệu BranhE, 1 input điều khiển trạng thái từ chân Pc\_takenE, ngõ ra là chân PcPredictedIF nếu là 1 thì địa chỉ ô lệnh ở 2 chu kỳ trước đó là địa chỉ nhảy, còn 0 là không nhảy.

Ban đầu ta cho máy trạng thái ở trạng thái 0 (not\_taken) nếu input Pc\_takenE là 1 thì trạng thái chuyển qua 1 (taken) và ngược lại, các trạng thái tuân theo sơ đồ trạng thái như sau:



Kết hợp PcPredictedIF và Hit từ BTB qua cổng And để đưa ra được khả năng dự đoán cuối cùng.



Sơ đồ khối của bộ pipeline 1-bit prediction

Bộ 1-bit prediction dự đoán khả năng chính xác khả tốt trong trường hợp chương trình là câu lệnh 1 vòng lặp. Ví dụ:

for (int i=0; i<10; i++) task A

Ở câu lệnh trên với bộ 1-bit predictor, processor chỉ dự đoán sai lần đầu tiên và lần thứ hai khi gặp câu lệnh, tức là tỉ lệ chính xác , với n= 10 tỉ lệ chính xác là 80% khi số vòng lặp rất lớn thì tỉ lệ chính xác gần như là 1. Tuy nhiên đối với các vòng lặp loop nest, tỉ lệ này sẽ giảm đi đáng kể

Giả sử:

for(int i=0; i<2; i++){

for(int j=0; j<5; j++){

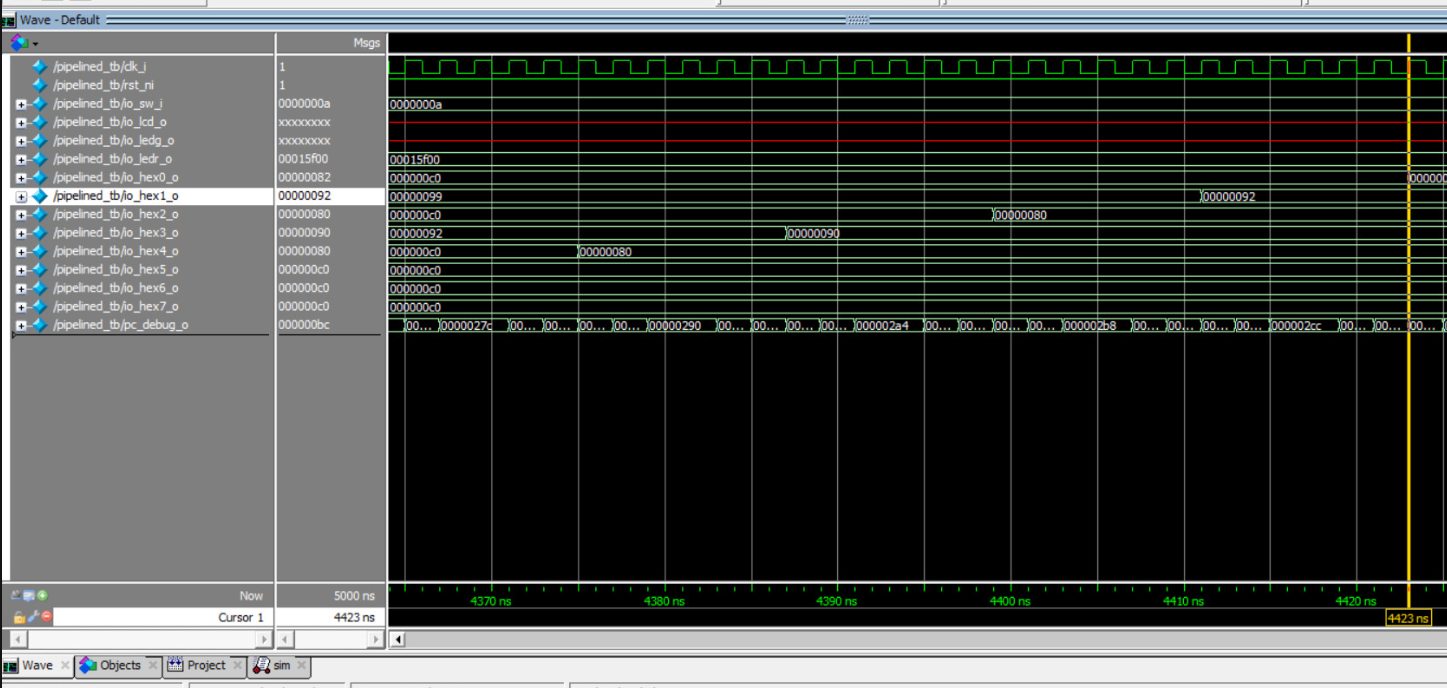
taskA

}

}

Khi đó 1 vòng lặp ở trong sẽ đoán sai 2 lần và vòng lặp ở trong được thực hiện 2 lần, như vậy số lần dự đoán lệnh nhảy sai của processor là 60%. Để giải quyết vấn đề này, chúng ta cần 1 bộ predictor có khả năng dự đoán chính xác cao hơn, đó là bộ 2-bit predictor có 4 trạng thái dự đoán thay vì là 2 như bộ 1-bit.

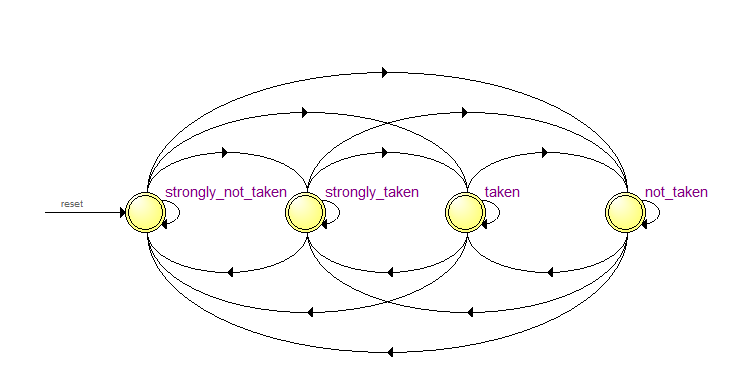
Mô phỏng mô hình 1-bit prediction khi nạp application code:



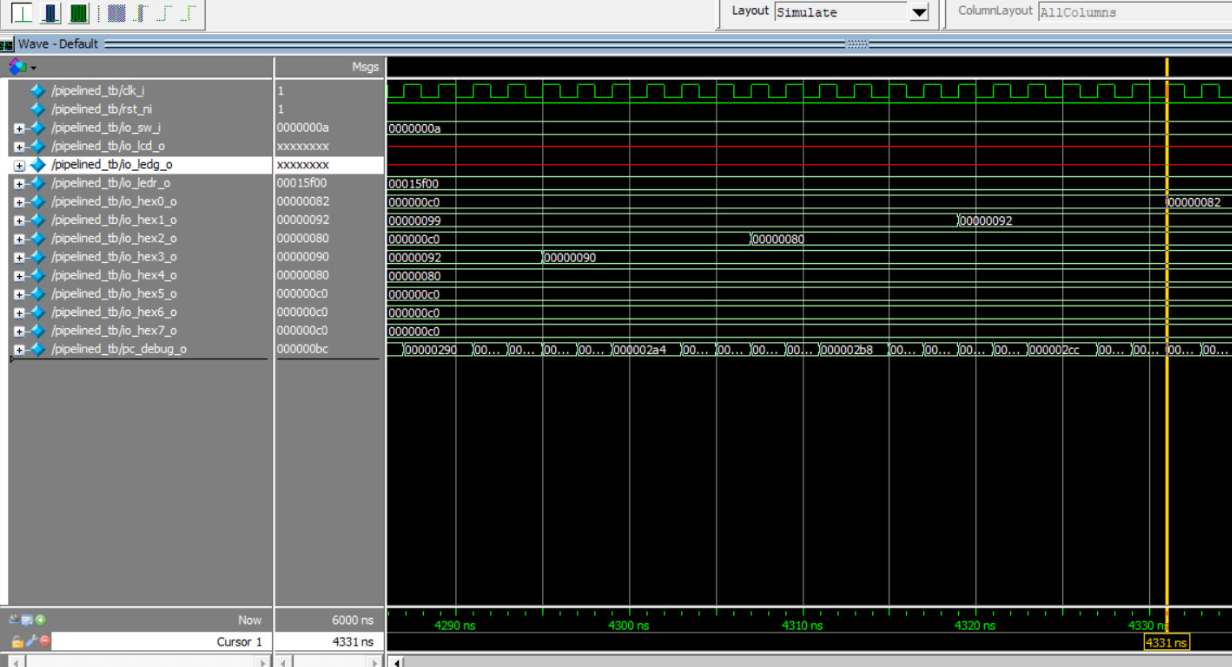
IPC của model = = 0.896

1. 2-bit predictor

Để thiết kế được 2-bit prediction model, ta thay thế bộ fsm 1-bit thành bộ fsm 2-bit với các chân tương tự. Bộ 2-bit predictor giải quyết được các trường hợp gặp các vòng lệnh loop nest và cũng như tăng khả năng dự đoán ở IF stage, vì bộ dự đoán có 4 trường hợp tương ứng với 4 trạng thái như sơ đồ:



Khi đó 1 địa chỉ được xác định từ nhảy hay không nhảy khi được dự đoán sai 2 lần, thay vì 1 lần như 1-bit predictor.



IPC của model = = 0.915

Kết luận: Ta thấy IPC(2-bit) > IPC(1-bit) > IPC(forwarding), điều đó cho thấy khả năng dự đoán tốt của model làm tăng IPC của processor và làm giảm thời gian xử lý chương trình của Program.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Forwarding | One bit prediction | Two bit prediction |
| IPC | 0,879 | 0,896 | 0,915 |