**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH­­**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HCM**

**BỘ MÔN ĐIỆN TỬ**

**Logo, company name

Description automatically generated**

EE3043: Computer Architecture

**Laboratory Report**

GVHD: Dr. Trần Hoàng Linh

TA: Cao Xuân Hải

Lớp: L01

|  |  |
| --- | --- |
| Thành viên nhóm | MSSV |
| Trần Thế Nhân | 2114274 |
| Nguyễn Đắc Tâm | 2114714 |
| Nguyễn Đăng Khoa | 2111527 |

Mục lục

[1. Introduction 2](#_Toc165559830)

[2. Design Strategy 3](#_Toc165559831)

[2.1. Non-forwarding 3](#_Toc165559832)

[2.2. Forwarding 4](#_Toc165559833)

[2.3. Always – taken 6](#_Toc165559834)

[3. Verification Strategy 9](#_Toc165559835)

[3.1. Non-forwarding 11](#_Toc165559836)

[3.2. Forwarding 13](#_Toc165559837)

[3.3. Always-taken 14](#_Toc165559838)

[3.4. Two-bit prediction 17](#_Toc165559839)

[4. Advanced Design 20](#_Toc165559840)

[4.1. Two-bit prediction 20](#_Toc165559841)

[5. Evaluation 23](#_Toc165559842)

[6. Conclusion 24](#_Toc165559843)

**Milestone 3**

**Design of Pipelined Processors**

# Introduction

RISC-V instruction set architecture (ISA) là một kiến trúc có mã nguồn mở dùng để thiết kế vi xử lý cho máy tính và đã trở nên phổ biến cho những ứng dụng trong công nghiệp và học thuật. Nó dựa trên nguyên tắc Reduced Instruction Set Computer (RISC) với mục đích đơn giản hóa tập lệnh và cải thiện tính hiệu quả cũng như hiệu năng cho vi xử lý. RV32I là những tập lệnh xử lý số nguyên nằm trong kiến trúc RISC-V và được thiết kế để triển khai số 32 bit. Trong báo cáo, chúng em sẽ trình bày thiết kế của vi xử lý RV32I pipelined 5 tầng dựa trên thiết kế single-cycle processor đã thiết kế trước đó. Pipelined processor được thiết kế với 4 models bao gồm non-forwarding, forwarding, always-taken và two-bit prediction.

Trong thiết kế vi xử lý, người ta luôn muốn tăng tốc độ của vi xử lý để đạt được những hiệu quả tốt nhất, và pipeline là một kỹ thuật giúp tăng đáng kể tốc độ lên nhiều lần. Trong thiết kế pipeline processor không thể tránh khỏi những loại hazards bao gồm structural hazard, data hazard và control hazard. Vì phiên bản single-cycle processor trước đó được thiết kế với cấu trúc Harvard nên structural hazard sẽ không phải vấn đề cần xử lý. Ở non-forwarding model, data hazard và control hazard sẽ được xử lý bằng cách stall những lệnh có dữ liệu phụ thuộc vào lệnh trước đó và flush những lệnh không đúng đã được đưa vào các tầng phía trước tầng execute. Forwarding model sẽ xử lý được data hazard bằng cách bypass những data wire cần thiết về tầng execute, control hazard vẫn được stall và flush như non-forwarding model. Always-taken model kế thừa toàn bộ forwarding model nhưng có thêm branch target buffer (BTB) để dự đoán trước địa chỉ cần nhảy khi gặp lệnh branch và jump. Two-bit prediction model kế thừa toàn bộ always-taken model và sử dụng phương pháp dự đoán 2 bit để cải thiện tính đúng đắn ở model trước.

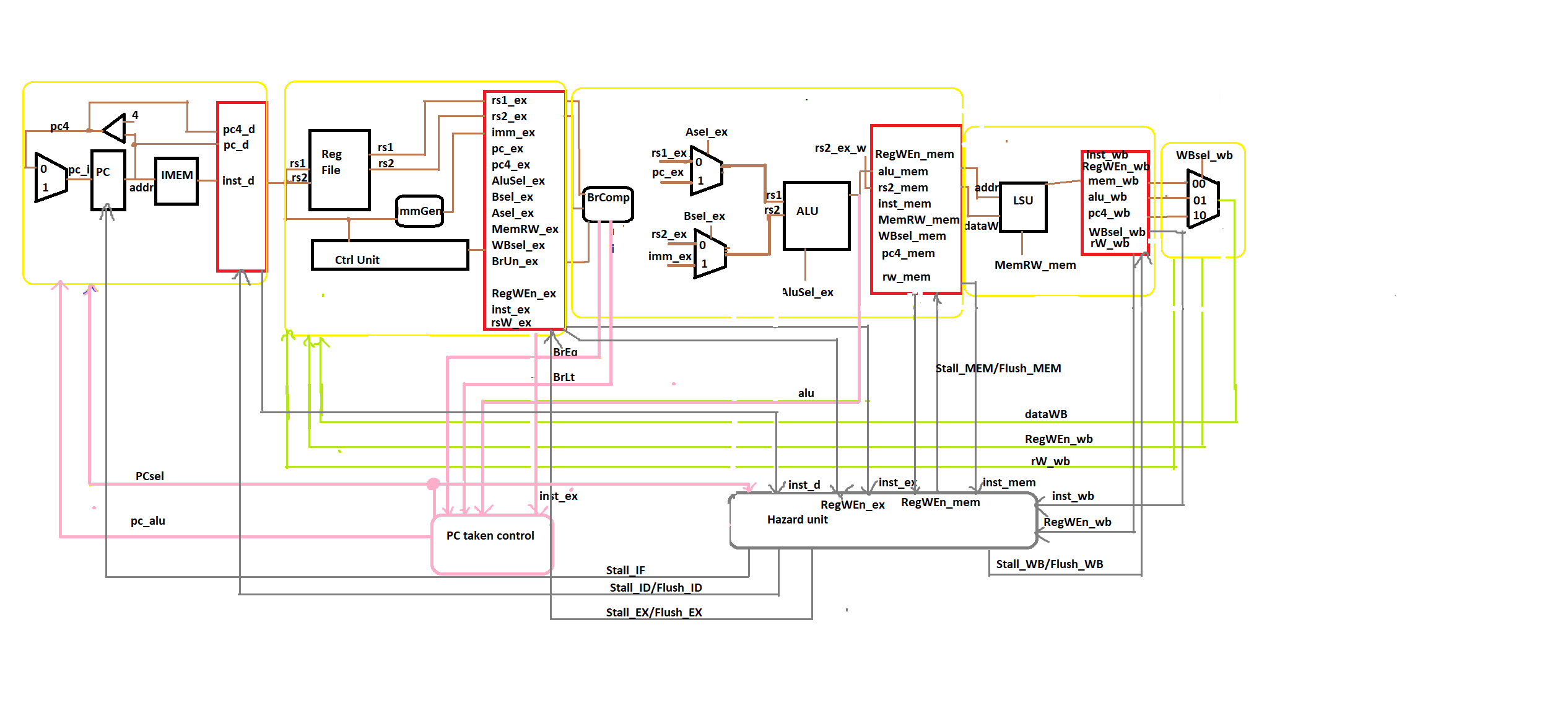
Để hoàn thành được thiết kế, chúng em xin cảm ơn thầy Trần Hoàng Linh đã dạy những kiến thức về cấu trúc máy tính và xin cảm ơn anh Cao Xuân Hải đã hướng dẫn cụ thể cách thực hiện thiết kế cũng như những kiến thức liên quan.

# Design Strategy

Pipelined processor được thiết kế với 5 tầng instruction fetch (IF), instruction decode (ID), execute (EX), memory (MEM) và write back (WB). 4 models được thiết kế bao gồm non-forwarding, forwarding, always-taken, two-bit prediction. Các model đều được dựa trên model trước đó để xây dựng thêm các khối bổ sung để giải quyết data hazard và control hazard.

## Non-forwarding

Ở model này, các instructions kế tiếp luôn được fetch vào pipeline, data hazard và control hazard chỉ được xử lý bằng cách flush những lệnh sai hoặc stall pipeline. Điều này sẽ làm giảm hiệu năng của vi xử lý. Stall và flush bằng cách sử dụng enable và reset bất đồng bộ trong flipflop. Khối hazard unit có chức năng phát hiện data hazard và control hazard để điều khiển tín hiệu stall và flush.



Hình 1. Block diagram của non-forwarding model

Data hazard bao gồm 3 trường hợp:

* Lệnh thứ 2 đọc thanh ghi được ghi ở lệnh thứ nhất. Khi này 1 trong 2 hoặc cả 2 thanh ghi đưa vào bộ ALU ở tầng execute chưa được cập nhật giá trị vì lệnh trước đó đang ở tầng memory và chưa được đưa vào tầng write back để lưu lại register file. Do đó, cần phải stall lệnh ở tầng IF, ID và flush lệnh ở tầng execute.
* Lệnh thứ 3 đọc thanh ghi được ghi ở lệnh thứ nhất. Khi này 1 trong 2 hoặc cả 2 thanh ghi đưa vào bộ ALU ở tầng execute chưa được cập nhật giá trị vì lệnh thứ nhất trước đó đang ở tầng write back và lưu lại register file nhưng lệnh thứ 3 đã lấy giá trị ở tầng ID trước đó. Do đó, cần phải stall lệnh ở tầng IF, ID và flush lệnh ở tầng EX, MEM.
* Đọc thanh ghi được ghi sau lệnh load. Khi lệnh hiện tại đang ở tầng ID để lấy dữ liệu thanh ghi thì lệnh load trước đó vẫn chưa tới tầng MEM để load giá trị và trả về register file. Do đó cần phải stall lệnh ở tầng IF, ID và flush lệnh ở tầng EX, MEM, WB.

Control hazard: có 2 trường hợp khi gặp lệnh branch là nhảy và không nhảy, trường hợp không nhảy sẽ không thành vấn đề vì những lệnh kế tiếp sau đó vẫn được thực thi. Nhưng trường hợp nhảy thì khi câu lệnh đi tới lệnh execute thì mới tính toán được có nhảy hay không, do đó cần flush đi 2 câu lệnh ở tầng IF và ID vì khi đã nhảy sẽ không thực hiện 2 lệnh đã đi vào ở 2 tầng trước đó.

**Tính toán IPC:**

Giả sử :

Xác suất gặp data hazard là

Số lượng câu lệnh cần delay trung bình là

Số lượng lệnh rẽ nhánh là

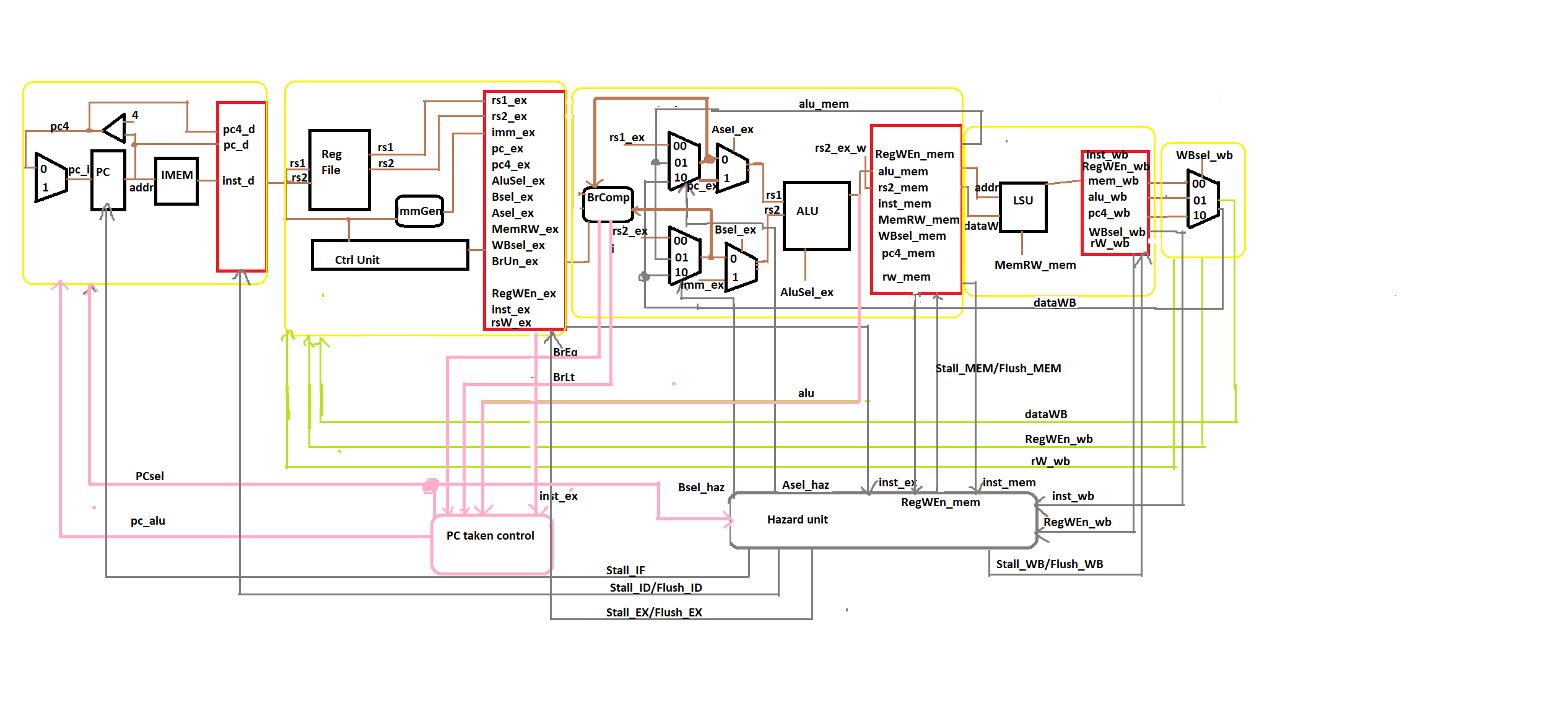
Xác suất nhảy khi gặp lệnh rẽ nhánh là

Số câu lệnh cần delay là

Ta tính được:

## Forwarding

Ở model này, các loại data hazard sẽ được xử lý bằng kỹ thuật forwarding. Các dây data chứa dữ liệu cần được ghi vào register file ở tầng EX và MEM sẽ được bypass vào các thanh ghi được đọc ở khối ALU.



Hình 2. Block diagram của forwarding model

Trong model này, tầng EX được thêm 2 bộ mux để lựa chọn dữ liệu bypass từ EX/MEM và MEM/WB khi hazard unit phát hiện data hazard. Riêng trường hợp đọc thanh ghi được ghi sau lệnh load, tầng IF, ID, EX cần được stall để lệnh load đưa dữ liệu vào thanh ghi sau đó chuyển tới tầng WB thì mới có thể tiếp tục pipeline. Control hazard vẫn được flush tương tự như non-forwarding model.

**Tính toán IPC:**

Giả sử :

Xác suất gặp load data hazard là

Số lượng câu lệnh cần delay là

Số lượng lệnh rẽ nhánh là

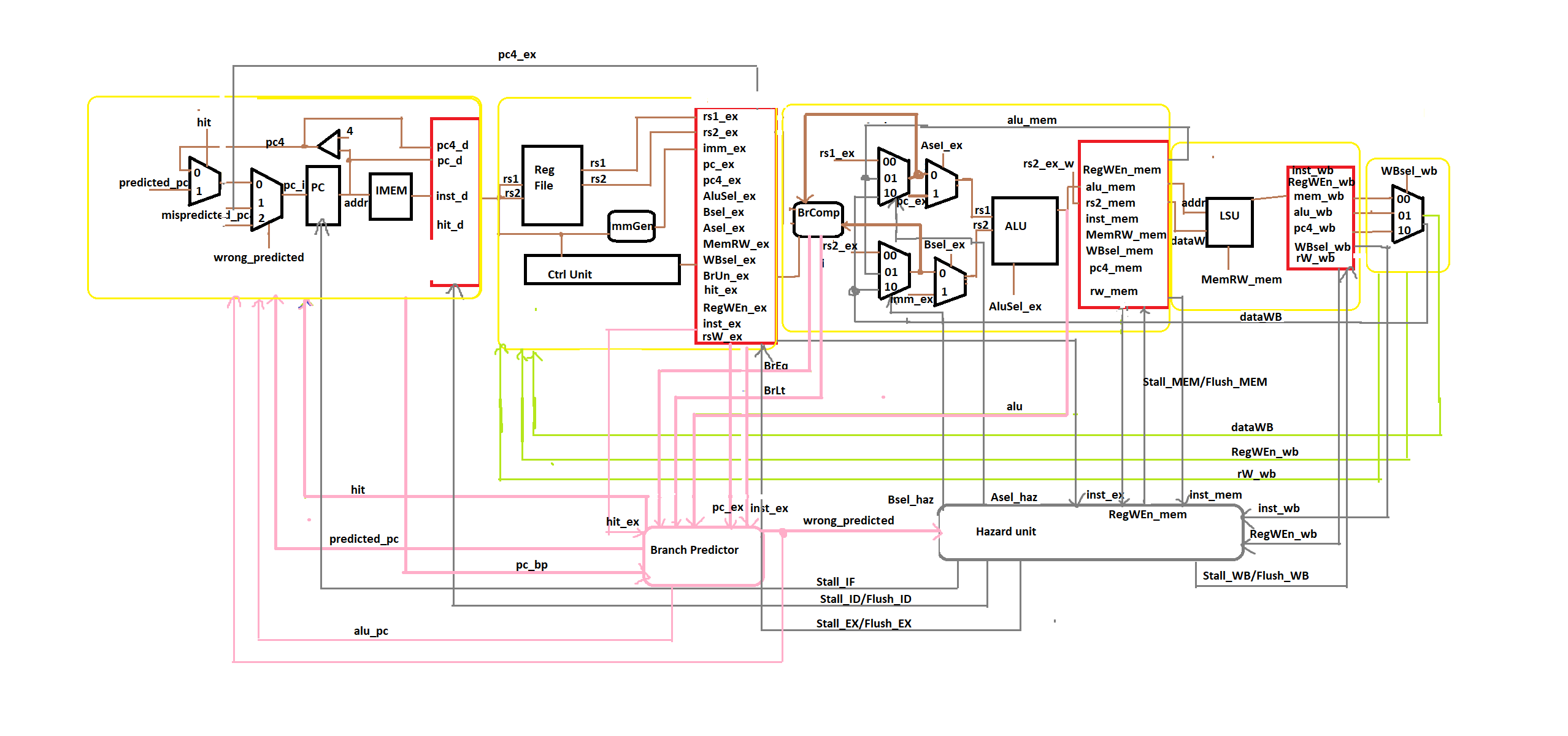
Xác suất nhảy khi gặp lệnh rẽ nhánh là

Số câu lệnh cần delay là

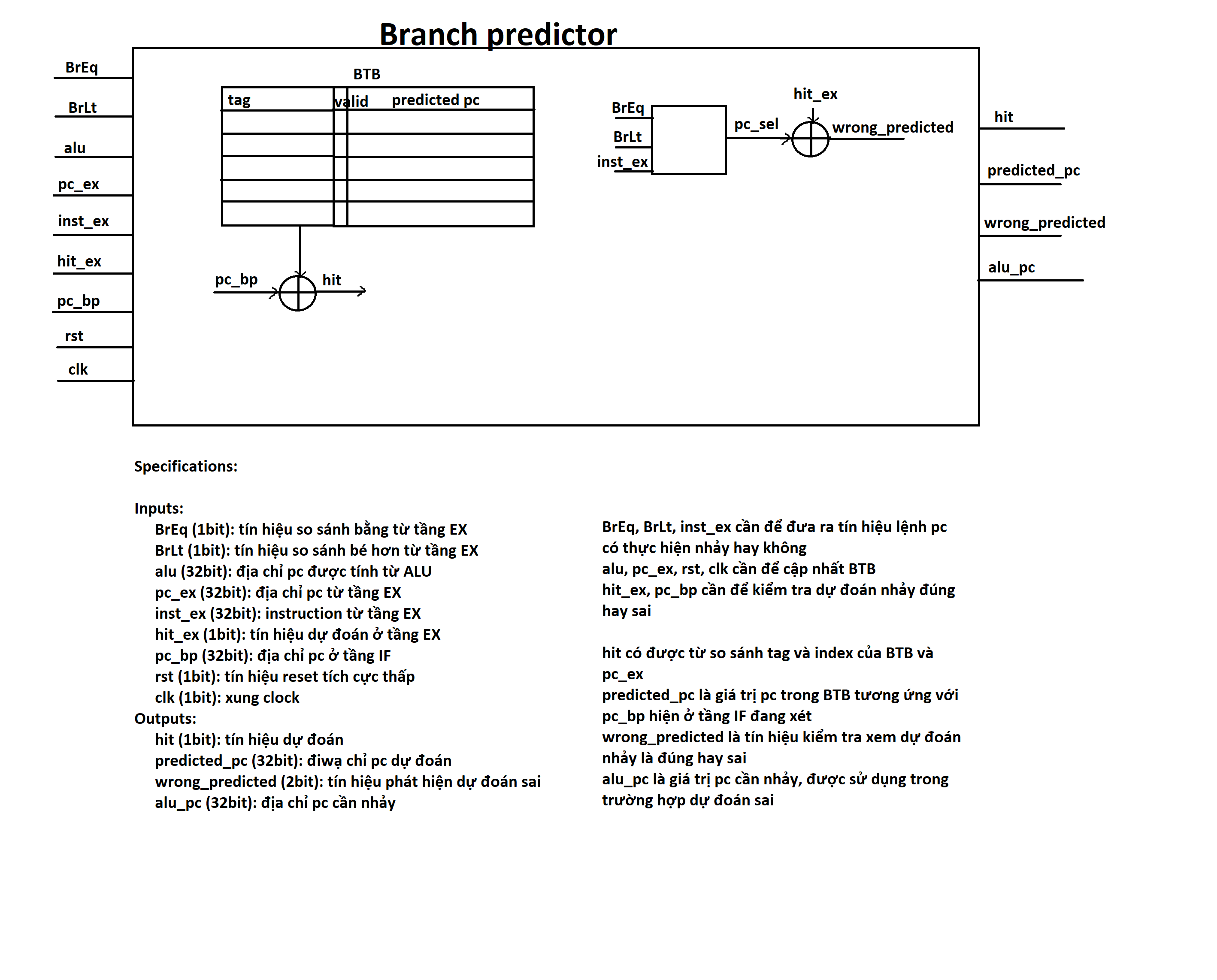
Ta tính được:

## Always – taken

Ở model này, kỹ thuật static branch prediction được sử dụng để dự đoán địa chỉ pc cần nhảy tới khi gặp lệnh branch và jump. Branch target buffer (BTB) được thiết kế để lưu lại predicted pc và giá trị địa chỉ của lệnh branch hoặc jump để nhận biết. Vì địa chỉ pc chứa 32 bit nên sẽ cần dung lượng lớn để lưu trữ, ở đây BTB được thiết kế với chiều dài là 32. Ở mỗi phần tử BTB, 20 bit đầu tiên là tag sẽ chứa 20 bit cao của địa chỉ pc ở lệnh branch hoặc jump, 1 bit valid để nhận biết phần tử BTB đã được cập nhật hay chưa, 32 bit cuối sẽ chứa địa chỉ pc mà lệnh branch hoặc jump nhảy tới. Giá trị index của BTB sẽ là 5 bit đầu của địa chỉ pc ở lệnh branch hoặc jump vì depth của BTB được thiết kế bằng 32.



Hình 3: Block diagram của always-taken model



Hình 4: Block diagram của khối branch predictor trong always-taken model

**Tính toán IPC:**

Giả sử :

Xác suất gặp load data hazard là

Số lượng câu lệnh cần delay là

Số lượng lệnh rẽ nhánh là

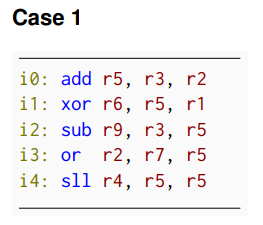
Xác suất nhảy khi gặp lệnh rẽ nhánh là

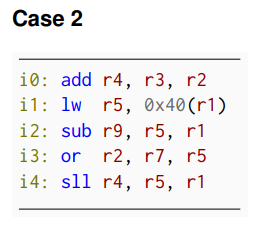
Số câu lệnh cần delay là

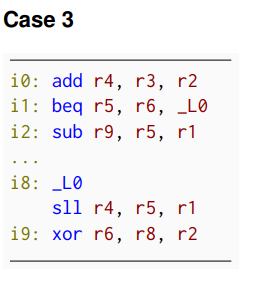
Ta tính được:

# Verification Strategy

Kiểm tra các model qua những trường hợp hazard như sau:







Các lệnh assembly được sử dụng trong từng trường hợp như sau:

* Case 1:

addi x1, x0, 1

addi x2, x0, 2

addi x3, x0, 3

addi x4, x0, 4

addi x5, x0, 5

addi x6, x0, 6

addi x7, x0, 7

addi x8, x0, 8

addi x9, x0, 9

add x5, x3, x2

xor x6, x5, x1

sub x9, x3, x5

or x2, x7, x5

sll x4, x5, x5

* Case 2:

addi x1, x0, 1

addi x2, x0, 2

addi x3, x0, 3

addi x4, x0, 4

addi x5, x0, 5

addi x6, x0, 6

addi x7, x0, 7

addi x8, x0, 8

addi x9, x0, 9

sw x6, 0(x1)

add x4, x3, x2

lw x5, 0(x1)

sub x9, x5, x1

or x2, x7, x5

sll x4, x5, x1

* Case 3:

addi x1, x0, 1

addi x2, x0, 2

addi x3, x0, 3

addi x4, x0, 4

addi x5, x0, 5

addi x6, x0, 6

addi x7, x0, 7

addi x8, x0, 8

addi x9, x0, 9

add x4, x3, x2

beq x5, x4, \_L0

sub x9, x5, x1

addi x5, x0, 50

addi x6, x0, 60

addi x7, x0, 70

addi x8, x0, 80

addi x9, x0, 90

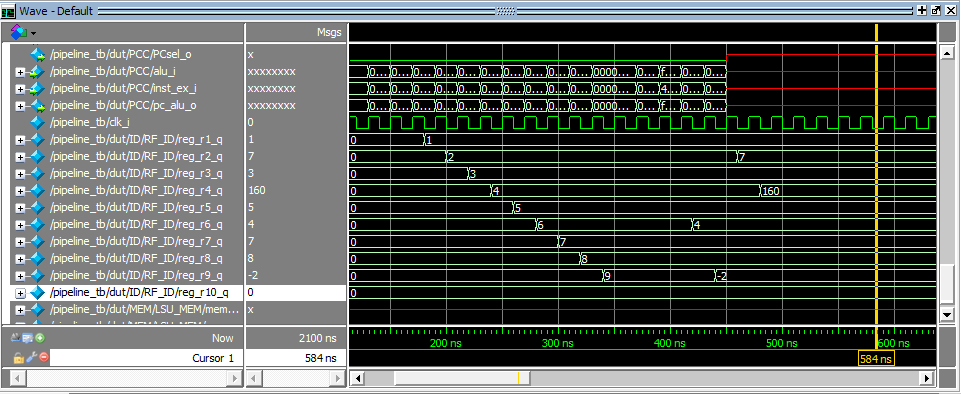
\_L0:

sll x4, x5, x1

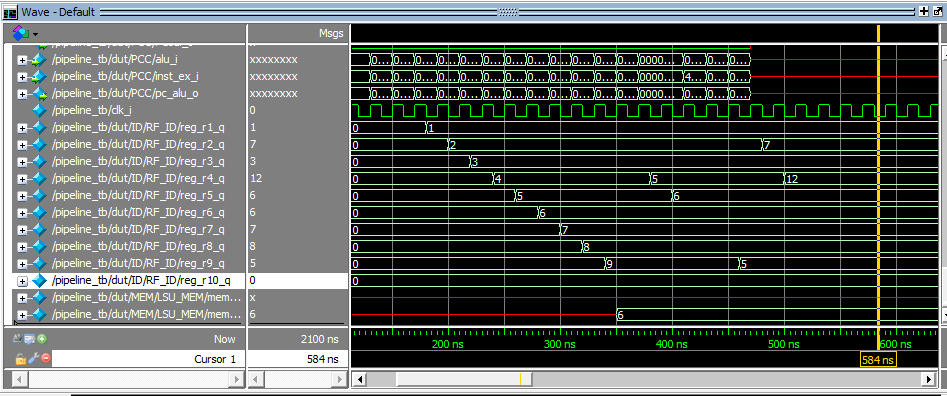
xor x6, x8, x2

## Non-forwarding

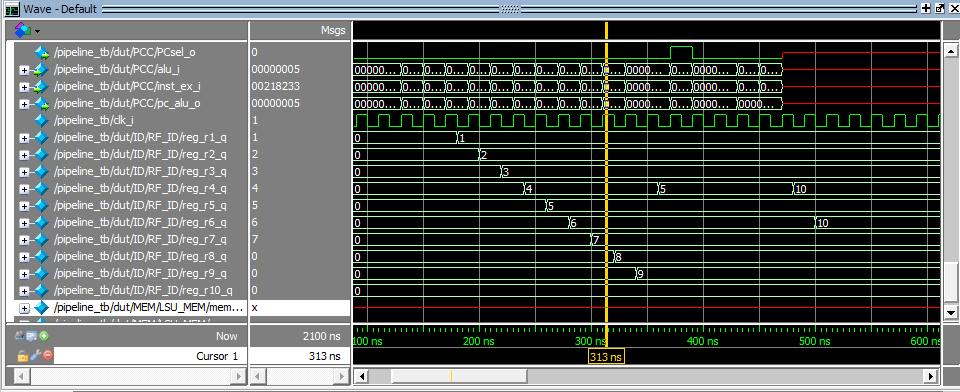
Case 1:



Case 2:



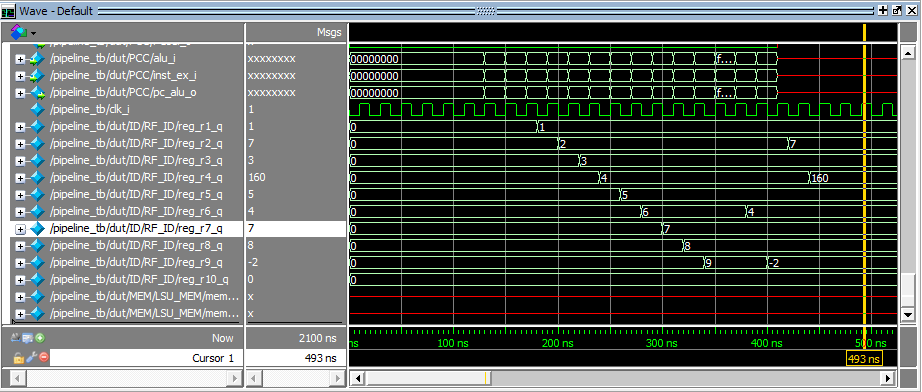
Case 3:



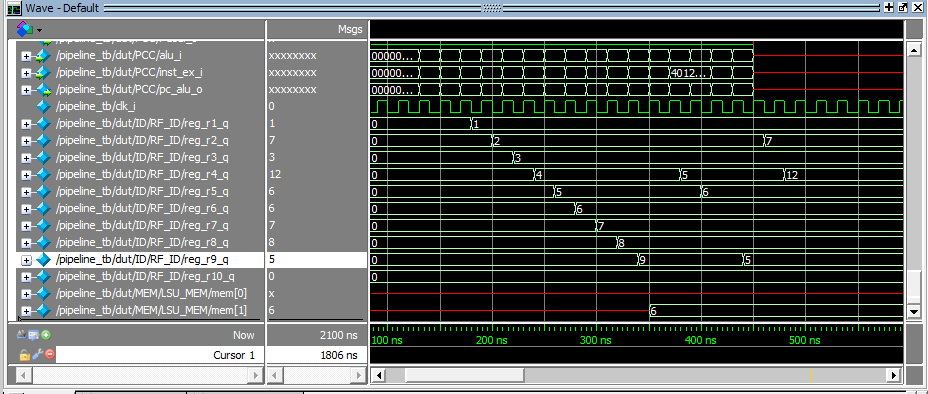
Các trường hợp gặp data hazard thì chương trình phải stall và flush một số lệnh do đó làm giảm hiệu suất của processor.

## Forwarding

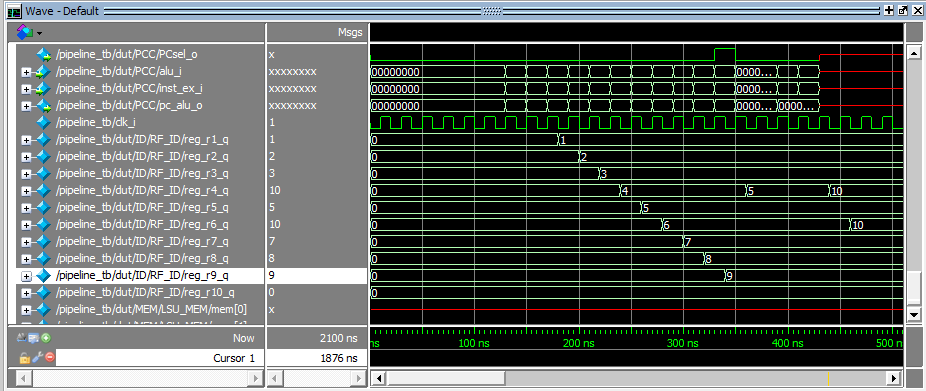
Case 1:



Case 2:



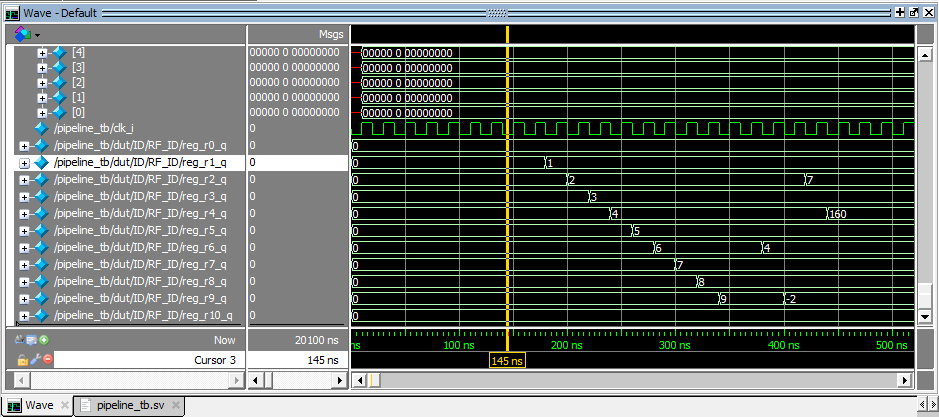
Case 3:



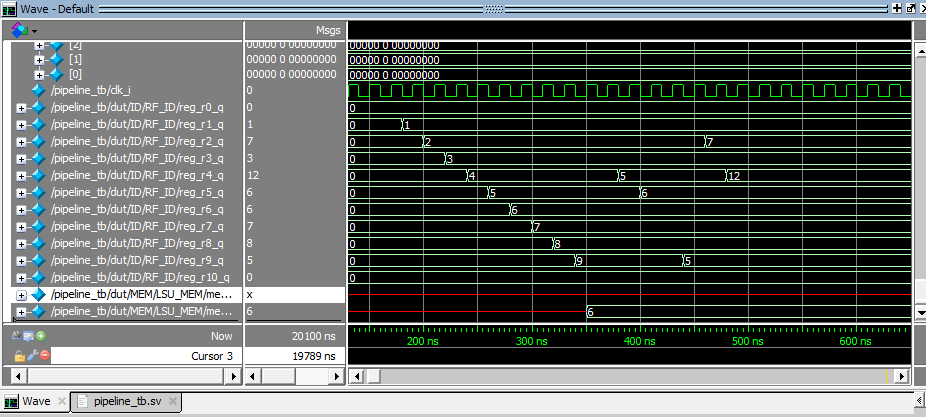
Data hazard ở forwarding model được giải quyết và riêng trường hợp đọc thanh ghi được ghi sau lệnh load bị stall 1 chu kỳ. Control hazard vẫn được flush như model trước.

## Always-taken

Case 1:



Case 2:



Case 3:

Ở trường hợp này, assembly sẽ thay đổi để có thể thực hiện nhiều lệnh nhảy giúp dễ kiểm chứng hơn.

Assembly:

addi x1, x0, 10

addi x8, x0, 1

add x3, x0, x0

\_COMPARE:

and x2, x1, x8

beq x2, x0, \_ADD\_EVEN

jal x10, \_DECREASE

\_ADD\_EVEN:

add x3, x1, x3

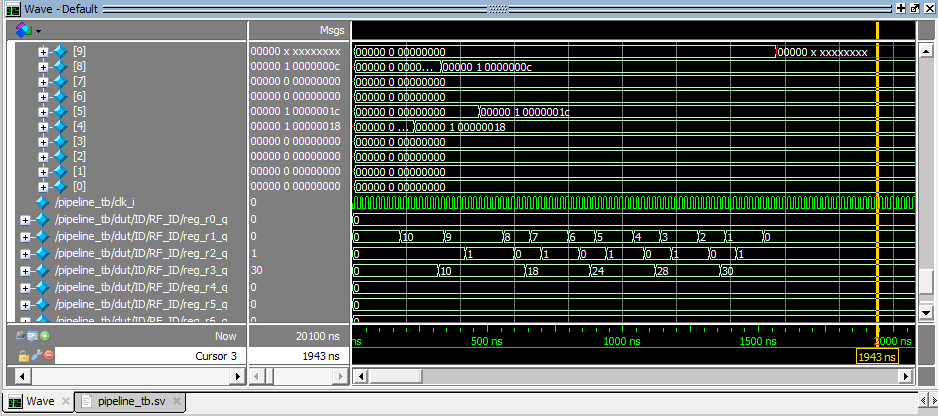
\_DECREASE:

sub x1, x1, x8

bne x1, x0, \_COMPARE

\_EXIT:

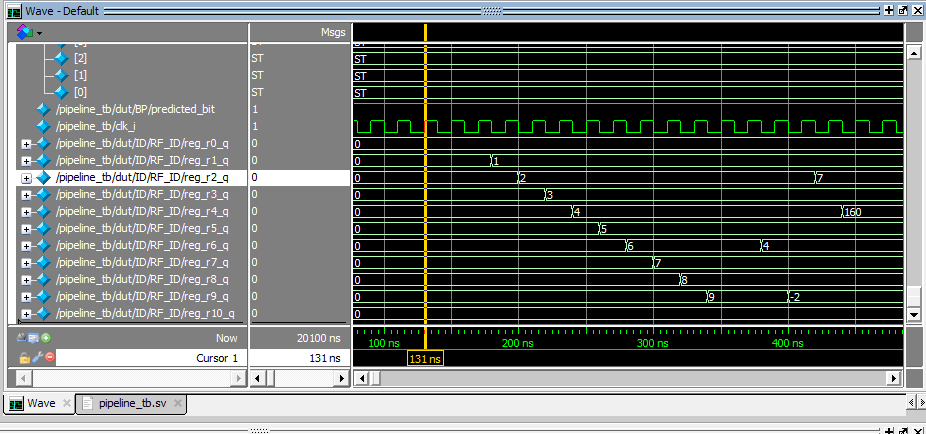
Chương trình assembly trên sẽ tính tổng các số chẵn từ 1 đến 10 và lưu vào thanh ghi x3.



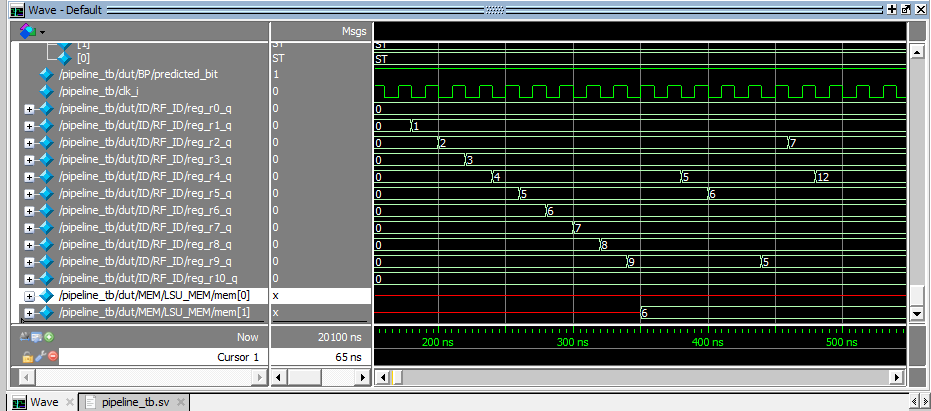
Như waveform trên, BTB[4], BTB[5], BTB[8] đã lưu giá trị tag và predicted pc. Khi gặp các lệnh nhảy thì ở model này bit dự đoán sẽ luôn là 1, nghĩa là luôn chọn nhảy đến địa chỉ pc được dự đoán.

## Two-bit prediction

Case 1:



Case 2:



Case 3:

Ở trường hợp này, assembly sẽ thay đổi để có thể thực hiện nhiều lệnh nhảy giúp dễ kiểm chứng hơn.

Assembly:

addi x1, x0, 10

addi x8, x0, 1

add x3, x0, x0

\_COMPARE:

and x2, x1, x8

beq x2, x0, \_ADD\_EVEN

jal x10, \_DECREASE

\_ADD\_EVEN:

add x3, x1, x3

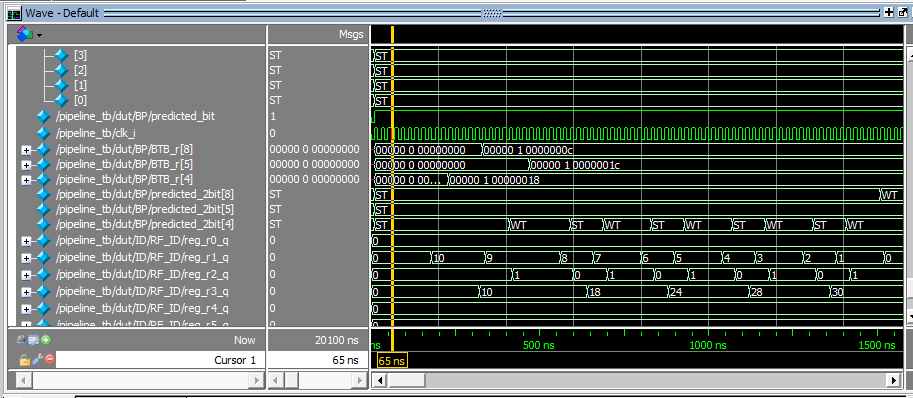
\_DECREASE:

sub x1, x1, x8

bne x1, x0, \_COMPARE

\_EXIT:

Chương trình assembly trên sẽ tính tổng các số chẵn từ 1 đến 10 và lưu vào thanh ghi x3.



Trong waveform, BTB đã được cập nhật giá trị tag và predicted pc ở index cụ thể, tương tự như BTB, trạng thái của 2 bit prediction cũng được cập nhật khi nhảy sai với dự đoán.

2 bit prediction có 4 trạng thái:

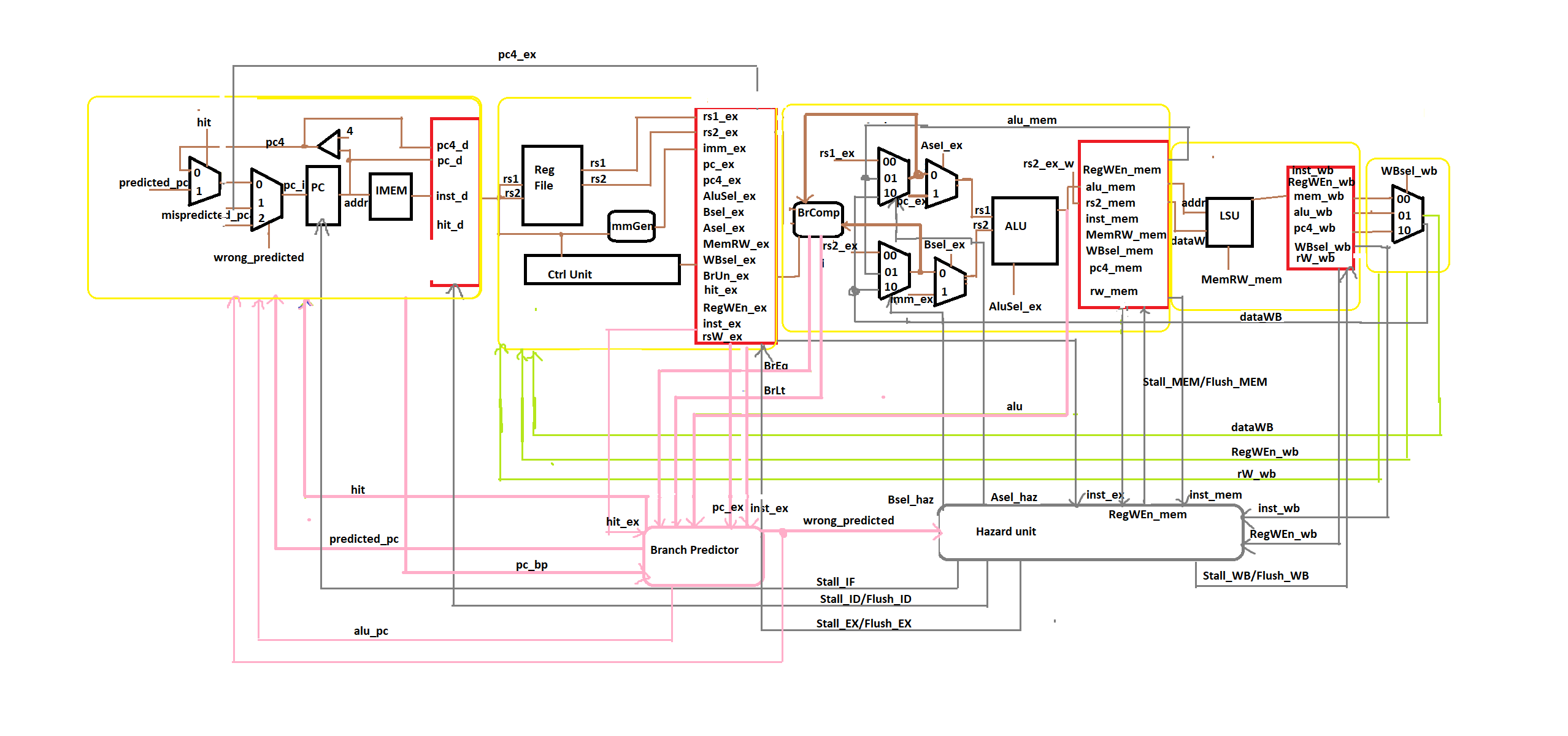
* ST (strongly taken)
* WT (weakly taken)
* SNT (strongly not taken)
* WNT (weakly not taken)

Chỉ khi lệnh nhảy được dự đoán sai 2 lần liên tiếp thì 2 bit prediction mới thay đổi từ taken sang not taken hoặc ngược lại. Trong waveform cho thấy lệnh nhảy beq x2, x0, \_ADD\_EVEN sẽ taken và not taken liên tiếp nhau nên 2 bit prediction chỉ thay đổi từ ST sang WT và ngược lại.

# Advanced Design

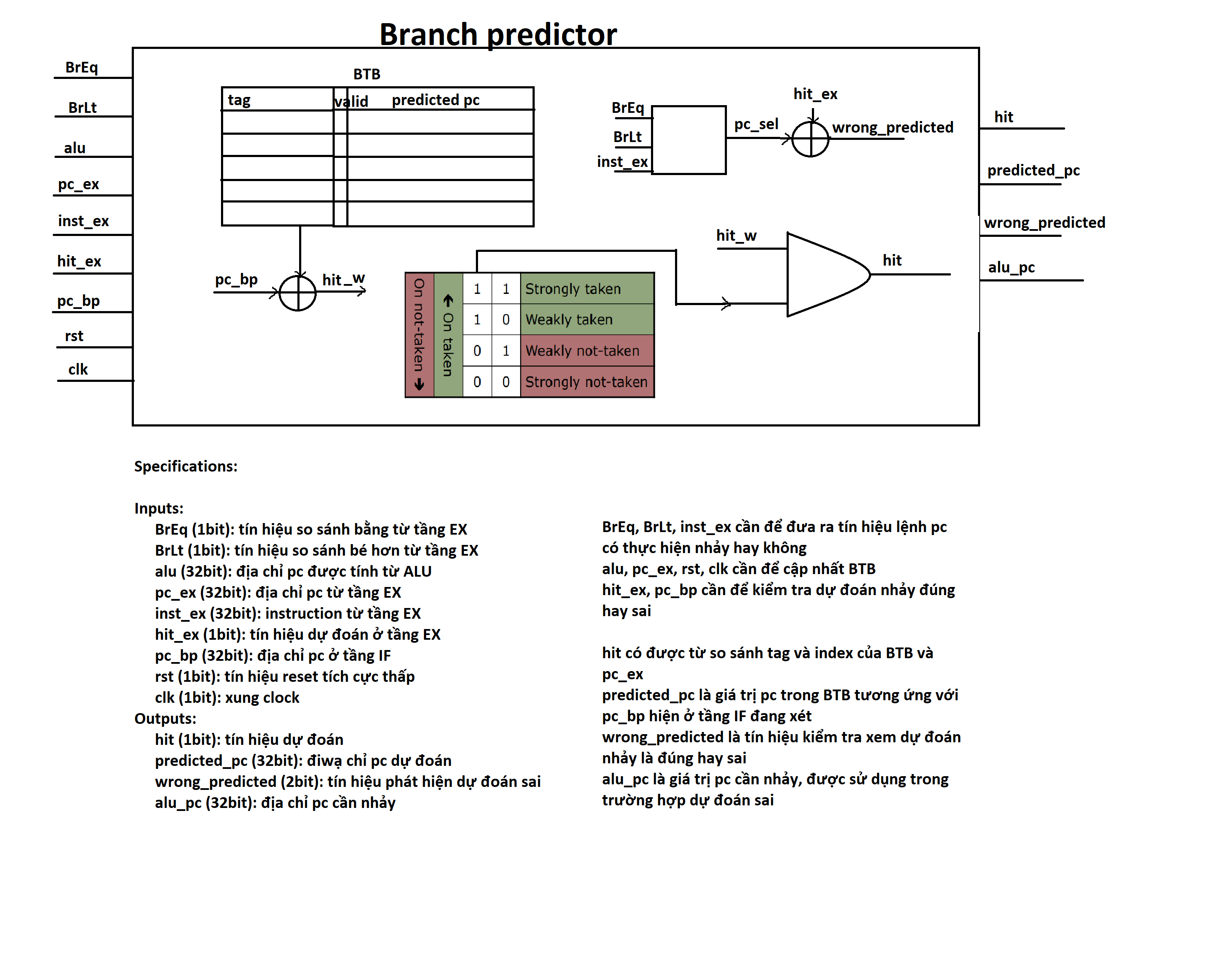
## Two-bit prediction

Ở model này được kế thừa hoàn toàn từ always-taken model, riêng phần dự đoán bit được cải thiện bằng kỹ thuật two-bit prediction. Kỹ thuật chính ở model này là two-bit dynamic branch prediction so với static branch prediction ở always-taken model.



Hình 3: Block diagram của always-taken model

Block diagram ở model này hoàn toàn giống always-taken model, riêng branch predictor có thay đổi.



Hình 5: Block diagram của branch predictor trong two-bit prediction model

**Tính toán IPC:**

Giả sử :

Xác suất gặp load data hazard là

Số lượng câu lệnh cần delay là

Số lượng lệnh rẽ nhánh là

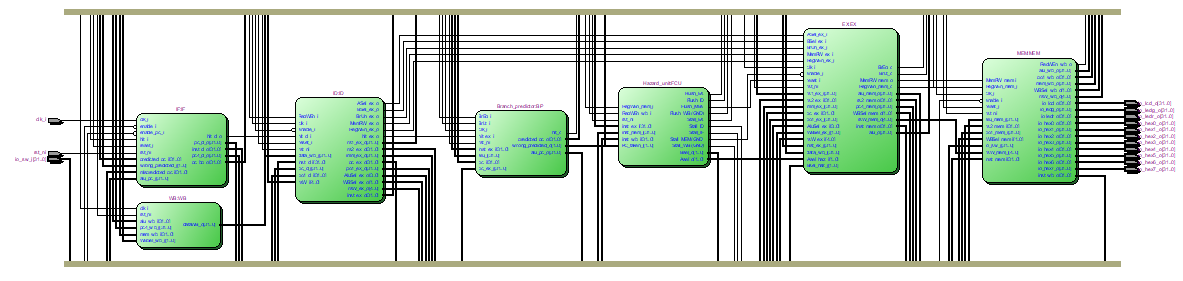
Xác suất nhảy khi gặp lệnh rẽ nhánh là

Số câu lệnh cần delay là

Ta tính được:

# Evaluation

Sau khi thiết kế hoàn chỉnh, mô phỏng được RTL của mạch như hình:



4 models được thiết kế ở trên đạt được hiệu năng theo thứ tự như sau: 2-bit prediction > always-taken > forwarding > non-forwarding.

Hiệu năng được đánh giá qua IPC và có thể cải thiện bằng cách thiết kế khối branch predictor với những kỹ thuật phức tạp hơn như G-share hoặc kết hợp giữa global và local history predictor.

# Conclusion

Piped line processor được thiết kế hoàn chỉnh với 4 model, mỗi model phía sau là phiên bản cải thiện của phiên bản trước. Một số ứng dụng cơ bản được viết bằng hợp ngữ và biên dịch ra mã hex, sau đó cho cpu chạy thử và thực hiện đúng chức năng. Chi tiết source code thiết kế, testbench và ứng dụng được đăng trên github:

<https://github.com/Stork1323/Risc-V-Pipelined-RV32I-.git>