VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS
----------o0o----------

EE3043: COMPUTER ARCHITECTURE
BIG PROJECT REPORT
MILESTONE 3: DESIGN OF PIPELINED RISCS-V PROCESSOR

Instructor : Dr. Trần Hoàng Linh
Teaching Assistant: Cao Xuân Hải
Group: 15
Student: Phạm Việt Hùng - 2113592
Nguyễn Đức Hoàng - 2110184
Lê Trung Hiếu - 2111185

HO CHI MINH CITY, DECEMBER 15, 2024

Mục lục

I. Objective:

- Ôn lại kiến thức liên quan đến SystemVerilog
- Ôn lại kiến thức liên quan đến tập lệnh RV32I
- Thiết kế pipeline RV32I processor

II. Design Strategy

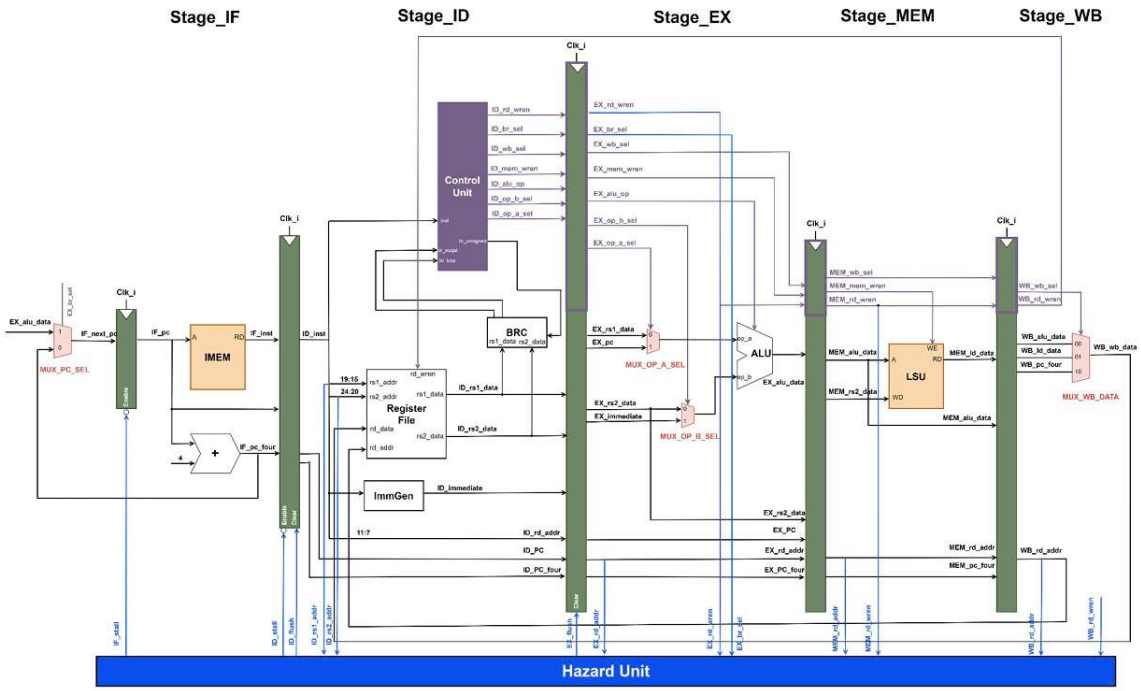
1. Non-forwarding:

Trong mô hình không chuyển tiếp (non-forwarding), pipeline được chia thành năm giai đoạn: Instruction Fetch (IF)**,** Instruction Decode (ID)**,** Execute (EX)**,** Memory Access (MEM) và Write Back (WB). Một thành phần quan trọng trong thiết kế này là HazardUnit, chịu trách nhiệm quản lý data hazard để đảm bảo tính đúng đắn của việc thực thi lệnh. HazardUnit sử dụng hai cơ chế chính — tín hiệu **Stall** và **Flush** — để xử lý các nguy cơ này và duy trì hoạt động chính xác của pipeline.

Data hazzard xảy ra khi một lệnh trong pipeline cần dữ liệu được tạo ra bởi một lệnh trước đó, nhưng dữ liệu này chưa sẵn sàng do tính chất tuần tự của xử lý pipeline. Ví dụ, nếu một lệnh trong giai đoạn ID cần kết quả của một phép toán vẫn đang được tính toán trong giai đoạn EX, một nguy cơ dữ liệu sẽ phát sinh. HazardUnit phát hiện các tình huống này bằng cách theo dõi sự phụ thuộc giữa các thanh ghi trong các giai đoạn pipeline, đặc biệt là giữa EX, MEM và WB. Khi phát hiện nguy cơ, HazardUnit phát ra tín hiệu **Stall** để tạm thời dừng tiến trình pipeline, giữ nguyên giai đoạn bị ảnh hưởng cho đến khi dữ liệu cần thiết sẵn sàng. Điều này đảm bảo rằng các lệnh phụ thuộc không được thực thi quá sớm, tránh tạo ra kết quả sai.

Ngược lại, **control hazard** xảy ra khi có sự thay đổi trong luồng điều khiển, chẳng hạn khi gặp một lệnh branch. Pipeline có thể dự đoán và lấy lệnh một cách suy đoán, dẫn đến việc giải mã các lệnh không chính xác nếu dự đoán nhánh sai hoặc quyết định nhánh được đưa ra muộn hơn trong pipeline. Để giải quyết điều này, HazardUnit tạo ra các tín hiệu **Flush**, xóa các lệnh trong các giai đoạn pipeline bị ảnh hưởng ngay khi quyết định nhánh được hoàn tất. Quá trình này loại bỏ các lệnh không hợp lệ, đảm bảo rằng chỉ những lệnh chính xác được tiếp tục thực thi trong pipeline.

Bằng cách sử dụng cơ chế **Stall** và **Flush**, HazardUnit đảm bảo tính đúng đắn của việc thực thi lệnh ngay cả khi đối mặt với các phụ thuộc dữ liệu và thay đổi luồng điều khiển. Tuy nhiên, việc phụ thuộc vào cơ chế dừng (stall) và xóa (flush) có thể ảnh hưởng đáng kể đến hiệu suất, vì pipeline phải chờ đợi hoặc loại bỏ lệnh thay vì duy trì hoạt động liên tục. Hạn chế này được khắc phục trong model **forwarding** giúp giảm thiểu số chu kỳ chờ và cải thiện hiệu quả của pipeline.



Specification:

Signal	Width	Direction	Description
EX_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn EX.
MEM_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn MEM.
WB_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn WB.
id_is_rs2	1	Input	Cho biết thanh ghi RS2 được sử dụng trong lệnh giai đoạn ID.
ID_br_sel	1	Input	Tín hiệu chọn nhánh trong giai đoạn ID.
EX_br_sel	1	Input	Tín hiệu chọn nhánh trong giai đoạn EX.
EX_rd_addr	5	Input	Địa chỉ của thanh ghi đích được ghi trong giai đoạn EX.
MEM_rd_addr	5	Input	Địa chỉ của thanh ghi đích được ghi trong giai đoạn MEM.
WB_rd_addr	5	Input	Địa chỉ của thanh ghi đích được ghi trong giai đoạn WB.
ID_rs1_addr	5	Input	Địa chỉ của thanh ghi RS1 trong lệnh giai đoạn ID.
ID_rs2_addr	5	Input	Địa chỉ của thanh ghi RS2 trong lệnh giai đoạn ID.

ID_inst	32	Input	Lệnh hiện tại trong giai đoạn ID.
pc_enable_o	1	Output	Bật hoặc tắt cập nhật bộ đếm chương trình (PC).
id_enable_o	1	Output	Bật hoặc tắt giai đoạn ID.
ex_enable_o	1	Output	Bật hoặc tắt giai đoạn EX.
mem_enable_o	1	Output	Bật hoặc tắt giai đoạn MEM.
wb_enable_o	1	Output	Bật hoặc tắt giai đoạn WB.
id_reset_no	1	Output	Tín hiệu reset giai đoạn ID (active low, dùng để flush).
ex_reset_no	1	Output	Tín hiệu reset giai đoạn EX (active low, dùng để flush).
mem_reset_no	1	Output	Tín hiệu reset giai đoạn MEM (active low).
wb_reset_no	1	Output	Tín hiệu reset giai đoạn WB (active low).
hazard_1	1	Internal Wire	Chỉ báo hazard WB -> ID.
hazard_2	1	Internal Wire	Chỉ báo hazard MEM -> ID.
hazard_3	1	Internal Wire	Chỉ báo hazard EX -> ID.

2. Forwarding:

Module hazard_detect đóng vai trò quan trọng trong việc duy trì hiệu suất và tính đúng đắn của bộ xử lý pipeline bằng cách xử lý các nguy cơ dữ liệu (data hazards) và điều phối luồng lệnh trong pipeline. Để hỗ trợ chuyển tiếp dữ liệu (forwarding), module này đã được bổ sung những cải tiến quan trọng so với mô hình không chuyển tiếp (non-forwarding), cho phép phát hiện và giải quyết các phụ thuộc dữ liệu một cách hiệu quả hơn. Những cải tiến này giúp module xác định cơ hội chuyển tiếp dữ liệu từ các tầng sau (MEM và WB) về các tầng trước (ID và EX), qua đó giảm đáng kể số chu kỳ chờ (stall). Bằng cách kết hợp các cơ chế phát hiện hazard, chuyển tiếp, và điều khiển pipeline, module đảm bảo quá trình thực thi diễn ra mượt mà và tối ưu, đồng thời giảm thiểu số lần xóa lệnh (flush) và tối đa hóa thông lượng lệnh.

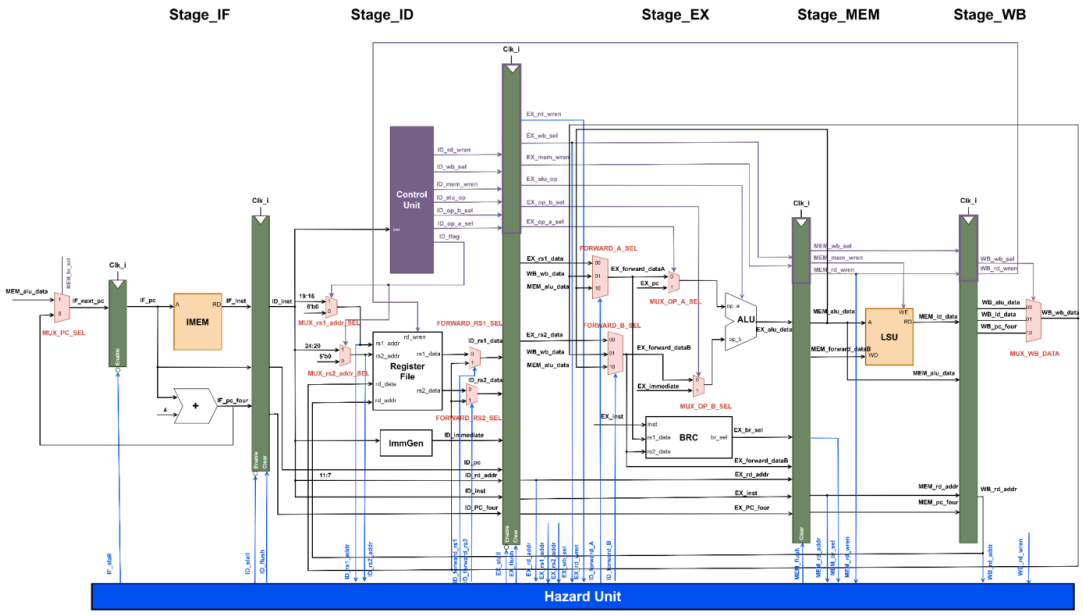
Chuyển tiếp dữ liệu (forwarding) được triển khai để giải quyết hiệu quả các phụ thuộc dữ liệu. Đối với tầng EX, logic chuyển tiếp chọn động (dynamically selects) nguồn dữ liệu phù hợp cho các toán hạng rs1 và rs2. Nếu dữ liệu cần thiết đã có sẵn ở tầng MEM (MEM_rd_addr khớp với EX_rs1_addr hoặc EX_rs2_addr), dữ liệu này được chuyển tiếp đến tầng EX thông qua các tín hiệu ID_forward_A và ID_forward_B. Tương tự, dữ liệu từ tầng WB (WB_rd_addr) có thể được chuyển tiếp đến tầng EX nếu không có dữ liệu

hợp lệ tại tầng MEM. Đối với tầng ID, các đường chuyển tiếp (forwarding paths) cho phép lấy dữ liệu từ tầng WB khi cần thiết (WB_rd_addr khớp với ID_rs1_addr hoặc ID_rs2_addr), giúp giảm thời gian chờ cho các lệnh phụ thuộc.

Module cũng phát hiện và xử lý các nguy cơ do phụ thuộc load-use hoặc các lệnh rẽ nhánh. Khi phát hiện phụ thuộc load-use (được chỉ báo bởi các tín hiệu EX_mem_rden hoặc MEM_mem_rden), module sẽ dừng pipeline bằng cách vô hiệu hóa bộ đếm chương trình (tín hiệu pc_enable_o) và tầng ID (id_enable_o). Đồng thời, các tầng phía sau vẫn được duy trì hoạt động để đảm bảo tiến trình thực thi lệnh mượt mà. Trong trường hợp nguy cơ điều khiển (control hazard), khi lệnh rẽ nhánh được giải quyết tại tầng EX (EX_br_sel), các tầng ID và EX sẽ được xóa (flush) bằng cách sử dụng các tín hiệu reset (id_reset_no và ex_reset_no) để ngăn chặn việc thực thi sai.

Các tín hiệu enable và reset được kiểm soát cẩn thận nhằm duy trì tính nhất quán của pipeline. Trong điều kiện bình thường, tất cả các tầng đều được kích hoạt, nhưng trong trường hợp xảy ra hazard, các tầng cụ thể sẽ bị tạm dừng (stall) hoặc xóa (flush) một cách linh hoạt. Cách tiếp cận này giảm thiểu gián đoạn và đảm bảo rằng chỉ các lệnh bị ảnh hưởng mới bị trì hoãn hoặc loại bỏ.

Tổng thể, module hazard_detect tích hợp các cơ chế chuyển tiếp, phát hiện hazard, và điều khiển pipeline để cải thiện hiệu suất của pipeline. Bằng cách giải quyết các phụ thuộc dữ liệu trong thời gian thực và giảm thiểu tác động của các loại hazard, module này giúp giảm số chu kỳ chờ (stall), tăng thông lượng lệnh, đảm bảo tính đúng đắn, và làm cho thiết kế trở nên hiệu quả và có khả năng mở rộng cho các bộ xử lý hiện đại.



Specification:

Signal	Width	Direction	Description
EX_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn EX.
MEM_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn MEM.
WB_rd_wren	1	Input	Tín hiệu cho phép ghi vào thanh ghi trong giai đoạn WB.
id_is_rs1	1	Input	Cho biết lệnh ở giai đoạn ID phụ thuộc vào thanh ghi rs1.
id_is_rs2	1	Input	Cho biết lệnh ở giai đoạn ID phụ thuộc vào thanh ghi rs2.
ex_is_rs1	1	Input	Cho biết lệnh ở giai đoạn EX phụ thuộc vào thanh ghi rs1.
ex_is_rs2	1	Input	Cho biết lệnh ở giai đoạn EX phụ thuộc vào thanh ghi rs2.
MEM_br_sel	1	Input	Tín hiệu chọn nhánh trong giai đoạn MEM.
EX_br_sel	1	Input	Tín hiệu chọn nhánh trong giai đoạn EX.
EX_mem_rden	1	Input	Tín hiệu cho phép đọc bộ nhớ trong giai đoạn EX.
MEM_mem_rden	1	Input	Tín hiệu cho phép đọc bộ nhớ trong giai đoạn MEM.
EX_rd_addr	5	Input	Địa chỉ của thanh ghi được ghi trong giai đoạn EX.
MEM_rd_addr	5	Input	Địa chỉ của thanh ghi được ghi trong giai đoạn MEM.
WB_rd_addr	5	Input	Địa chỉ của thanh ghi được ghi trong giai đoạn WB.
ID_rs1_addr	5	Input	Địa chỉ của thanh ghi rs1 trong giai đoạn ID.
ID_rs2_addr	5	Input	Địa chỉ của thanh ghi rs2 trong giai đoạn ID.
EX_rs1_addr	5	Input	Địa chỉ của thanh ghi rs1 trong giai đoạn EX.
EX_rs2_addr	5	Input	Địa chỉ của thanh ghi rs2 trong giai đoạn EX.
ID_inst	32	Input	Lệnh trong giai đoạn ID.
ID_forward_A	2	Output	Tín hiệu điều khiển chuyển tiếp cho toán hạng rs1 trong giai đoạn EX.
ID_forward_B	2	Output	Tín hiệu điều khiển chuyển tiếp cho toán hạng rs2 trong giai đoạn EX.
ID_forward_rs1	1	Output	Tín hiệu cho phép chuyển tiếp cho toán hạng rs1 trong giai đoạn ID.

ID_forward_rs2	1	Output	Tín hiệu cho phép chuyển tiếp cho toán hạng rs2 trong giai đoạn ID.
pc_enable_o	1	Output	Tín hiệu cho phép bộ đếm chương trình (PC).
id_enable_o	1	Output	Tín hiệu cho phép giai đoạn ID.
ex_enable_o	1	Output	Tín hiệu cho phép giai đoạn EX.
mem_enable_o	1	Output	Tín hiệu cho phép giai đoạn MEM.
wb_enable_o	1	Output	Tín hiệu cho phép giai đoạn WB.
id_reset_no	1	Output	Tín hiệu reset giai đoạn ID (active low).
ex_reset_no	1	Output	Tín hiệu reset giai đoạn EX (active low).
mem_reset_no	1	Output	Tín hiệu reset giai đoạn MEM (active low).
wb_reset_no	1	Output	Tín hiệu reset giai đoạn WB (active low).

III. Verification Strategy:

1. Tính ICP của chương trình chuyển giá trị nhị phân từ SWITCH và xuất ra giá trị của LED 7 đoạn:

Tiến hành chạy thử một program để xác minh thiết kế từ đó có thể kết luật được sự hiệu quả và đánh đổi khi áp dụng các kỹ thuật non forwarding, forwarding so với single cycle đã làm ở milestone trước.

Program là đọc giá trị từ switch và từ mã BCD chuyển thành mã led 7 đoạn và xuất ra led 7 đoạn tương ứng.

```
addi x31, x0, 0x78
slli x31, x31, 8
lw x9, 0(x31)

addi x20, x0, 10
add x21, x0,x9
add x22, x0, x0
addi x8,x0,7

addi x31, x0, 0x70
slli x31, x31, 8
addi x31, x31, 0x20



Loop1:
   blt x21, x20, Done1
   addi x22, x22, 1
   sub x21, x21, x20
   jal Loop1
Done1:
   addi x11, x21, 0
   add x25, x11, x0
   jal LOOK_UP
   sb x25, 0(x31)
   addi x31,x31,1

   addi x21,x22,0
   addi x22,x0,0
   addi x8,x8,-1
   bne x8,x0,Loop1



   add x25, x21, x0
   jal LOOK_UP
   sb x25, 0(x31)

laplai: jal laplai

# LOOK_UP:
LOOK_UP:
   addi x26, x25, 0
   beq x26, x0, MA_0
   addi x26, x25, -1
   beq x26, x0, MA_1
   addi x26, x25, -2
   beq x26, x0, MA_2
   addi x26, x25, -3
   beq x26, x0, MA_3
   addi x26, x25, -4
```

```
   beq x26, x0, MA_4
   addi x26, x25, -5
   beq x26, x0, MA_5
   addi x26, x25, -6
   beq x26, x0, MA_6
   addi x26, x25, -7
   beq x26, x0, MA_7
   addi x26, x25, -8
   beq x26, x0, MA_8
   addi x26, x25, -9
   beq x26, x0, MA_9
MA_0:
   addi x25,x0, 0xC0
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_1:
   addi x25,x0, 0xF9
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_2:
   addi x25,x0, 0xA4
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_3:
   addi x25,x0, 0xB0
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_4:
   addi x25,x0, 0x99
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_5:
   addi x25,x0, 0x92
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_6:
   addi x25,x0, 0x82
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_7:
   addi x25,x0 ,0xF8
   jalr  x0, x1, 0  #Jump back to the
address saved in x1
MA_8:
   addi x25,x0 ,0x80
   jalr x0, x1, 0  #Jump back to the
address saved in x1
MA_9:
   addi x25,x0 ,0x90
   jalr x0, x1, 0  #Jump back to the
address saved in x1
```

	MSB							LSB
Đoạn	dp	g	f	e	d	c	b	a
Data	D7	D6	D5	D4	D3	D2	D1	D0

Ký tự (HEX)	Anode chung	Cathode chung	Ký tự (HEX)	Anode chung	Cathode chung
0	C0	3F	8	80	7F
1	F9	06	9	90	6F
2	A4	5B	A	88	77
3	B0	4F	B	83	7C
4	99	66	C	C6	39
5	92	6D	D	A1	5E
6	82	7D	E	86	79
7	F8	07	F	8E	71

Ở Module fpga DE2 dùng Anode chung sơ đồ nối chân không có bit MSB là dp do đó giá trị xuất ra sẽ thay đổi như bảng sau

HEX	Anode chung	Giá trị xuất ra từ o_hex
0	C0	40
1	F9	79
2	A4	24
3	B0	30
4	99	19
5	92	12
6	82	2

Để test chương trình ta nối module thiết kế vào testbench, gán giá chi switch là d'14 để test mã xuất ra từng hex xem đã đúng với mong muốn chưa đồng thời quan sát runtime chương trình.

```
    // Apply some test vectors
    // Test case 1: Simple instruction fetch
    i_io_sw = 32'd14;
    i_io_btn = 32'h00000002;
    #25000;

    $finish;
  end

  // Monitoring Outputs
  initial begin
    $monitor("Time=%0t | LEDR=%h | LEDG=%h | HEX7=%h | HEX6=%h | HEX5=%h | HEX4=%h | HEX3=%h | HEX2=%h | HEX1=%h | HEX0=%h | LCD=%h ",
             $time, o_io_ledr, o_io_ledg,
             o_io_hex7, o_io_hex6, o_io_hex5, o_io_hex4, o_io_hex3, o_io_hex2, o_io_hex1, o_io_hex0, o_io_lcd );
  end
```

1. Testbench nối với module thiết kế

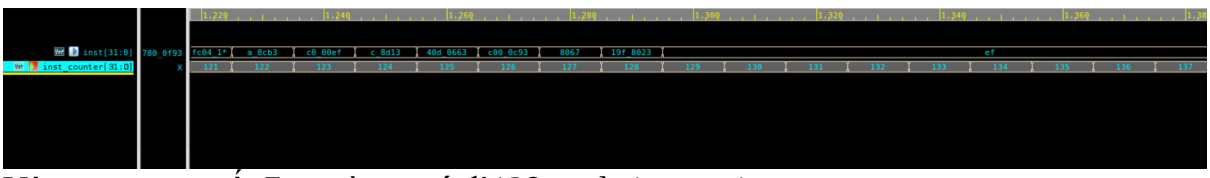
a. Tính số lệnh chương trình thực hiện bằng việc gắn module đếm lệnh vào single cycle

Số lệnh có ích của chương trình chính là số lệnh thực hiện bởi thiết kế single cycle. Giả sử program không có bất kì lệnh nhiễu nào (lệnh không có trong ISA), ta đếm số lệnh bằng cách gắn vào một IP đếm lệnh, counter tự tăng lên thêm một khi có lệnh khác không thực thực thi.

```
module pulse_counter (
   input  wire clk,
   input  wire rst,
   input  wire [31:0] inst,
   output reg [31:0] inst_counter
);
   always @(posedge clk) begin
      if (rst) begin
         inst_counter <= 32'd0;
      end else if (inst != 0) begin
         inst_counter <= inst_counter + 32'd1;
      end
   end

endmodule
```

```
*Verdi* : Begin traversing the scope (design_test), layer (0).
*Verdi* : Enable +all and +mda dumping.
*Verdi* : End of traversing.
Time=0 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=00 | LCD=00000000
Time=315 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=19 | LCD=00000000
Time=475 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=79 | HEX0=19 | LCD=00000000
Time=615 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=755 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=895 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1035 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1175 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1295 | LEDR=00000000 | LEDG=00000000 | HEX7=40 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
$finish called from file "../01_bench/testbench.sv", line 86.
$finish at simulation time              10010
           V C S   S i m u l a t i o n   R e p o r t
Time: 10010 ps
CPU Time:      0.330 seconds;       Data structure size:   0.1Mb
Sun Dec 15 23:15:19 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/last_single_fpga/12_vcs$
```

Nhận xét: Các mã led lần lượt được xuất ra đúng như assembly. Chương trình kết thúc tại lệnh nhảy tại chỗ do đó counter đếm câu lệnh kết thúc khi hoàn thành câu lệnh nhảy tại chỗ (ef)



Vậy program xuất 7seg từ sw có d’129 cycle instruction

b. Chạy thử thiết kế và tính cycle nop của non-forwading.

Để đếm cycle bị nop ta tận dụng xung o_insn_vld từ control unit ở tầng DECODE lần lượt cho đi qua từng tầng và ở tầng WB thì kết nối vào module đếm lệnh nop.

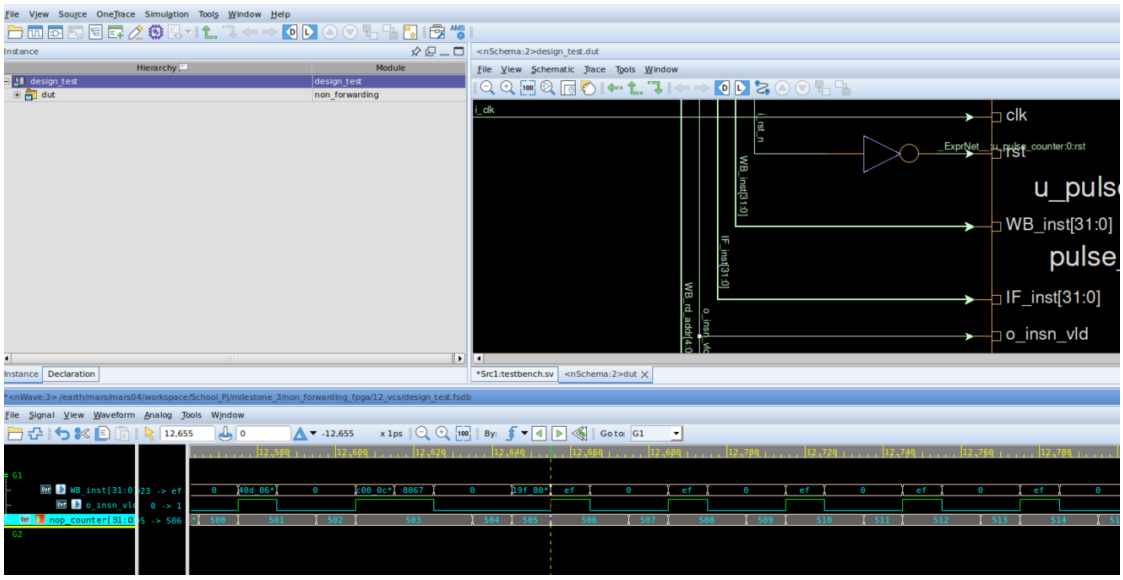
Số lệnh bị nop tương đượng với số lần xung WB_insn tích cực thấp ở mỗi chu kì clk.

Gắn WB_insn từ thiết kế vào bộ đém xung sau

```
module pulse_counter (
   input  wire clk,
   input  wire rst,
   input  wire [31:0] WB_inst,
   input  wire [31:0] IF_inst,
         input wire o_insn_vld,
   output reg [31:0] inst_counter,
   output reg [31:0] nop_counter
);

   reg start_nop_counting;
   always @(posedge clk) begin
      if (rst) begin
         start_nop_counting <= 1'b0; // Reset trạng thái
      end else if (WB_inst != 32'd0) begin
         start_nop_counting <= 1'b1; // Bắt đầu đếm khi WB_inst khác 0
      end
   end
   // Đếm số lần WB_inst = 0 sau khi bắt đầu đếm
   always @(posedge clk) begin
      if (rst) begin
         nop_counter <= 32'd0;   // Reset counter về 0
      end else if (start_nop_counting && o_insn_vld == 1'd0) begin
         nop_counter <= nop_counter + 32'd1;  // Tăng biến đếm khi WB_inst = 0
      end
   end

endmodule
```

```
*Verdi* : Begin traversing the scope (design_test), layer (0).
*Verdi* : Enable +all and +mda dumping.
*Verdi* : End of traversing.
Time=0 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=00 | LCD=00000000
Time=9175 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=30 | LCD=00000000
Time=10445 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=24 | HEX0=30 | LCD=00000000
Time=11035 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
Time=11415 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=79 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
Time=11745 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=40 | HEX3=79 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
Time=12075 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=40 | HEX4=40 | HEX3=79 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
Time=12405 |  LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=79 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
Time=12645 |  LEDR=00000000 | LEDG=00000000 | HEX7=40 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=79 | HEX2=19 | HEX1=24 | HEX0=30 | LCD=00000000
$finish called from file "./../01_bench/testbench.sv", line 88.
$finish at simulation time              50010
         V C S   S i m u l a t i o n   R e p o r t
Time: 50010 ps
CPU Time:      0.410 seconds;       Data structure size:   0.1Mb
Sat Dec 14 23:45:03 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/non_forwarding_fpga/12_vcs$
```

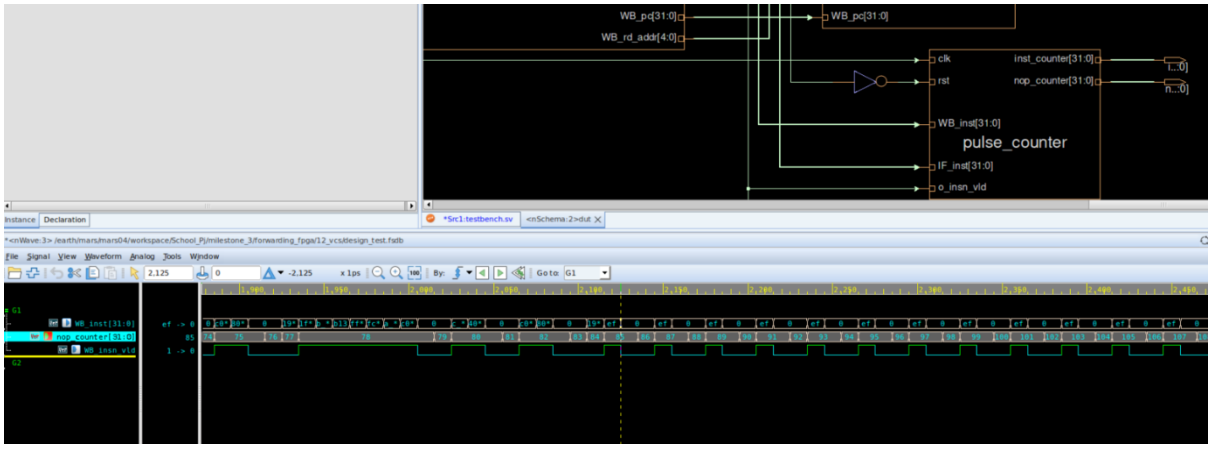
Nhận xét: tương tự như single cycle nhưng ở Non forwarding runtime tăng lên 10 lần. Nguyên nhân là do xuất hiện quá nhiều data hazard và program có nhiều lần nhảy. Do đó thời gian nop rất dài và nhiều lần ảnh hưởng tới runtime nhưng tần số của Non-fw được tăng lên 2 lần.



Kết quả từ module điếm lệnh nop cho lấy có 565 cycle bị nop rất nhiều ảnh hưởng tới runtime của chương trình.

c. Chạy thử chương trình với thiết kế Forwarding và tính số lệnh nop của forwarding:

```
FSDB Dumper for VCS, Release Verdi_V-2023.12-SP2, Linux x86_64/64bit, 05/26/2024
(C) 1996 - 2024 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi* : Create FSDB file 'design_test.fsdb'
*Verdi* : Begin traversing the scope (design_test), layer (0).
*Verdi* : Enable +all and +mda dumping.
*Verdi* : End of traversing.
Time=0 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=00 | LCD=00000000
Time=465 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=00 | HEX0=19 | LCD=00000000
Time=725 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=00 | HEX1=79 | HEX0=19 | LCD=00000000
Time=965 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=00 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1205 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=00 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1445 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=00 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1685 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=00 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=1925 | LEDR=00000000 | LEDG=00000000 | HEX7=00 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
Time=2105 | LEDR=00000000 | LEDG=00000000 | HEX7=40 | HEX6=40 | HEX5=40 | HEX4=40 | HEX3=40 | HEX2=40 | HEX1=79 | HEX0=19 | LCD=00000000
$finish called from file "./../01_bench/testbench.sv", line 88.
$finish at simulation time              25010
           V C S   S i m u l a t i o n   R e p o r t
Time: 25010 ps
CPU Time:      0.350 seconds;       Data structure size:   0.1Mb
Sun Dec 15 10:39:48 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/forwarding_fpga/12_vcs$
```



Do có lệnh nhảy tại chỗ (mã lệnh là EF) nên chương trình sẽ thực hiện nop 2 lệnh tiếp theo sau đó nên cuối chương trình sẽ là vòng lặp của EF và 2 lệnh 00,

Vậy số lệnh nop của FW là 85

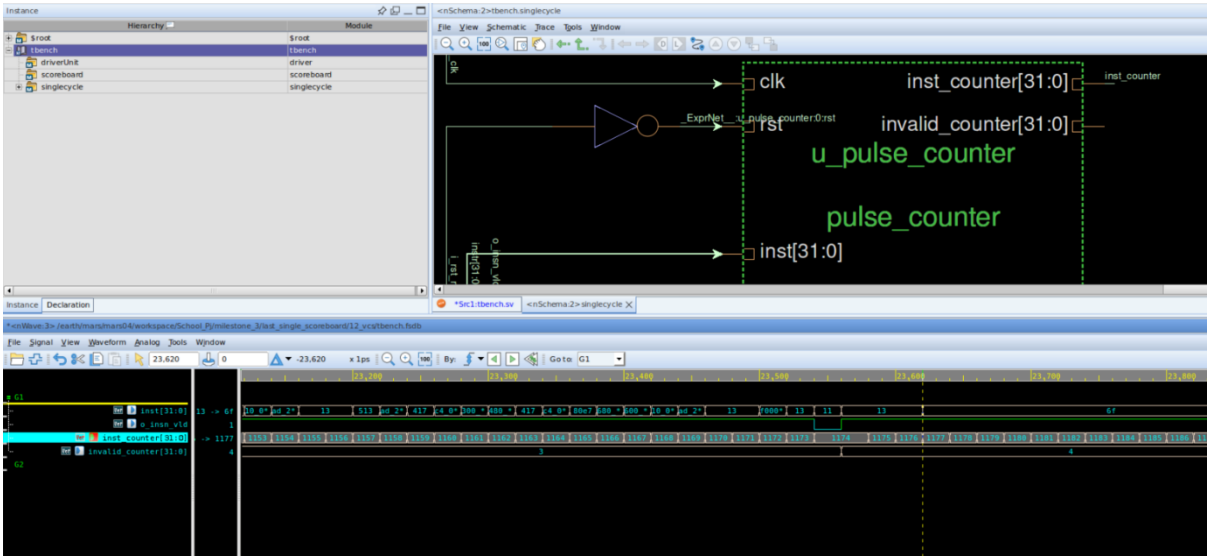
Kết luận forwarding đã giải quyết được các data hazard ngoại trừ các nhóm lệnh load vẫn có penalty là 1 cycle

	Single cycle	Nop cycle	Total cycle	ICP	Run-time
Single cycle	129	0	129	1	1295ps
Non-fw	129	506	635	0.203	12646ps
Fw	129	85	214	0.602	2105ps

2. Tính CPI của testcase Scoreboard:

a. Tính số single cycle instruction:

```
[    14620]::19::SLTIU.............PASSED
[    15140]::20::LUI...............PASSED
[    15600]::21::AUIPC.............PASSED
[    15900]::22::LW................PASSED
[    16720]::23::LH................PASSED
[    17280]::24::LB................PASSED
[    17760]::25::LHU...............PASSED
[    18340]::26::LBU...............PASSED
[    18820]::27::SW................PASSED
[    19260]::28::SH................FAILED
[    19880]::29::SB................PASSED
[    20480]::30::misaligned........FAILED
[    20820]::31::BEQ...............PASSED
[    21040]::32::BNE...............PASSED
[    21260]::33::BLT...............PASSED
[    21660]::34::BGE...............PASSED
[    22060]::35::BLTU..............PASSED
[    22460]::36::BGEU..............PASSED
[    22860]::37::JAL...............PASSED
[    23160]::38::JALR..............PASSED
[    23460]::39::illegal_insn......FAILED

Timeout...

DUT is considered       P A S S E D

$finish called from file "../01_bench/tlib.svh", line 22.
$finish at simulation time                50000
           V C S   S i m u l a t i o n   R e p o r t
Time: 50000 ps
CPU Time:      0.380 seconds;       Data structure size:   0.1Mb
Sun Dec 15 10:54:34 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/last_single_scoreboard/12_vcs$
```

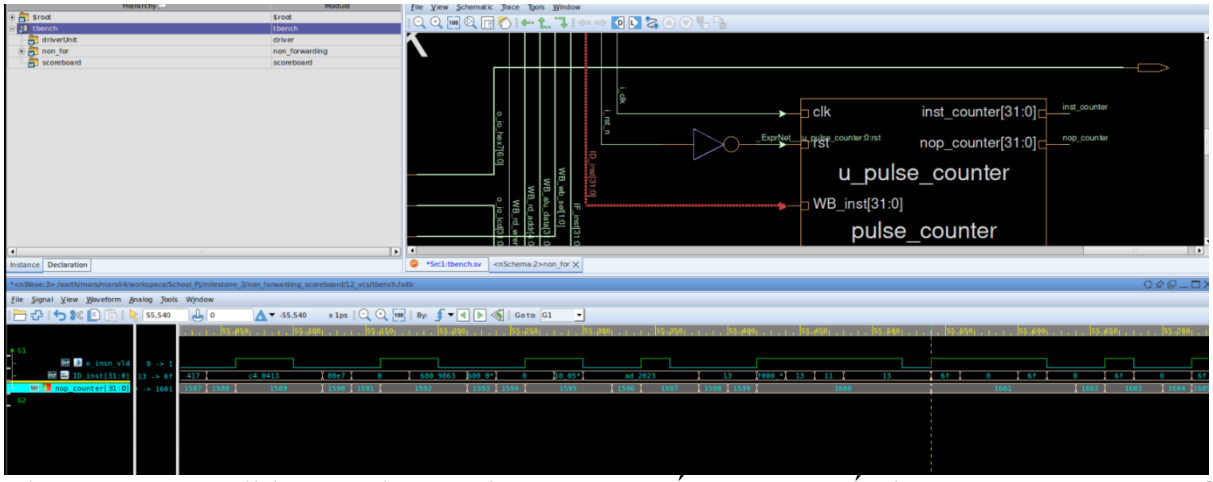


Scoreboard có 1177 câu lệnh với 4 câu lệnh ảo

b. Tính số lệnh nop của non forwarding:

Do scoreboard có test 4 lệnh ảo nên số lệnh bị data hazard và control hazard của non fw là

```
[    44560]::28::SH..............FAILED
[    46220]::29::SB..............PASSED
[    47700]::30::misaligned.....FAILED
[    48600]::31::BEQ.............PASSED
[    49160]::32::BNE.............PASSED
[    49720]::33::BLT.............PASSED
[    50780]::34::BGE.............PASSED
[    51880]::35::BLTU............PASSED
[    52940]::36::BGEU............PASSED
[    54040]::37::JAL.............PASSED
[    54740]::38::JALR............PASSED
[    55440]::39::illegal_insn.....FAILED

Timeout...

DUT is considered       P A S S E D

$finish called from file "./../01_bench/tlib.svh", line 22.
$finish at simulation time              60000
          V C S   S i m u l a t i o n   R e p o r t
Time: 60000 ps
CPU Time:      0.420 seconds;       Data structure size:   0.1Mb
Sun Dec 15 14:40:53 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/non_forwarding_scoreboard/12_vcs$
```

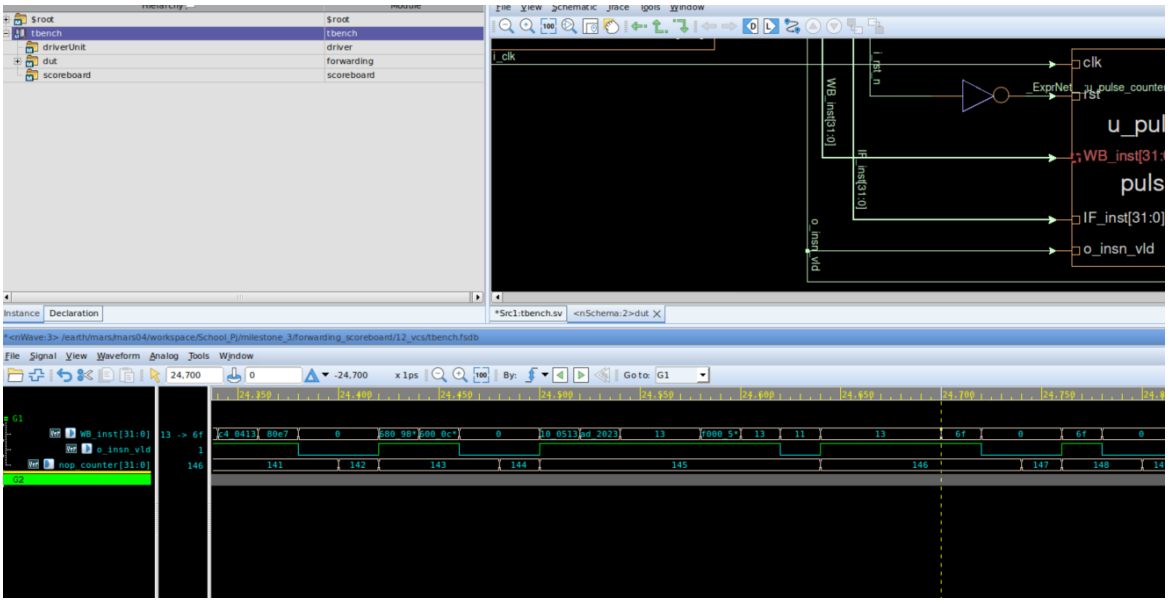


Lệnh có 1601 unvalid trong đó có 4 lệnh ảo vậy số lệnh thức tế bị nop do data hazard và nhóm lệnh nhảy là 1597

c. Tính số lệnh nop của forwarding:

Ở forwarding, data hazard đã được xử lý chỉ còn lệnh nop do các nhóm lệnh nhảy và lệnh load vẫn bị nop 1 clock:

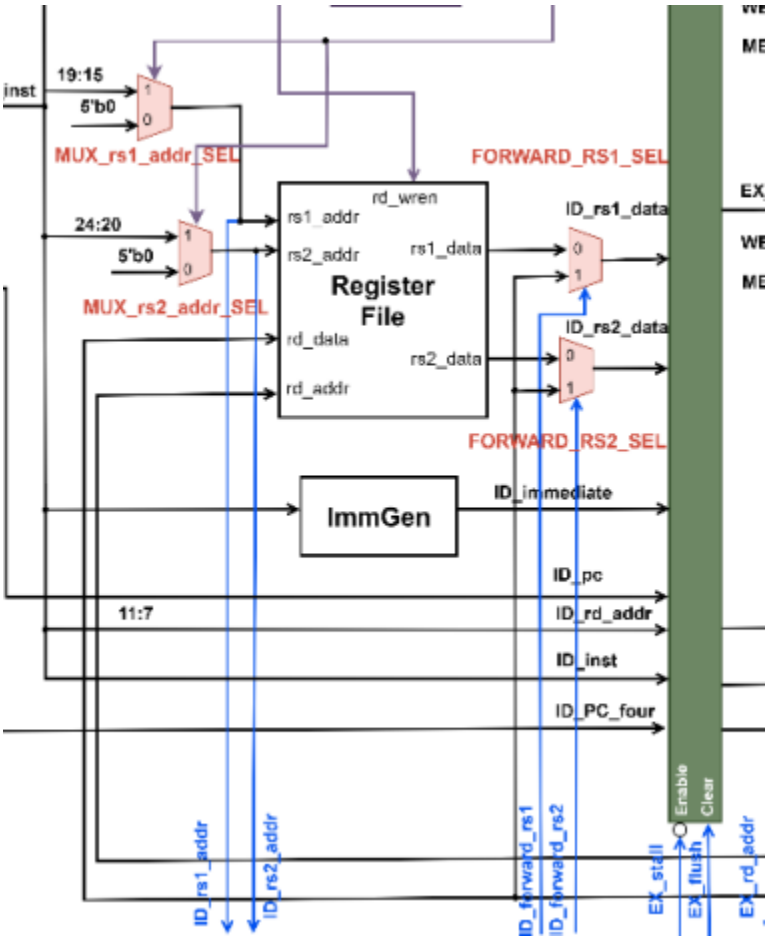
```
[    16940]::22::LW..............FAILED
[    17620]::23::LH..............FAILED
[    18060]::24::LB..............FAILED
[    18460]::25::LHU.............FAILED
[    18900]::26::LBU.............FAILED
[    19300]::27::SW..............FAILED
[    19680]::28::SH..............FAILED
[    20140]::29::SB..............FAILED
[    20560]::30::misaligned......FAILED
[    21020]::31::BEQ.............FAILED
[    21280]::32::BNE.............FAILED
[    21540]::33::BLT.............FAILED
[    22060]::34::BGE.............FAILED
[    22620]::35::BLTU............FAILED
[    23140]::36::BGEU............FAILED
[    23700]::37::JAL.............FAILED
[    24120]::38::JALR............FAILED
[    24540]::39::illegal_insn.....FAILED

Timeout...

DUT is considered       P A S S E D

$finish called from file "./../01_bench/tlib.svh", line 22.
$finish at simulation time              50000
          V C S   S i m u l a t i o n   R e p o r t
Time: 50000 ps
CPU Time:      0.410 seconds;       Data structure size:   0.1Mb
Sun Dec 15 14:29:42 2024
mars04@himalia:~/workspace/School_Pj/milestone_3/forwarding_scoreboard/12_vcs$ make verdi
verdi -sverilog \
```



Vậy số lệnh nop thực tế còn lại của model còn 142

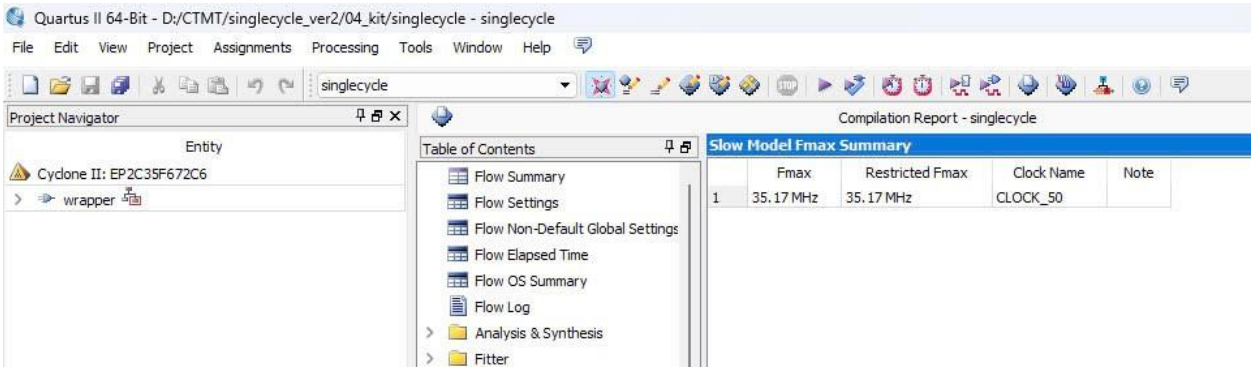
	Single cycle	Nop cycle	Total cycle	ICP	Run-time
Single cycle	1177	0	1177	1	23406ps
Non-fw	1177	1597	2774	0.424	55440ps
Fw	1177	142	214	0.89	25540ps

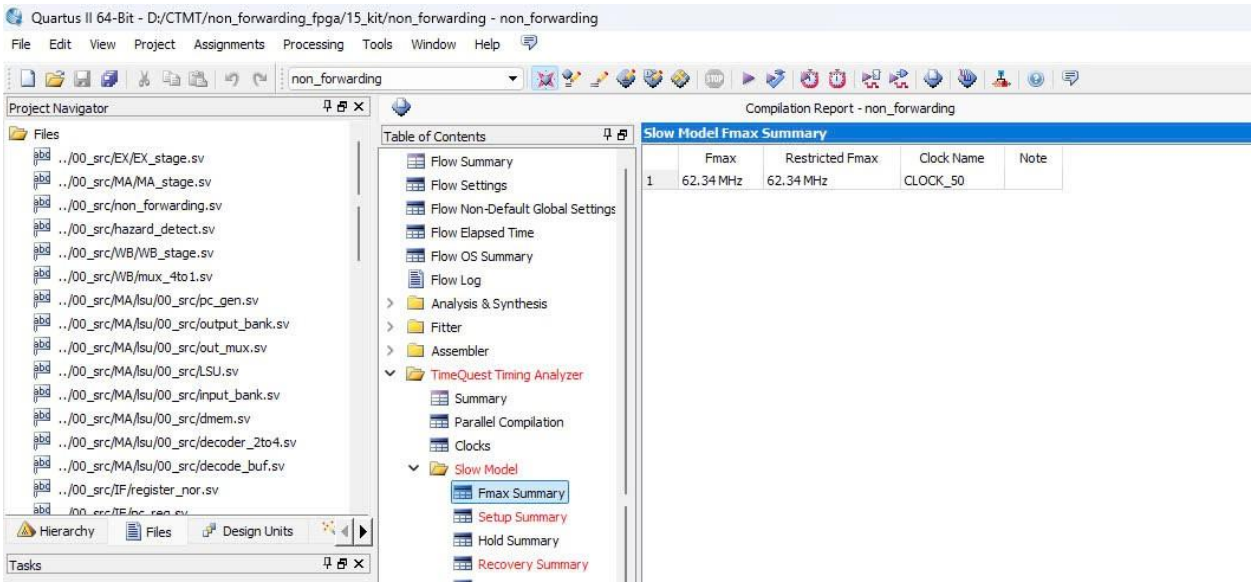
IV.Advanced Design:

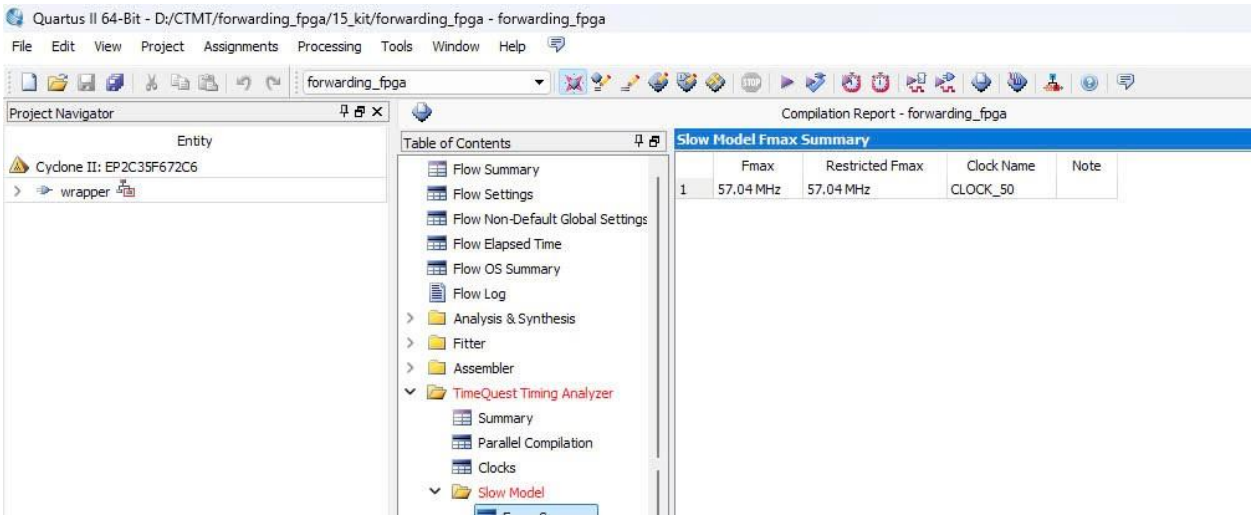


Ở thiết kết forwarding tất cả các loại data hazard đã được xử lý triệt để bằng forward data để không phải nop câu lệnh nhiều lần. Nhưng việc xử lý data hazard ở lệnh load là không thể nhưng nếu có thể forward data từ WB ra thẳng buffer ở cuối tầng ID thì có thể tránh được một cycle penalty khi mà rd phải đi qua Regfile.

V. Evaluation:







Nhận xét: F_{max} của non-fowarding và forwarding đã tăng gấp đôi so với single cycle

VI.Conclusion:

Với thiết kế non-forwarding và forwarding đã hoàn thành được mục tiêu thiết kế và tăng Fmax của cpu lên gấp đôi so với single cycle.

	Single cycle	Nop cycle	Total cycle	ICP	Run-time
Single cycle	129	0	129	1	1295ps
Non-fw	129	506	635	0.203	12646ps
Fw	129	85	214	0.602	2105ps

Với số liệu trên từ program xuất led sủ dụng rất nhiều lệnh nhảy cho thấy. Non-forward mặc du Fmax tăng nhưng runtime giảm bởi vì trễn lan truyền khi ta thêm quá nhiều thiết kế combinational, và xảy ra nhiều nop cycle.

Forwarding thì giải quyết được triệt để việc data hazard nhưng không thể xử lý triệt để việc khi gặp lệnh branch hoặc jump phải nop do đó ICP chỉ lên được 0.6.

Giải pháp tối ưu hơn là thiết kế Branch prediction để giải quyết các nop cycle nêu trên.

REFERENCES

[1] Dan Garcia and Justin Yokota. Pipelining I, II, III – Great Ideas in Computer Architecture (Machine Structures). CS61C Course Material. University of California, Berkeley. 2023.

[2] Hai Cao. Pipelining and Branch Prediction. EE4423. Dept. of Electronics Engineering, Ho Chi Minh City University of Technology. Apr. 2023.

[3] Ben H. Juurlink. Course 2 ILP, DLP, TLP. YouTube, 2018. url: https://www. youtube.com/playlist?list=PLeWkeA7esB-PcOTrTCvAsaCArnCMQkcNv.

[4] Hiromu Miyazaki et al. "RVCoreP: An optimized RISC-V soft processor of fivestage pipelining". In: The Institute of Electronics, Information and Communication Engineers (2020). url: https://arxiv.org/pdf/2002.03568.pdf.

[5] Onur Mutlu. Digital Design and Computer Architecture - Lecture 17: Advanced Branch Prediction. YouTube, Apr. 2023. url: https://www.youtube.com/watch?v=g9H_ 79ITdbM.

[6] Shaaban. CPU Performance Evaluation: Cycles Per Instruction (CPI). EECC550 - Shaaban. Dec. 2011. url: http://meseec.ce.rit.edu/eecc550-winter2011/550- 12-6-2011.pdf.

[7] University of Utah. Lecture 8: Branch Prediction, Dynamic ILP, CS6810 Course. CS6810 Course Material. University of Utah