BÁO CÁO ĐỒ ÁN CUỐI KÌ

Implement a simple 5 stage pipeline 16bit/32bit processor using Verilog

GVHD: ThS. Hồ Ngọc Diễm

Họ tên: Trần Triều Trung

MSSV: 21522727

MỤC LỤC

1. Giới thiệu đề tài

Pipeline là kỹ thuật mà các công việc tiếp theo được thực hiện khi công việc hiện tại chưa hoàn tất nhằm tận dụng tối đa tài nguyên thiết bị để tăng hiệu suất công việc.

Khi thực thi, các lệnh MIPS được chia làm 5 công đoạn:

1. Nạp lệnh từ bộ nhớ (Fetch)

2. Giải mã lệnh và đọc các thanh ghi cần thiết (MIPS cho phép đọc và giải mã đồng thời) (Decode)

3. Thực thi các phép tính hoặc tính toán địa chỉ (Execute)

4. Truy xuất các toán hạng trong bộ nhớ (Mem)

5. Ghi kết quả cuối vào thanh ghi (Write back)

- Trong đề tài gồm nội dung thực hiện sau:
  - Sử dụng ngôn ngữ Verilog
  - Thiết kế các lệnh cơ bản trong MIPS
  - Đưa ra vấn đề và giải quyết xung đột cơ bản trong MIPS pipeline

2. Phân tích đề tài

a. Datapath

Để thiết kế ta dựa vào sơ đồ:

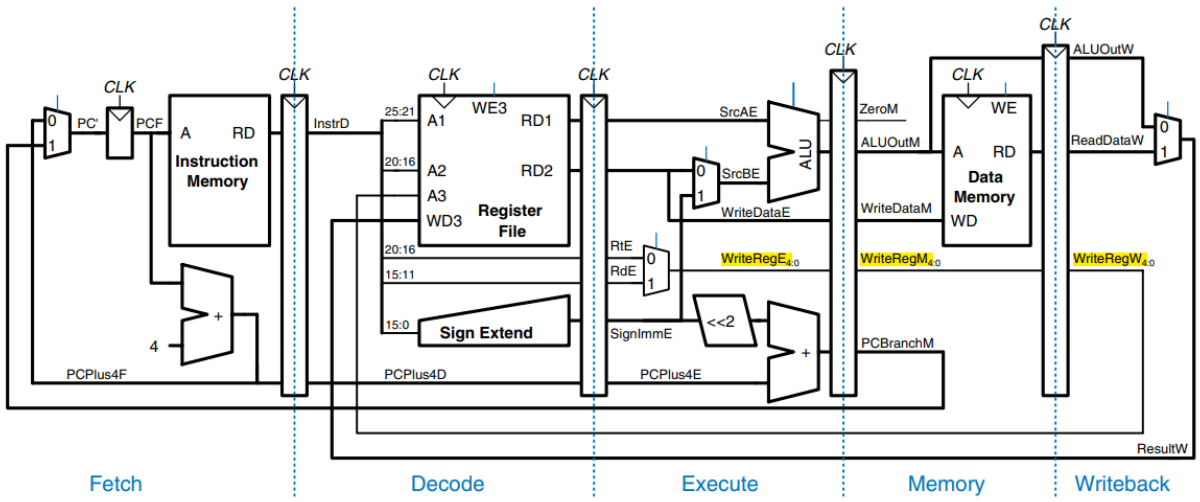


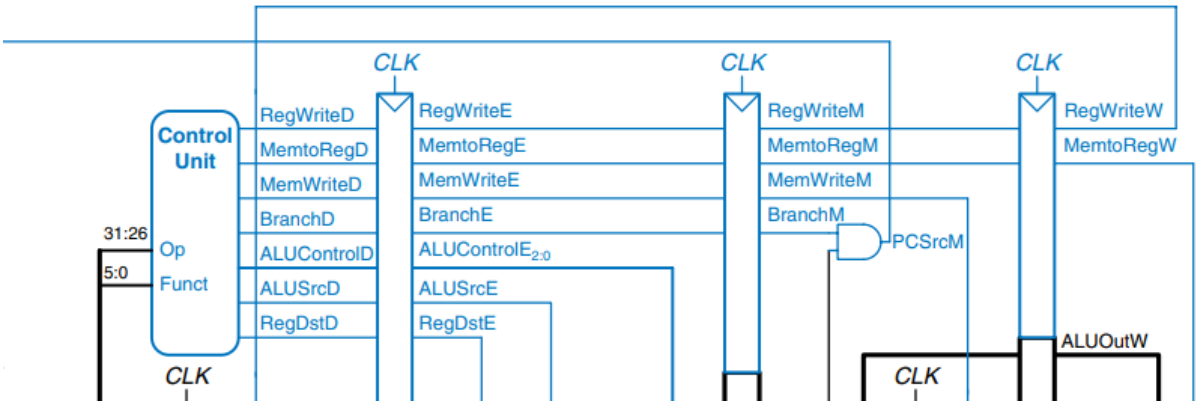
Figure 7.46 Corrected pipelined datapath

Trong Datapath này gồm các modules sau:

- Imem
- Register File
- Sign Extend
- ALU
- MUX
- Shift
- Dmem
- Add4
- Add

Code được thể hiện sau: [PipelineMips\Modules]

b. Controller



Trong controller này gồm có các thành phần sau:

- 5 reg Fetch/ Decode/ Excute/ Memory/ Writeback
- 5 reg IFID/ IDIE/ IEMem/ MemWB

Code được thể hiện sau: PipelineMips\PipelineStage , PipelineMips\PipelineReg

3. Xung đột

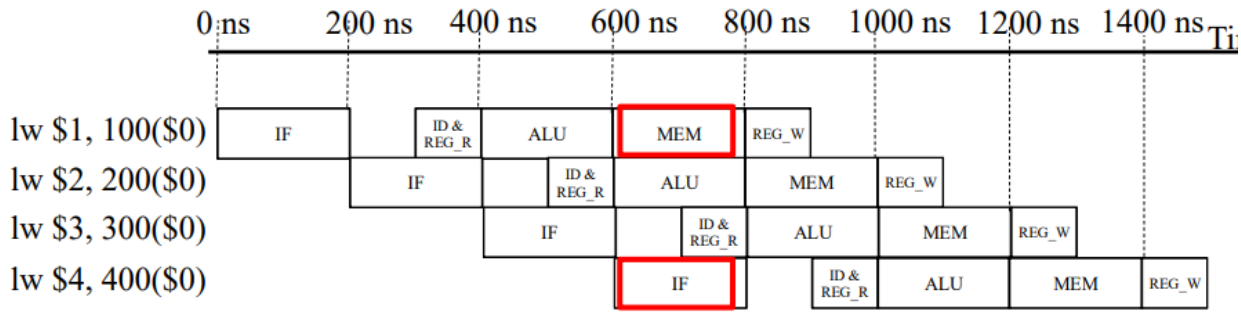
Các xung đột có thể xảy ra khi áp dụng kỹ thuật pipeline (Pipeline Hazards):

Xung đột là trạng thái mà lệnh tiếp theo không thể thực thi trong chu kỳ pipeline ngay sau đó (hoặc thực thi nhưng sẽ cho ra kết quả sai), thường do một trong ba nguyên nhân sau:

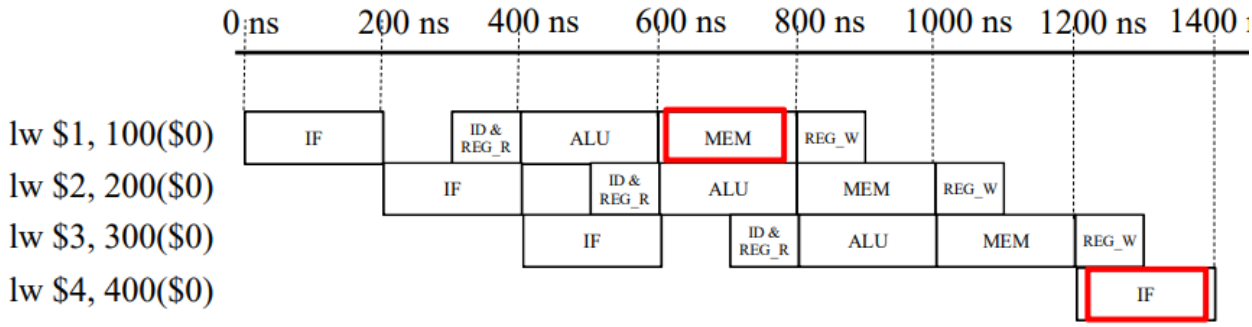
a. Xung đột cấu trúc

Xảy ra khi hai hoặc nhiều công đoạn của các lệnh khác nhau sử dụng cùng một tầng cấu trúc phần cứng (tài nguyên phần cứng trong cùng thời gian). Nói cách khác, xung đột cấu trúc xảy ra khi có hai lệnh cùng truy xuất vào một tài nguyên phần cứng nào đó cùng một lúc.

Ví dụ sau:



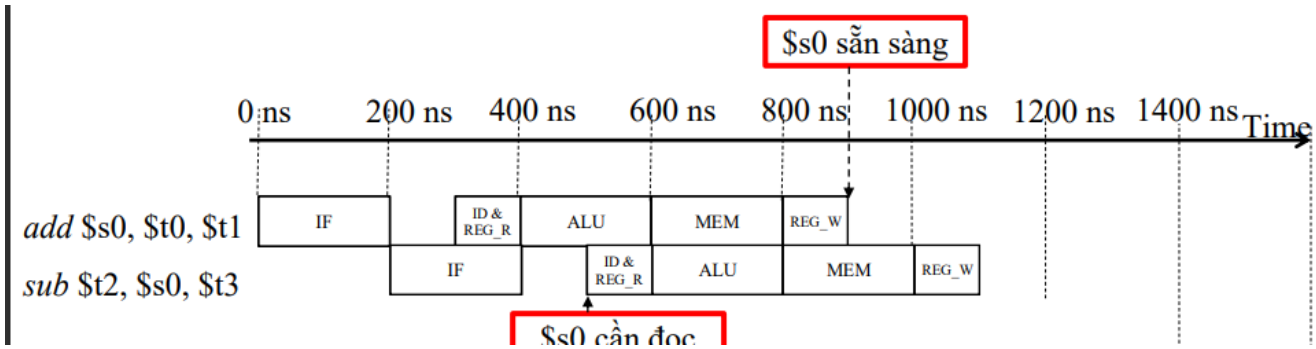
Có thể thấy rằng tại đây dùng chung Memory dẫn đến không thể truy cập cùng lúc, ta giải quyết bằng cách dời đi:



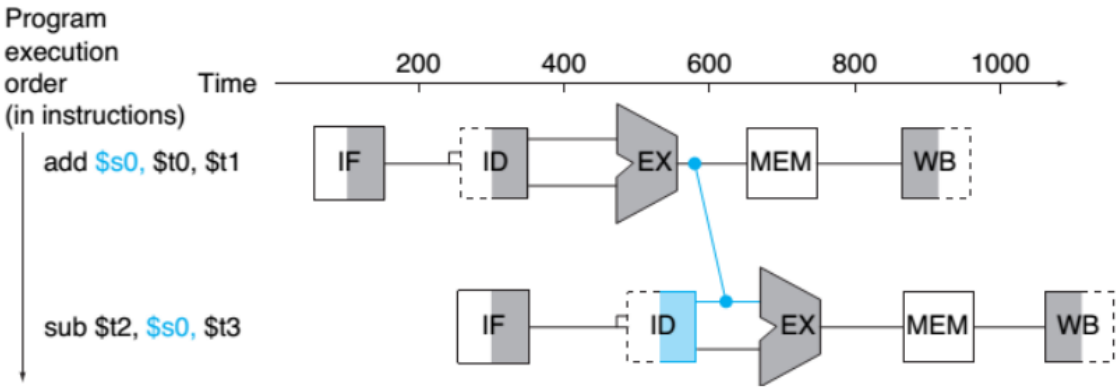
b. Xung đột dữ liệu

Xảy ra khi tại thời điểm lệnh kế tiếp cần sử dụng kết quả dữ liệu của lệnh trước đó, nhưng lệnh trước đó chưa hoàn thành việc thực thi

Ví dụ sau:



Tại đây cần lấy dữ liệu của s0 nhưng chưa được thực thi xong nên giải quyết bằng cách Forwarding(Nhìn trước) để giải quyết vấn đề:



Trong thiết kế của Datapath sẽ chỉnh sửa như sau:

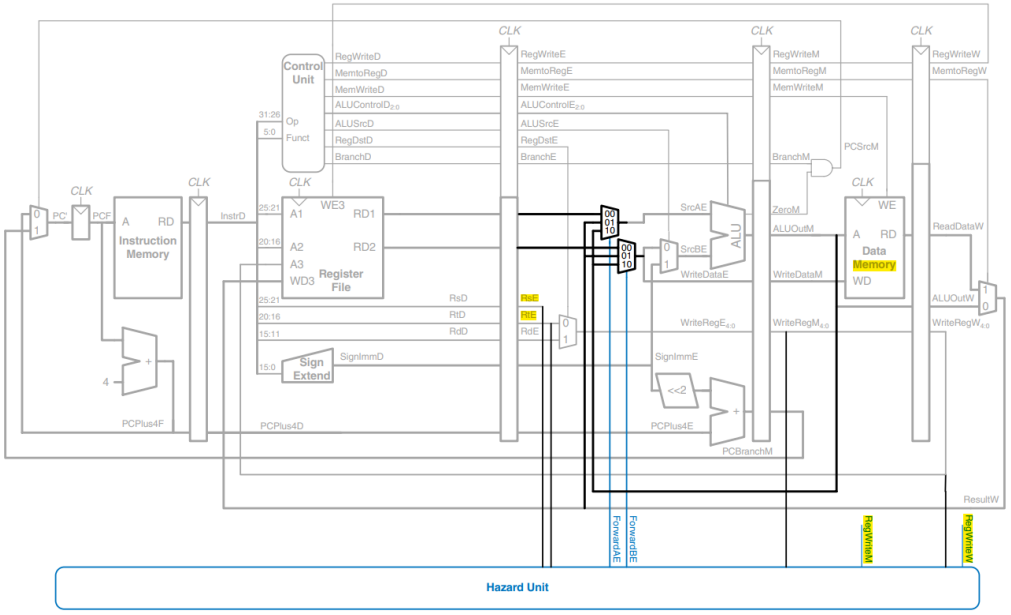


Figure 7.50 **Pipelined processor with forwarding to solve hazards**

Ở kỹ thuật này ta sẽ dựa vào thanh ghi cần ghi ở **Stage Memory** hoặc **Stage WriteBack** kiểm tra xem có giống với thanh ghi cần ghi ở **Stage Excute** hay không, nếu giống sẽ lấy kết quả có được từ 1 trong 2 Stage đó đưa vào bằng MUX.

c. Xung đột điều khiển

Xảy ra ngay sau các lệnh điều khiển nhảy - lệnh liền kề được thực thi khi mà giá trị địa chỉ của lệnh kế tiếp chương trình muốn nhảy đến chưa được tính toán xong bởi lệnh hiện tại.

Ví dụ sau:

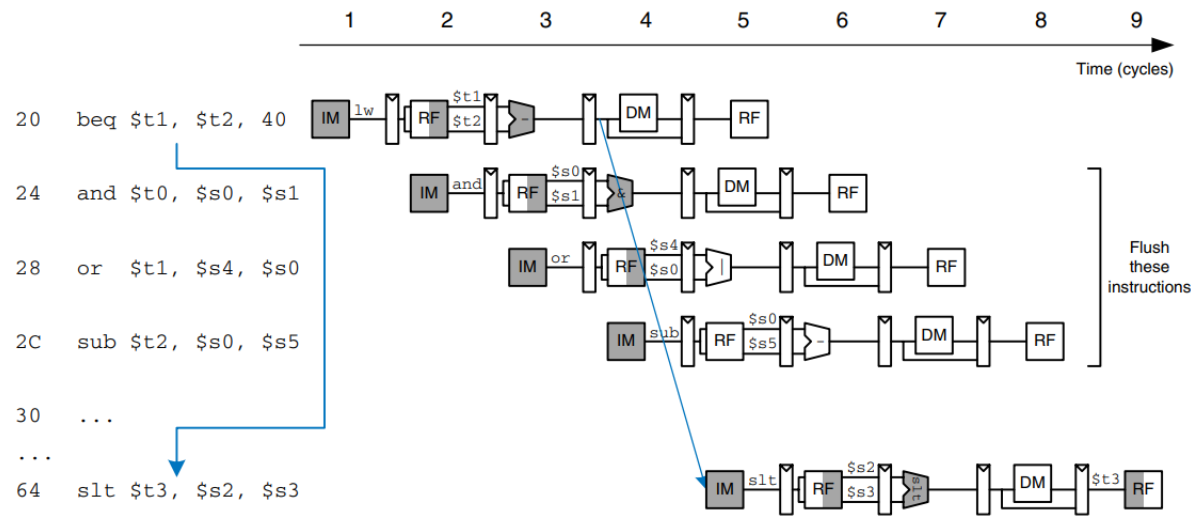


Figure 7.54 **Abstract pipeline diagram illustrating flushing when a branch is taken**

Chúng ta sẽ cần **3** clock mới xử lý được vấn đề Nhảy nếu nó xảy ra, để giải quyết vấn đề ta sẽ dời phần nhảy lên ID giải quyết nhanh hơn.

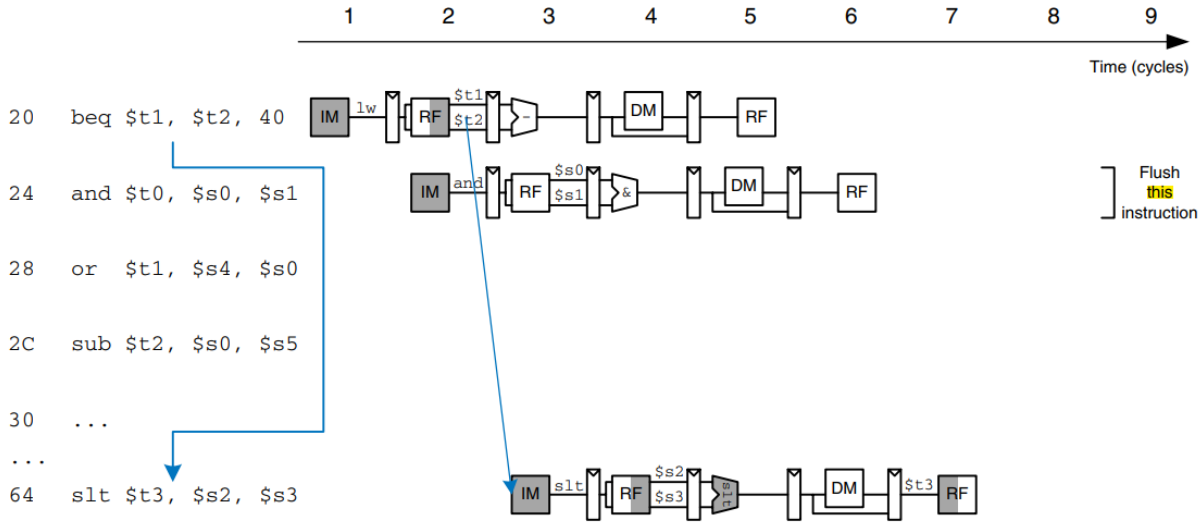


Figure 7.55 **Abstract pipeline diagram illustrating earlier branch decision**

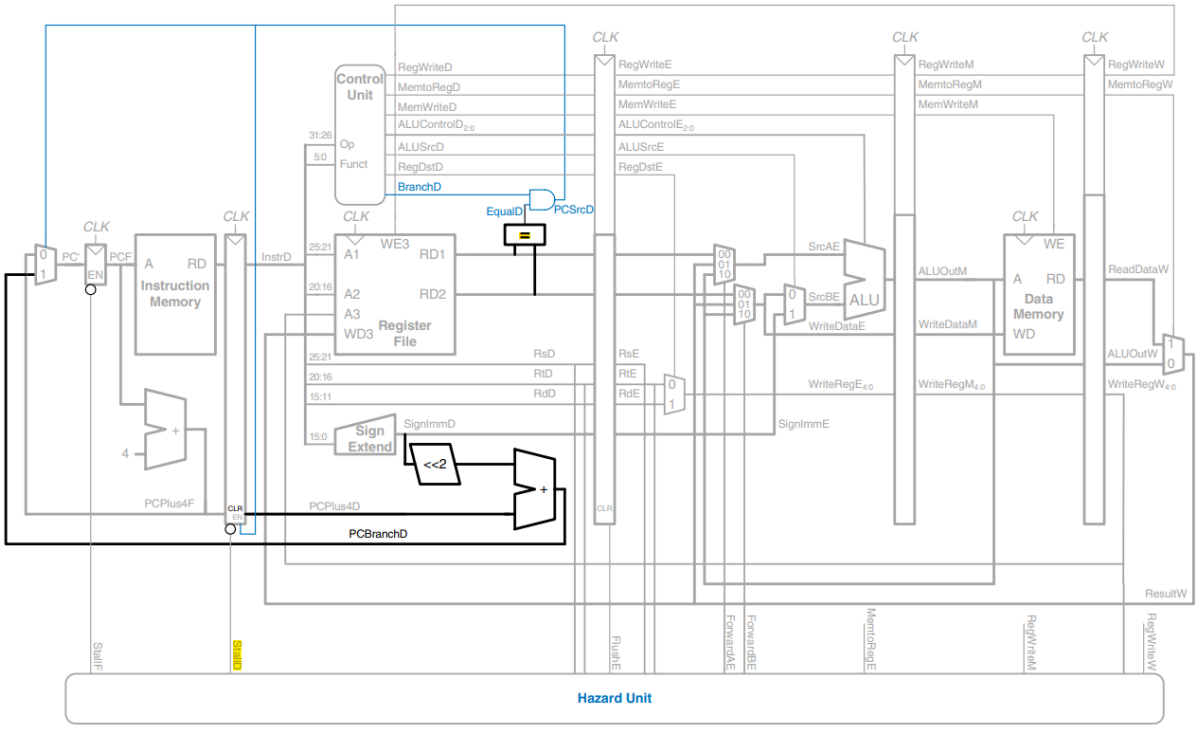


Figure 7.56 **Pipelined processor handling branch control hazard**

Nếu dời sớm sẽ gây ra vấn đề Nạp lệnh tiếp theo vào ảnh hưởng nên sẽ đưa ra 1 signal mới từ lệnh Branch như sau: Nếu PCSrcD = 1 sẽ Flush thanh ghi IFID tránh nạp dữ liệu mới

Đây là code giải quyết các vấn đề xung đột trên: PipelineMips\PipelineReg\Hazard.v

Còn về việc các thành phần thanh đổi ở Datapath đã được code bổ sung vào phần **Stage** và **Reg** đã **đề cập**.

4. Kết quả giải quyết xung đột

Sau khi đã giải quyết xong thì sẽ đưa ra 1 bài toán cụ thể kiểm tra, ở bài này sẽ là Fibonanci

Code **Assembly**:

```
addi $6, $0, 20
addi $2, $0, 0
addi $3, $0, 1
addi $7, $0, 1
beq $6, $0, Out
Loop:
sub $6, $6, $7
add $4, $2, $3
add $3, $2, $3
sub $2, $4, $2
sw $4, 0($t1)
addi $t1, $t1, 4
bne $6, $0, Loop
Out:
and $4, $4, $4
```

Chuyển sang **mã máy**:

```
00000000
20060020
20020000
20030001
20070001
10c00007
00c73022
00432020
00431820
00821022
Ad240000
21290004
14c0fff9
00842024
```

Ở Fibo này sẽ lưu 32 số fibo vào Dmem, như sau:

1	2	3	5	8	13	21
34	55	89	144	233	377	610
987	1597	2584	4181	6765	10946	17711
28657	46368	75025	121393	196418	317811	514229
832040	1346269	2178309	3524578	0	0	0
0	0	0	0	0	0	0

5. Kết luận và hướng phát triển

■ Kết luận

Giải quyết được các vấn đề cơ bản của pipeline

■ Hướng phát triển và công việc tiếp theo

Phát triển thêm lệnh

Thêm bộ nhân, chia

Superpipelining

Phát triển bộ dịch sang mã máy

6. Tài liệu tham khảo

- Digital Design and Computer Architecture by David Money Harris & Sarah L.Harris

  textbook-ddca-en.pdf

----------HẾT----------