ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Computer Architecture (CO2008)

Bài báo cáo Chapter 4.2: PIPELINE (Lab 7)

Họ và tên	Nguyễn Phúc Tiến
MSSV	2014725
Lớp	L03

Ngày 14 tháng 5 năm 2022

Mục lục

1 Xác định clock cycle

Cho thời gian delay của các khối như hình 1

Phần cứng	Delay (ns)
Instruction memory	150
Register	100
ALU	100
Data memory	150
Các bộ phần cứng khác	0

Hình 1: *delay của các khối phần cứng*

Xét đoạn chương trình như sau:

```
        addi    $t1, $zero, 100
        addi    $t2, $zero, 0
loop:
        beq     $t1, $t2, exit
        addi    $t1, $t1, -1
        addi    $t2, $t2, 1
        j       loop
```

(a) Xác định clock cycle của hệ thống single clock, multi clock và pipeline clock.

- Single clock cycle = thời gian thực thi lệnh dài nhất (lệnh load).
  = I-Mem + Register + ALU + D-Mem
  = (150 + 100 + 100 + 150)ns = 500ns.
- Multiple clock cycle = max của khối có độ delay cao nhất.
  = max(I-Mem, Register, ALU, D-Mem) = 150ns.
- Pipeline clock cycle cũng giống như multiple cycle = 150ns.

(b) Xác định thời gian thực thi của chương trình trên khi chạy với hệ thống single cycle, multi cycle và pipeline cycle (không xét stall).

- Vòng lặp thực thi 50 lần ⇒ Tổng số lệnh trong vòng lặp = 4 * 50 = 200, lệnh beq cuối cùng để thoát ra khỏi vòng lặp và 2 lệnh đầu tiên nữa. Vậy tổng số lệnh của chương trình = 200 + 1 + 2 = 203 lệnh.
- Lệnh addi thực thi trong 4 giai đoạn, beq thực thi trong 3 giai đoạn và lệnh jump thực thi trong 2 giai đoạn.
- CPI của hệ thống single cycle = 1 (1 chu kỳ thực thi 1 lệnh).
- CPI của hệ thống Multiple cycle = $\frac{2*4+(3+4*2+2)*50+3}{203} = \frac{661}{203}$.
- CPI lý tưởng của hệ thống Pipeline = 1.

Thời gian thực thi của chương trình = IC * CPI * Cycle time.
⇒ Thời gian chạy trên hệ thống single cycle = 1 x 203 x 500ns = $0.1015ms$
⇒ Thời gian chạy trên hệ thống multiple cycle = $\frac{661}{203}$ x 203 * 150ns = $0.09915ms$
⇒ Thời gian chạy trên hệ thống pipeline = 1 x (203 + 5 - 1) * 150ns = $0.03105ms$

(c) Tính speed up của hệ thống pipeline với hệ thống multi cycle và với single cycle.

Speedup pipeline/single = $\frac{0.1015m}{0.03105m} = 3.2689$

Speedup pipeline/multiple = $\frac{0.09915m}{0.03105m} = 3.1932$

(d) Khi delay ALU thay đổi từ 100 –> 150. Tính lại kết quả câu a,b,c.

- Kết quả sau khi tính lại câu a.
  + Single clock cycle = 550ns
  + Multiple clock cycle = 150ns
  + Pipeline clock cycle = 150ns
- Kết quả sau khi tính lại câu b.
  + Thời gian chạy trên hệ thống single cycle = 1 x 203 x 550ns = $0.11165ms$
  + Thời gian chạy trên hệ thống multiple và pipeline cycle không đổi.
- Kết quả sau khi tính lại câu c.
  Speedup pipeline/single = $\frac{0.11165m}{0.03105m} = 3.5958$
  Speedup pipeline/multiple là không đổi.

2 Xử lý Hazard

Dùng lại đoạn code của Bài 1:

```
        addi    $t1, $zero, 100
        addi    $t2, $zero, 0
loop:
        beq     $t1, $t2, exit
        addi    $t1, $t1, -1
        addi    $t2, $t2, 1
        j       loop
```

(a) Xác định sự phụ thuộc dữ liệu trong đoạn chương trình trên.

- Sự phụ thuộc dữ liệu ở đây là: Read After Write – RAW Hazard.
- Trước khi vòng lặp bắt đầu. Lệnh (3) - beq - phụ thuộc lệnh (1) và (2). Trong vòng lặp, lệnh (3) - beq - phụ thuộc lệnh (4), (5) - addi.

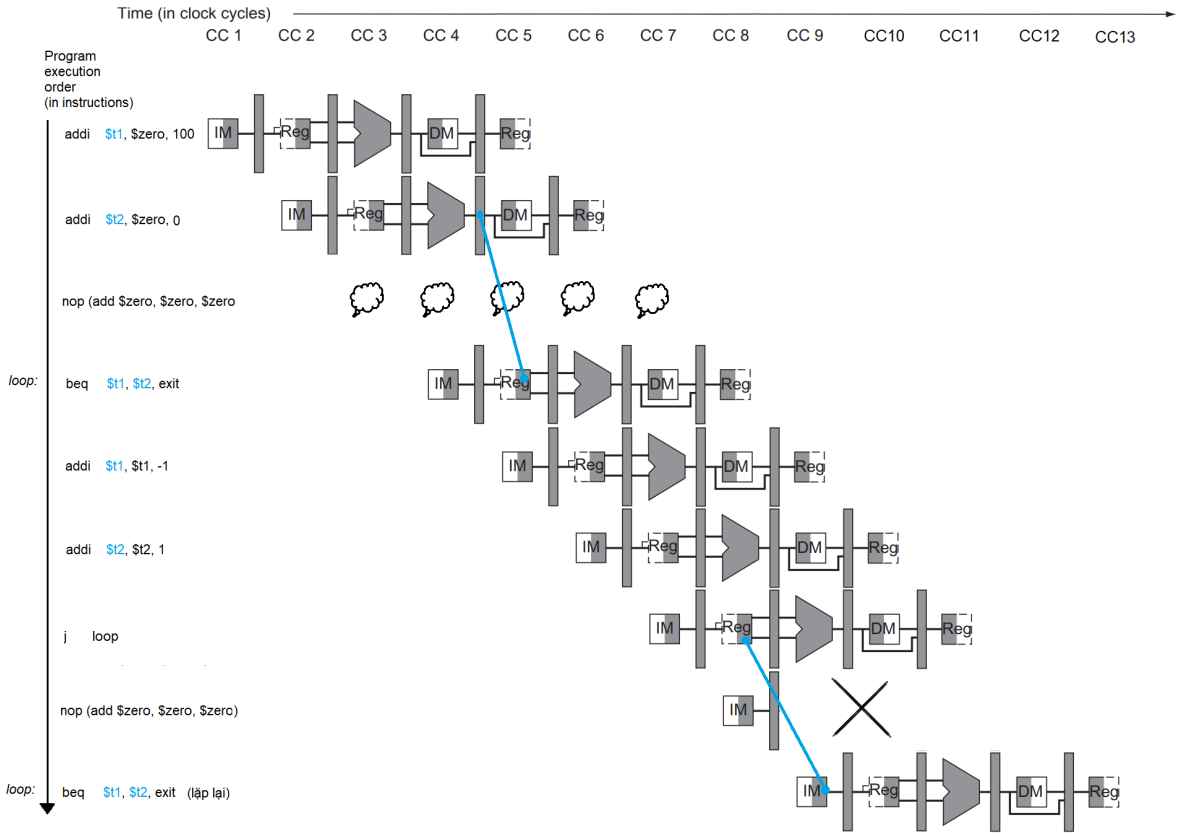
(b) Giải quyết data hazard bằng chèn stall (giải quyết bằng phần mềm), khi thực thi đoạn code trên với hệ thống pipeline thì cần chèn vào bao nhiêu stall (khựng lại) ?

- Giải quyết bằng phần mềm: nop ngay nhãn loop để giải quyết lệnh (3) phụ thuộc lệnh (5), và 2 nop để giải quyết lệnh (3) phụ thuộc lệnh (1), (2) trước khi vòng lặp bắt đầu.

```
        addi    $t1, $zero, 100
        addi    $t2, $zero, 0
        nop
loop:   nop
        beq     $t1, $t2, exit
        addi    $t1, $t1, -1
        addi    $t2, $t2, 1
        j       loop
```

- Khi thực thi đoạn code trên với hệ thống pipeline ta cần chèn tổng cộng 53 stalls, 2 stalls trước khi vòng lặp bắt đầu để giải quyết sự phụ thuộc thanh ghi $t2 và $t1 và 51 stalls tương ứng 51 vòng lặp để giải quyết lệnh (3) - beq - phụ thuộc lệnh (5) - addi $t2, Vòng lặp cuối cùng là 51 chỉ để xét điều kiện thỏa và rẽ nhánh tới label exit.

(c) Dùng cơ chế forward để giải quyết hazard (giải quyết bằng phần cứng), khi đó có bao nhiêu stall? Vẽ hình minh họa.



Theo hình thì ta thấy, ta cần thêm 1 stall để giải quyết data hazard và 1 stall để giải quyết control hazard của lệnh jump.

(d) Dùng cơ chế forward, stall, để giải quyết hazard (control và data), khi đó có bao nhiêu stall?

- Data hazard: Ta cần 1 stall để giải quyết data hazard.
- Control hazard: Ta biết bước EXE là bước hiện thực việc so sánh điều kiện, sau đó bước MEM sẽ cập nhập thanh ghi PC. Sau bước MEM ta mới biết chính xác là lệnh tiếp theo sau lệnh branch là lệnh nào. Nên ta cần phải thêm 2 stalls (khi đó bước IF nằm sau bước MEM của lệnh branch) trước đó đã có 1 stall để loại bỏ control hazard của lệnh beq. Vì jump thực hiện 2 khối là IF và ID, khi thực hiện decode lệnh jump thì cùng chu kì nó cũng đã fetch lệnh tiếp theo. Do đó để giải quyết vấn đề này, ta cần thêm một stall (ảnh ở câu c).
  ⇒ Như vậy ta cần tổng: 1 + 50 * 2 + 1 * 50 = 151 stalls.

(e) Ngoài 2 cơ chế ở trên, ta có thể giảm stall bằng cách sắp xếp lại thứ tự code (giải quyết bằng trình biên dịch compiler). Hãy sắp xếp lại code sao cho ít stall nhất.

- Với đoạn code trên, không thể giảm stall bằng cách sắp xếp lại code vì nó đã tối ưu rồi.

3 Xử lý Hazard (lệnh load)

Cho đoạn code sau:

```
addi    $t1, $zero, 100
addi    $t2, $zero, 100
add     $t3, $t1, $t2
lw      $t4, 0($a0)
lw      $t5, 4($a0)
and     $t6, $t4, $t5
sw      $t6, 8($a0)
```

(a) Xác định sự phụ thuộc dữ liệu trong đoạn chương trình trên.

– Sự phụ thuộc dữ liệu ở đây là: Read After Write – RAW Hazard.
Lệnh (3) phụ thuộc lệnh (1), (2).
Lệnh (6) phụ thuộc lệnh (4), (5).
Lệnh (7) phụ thuộc lệnh (6).

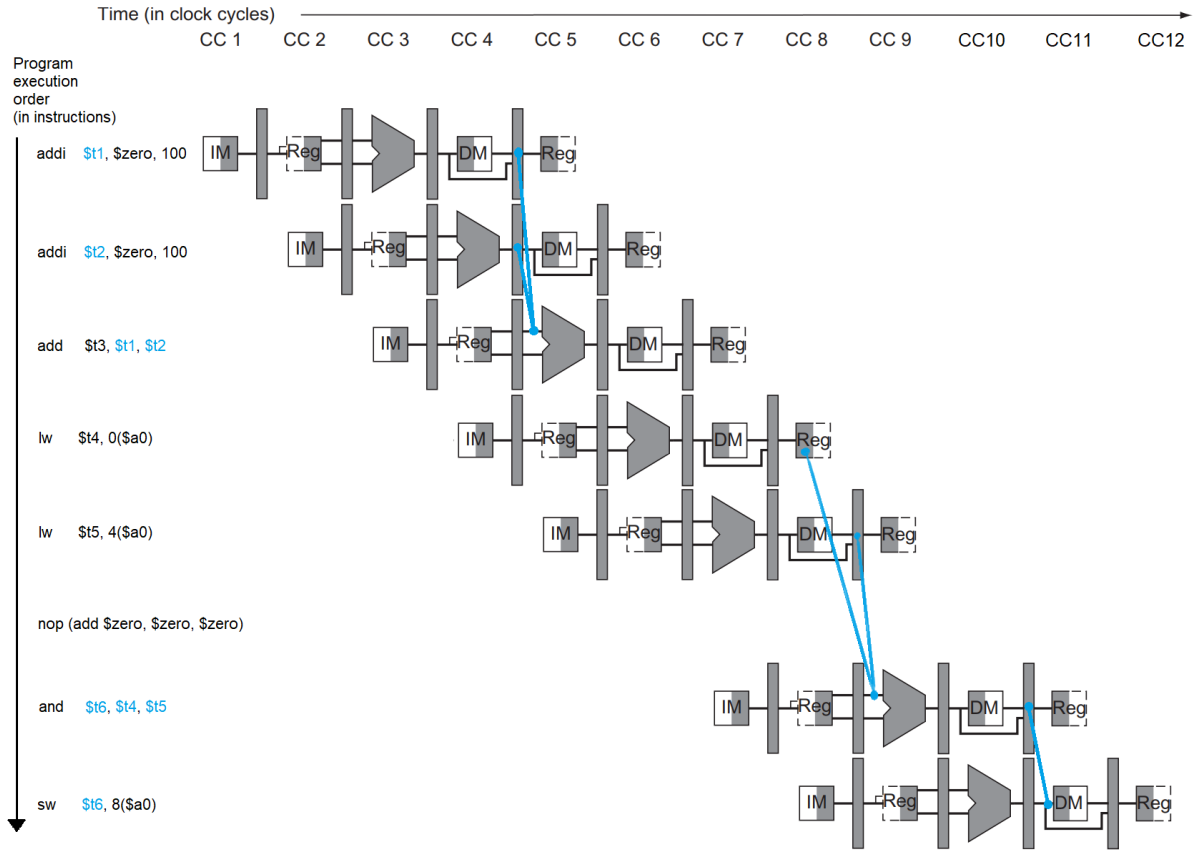
(b) Giải quyết data hazard bằng chèn stall (giải quyết bằng phần mềm), khi thực thi đoạn code trên với hệ thống pipeline thì cần chèn vào bao nhiêu stall (khựng lại) ?

– Giải quyết bằng phần mềm: 2 nop trước lệnh (3) để giải quyết lệnh (3) phụ thuộc lệnh (1), (2), và 2 nop để giải quyết lệnh (6) phụ thuộc lệnh (4), (5).

```
1  addi    $t1, $zero, 100
2  addi    $t2, $zero, 100
3  nop
4  nop
5  add     $t3, $t1, $t2
6  lw      $t4, 0($a0)
7  lw      $t5, 4($a0)
8  nop
9  nop
10 and     $t6, $t4, $t5
11 nop
12 nop
13 sw      $t6, 8($a0)
```

– Khi thực thi đoạn code trên với hệ thống pipeline ta cần chèn 6 stall.
2 stall giữa (2) và (3). 2 stall giữa (5) và (6).
2 stall giữa (6) và (7).

(c) Dùng cơ chế forward để giải quyết hazard (giải quyết bằng phần cứng), khi đó có bao nhiêu stall? Vẽ hình minh họa.



(d) Dùng cơ chế forward, stall, để giải quyết hazard (control và data), khi đó có bao nhiêu stall?

- Còn 1 stall giữa lệnh (5) và (6) để giải quyết data hazard. Vì không có lệnh rẽ nhánh nên không tồn tại control hazard.

(e) Ngoài 2 cơ chế ở trên, ta có thể giảm stall bằng cách sắp xếp lại thứ tự code (giải quyết bằng trình biên dịch compiler). Hãy sắp xếp lại code sao cho ít stall nhất.
(4) → (5) → (1) → (2) → (6) → (3) → (7).

```
lw      $t4, 0($a0)          #4
lw      $t5, 4($a0)          #5
addi    $t1, $zero, 100      #1
addi    $t2, $zero, 100      #2
and     $t6, $t4, $t5        #6
add     $t3, $t1, $t2        #3
sw      $t6, 8($a0)          #7
```

Còn lại 1 stall giữa (6) và (3) để giải quyết thanh ghi $t2.

4 Bài tập Textbook

4.1 Bài 4.16

This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT.

4.1.1 What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

- Dự đoán Always-taken:
  Branch outcomes: T, NT, T, T, NT.
  Dự đoán: T, T, T, T, T.
  ⇒ Accuracy $= \frac{3}{5} = 0.6$.
  ⇒ Độ chính xác của dự đoán Always-taken là 60%.
- Dự đoán Always-not-taken:
  Branch outcomes: T, NT, T, T, NT.
  Dự đoán: NT, NT, NT, NT, NT.
  ⇒ Accuracy $= \frac{2}{5} = 0.4$.
  ⇒ Độ chính xác của dự đoán Always-not-taken là 40%.