ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO LAB 7, 8

LỚP: L07

NHÓM: 7

Danh sách thành viên

Họ tên	Mã số sinh viên
Võ Nguyên Giáp	2110142
Trần Thành Tài	2213001
Phạm Gia Trí	2213656
Đào Anh Tuấn	2213763

Thành phố Hồ Chí Minh, Tháng 12 – 2023

MỤC LỤC

A. Bài tập/ Thực hành 7

1. Xác định clock cycle

Cho thời gian delay của các khối như Bảng 1

Bảng. 1: delay của các khối phần cứng

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

Xét đoạn chương trình như sau:

```
1        addi $t1, $zero, 100
2        addi $t2, $zero, 0
3  loop:
4        beq  $t1, $t2, exit
5        addi $t1, $t1, -1
6        addi $t2, $t2, 1
7        j loop
```

(a) Xác định thời gian một clock của hệ thống single clock, multi clock và pipeline clock.

(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).

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

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

a. Xác định chu kì CR

- Single clock: IM + REG + ALU + DM + REG = 600ns
- Multi clock: Max(IM, REG, ALU, DM) = 150ns
- Pipeline clock: Max(IM, REG, ALU, DM) = 150ns

b. Đọc hiểu đoạn code, ta xác định được số lần lặp là 50

Tính IC theo số lệnh thực thi:

IC = addi + addi + 50 (beq + addi + addi + j) + beq = 203 lệnh

- Single clock = 203 x 1 x 600 = 121800ns
- Multi clock = 150 x 4 + 150 x 4 + 50 x 150 x (3 + 4 + 4 + 2) + 150 x 3 = 99150ns
- Pipeline clock: (203 + 5 – 1) x 150 = 31050ns

c. Speedup (so với SC) = 121800/31050 = 3.92

Speedup (so với MC) = 99150/31050 = 3.19

d. Thay đổi ALU 100 -> 150

- Single clock = 650ns
- Multi clock không đổi = 150ns
- Pipeline clock không đổi = 150ns
- Speedup (so với SC) = 203x650 / 31050 = 4.25
- Speedup (so với MC) không đổi = 3.19

2. Xử lý Hazard

Dùng lại đoạn code của **Bài 1**:

Xét đoạn chương trình như sau:

```
1         addi $t1, $zero, 100
2         addi $t2, $zero, 0
3   loop:
4         beq  $t1, $t2, exit
5         addi $t1, $t1, -1
6         addi $t2, $t2, 1
7         j loop
```

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

(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) ?

(c) Dùng cơ chế forward để giải quyết data 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?

(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.

a. Sự phụ thuộc dữ liệu gồm:

- 1: lệnh addi $t1, $zero, 100 với beq $t1, $t2, exit (thanh ghi $t1),
- 2: lệnh addi $t2, $zero, 0 với beq $t1, $t2, exit (thanh ghi $t2),
- 3: lệnh addi $t2, $t2, 1 với beq $t1, $t2, exit (thanh ghi $t2).

b. Tổng stall cần chèn

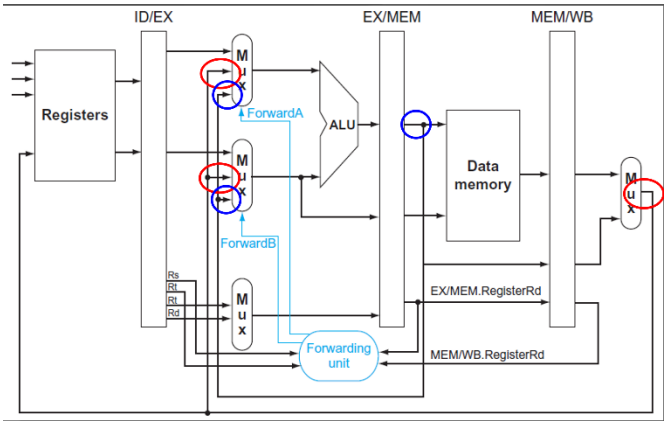
- sự phụ thuộc dữ liệu 1, 2 cần 2 stalls
- sự phụ thuộc dữ liệu 3 cần chèn 1 stall

Tổng: 2 + 50 (lặp lại 50 lần) x 1 = 52 stalls

c. Khi dùng cơ chế forward, ta không còn stall nào

- vòng màu xanh biểu hiện dùng forwarding cho sự phụ thuộc dữ liệu 1 và 3

- vòng màu đỏ biểu hiện dùng forwarding cho dữ liệu 2



d. Khi sắp xếp lại lệnh và dùng forwarding, ta có kết quả không đổi so với việc chỉ dùng forwarding là không còn stall nào

3. Xử lý Hazard (lệnh load)

Cho đoạn code sau:

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

Trả lời câu hỏi trong **Bài 2**:

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

(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) ?

(c) Dùng cơ chế forward để giải quyết data 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?

(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.

B. Bài tập/ Thực hành 8

1. (Xác định tag, index, offset)

Cho bộ nhớ chính có không gian 32bit, bộ nhớ cache có kích thước là 4MB, 1 block 256B, Đơn vị truy xuất của hệ thống là 1 byte.
Xác định tag, index, byte-offset với cấu hình cache sau:

(a) Direct mapped
(b) 4-way set associative
(c) Fully associative

Số phần tử trong 1 block = (size of block) / (size of phần tử truy xuất) = 256B / 1byte = 2^8

Số block trong cache = (size of cache) / (size of block) = 4MB / 256B = 2^2 x 2^20 / 2^8 = 2^14

Không gian địa chỉ là 32-bit.

- Direct mapped: byte-offset 8 bits, index = 14 bits, tag = 32 – 14 – 8 = 10 bits
- 4-way set asociative: 4 block tạo thành 1 set mà có 2^14 blocks nên có 2^12 sets, byte-offset 8 bits, index = 12, tag = 32 – 12 – 8 = 12 bits
- Fully associative: byte-offset 8 bits, index = 0 bit, tag = 32 – 0 – 8 = 24 bits

2. (Xác định tag, index, offset)

Cho bộ nhớ chính tổng dung lượng là 256M, bộ nhớ cache có kích thước là 256KB, 1 block 64 words, Đơn vị truy xuất của hệ thống là 2 byte. Xác định tag, index, half-word offset với cấu hình cache sau:

(a) Direct mapped

(b) 4-way set associative

(c) Fully associative

Số phần tử trong 1 block = (size of block) / (size of phần tử truy xuất) = 64 words / 2 bytes = 64x4 bytes / 2 bytes = 128 = 2^7

Số block trong cache = (size of cache) / (size of block) = 256KB / 64 words = 2^8 x 2^10 / 64x4 = 2^10

Không gian địa chỉ là 256M, do đó ta dùng thanh ghi 28 bit tính theo byte-offset.

- Direct mapped: half-word offset 7 bits, index = 10 bits, tag = 28 – 10 – 7 = 11 bits
- 4-way set asociative: 4 block tạo thành 1 set mà có 2^10 blocks nên có 2^8 sets, half-word offset 7 bits, index = 8, tag = 28 – 8 – 7 = 13 bits
- Fully associative: half-word offset 7 bits, index = 0 bit, tag = 28 – 0 – 7 = 21 bits

3. (Xác định HIT/MISS)

Cho dãy địa chỉ (words) sau:
0, 4, 1, 5, 65, 1, 67, 46, 1, 70, 2, 0
Biết hệ thống có 256Bytes caches, 4-word block, đơn vị truy xuất là byte.

Xác định số lần HIT/MISS khi chạy chương trình trên với các cấu hình caches sau:

(a) Direct mapped.

(b) 2-way set associative.

(c) Fully associative.

Chuyển đổi địa chỉ decimal => hex

Decimal	Hex	Decimal	Hex
0	0x00000000	67	0x00000043
4	0x00000004	46	0x0000002E
1	0x00000001	1	0x00000001
5	0x00000005	70	0x00000046
65	0x00000041	2	0x00000002
1	0x00000001	0	0x00000000

Với hệ thống trên, cách tính tương tự bài 2, ta được byte-offset 4 bits, index = 4 bits. Do đó, ta phân tích địa chỉ như bên dưới:

- Direct mapped

Address	Tag	Index	Offset	Miss/Hit	Explain
0x00000000	0000 0000 0000 0000 0000 0000	0000	0000	M	First access
0x00000004	0000 0000 0000 0000 0000 0000	0000	0100	H	
0x00000001	0000 0000 0000 0000 0000 0000	0000	0001	H	
0x00000005	0000 0000 0000 0000 0000 0000	0000	0101	H	
0x00000041	0000 0000 0000 0000 0000 0000	0100	0001	M	First access
0x00000001	0000 0000 0000 0000 0000 0000	0000	0001	H	
0x00000043	0000 0000 0000 0000 0000 0000	0100	0011	H	
0x0000002E	0000 0000 0000 0000 0000 0000	0010	1110	M	First access
0x00000001	0000 0000 0000 0000 0000 0000	0000	0001	H	
0x00000046	0000 0000 0000 0000 0000 0000	0100	0110	H	
0x00000002	0000 0000 0000 0000 0000 0000	0000	0010	H	
0x00000000	0000 0000 0000 0000 0000 0000	0000	0000	H	

- 2-way set associative

Address	Tag	Index	Offset	Miss/Hit	Explain
0x00000000	0000 0000 0000 0000 0000 0000 0	000	0000	M	First access
0x00000004	0000 0000 0000 0000 0000 0000 0	000	0100	H	
0x00000001	0000 0000 0000 0000 0000 0000 0	000	0001	H	
0x00000005	0000 0000 0000 0000 0000 0000 0	000	0101	H	
0x00000041	0000 0000 0000 0000 0000 0000 0	100	0001	M	First access
0x00000001	0000 0000 0000 0000 0000 0000 0	000	0001	H	
0x00000043	0000 0000 0000 0000 0000 0000 0	100	0011	H	
0x0000002E	0000 0000 0000 0000 0000 0000 0	010	1110	M	First access
0x00000001	0000 0000 0000 0000 0000 0000 0	000	0001	H	
0x00000046	0000 0000 0000 0000 0000 0000 0	100	0110	H	
0x00000002	0000 0000 0000 0000 0000 0000 0	000	0010	H	
0x00000000	0000 0000 0000 0000 0000 0000 0	000	0000	H	

- Fully associative

Address	Tag	Offset	Miss/Hit	Explain
0x00000000	0000 0000 0000 0000 0000 0000 0000	0000	M	First access
0x00000004	0000 0000 0000 0000 0000 0000 0000	0100	H	
0x00000001	0000 0000 0000 0000 0000 0000 0000	0001	H	
0x00000005	0000 0000 0000 0000 0000 0000 0000	0101	H	
0x00000041	0000 0000 0000 0000 0000 0000 0100	0001	M	First access
0x00000001	0000 0000 0000 0000 0000 0000 0000	0001	H	
0x00000043	0000 0000 0000 0000 0000 0000 0100	0011	H	
0x0000002E	0000 0000 0000 0000 0000 0000 0010	1110	M	First access
0x00000001	0000 0000 0000 0000 0000 0000 0000	0001	H	
0x00000046	0000 0000 0000 0000 0000 0000 0100	0110	H	
0x00000002	0000 0000 0000 0000 0000 0000 0000	0010	H	
0x00000000	0000 0000 0000 0000 0000 0000 0000	0000	H	

Theo cả 3 cách trên đều cho kết quả giống nhau: 3 lần miss và 9 lần hit

4. (Tính thời gian truy xuất trung bình – AMAT)

Xác định thời gian truy xuất trung bình(AMAT) ở **Bài 3**:, biết rằng Hit time = 5 cycles, thời gian truy xuất RAM là 10 ns, tần số máy tính là 2Ghz.

Ta có tần số là 2GHz => 1 chu kỳ 0.5ns

Thời gian truy xuất RAM là 10ns => Miss penalty là 10/0.5 = 20 chu kỳ

Miss rate = 3/12 = 0.25

AMAT (cycles) = 5 + 0.25 x 20 = 10 cycles

AMAT (time) = 10 x 0.5 = 5ns

5. (Tính thời gian truy xuất trung bình – AMAT)

Cho biêt hit time của L1 là 10 cycles, hit time của L2 là 15 cycle, thời gian truy xuất của RAM (main memory) là 100 cycles. L1 tỉ lệ miss là 20%, L 2 tỉ lệ miss là 10%. Xác định thời gian truy xuất vùng nhớ trung bình của hệ thống trên.

AMAT (cycles) = 10 + 0.2 x (15 + 0.1 x 100) = 15 cycles

6. (Tính CPI trung bình)

Tính CPI trung bình của hệ thống pipeline khi biết tỉ lệ miss của bộ nhớ lệnh là 5%, tỉ lệ miss của bộ nhớ dữ liệu là 10%. Biết đoạn chương trình có 1000 lệnh, trong đó có 100 lệnh là lệnh load và store. Thời gian miss penalty là 100 cycles.

- Mem stalls cho mỗi lệnh = 0.05 x 100 + 100/1000 x 0.1 x 100 = 6
- CPI Memory Stalls = CPI Perfect Cache + 6