LAB 7

Bài 1:

```
1.      addi $t1, $zero, 100    #t1 = 100

2.      addi $t2, $zero, 0      #t2 = 0

loop:

3.      beq $t1, $t2, exit      #t1 = t2 => exit

4.      addi $t1, $t1, -1       #t1 -- => t1 = 100, 99,... 50

5.      addi $t2, $t2, 1 #t2++ => t2 = 0, 1, 2,... 50

6.      j loop
```

a> Xác định thời gian một clock

- single clock: 2 + 4*50 + 1 = 203 clock cycles
- multi clock: 4*2 + (3+4*2+2)*50 + 3 = 661 clock cycles
- pipeline: 5 + 203 - 1 = 207 clock cycles

b> Xác định thời gian thực thi

- single clock : thời gian 1 chu kỳ = 150 + 100 + 100 + 150 = 600 (ns) => Ts = 203*500 = 121800 ns
- 1 chu kỳ = thời gian thực thi lớn nhất của 1 khối = 150 ns => dựa vào câu a ta cần : 661*150 = 99150 ns
- pipeline : xét thời gian 1 chu kỳ = thời gian lớn nhất thực thi trong 1 khối = 150 ns => dựa vào câu a ta có 207*150 = 31050 ns

c> Tính speed up của hệ thống pipeline với hệ thống multi cycle và với single cycle

- so với multi : 3.1932
- so với single clock : 3.9227

d> Khi delay ALU thay đổi từ 100 –> 150

- single clock : thời gian 1 chu kỳ = 150+100+150+150 = 650 ns => Ts = 203*650 = 131950 ns
- multi clock: thời gian thực thi lớn nhất của 1 khối không đổi => vẫn là 99150 ns
- pipeline : thời gian thực thi tối đa của 1 khối không đổi => vẫn là 31050 ns
- so với multi : 3.1932
- so với single clock : 4.2496

Bài 2:

a> Xác định sự phụ thuộc dữ liệu trong đoạn chương trình trên:

- lệnh 3 cần dữ liệu được ghi vào $t1 và $t2 của lệnh 1, 2 và lệnh 4, 5

b> chèn 2 stall giữa (2) và (3), 1 stall sau lệnh 6 (vì sau J là beq)

loop 50 lần nên cần tổng cộng 2 + 50 = 52 stalls

c> Không cần chèn stall

d> Không có so sánh sớm:

3 stall giữa (3) và (4) ; 1 stall sau (6)

tổng : 4*50 + 3 = 203 stalls

Có so sánh sớm

1 stall giữa (2) và (3); 1 stall giữa (3) và (4); 1 stall sau (6)

tổng : 1 + 2*50 + 1 = 102 stalls

e>

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

Không thể sắp xếp để tối ưu hơn được nữa.

Bài 3:

a> Sự phụ thuộc dữ liệu:

- Dòng 3 phụ thuộc vào kết quả của dòng 1 và dòng 2.

- Dòng 4 và dòng 5 phụ thuộc vào giá trị tại địa chỉ bộ nhớ được lấy từ thanh ghi $a0.

- Dòng 6 phụ thuộc vào kết quả của dòng 4 và dòng 5.

- Dòng 7 phụ thuộc vào kết quả của dòng 6.

(b) Giải quyết data hazard bằng chèn stall:

- Dòng 3 phải chờ cho đến khi dòng 1(cần 1 stall) và dòng 2(cần 2 stall) kết thúc để lấy giá trị của $t1 và $t2 → cần 2 stall

- Dòng 6 phải chờ cho đến khi dòng 4((cần 1 stall)) và dòng 5(cần 2 stall) kết thúc để lấy giá trị của $t4 và $t5 → cần 2 stall

- Dòng 7 phụ thuộc vào kết quả của dòng 6 → cần 2 stall

→ Vậy cần chèn: 6 stall

c> Giải quyết data hazard bằng cơ chế forward:

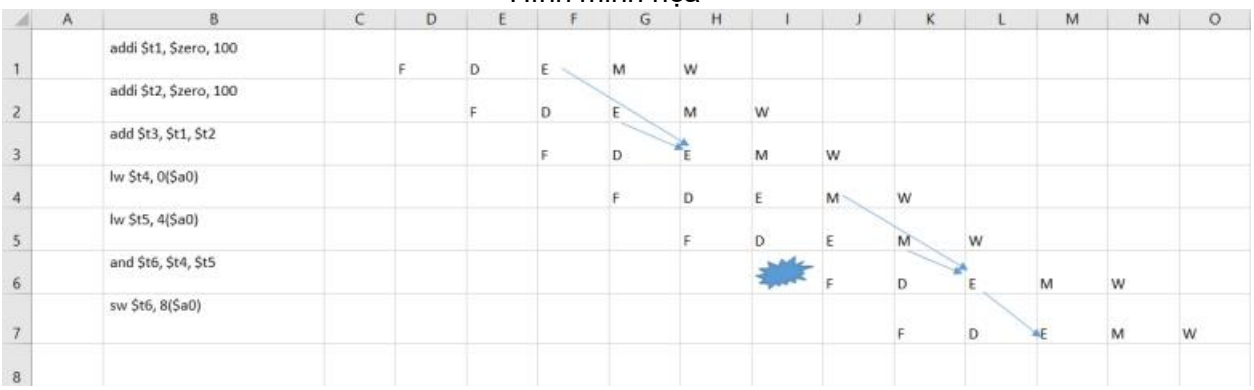
- Dòng 3: Sử dụng forward từ dòng 1 và dòng 2→ không chèn stall

(Như đã biết thì cơ chế xử lý data hazard bằng forwarding có thể xử lí cho mọi loại lệnh mà không cần stall TRỪ lệnh lw(áp dụng cho lệnh phụ thuộc ngay sau lệnh lw), để giải quyết cần tiêu tốn 1 stall). Từ đó ta dễ dàng thấy rằng:

- Dòng 6: Sử dụng forward từ dòng 4(ko chèn stall vì hai lệnh đã cách nhau bởi lệnh 5) và dòng 5(chèn 1 stall) → cần 1 stall.

→ Vậy cần chèn: 6 stall

Hình minh họa



Mỗi lệnh 5 chu kỳ tương ứng với 5 bước là: Fetch(F)- Decode(D)- Execute(E)- Memmory(M)- Writeback(W)

d> Giải quyết cả data hazard và control hazard:

- Rõ ràng là đoạn code hợp ngữ trên không có bất kì lệnh rẽ nhánh nào nên chúng ta ko cần quan tâm đến control hazard. Nên viêcj còn lại là giải quyết data hazard(xem câu a>, b>).

e> Sắp xếp lại thứ tự code:

1 addi $t1, $zero, 100

2 lw $t4, 0($a0)

3 lw $t5, 4($a0)

4 addi $t2, $zero, 100

5 and $t6, $t4, $t5

6 add $t3, $t1, $t2

7 sw $t6, 8($a0)

Khi sắp xếp lại thứ tự code như trên, không cần chèn stall nếu sử dụng cơ chế forward.