컴퓨터구조 4장 연습문제 풀이.

4.4.1

메모리에서 데이터를 추출하여 레지스터에 저장하는 명령어들을 사용하지 못함으로 load명령어를 사용하지 못한다.

4.4.2

피연산자에 레지스터가 아닌 숫자를 사용할수없음으로 load, store 같은 명령어를 사용하지 못한다.

4.7.1~6

RegisterRead는 5가지 명령어 모두 사용한다!

Register Setup은 레지스터에 데이터 입력이 없는 sw 명령어를 제외하고 모두 사용한다.!!

4.7.1

Reg.Read + I-Mem+ Reg.file + Mux+ ALU + Mux+ Reg.setup

30+250+150+25+200+25+20 = 700ps

4.7.2

Reg.Read + I-Mem+ Reg.file + Mux+ ALU + D-Mem + Mux + Reg.Setup

30+250+150+25+200+250+25+20 = 950ps

4.7.3

Reg.Read + I-Mem+ Reg.file + ALU + Mux + D-Mem

30+250+150+200+25+250 = 905ps

4.7.4

Reg.Read + I-Mem+ Reg.file + Mux + ALU + SingleGate + Mux + Reg.Setup

30+250+150+25+200+5+25+20 = 705ps

4.7.5

Reg.Read + I-Mem+ Reg.file + Mux + ALU + Mux + Reg.Setup

30+250+150+25+200+25+20 = 700ps

4.7.6

제일 긴시간인 950ps가 된다.

4.19

Addi x11 x12 5 에서 rd값이 x11이기 때문에 다음 명령어인

Add x13 x11 x12

Addi x14 x11 15 에서는 데이터 해저드가 발생 한다!!

하지만 위 둘 명령어가 끝난뒤엔 x11의 WB이 완료 된 상태이므로

X11엔 27이 들어있고 x15엔 54가 대입된다.

4.23.1~3

4.23.1

가장 주기가 긴 명령어는 load 이다. 여기서 ALU계산 기능이 빠진다면 빠진만큼 빨라지거나 LOAD 다음으로 주기가 긴 명령어의 사이클 최소 주기가 될것이다.

4.23.2

사이클의 주기가 빨라지므로 속도가 빨라지는걸 확인할수 있을것이다.

4.23.3

주소를 레지스터에 저장한다곤 하지만 offset을 계산하기 위해선 lw 또는 sw 명령어 앞쪽에

addi 또는 add 명령어가 추가되어야 하므로 총 명령어가 증가할 것이다.

4.27.1~7

1. Add x15 x12 x11
2. Lw x13, 8(x15) //1.에서 계산된 x15를 사용하려함!! 해저드 발생!!
3. Lw x12, 0(x2)
4. Or x13 x15 x13 //2. 에서 계산된 x13를 사용하려함!! 해저드 발생!!
5. Sw x13 0(x15) //4. 에서 계산된 x13를 사용하려함!! 해저드 발생!!

4.27.1

Add x15 x12 x11

NOP //x15의 값을 얻기 위해 2주기 버리기

NOP

Lw x13, 8(x15)

Lw x12, 0(x2)

NOP //x13의 값을 얻기 위해 1주기 버리기

Or x13 x15 x13

NOP //x13의 값을 얻기 위해 2주기 버리기

NOP

Sw x13 0(x15)

4.27.2

무조건 띄어놔야하는 부분을 buble로 표시한다면

Add x15 x12 x11

buble

buble

Lw x13, 8(x15)

Buble

buble

Or x13 x15 x13

buble

buble

Sw x13 0(x15)

남은 명령어는 어디에 들어가도 상관없는

Lw x12, 0(x2)

명령어만 남았다. 결론은 nop의 개수는 최소 5개로 고정이다.

4.27.3

코드가 문제없이 실행된다!!

Add x15 x12 x11

Lw x13, 8(x15) //x15가 전방 전달된다

Lw x12, 0(x2)

Or x13 x15 x13 //x13가 전방전달된다.

Sw x13 0(x15) //x13가 전방전달된다.

4.27.4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1(cylcle) | 2 | 3 | 4 | 5 | 6 | 7 |
| IM | Reg | Ex | DM | WB |  |  |
|  | IM | Reg | Ex | DM | WB |  |
|  |  | IM | Reg | Ex | DM | WB |
|  |  |  | IM | Reg | Ex | DM |
|  |  |  |  | IM | Reg | Ex |

1,2에서는 계산이 없기 때문에 신호가 없다.

3. 레지스터를 직접읽어 사용하기 때문에 0 이 출력신호이다.

4. 바로 앞 명령어의 계산값을 피연산자 1 로 사용하므로 FwardingA = 2 이 출력되고 나머진 0이다.

5. 레지스터를 직접 읽어 사용하기 때문에 0이 출력신호이다.

1. 두 단계 앞 명령어의 계산값을 피연산자 2로 사용하므로 FowardingB = 1이 출력되고 나머진 0이다.
2. 바로 앞 명령어의 계산값을 피연산자2로 사용하므로 FowardingB = 2 이 출력되고 나머진 0이다.

4.27.5

이전 명령어의 rd값이 다음명령어의 소스(r1,r2)값으로 사용된다면 유닛에서는 검출 출력값이 출력되어야하고 이런 신호들은 해저드들이 발생하지 않게 하기 위해 필요하다.

4.27.6

해저드 검출 유닛의 3가지 신호!

1. PC쓰는 것 제어
2. IF/ID 레지스터 쓰는 것 제어
3. 멀티플렉서가 실제 제어값과 0 중 하나를 선택하도록

|  |  |  |  |
| --- | --- | --- | --- |
|  | PC쓰는 것 제어 | IF/ID 레지스터 제어 | 멀티플렉서 |
| 1(cylcle) | 1 | 1 | 0 |
| 2 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 |
| 5 | 0 | 0 | 1 |