Midterm Examination #2 - Solution

시험시간: 11월 16일(화) 오후 7:00-9:00

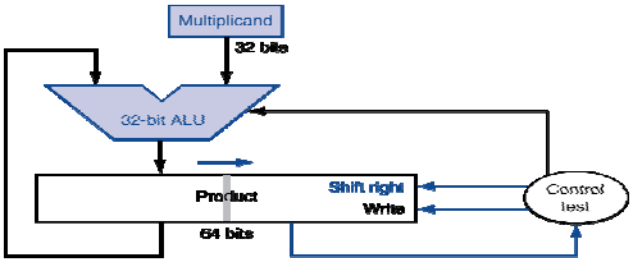
주의사항:

- 답안지에 이름, 학번을 반드시 기입, 답안지는 앞면, 뒷면을 모두 사용할 것
- 시험지 앞, 뒷면에 문제가 모두 인쇄되었는지 확인할 것
- 답이 도출되는 과정을 명확하고 알아보기 쉽게 적을 것
- 문제에 애매한 부분이 있다고 판단되는 경우 합리적인 가정을 한 후 문제를 풀어도 무방함
- 문제들마다 난이도가 다르므로 먼저 전체를 한번 훑어 본 후 시간을 잘 조절하여 문제를 풀 것

---

1. 10진수 A=69, B=90가 주어져 있을 때,
   A. [5 points] A와 B가 unsigned 8-bit 정수라고 가정할 때 A-B를 계산하고, overflow/underflow인지 나타내시오.
   Underflow (-21)
   B. [5 points] A와 B가 sign-magnitude 형태의 signed 8-bit 정수라고 가정할 때 A+B를 계산하고, overflow/underflow인지 나타내시오.
   Overflow (result = 159, which does not fit into an 8-bit SM format)
   C. [5 points] A와 B가 sign-magnitude 형태의 signed 8-bit 정수라고 가정할 때 A-B를 계산하고, overflow/underflow인지 나타내시오.
   Neither (–21)

2. [10 points] 아래와 같은 Multiplier를 이용하여 4-bit으로 표현된 두 정수의 곱셈, 2 x 3, 또는 0010_2 x 0011_2를 계산하기 위해 필요한 각 단계들에 해당되는 값들을 아래의 테이블에 채우시오. Step 항목에는 아래의 동작 중 해당하는 것을 기입하시오
   SL (shift left multiplicand)
   SR (shift right multiplier)
   ADD (product = product + multiplicand)
   NOP (No operation)



Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1-1	ADD	0011	0000 0010	0000 0010
1-2				
1-3				
2-1				
2-2				
2-3				

3-1				
3-2				
3-3				
4-1				
4-2				
4-3				

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

3. NVIDIA에서 사용하는 floating point 형식은 IEEE754 floating format과 유사하지만 전체 길이는 16bit로 구성되어 있다. 최상위 1 bit는 부호를 나타내고, 지수부분(exponent)은 5 bit를 사용하며 유효숫자(fraction)의 표시를 위해 10 bit를 사용한다. 또한 IEEE754 format과 마찬가지로 normalized된 숫자를 가정한다. 또한 지수부분을 표시할 때 "bias"된 표시 방법을 사용하는데, 32-bit IEEE754 format에서 지수를 나타내기 위해 normalized된 숫자의 지수에 127을 더했던 것 대신에 NIVIDA 형식에서는 16을 더한다. 다음의 숫자를 NVIDIA floating point 형식으로 나타내시오. 만약 표시할 수 없다면 그 이유는?

   A. [7 points] 5.00736125×10^5

$5.00736125 \times 10^5 = 500736.125 \times 10^0 = 0x7A400.2 \times 16^0 =$

$11110100100000000000.0010_2 \times 2^0$

move the binary point 18 to the left = $1.11101001000000000000010 \times 2^{18}$

exponent = +18, fraction = +11101001000000000000010

answer: Cannot represent +18+16=+34

   B. [7 points] $-0.00000110111001 \times 2^0$

$-2.691650390625 \times 10^{-2} = -.02691650390625 \times 10^0 = -.00000110111001 \times 2^0$

move the binary point 6 to the right = $-1.10111001 \times 2^{-6}$

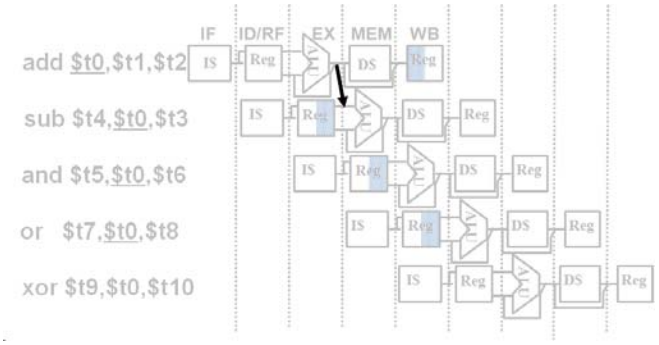
exponent = $-6 = -6 + 16 = 10$, fraction = $-.10111001$

answer: $1\_01010\_1011100100_2$

4. 파이프라인 시스템의 각 stage들에서의 수행시간이 다음과 같이 주어져 있다.
   Stage 1 = 40 ns, Stage 2 = 10 ns, Stage 3 = 30 ns, Stage 4 = 5 ns.
   A. [7 points] 파이프라인 시스템의 clock 주기는 얼마인가?
      40 ns
   B. [7 points] 파이프라인 기법을 사용하지 않고 하나의 명령어를 수행하는데 걸리는 시간은 얼마인가?
      40+10+30+5 = 85 ns
   C. [7 points] 100개의 명령어를 수행했을 때 파이프라인 기법으로 throughput(단위 시간당 명령어 처리 개수)이 얼마나 향상되는가?
      Non-pipeline: 85xN (N: 명령어 개수), Pipeline: 40x4 + (N-1)x40
      그러므로 (85x100)/(160+99x40) = 8500/4120 = 2.06배의 성능향상이 있다.

5. 다음의 문제들에 대해 답하시오.
   A. [10 points] 아래에 주어진 MIPS instruction들은 레지스터 r0와 관련해 data hazard가 존재한다. 어떤 data hazard인지 설명하고 이를 해결하기 위한 forwarding path(들)을 만들어 보시오. 예를 들어 첫 번째와 두 번째 instruction들 사이의 hazard를 해결하기 위한 forwarding path는 **add.mem→sub.ex**과 같이 나타낸다.

```
add    r0, r1, r2    # r0←r1 AND r2
sub    r4, r0, r3    # r4←r0 - r3
and    r5, r0, r6    # r5←r0 AND r6
or     r7, r0, r8    # r7←r0 AND r8
xor    r9, r0, r10   # r9←r0 XOR r10
```



   레지스터들 사이의 forwarding은 add.wb→and.ex가 정답이지만 pipeline 단계를 기준으로 forwarding path를 생각하면 add.mem→and.ex로 가능하므로 모두 정답으로 인정함

   B. [10 points] 아래의 instruction들에서는 또 다른 종류의 data hazard가 존재한다.

```
lw      r0, 0(r1)      # M[0+r1] ← r0
sub     r3, r0, r2     # r3 ← r0 - r2
and     r5, r0, r4     # r5 ← r0 AND r4
or      r7, r0, r6     # r7 ← r0 OR r6
```

   위의 instruction들 사이에 존재하는 data hazard가 왜 A번 문제의 forwarding 방법으로 해결될 수 없는지 설명하고 이를 해결하기 위한 방법을 간략히 기술하시오.

   Load-use hazard

6. 아래와 같은 MIPS 코드가 주어져 있다.

```
lw     $1, 40($6)
beq    $2, $0, Label      ; Assume $2 == $0
sw     $6, 50($2)
```

```
Label:      add    $2, $3, $4
            sw     $2, 50($4)
```

A. [10 points] 모든 branch들은 decode stage에서 완벽하게 예측된다고 가정한다. (즉 control hazard는 존재하지 않는다). Instruction와 data는 2개의 메모리에 분리되어 존재하고 (즉 structural hazard는 존재하지 않는다) 메모리에서 32-bit word를 읽는데 1 사이클이 걸린다. 그러나 data hazard를 해결하기 위한 forwarding 기법은 구현되어 있지 않다. 이 때 위의 MIPS 명령어들이 한번씩 실행하기 위해서는 몇 사이클이 필요한가? 아래의 테이블에 수행되는 명령어 별로 5-stage pipeline (F, D, X, M, W)들이 진행되는 과정을 채우시오.

	Cycle																
instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`lw $1, 40($6)`	F	D	X	M	W												
`beq $2, $0, Label`		F	D	X	M	W											
`add $2, $3, $4`			F	D	X	M	W										
`sw $2, 50($4)`				F	D	D	D	X	M	W							

B. [10 points] 위의 문제에서 다음과 같이 가정을 바꾼다. 먼저 instruction과 data는 하나의 메모리에 같이 존재한다 (즉 structural hazard가 존재함). 만약 어떤 사이클에서 instruction fetch를 수행하는 명령어와 data memory access를 수행하는 명령어가 존재할 때는 instruction fetch를 수행하는 명령어에 메모리 접근에 대한 우선권을 주기로 한다. 그리고 data hazard를 해결하기 위한 forwarding logic이 구현되어 있다. 이 때 위의 MIPS 명령어들이 한번씩 실행하기 위해서는 몇 사이클이 필요한가?

	Cycle																
instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`lw $1, 40($6)`	F	D	X	M	M	W											
`beq $2, $0, Label`		F	D	X	X	M	W										
`add $2, $3, $4`			F	D	D	X	M	W									
`sw $2, 50($4)`				F	F	D	X	M	W								