SSN:________________________

CS 473
Midterm Exam
Solutions

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:20 to finish the exam. The questions are equally weighted. Some of the questions (2, 3 and 4) might be more easily answered on the exam paper; you may do so and turn in those pages (but remember to put your SSN on them if you do!).

1. Floating Point

   (a) Convert the following decimal number to IEEE floating point format:

   `13.375`

   Your final answer should be an eight digit hexadecimal number.

Integer Part:

Old	*New*	*Bit*
13	6	1
6	3	0
3	1	1
1	0	1

So the integer part is `1101`.

Fraction Part:

Old	*Bit*	*New*
.375	0	.75
.75	1	.5
.5	1	.0

So the fraction part is `.011`

Putting them together, in binary the number is `1101.011` = `1.101011`$\times 2^3$

The sign is `0` *(positive)*

The exponent field is `127 + 3 = 130`$_{10}$ = `10000010`$_2$

The fraction field is `101011` *(remember to take away the phantom bit!*

Putting it all together, we get `0 10000010 101011` *which translates into hexadecimal as*

```
 0100  0001  0101  0110  0000  0000  0000  0000
   4     1     5     6     0     0     0     0
```

Answer: ***`41560000`***

Points	*Error*
-5	*Only took to binary; no IEEE format*
-3	*Added int part to 127 (???)*
-1	*Read integer part top-to-bottom*
-2	*Forgot phantom bit*

SSN:________________________

(b) Use the floating point multiplication algorithm to multiply the following two IEEE floating point numbers :

`0x3fc00000`$\times$`0xc0980000`

Your final answer should be an eight digit hexadecimal number.

First, we break the two numbers apart into fields:

```
  3     f     c     0     0     0     0     0
0011  1111  1100  0000  0000  0000  0000  0000
```

Sign	0
Exponent	01111111
Significand	1.1

```
  c     0     9     8     0     0     0     0
1100  0000  1001  1000  0000  0000  0000  0000
```

Sign	1
Exponent	10000001
Significand	1.0011

Then we multiply:

The sign is $1 \oplus 0 = 1$

The exponent is 10000001 + 01111111 - 01111111 = 10000001

The significand is

```
    1.0011
 ×  1.1
 ---------
    1.0011
    0.10011
 ---------
    1.11001
```

It's already normalized, so we can just reassemble it:

```
1100  0000  1110  0100  0000  0000  0000  0000
  c     0     e     4     0     0     0     0
```

Answer: ***c0e40000***

Points	*Error*
-2	*Forgot phantom bit*
-5	*No IEEE result*
-1	*Converted exp to decimal, got it wrong.*
-3	*Multiplied in decimal*

(c) Convert the following IEEE floating pont number into "human-readable" decimal.

`0xc0980000`

Your final answer should be in the same form as the "`13.375`" in part 1a.

First, convert the number into binary and convert it into fields:

```
  c     0     9     8     0     0     0     0
1100  0000  1001  1000  0000  0000  0000  0000
```

Sign	1
Exponent	10000001
Significand	**1**.0011

The value of the exponent is 1000001 - 01111111 = $10_2 = 2_{10}$

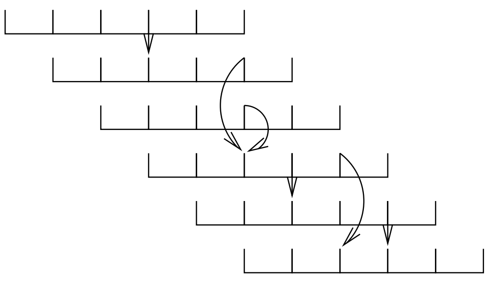
So, in binary, the value of the number is $-1.0011 \times 2^2 = -100.11$

We convert this to decimal and get ***-4.75***

Points	*Error*
-2	*Forgot to denormalize*
-5	*Didn't properly convert to decimal*
-2	*Converted first, then denormalized*
-1	*Forgot negative*
-3	*Didn't convert fraction part*

2. Suppose the following sequence of instructions is to be executed on a MIPS processor.

```
lui $2, 100
lw  $1, 100($2)
add $2, $3, $4
sub $5, $1, $2
lw  $7, 100($5)
sw  $7, 200($5)
```



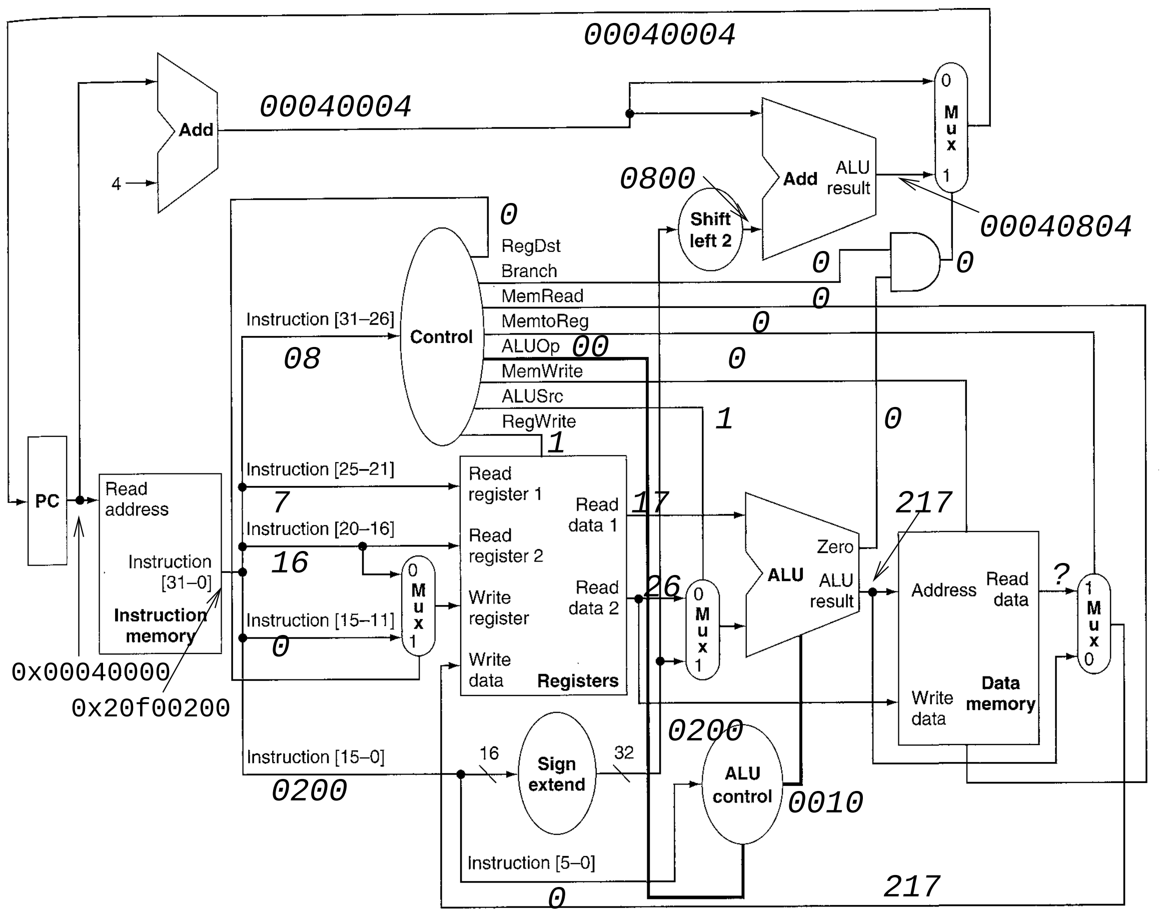
(a) Use circles and arrows to show all the dependences between these instructions.

(b) Draw a timing chart showing the execution. Assume all possible forwarding and all necessary stalls occur as needed. Be sure to use arrows showing forwarding in your answer.

Points	*Error*
-2	*Each missed dependence*
-2	*Each missed forward*
-2	*Each extra stall*

3. In the following figure (reproduced from page 307 of the text and modified by me for this question) address `0x00040000` has just been read from the instruction memory, and the instruction `0x20f00200` has been returned . Show the value that appears on every line in the diagram as the instruction is executed. Use our standard assumptions for register contents, ie that every register `$i` except `$0` returns a value of `i+10` when it is read for the first time.

The instruction is `20f00200`*; that turns out to be* `001000 00111 10000 0000001000000000`*, or* `addi $16, $7, 0x0200`.



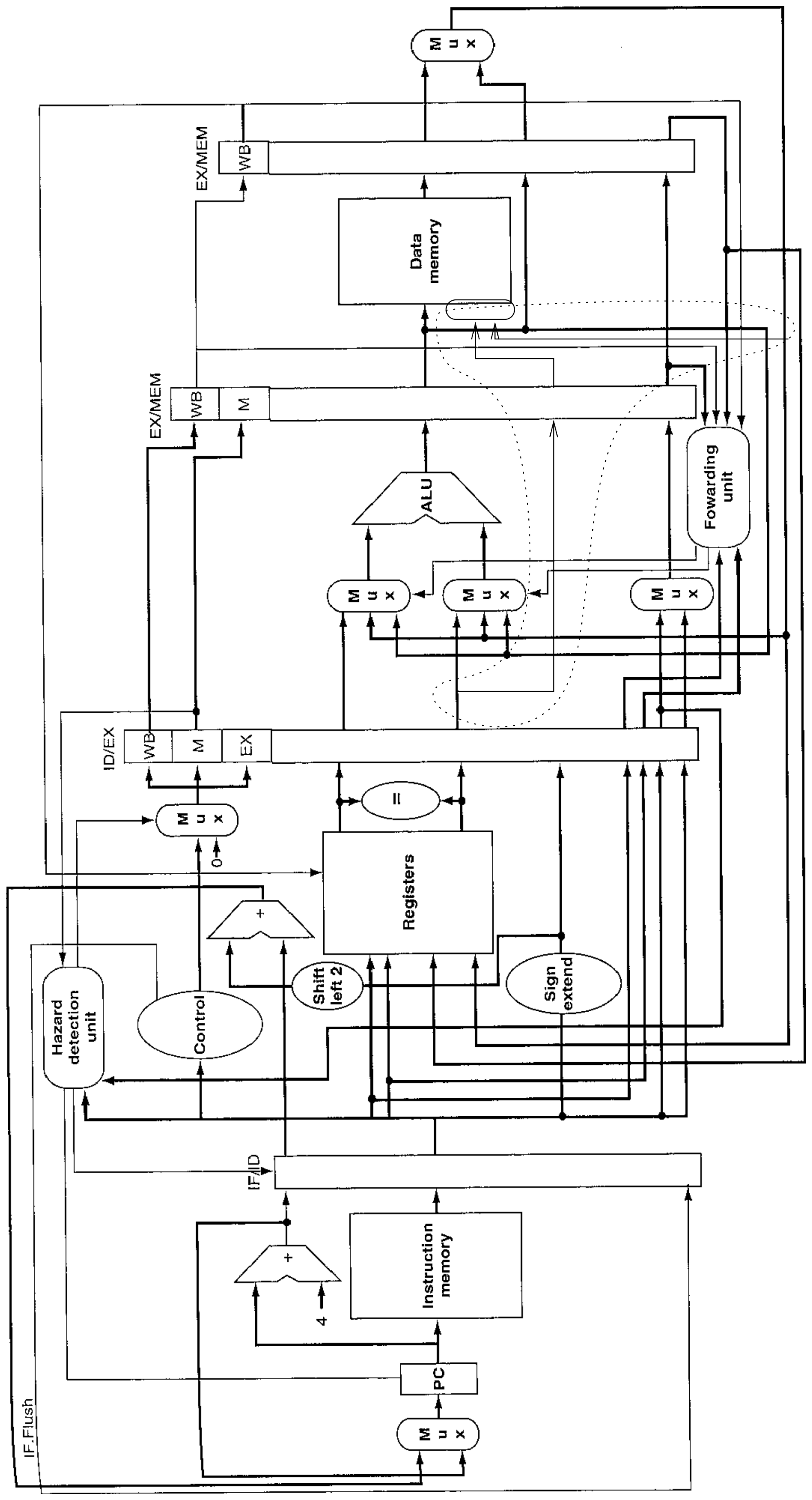
SSN:________________________

Points	*Error*
-1	*Each missed value*
5	*Just decode instruction*

4. We've made the comment in class that if we had a pair of instructions like this:

```
lw $2, 100($6)
sw $2, 400($5)
```

(in which the sw writes the same register out to memory that the lw has fetched) can execute without stalling by forwarding the register contents from the first instruction to the first. On figure on the next page (reproduced from page 427 of the book), modify the datapaths so this forwarding can actually take place.


IF.Flush
Mux
PC
4
+
Instruction memory
IF/ID
Hazard detection unit
Control
Mux
0
Shift left 2
+
Registers
=
Sign extend
ID/EX
WB
M
EX
Mux
Mux
ALU
Mux
Fowarding unit
EX/MEM
WB
M
Data memory
EX/MEM
WB
Mux