EE380 Spring 2018 Exam 1

Your name is: `Sample solution`

Your account name (as used for assignments) is:

The two questions you skipped are:

There are 15 questions. Each of the first 14 questions is worth 8 points; you must answer any 12 and indicate above which 2 you skipped (no extra credit). You also must answer question 15, which is worth just 4 points. For each question with boxes, ☐, in front of each potential answer, read the question carefully to see if you should check a single answer or all answers that apply. The short answer questions should get **short** answers; excessively long answers may be considered incorrect. The test is without calculators, closed book, closed notes, closed minds (no reading your neighbor's mind nor test paper. You may detach and keep the reference sheet at the end of this exam.

1. What is the range of values you can represent in a 4-bit two's complement integer? Give the minimum and maximum values and their bit patterns.

`1000 to 0111, decimal -8 to +7`

2. For this question, *mark all that apply*. Which of the following describe an instruction in the MIPS ISA as discussed in this course?

☐ Store a 16-bit immediate value into memory

■ Call a function placing the return address in a register

■ Add two values from registers, placing the result in a third register

■ Set a register to 1 if another register's value is less a third register's value, 0 if not

3. Using *only* MIPS instructions listed on the last page of this exam, write assembly code to make register `$t0` hold the absolute value of `$t0`'s original two's complement value. You may use other t registers as temporaries. Comment any non-obvious things you do.

```
abst0:  slt $t1,%t0,$0
        beq $t1,$0,okasis
        sub $t0,$0,$t0
okasis:

This negates if $t0 is less than 0;
this is same as negate if bit 0x80000000 was 1.
```

4. Using *only* MIPS instructions listed on the last page of this exam, write assembly code to implement the following C code. Variables should be placed in the registers with the same name, i.e., t0 is kept in the register named **$t0**; other t registers may be used as temporaries. Comment any non-obvious things you do.

```
while (t0 != t1) { t0 = t0 + 5; }
```

```
while:  beq $t0,$t1,elihw
        addi $t0,$t0,5
        j while
elihw:
```

Note that add can overflow, so addu might be better....

5. Using IEEE 754 single-precision (32-bit) floating point, what is **1073741824 + 1**? Hint: 2^{30} is 1073741824. Explain.

1073741824 + 1 is 1073741824.

The problem is that 1 is 2^0, so the exponents differ by 30... and the 32-bit float mantissa doesn't have that many bits, so the denormalize step makes 1 into 0.

6. For this question, *mark all that apply*. Consider the following four ways to build a 32-bit binary adder. Which of these methods would be able to make good use of very wide (many input) AND and OR gates?

- ☐ Ripple carry
- ☐ Carry select
- ■ Carry lookahead
- ☐ Speculative carry

7. In linguistics, **n-gram** analysis is commonly used to quantify similarity between documnts. Typically, 3-grams are used; for example, the word **`test`** contains the 3-grams **`_te`**, **`tes`**, **`est`**, and **`st_`**. For analysis, the probability of each 3-gram is computed; if document A contains 1000 3-grams and **`tes`** appears 23 times, we would say **`tes`** has a probability of 2.3% in A; call that A(tes). The similarity of two documents, A to B, is computed as the sum of the squares of probability differences for all 3-grams occuring in both documents. Thus, if B(tes) was 5%, for **`tes`** we would add (0.023-0.05)*(0.023-0.05) to the sum — which we'll accumulate in an IEEE 754 **`float`**. The sum is computed in the order in which the 3-grams first appeared in the document specified first, and only 3-grams that appear in both documents are used. Given this similarity computation, would you expect the similarity of document C to D to be the same as the similarity of D to C? Explain.

Summing in different orders can produce different sums if any pair of values being summed could differ in magnitude by more than the mantissa can hold; there is nothing here that would limit the range of values. Thus, there could be differences in the similarity of C to D and D to C.

8. For this question, *mark all that apply*. Consider the figure shown at the end of this exam. Which of the following four control lines *must be* set to **logic 1** to ensure correct operation while executing an **`xor`** instruction?

- ■ RegDst
- ☐ ALUSrc
- ■ RegWrite
- ☐ MemToReg

9. For this question, *mark all that apply*. Consider the figure shown at the end of this exam (the same one used in the previous question). For this single-cycle implementation, suppose we were to probe a few of the control signals and found that for a particular clock cycle, **ALUSrc=0**, **MemtoReg=0**, and **MemWrite=1**. Which of the following MIPS instructions might have been executing?

☐ **`lw`**

☐ **`beq`**

☐ **`and`**

☐ **`xori`**

10. For this question, *mark all that apply*. Consider the figure shown at the end of this exam (the same one in the previous two questions). For this single-cycle implementation, which of the following control signals *cannot* have the same value for all three of the **`ori`**, **`lw`**, and **`sw`** instructions?

☐ ALUsrc

☐ RegDst

■ RegWrite

■ MemToReg

11. What does the following C code do if `f` is an IEEE754 32-bit float and `i` is a 32-bit two's complement integer? Why?

```
main() {
  float f; int i = 0;
  for (f=0; f<1000000; ++f) ++i;
  if (f == i) printf("yes!\n"); else printf("no!\n");
}
```

This loop is using a 32-bit float index variable, so it is possible we'd have it never terminate -- BUT that does NOT happen in this case. 1000000 is slightly less than 2^20 and 1 (for ++f) is 2^0, so the values fit in the mantissa and the loop will print "yes."

12. For this question, *mark all that apply*. Remember the shift-and-add multiplier and Booth's algorithm? Using the constants listed below to control the multiplier, which of the following multiplies would take fewer clock cycles using Booth's algorithm than using a multiplier that just skips 0 bits?

- [✓] `x*7`
- [ ] `x*8`
- [✓] `x*56`
- [ ] `x*72`

13. What is the effect of executing the MIPS subroutine `f`?

```
    .data
x:  .word   22
    .word   380
    .word   13
    .word   42
    .word   601
    .word   1
    .word   2
    .word   3
    .word   0
    .text
f:  la  $t0, x
    ori $t1, $0, 480
    sw  $t1, 8($t0)
    jr  $ra
```

It is storing 480 into the location two words (8 bytes) past x. That means it overwrites 13 with 480.

14. Think about how we implemented a (leaf) subroutine call using the MIPS instruction set. Give the MIPS instruction that would be used to call the subroutine at the address labeled `sub`. Also give the MIPS instruction that would be used to return from that subroutine.

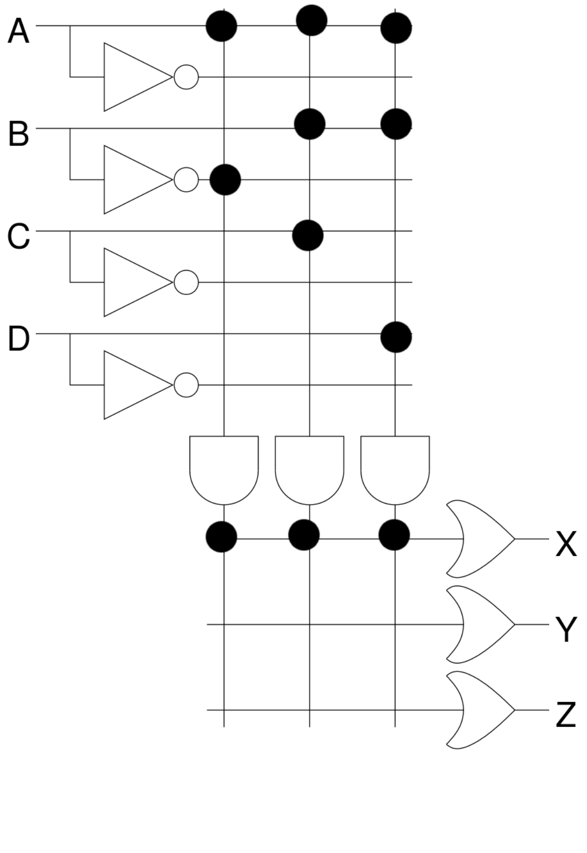
You would call the subroutine sub using:

```
jal sub
```

which would put the return address in $ra ($31).
Return would simply go back to that address:

```
jr $ra
```

15. Given the following logic array diagram, show the connections needed to make `X = OR(AND(A, NOT(B)), AND(A, B, C), AND(A, B, D))`



This page was intentionally blank before I printed this note here. ;-)

Reference Information

Feel free to separate (and discard) this sheet from your exam.

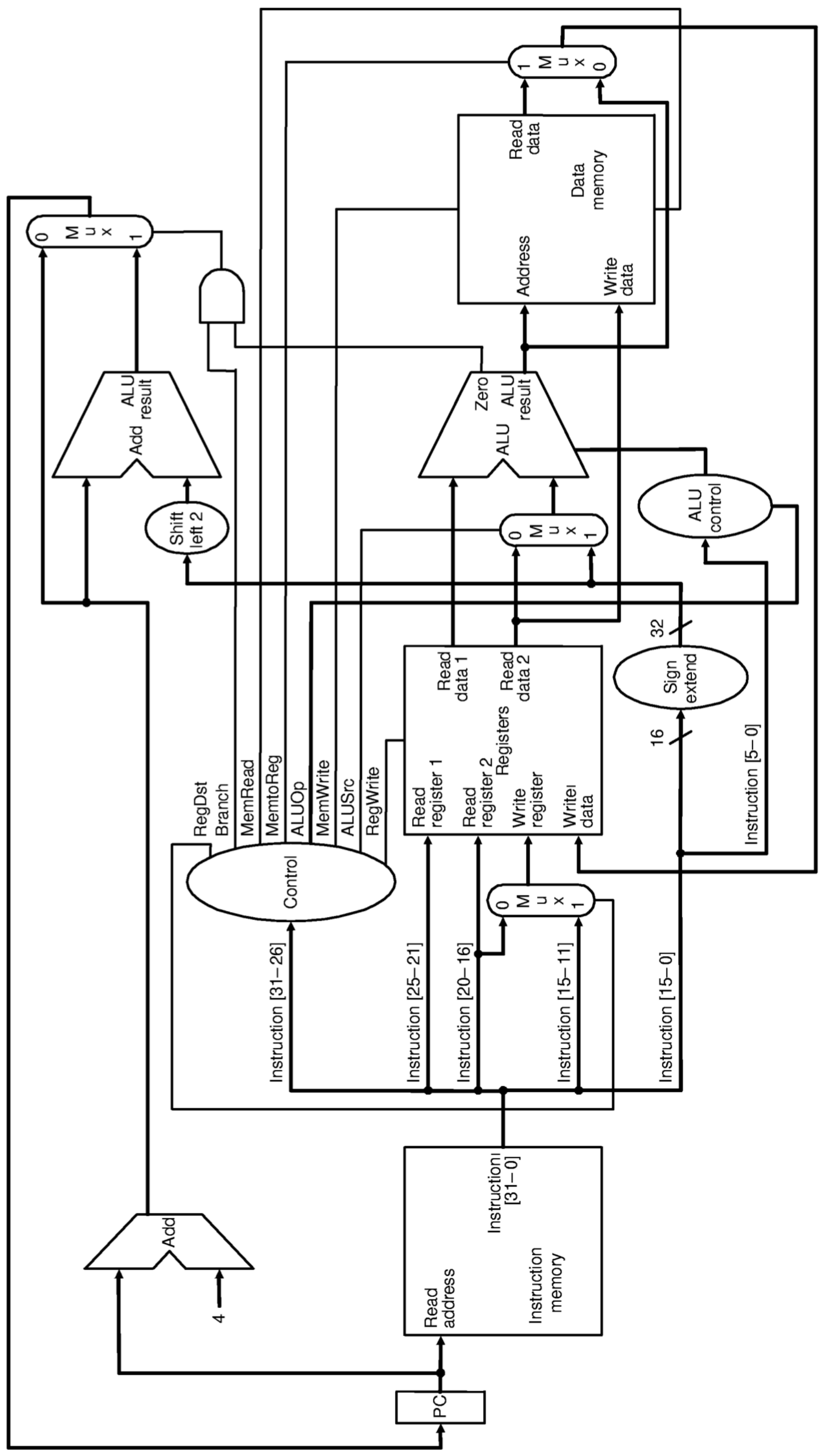
It is intended as a reference only.

MIPS Instructions (a subset)

```
add     r1,r2,r3
addu    r1,r2,r3
addi    r1,r2,immediate
addiu   r1,r2,immediate
and     r1,r2,r3
andi    r1,r2,immediate
or      r1,r2,r3
ori     r1,r2,immediate
sub     r1,r2,r3
subu    r1,r2,r3
subi    r1,r2,immediate
subiu   r1,r2,immediate
xor     r1,r2,r3
xori    r1,r2,immediate
lui     r1,immediate
slt     r1,r2,r3
sll     r1,r2,shamt      # r1=r2<<shamt
sra     r1,r2,shamt      # r1=r2>>shamt (signed)
srl     r1,r2,shamt      # r1=r2>>shamt
beq     r1,r2,label
bne     r1,r2,label
j       label
jal     label
jr      r1
lw      r1,offset(r2)
sw      r1,offset(r2)
```

Note that for this exam, you **may** use `li` and/or `la` even though they are not really instructions.

Single-Cycle Implementation Architecture



PC
Read address
Instruction [31–0]
Instruction memory
Add
4
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign extend
32
Instruction [5–0]
ALU control
Shift left 2
Add
ALU result
ALU
Zero
ALU result
M u x
0
1
Address
Write data
Read data
Data memory