1. (1 point) Write the 8-bit sign-magnitude binary number 11110000 in the form of a 16-bit sign-magnitude binary number.

1000 0000 0111 0000

2. (3 points) Write the value 18.125 x 10^0 in 32-bit IEEE Floating Point format.

$18.125 \times 10^0 = 18.125 \times 2^0$

$= 10010.001 \times 2^0 = 1.0010001 \times 2^4$

$Exp = 4 + 127 = 131 = 10000011$

$\Rightarrow$ 0 10000011 0010001000 ------ 0

3. (2 points) Suppose there was a large floating point representation which used 16 bits for the exponent. If the exponent was represented using a bias, what would you expect the bias to be?

for 8 bits ~~Exp~~ $2^8/2 - 1 = 127$ is the bias

$\Rightarrow$ for 16 bit Exp $2^{16}/2 - 1 = 2^{15} - 1 = 32767$

4. (2 points) Give the **minimal** sequence of actual MIPS instructions to perform the pseudo instruction

```
ble $t5, $t3, L    (branch to label L if $t5 <= $t3)
```

slt \$t0, \$t3, \$t5

beq \$t0, \$0, L. ($\Rightarrow$ go to L if $t3 \not< t5$)

$\Rightarrow t3 \geq t5$

or $t5 \leq t3$

5. Consider the following code sequence in MIPS assembly:

```
here: beq $t1, $t2, there
 . . .
   . . .

there: add $t1, $t1, $t1
```

(a) (2 points) Can you think of a situation where this code sequence might cause issues given the semantics of the (branch) instructions?

# of instructions between "here" & "there" might be more than what can be represented in the range of a I type instruction.

(b) (3 points) Rewrite the code sequence to fix the problem(s).

still need to test t1 = t2?

```
here: bne $t1, $t2, closeby
        j there
closeby: . . . .

there: add $t1, $t1, $t1
```

6. (10 points) State whether each of the following are [T]rue or [F]alse next to the statement in the space provided. Each carries 1 point.
   (a) An exclusive-NOR gate output is HIGH when the inputs are unequal. [F]
   (b) A circuit containing only OR and NOT gates must be a combinational circuit.[F]
   (c) For two's complement numbers, the negative of a number can be found by adding one and then inverting the bits.[F]
   (d) The word address 0x01FF044C is word aligned in a 32 bit computer with byte addressed memory.[T]
   (e) Doubling the number of registers of a register file (but leaving everything else the same) will double the number of input lines to the register file.[F]
   (f) To perform the operation $A - B$, where A and B are numbers represented in twos complement, one can build hardware to perform the following steps: flip the bits of A, add B with a carry-in of 1, flip the bits of the result, and then add 1.[T]
   (g) The opcode field of all J-format MIPS instructions is 0. [F]
   (h) Since MIPS provides instructions to access memory directly (such as `lw` and `sw`) MIPS is NOT a load-store architecture. [F]
   (i) A CPU with a faster clock frequency always has higher performance than one with a slower clock. [F]
   (j) The instruction pair used for function calls in MIPS are `JALR` and `JR`. [F] JAL & JR.

7. (3 points) State one instruction each of I-type, R-type and J-type that allows us to change control flow in MIPS.

bne/beq

jr

jal

8. (4 points) Consider the following program fragment in MIPS32 assembly code:

```
li $s0, -1
srl $v0, $s0, 1
addiu $a0, $v0, 1
```

pseudo instruction = 2 real instructions

$\Rightarrow$ 4 total instructions = 128 bits

Which of the following statements are correct? Justify.

(i) The fragment of code will occupy 128 bits in memory
(ii) Register $a0 will contain the largest positive representable signed number after executing the fragment
(iii) Register $a0 will contain the least negative representable signed number after executing the fragment
(iv) The fragment will raise an overflow exception
(v) Register $v0 will contain −1 after executing the fragment

0 0 00 000 0 000 1

11 11 111 1110

1

0x FFFFFFFF

0111 FFFF ... f

0111 1111 1111 1111 1111 1111 1111 1111

1

1000 0000 0000 00

```
# x is the first argument and has been stored in $a0
# y is the second argument and has been stored in $al
GCD:    subi $sp, $sp, 12   # create stack frame
        sw  $a0, 0($sp)     # save x
        sw  $al,  4($sp)    # save y
        sw  $ra,  8($sp)    # save return address

        bne $a0, $al, rec  # if x != y, jump to 'rec'

        # if x == y, return x
        move $v0, $a0
        addi $sp, $sp, 12
```

```
jr $RA .        # return
```

```
    # The recursion begins
rec:  bgt $a0, $al, xgty    # if x > y, jump to xgty
xlty: sub $al, $al, $a0     # a1 <- y-x
```

```
jal GCD          # call GCD(x, (y-x))
```

```
        # after returning from GCD(x, (y-x))
        lw    $a0, 0($sp)       # restore x
        lw    $al, 4($sp)       # restore y
        lw    $ra, 8($sp)       # restore return address
        addi  $sp, $sp, 12      # destroy stack frame
```

```
jr $RA .        # return
```

```
xgty:
    sub $a0 $a0, $a1
    jal GCD
    lw $a0, 0($SP)
    lw $a1, 4($SP)
    lw $ra, 8($SP)
    addi $SP, $SP, 12.
    jr $ra.
```