Question 0 (1 point): Fill out this front page correctly and put your name and login correctly at the top of each of the following pages. It is AMAZING the number of people which got this question wrong!

This booklet contains seven numbered pages including the cover page, plus a copy of the back inside cover of Patterson & Henessey. Put all answers on these pages, please; don’t land in stray pieces of paper. This is a closed book exam, calculators are allowed.

When writing procedures, write straightforward code. Do not try to make your program slightly more efficient at the cost of making it impossible to read and understand.

When writing procedures, don’t put in error checks. Assume that you will be given arguments of the correct type and specified format.

Our expectation is that many of you will not complete one or two of these questions. If you find one question especially difficult, leave it for later; start with the ones you find easier. We will use truncate as our rounding mode to round all fractional points to integer values.

Question 1 (5 points):
Consider the following 32 bit binary value 11111111111111111111111111111110.

Each part is 5/6th of a point, which since we truncate on the rounding, means that you lose a point for each missed question, with a minimum of 0.

(a) Write this value out in hexadecimal.

0xffffffff

(b) decimal, interpreting it as an unsigned value. Write this as the nearest power of 2 and add or subtract the appropriate offset. (EG, if you want to write 9, write $2^3 + 1$.)

$2^{32} - 2$ Simply 2 less that $2^{32}$.

(c) decimal, interpreting it as a sign/magnitude value. Write this as the nearest power of 2 and add or subtract the appropriate offset. (EG, if you want to write -9, write $-(2^3 + 1)$.)

$-(2^{31} - 2)$ Its negative (because of the sign bit).

(d) decimal, interpreting it as a ones complement signed value. Write this as the nearest power of 2 and add or subtract the appropriate offset.

$-1$ aka $-(2^0)$. Just invert the bits, since it is negative.

(e) decimal, interpreting it as a two's complement signed value. Once again, write this as the nearest power of 2 and add or subtract the appropriate offset.

$-2$ aka $-(2^1)$ Invert the bits and add 1

(f) What is this value if you interpret it as IEEE single precision floating point? (Remember, IEEE single precision floating point has an 8 bit exponent with a bias of 127 and a 23 bit significand).

NaN (Exponent is maximum and mantissa is nonzero)
Question 2 (6 points):

Each part is 1.5 points, which means that 1 right gives 1 point, 2 right gives 3 points, 3 right gives 4 points, and 4 right gives 6 points. When I wrote -1/2 it means you didn’t lose half a point, you lost the credit for that question.

(a) Encode the following MIPS instruction in its binary representation: `lui $20 0xf00d 001111 00000 10100 1111000000001110`

If you get the order of the Rs and Rd fields wrong, you get half credit.

(b) Decode the following binary number as a MIPS instruction and give the equivalent MIPS assembly language statement:

```
000000111010000101000000010000
00000 01110 10000 10100 00000 10000
add $20 $14 $16
```

Once again, mixed up fields gives half credit.

(c) `li $Rd high` is a pseudo instruction with a 32 bit immediate. Convert it to a series of actual MIPS instructions. For credit, you need to use the exact minimum of MIPS instructions. (You can use high to signify the upper 16 bits of the immediate, and low to signify the lower 16 bits.)

```
lui $Rd high
ori $Rd low
```

You can’t use something like `addi` or `addiu` since they sign extend the value. You will get 1/2 points if you did `add` or `addiu` but it was otherwise optimal (2 instructions). If you do something which works but is non-optimal in about 3 instructions, you will get 1/2 points. If you forgot the source register in the ci, you got 1/2 the points.

(d) The `addiu` instruction uses a 16 bit immediate. What is the largest constant which can be added with the `addiu` instruction? (HINT: The immediate is sign extended).

$2^{15} = 1$ is one answer. $2^{16} = 1$ is the other possibility, if you wish to treat all 1s as a positive number.

Question 3 (6 points):

Each question is worth 6/5 of a point. 1 right gives 1 point, 2 right gives 2 points, 3 right gives 3 points, 4 right gives 4 points, and 5 right gives 6 points.

Answer the following short questions. Be sure to read the questions carefully before answering:

(a) True or false: For every 32 bit signed, two’s complement number there exists a corresponding IEEE double precision floating point number. True. Double precision has enough bits in the significand to represent all 32 bit signed integers. Single precision does not.

(b) What is the difference between the `add` and `addu` instructions in MIPS? `addu` does not detect overflow. Just saying `addu` is for unsigned only gives you 1/2 the points. I really should have given nothing, since it is quite clear in the back of P&H that the difference is overflow detection.

(c) True or false: Two’s complement integer addition is not associative. False. Two’s complement integer is associative. Floating point is not.

(d) Using your 1st grade math, add the following pairs of 8 bit unsigned numbers together

```
11111111 + 10110101
00101111 + 01101011
--------- + ---------
11001010 00101010
```

(e) Using saturating arithmetic, add the following 8 bit signed, two’s complement numbers together.

```
11111111 + 00101010
00101010 + 01101011
--------- + ---------
01100100 10101011 -> 01111111
```
Question 4 (7 points):

Consider this C program definition:

```c
int foo(int a){
    int i;
    int result = 1;
    for(i = 0; i < a; ++i){
        result = result + bar(i);
    }
    return result;
}
```

Grading. -1 point for forgetting to save Ra. -2 points for using temporary registers and thinking they would be saved across function calls. -1 point for needlessly saving results on the stack instead of copying the value to a saved register for something like i or a. -1 point for calling bar wrong. -1 point for getting the test in the loop wrong. -2 points for clobbering saved registers. -1 point for not restoring the stack correctly. Beyond a certain level of errors, it’s up to the charity of the grader. People really seemed to have trouble with this, so I will go over the MIPS calling convention again on Monday.

Your name ____________________________ login cs61c--

Question 5 (5 points):

Write a MIPS function `div4` which accepts a single argument which is an IEEE double precision floating point number in `$a0` and `$a1` (with the most significant bits in `$a0`), divides it by 4, and returns that value **without using any floating point instructions**. You do not need to and **should not include** code to handle underflow, subnormal values, ±∞, or NaN. (Remember, IEEE double precision floating point has an 11 bit exponent with a bias of 1023 and a 52 bit significand).

2 points for the basic concept, getting function calling right, etc etc. The other 3 points are for details like making sure you actually replace the exponent with the new value, subtracting 2 instead of doing something funky to the exponent, returning both parts of the arguments. If you did something weird like trying to divide the significand, you will get at most 2 points. You didn’t need to worry about O. (which would have only added a test on the exponent in any case, and nobody did).

```
div4: # This is VERY similar to the homework problem where you had to
# multiply a single precision floating point number by 2. The idea
# is to accomplish the division by modifying the exponent, so
# we isolate the exponent, subtract 2 (equivalent to dividing
# the number by 4) and replace it. We don’t touch anything else
# srl $t0 $a0 2
# addi $t0 $t0 0xffff
# sll $t0 $t0 2
# or $a0 $a0 $t1
# or $v0 $a0 $t1
# move $v1 $a1
# jr $ra
```

```
# My implementation uses $s0 for storing a, $s1 for i
# and $s2 for result for the bulk of the function.
foo: addi $sp $sp -16
sw $s0 0($sp)
sw $s1 4($sp)
sw $s2 8($sp)
sw $ra 12($sp)
move $s0 $a0
li $s2 1
li $s1 0
j test
move $s0 $s1
jal bar
add $s2 $s2 $v0
addi $s1 $s1 1
i++
```

```
test: bll $s1 $s0 loop
move $v0 $s2
lw $s0 0($sp)
```

```
lw $s1 4($sp)
lw $s2 8($sp)
lw $ra 12($sp)
add $sp $sp 16
jr $ra
```

5