Name (Last, First): Solutions

This exam consists of 5 questions on 7 pages. Be sure you have the entire exam before starting. The point value of each question is indicated at its beginning; the entire exam is worth 100 points. Individual parts of a multi-part question are generally assigned approximately the same point value; exceptions are noted. For this exam you are allowed to bring a single page of notes (front and back). You may NOT share material with another student during the exam. Use of electronic devices is not allowed.

Be concise and clearly indicate your answers. Presentation and simplicity of your answers may affect your grade. Answer each question in the space following the question. If you find it necessary to continue an answer elsewhere, clearly indicate the location of its continuation and label its continuation with the question number and subpart if appropriate.

You should read through all the questions first, then pace yourself.

Problem	Possible	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total	100	

1 (20 points)

Short answer questions

(a) Consider the following C code snippet:

```
int *p = (int *) 0x2ABC3348;
p = p + 3;
```

What is the value of p after executing this snippet? Give the value in hexadecimal.

Due to pointer arithmetic
p = p+3 is really p = p+(3*4)
or p = p+12 12 = 0xC

```
   1
0x2ABC3348
+ 0x       C
0x2ABC3354
```

(b) Consider the following C code snippet:

```
uint32_t r = ((0xAABBCC) >> 12) & 0xFF;
```

What is the value of r in hexadecimal after executing this statement?

0xAABBCC >> 12 = 0xAAB
& 0x0FF
0x AB

0xAB

(c) Give the 8-bit binary 2's complement value for the integer -7. Show your work.

+7 in binary is ⟶ 00000111
inv 11111000
To get -7 in binary we +1 1
need to invert and add 1.
11111001

(d) If we have a byte address, `addr` = 28, what is the word address (`addr_word`) for this address? Explain your answer.

To get the word address from a byte address we need to divide by 4.

addr_word = addr / 4
= 28 / 4 = 7

2 (20 points)

RISC-V Assembly Part 1

Consider the following RISC-V assembly code snippets and initial register values. Show the values of each register used next to each instruction. For example:

```
li  t0, 1      # t0 <- 1
add t0, t0, t0 # t0 <- 1 + 1 = 2
```

You must show your work to get credit.

(a) Assume `a0 = 2, a1 = 3, a2 = 7`. What is the value of `a0` after executing this snippet?

```
add a0, a1, a2
mul a0, a0, a1
slli a0, a0, 1
```

a0 = a1 + a2 = 3 + 7 = 10
a0 = a0 * a1 = 10 * 3 = 30
a0 = a0 << 1 = 30 << 1 = 60

a0 = 60

(b) Assume `a0 = 2, a1 = 3`. What is the value of `a0` after executing this snippet?

```
    beq a0, a1, foo
    addi a0, a0, 1
    j boo
foo:
    addi a0, a0, 22
    j goo
boo:
    beq a0, a1, goo
    addi a0, a0, 33
goo:
    addi a0, a0, 1
```

a0 == a1 ? 2 ≠ 3
a0 = a0 + 1 = 2 + 1 = 3
a0 == a1 ? 3 == 3
a0 = a0 + 1 = 3 + 1 = 4

a0 = 4

(c) Assume `a0 = 10, a1 = 22`. What is the value of `a0` after executing this snippet?

```
addi sp, sp, -16
sw a1, (sp)
addi a1, a1, 20
sw a1, 4(sp)
addi a0, sp, 4
lw a0, (a0)
addi sp, sp, 16
```

mem[sp] = a1 = 22
a1 = a1 + 20 = 22 + 20 = 42
mem[sp+4] = 42
a0 = sp + 4
a0 = mem[sp+4] = 42

a0 = 42

(d) Assume `a0 = 3, a1 = 2`. What is the value of `a0` after executing this snippet? How many instructions are executed in this snippet? Labels do not count as instructions, but branches count if they are taken or not.

```
loop:
    beq a1, zero, done
    addi a0, a0, 1
    addi a1, a1, -1
    j loop
done:
    addi a0, a0, 2
```

a1 == 0 ? 3 ≠ 0
a0 = a0 + 1 = 3 + 1 = 4
a1 = a1 - 1 = 2 - 1 = 1
j loop

a1 == 0 ? 1 ≠ 0
a0 = a0 + 1 = 4 + 1 = 5
a1 = a1 - 1 = 1 - 1 = 0
j loop

a1 == 0 ? 0 == 0
a0 = a0 + 2 = 5 + 2 = 7

a0 = 7

10 instructions

3 (20 points)

RISC-V Assembly Part 2

Consider the following RISC-V assembly function. Answer the questions below.

```
add2_s:
    add a0, a0, a1
    ret

add4f_s:
    addi sp, sp, -48
    sd ra, (sp)
    sd s0, 8(sp)
    sd s1, 16(sp)
    sd s2, 24(sp)
    sd s3, 32(sp)

    mv s0, a2
    mv s1, a3
    call add2_s

    mv s2, a0
    mv a0, s0
    mv a1, s1
    call add2_s

    mv s3, a0
    mv a0, s2
    mv a1, s3
    call add2_s

    ld s3, 32(sp)
    ld s2, 24(sp)
    ld s1, 16(sp)
    ld s0, 8(sp)
    ld ra, (sp)
    addi sp, sp, 48
    ret
```

8 8 8 8 8 8
40

(a) What does the `add4f_s` function do? Give a possible C prototype for this function.

add4f_s adds four integers in a0, a1, a2, a3

int add4f_c(int a, int b, int c, int d);

(b) What caller-saved registers are preserved if any in `add4f_s`?

ra

(b) What callee-saved registers are preserved if any in `add4f_s`?

s0, s1, s2, s3

(d) How much stack space is actually used by this function? Note that this may not be the same as how much is allocated. Explain your answer.

40 bytes

4 (20 points)

RISC-V Is-Uppercase

Consider the following C function, `is_uppercase_c()` that takes an `char c` and determines if `c` is upper case.

```
int is_uppercase_c(char c) {
    int r;

    if (c >= 'A' && c <= 'Z') {
        r = 1;
    } else {
        r = 0;
    }

    return r;
}
```

Write an equivalent RISC-V assembly version of this function called `is_uppercase_s`. Your implementation must follow the logic in the C version and must follow the RISC-V calling conventions. Hint: the ASCII value of 'A' is 65.

```
.global is_uppercase_s

is_uppercase_s:
    li t0, 65
    li t1, 91
    blt a0, t0, else
    bgt a0, t1, else
    li a0, 1
    j done
else:
    li a0, 0
    ret
```

```
 1
65
26
--
91
```

5 (20 points)

RISC-V Instruction Word Decoding

For this problem we are going to decode parts of RISC-V instruction words.

First, consider the R-type instruction format:

```
|31               25|24     20|19      15|14    12|11      7|6         0|
|       funct7      |   rs2   |    rs1   | funct3 |    rd    |  opcode   |
```

Here is an implementation of `get_opcode(uint32_t iw)`:

```
uint32_t get_opcode(uint32_t iw) {
    uint32_t opcode;
    opcode = iw & 0b1111111;
    return opcode;
}
```

(a) Write a C function called `uint32_t get_funct3(uint32_t iw)` that returns the value of `funct3` from an R-type instruction word. You must write the function directly using C bitwise operators, you cannot call other functions like `git_bits()`:

```
uint32_t get_funct3(uint32_t iw) {
    uint32_t funct3
    funct3 = (iw >> 12) & 0b111;
    return funct3;
}
```

Now, consider the J-type instruction word format:

```
|31                                              12|11      7|6          0|
|          imm[20|10:1|11|19:12]                    |    rd    |   opcode   |
```

(b) Write a C function called `is_j_type_imm_negative(uint32_t iw)` that will return `1` if the J-type immediate value is negative and `0` if the J-type immediate is positive. Hint: you do not need to extract the entire immediate value.

```
uint32_t is_j_type_imm_negative(uint32_t iw) {
    return (iw >> 31) & 0b1;
}
```

Continue your answers here if necessary.