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 + 2;
```

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

Due to pointer arithmetic

p = p + 2 really adds 2*4 = 8

```
  0x2ABC3348
+          8
------------
  0x2ABC3350
```

(b) Consider the following C code snippet:

```
uint32_t r = ((0xAABBCC) >> 4) & 0xFF;
```

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

```
0xAABBCC >> 4 = 0xAABBC
  0x      FF
-------------
  0x      BC
```

→ 0xBC

(c) Give the 8-bit binary 2's complement value for the integer -11. Show your work.

+11 in binary is 00001011

To get -11 we just need to invert and add one.

```
invert  11110100
(+1)           1
----------------
  11110101
```

(d) If we have a word address, `addr_word` = 15, what is the byte address (`addr`) for this word address? Explain your answer.

To get the byte address from the word address we multiply by 4.

byte address = 15 x 4 = 60

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
srl a0, a0, 1
```

a0 = a1 + a2 = 3 + 7 = 10
a0 = a0 * a1 = 10 * 3 = 30
a0 = a0 >> 1 = 15

a0 = 15

(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, 10
sw a1, 4(sp)
addi a0, sp, 4
lw a0, (a0)
addi sp, sp, 16
```

mem[sp] = a1 = 22
a1 = a1 + 10 = 22 + 10 = 32
mem[sp+4] = 32
a0 = sp + 4
a0 = stack[sp+4] = 32

a0 = 32

(d) Assume `a0 = 4, 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? 2 ≠ 0
a0 = a0 + 1 = 4 + 1 = 5
a1 = a1 - 1 = 2 - 1 = 1
j loop

a1 == 0? 1 ≠ 0
a0 = a0 + 1 = 5 + 1 = 6
a1 = a1 - 1 = 1 - 1 = 0
j loop

a1 == 0? 0 == 0

a0 = a0 + 2 = 6 + 2 = 8

a0 = 8

#instructions = 10

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, -64
    sd ra, (sp)      8

    sd a2, 8(sp)     8
    sd a3, 16(sp)    8

    call add2_s

    sd a0, 24(sp)    8
    ld a0, 8(sp)
    ld a1, 16(sp)

    call add2_s

    mv a1, a0
    ld a0, 24(sp)

    call add2_s

    ld ra, (sp)
    addi sp, sp, 64
    ret
```

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

add4f_s adds 4 integers a0, a1, a2, a3.
int add4f(int a, int b, int c, int d);

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

ra, a2, a3, a0

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

none

(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.

32 bytes

4 (20 points)

RISC-V In-Range Checker

Consider the following C function, `inrange_c()` that takes an `int x` and a range defined by `int a` and `int b`. The function will return true (1) if `a >= x <= b`, and false (0) otherwise.

```
int inrange_c(int x, int a, int b) {
    int r;

    if (x >= a && x <= b) {
        r = 1;
    } else {
        r = 0;
    }

    return r;
}
```

Write an equivalent RISC-V assembly version of this function called `inrange_s`. Your implementation must follow the logic in the C version and must follow the RISC-V calling conventions.

```
.global inrange_s

inrange_s:

        blt a0, a1, else
        bgt a0, a2, else
        li a0, 1
        j done
else:
        li a0, 0
done:
        ret
```

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_rs1(uint32_t iw)` that returns the value of `rs1` 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_rs1(uint32_t iw) {
    uint32_t rs1
    rs1 = (iw >> 15) & 0b11111;
    return rs1;
}
```

Now, consider the B-type instruction word format:

```
|31              25|24     20|19      15|14    12|11              7|6       0|
|  imm[12]imm[10:5] |   rs2   |    rs1   | funct3 | imm[4:1]imm[11] | opcode |
```

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

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

Continue your answers here if necessary.