1	/10
2	/18
3	/18
4	/17
5	/21

M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Fall 2020

Practice Quiz #1B
(adapted from Quiz #1 Spring 2019)

Name	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-302 (Kade)	☐ WF 12, 34-302 (Driss)	☐ WF 2, 34-302 (Jeremy)
☐ WF 10, 34-301 (Intae)	☐ WF 12, 35-310 (Jason)	
☐ WF 11, 34-302 (Kade)	☐ WF 1, 34-302 (Driss)	☐ WF 3, 34-302 (Jeremy)
☐ WF 11, 34-301 (Intae)	☐ WF 1, 35-310 (Jason)	

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. Show your work for partial credit. You can use the extra white space and the backs of the pages for scratch work.

Problem 1. Binary Arithmetic (10 points)

(A) (4 points) What is ~(0xC7) ^ 0x1F, where ~ is bitwise NOT and ^ is bitwise XOR? Provide your result in both binary and hexadecimal. Show your work for partial credit.

Result in binary (0b):_____

Result in hexadecimal (0x):_____

(B) (3 points) Multiply 9 by 5 using unsigned binary numbers. **Show all of your work by filling in the missing rows in the table below** and provide your final answer in binary.

				1	**0**	**0**	**1**
				0	**1**	**0**	**1**

9 x 5 (0b):_____

(C) (3 points) What is -41 in 8-bit 2's complement notation? Provide your answer in binary and hexadecimal.

-41 in binary (using 8-bit 2's complement notation) (0b):_____

-41 in hexadecimal (0x):_____

Problem 2. RISC-V Assembly (18 points)

(A) You are given the following C code along with an incomplete translation of this code into RISC-V assembly. Assume that execution ends when it reaches the `unimp` instruction.

```
int a[4] = {0x1, 0x2, 0x4, 0x8}; // a[0]=0x1, a[1]=0x2, etc.
int b = 0x5;
int c = 0;
for (int i = 3; i >= 0; i--) {
  c = a[i] & b;  // & is bitwise-AND
  if (c != 0){
    break;
  }
}

// Translation into RISC-V assembly:
. = 0x0
  li a0, 0x800   // starting address of array
  li a1, 0x5
  li a2, 0x3
  mv t0, a2
```


```
  add t0, a0, t0
  lw t1, 0(t0)
  and t1, t1, a1
  bnez t1, end;
  addi a2, a2, -1
  bgez a2, loop
end: unimp

. = 0x800        // contents of array a, where a[0] is at 0x800
.word 0x1
.word 0x2
.word 0x4
.word 0x8
```

1) (4 points) Complete the RISC-V translation by:
   a. Inserting the loop label where it belongs in the code.
   b. Filling in the blank box with the missing instruction.

2) (6 points) Provide the values left in the registers below following the execution of the corrected code:

Value left in t0: 0x_____

Value left in t1: 0x_____

Value left in a2: 0x_____

(B) (8 points) Provide the values left in the registers after executing the code below. Assume that execution ends when it reaches the `unimp` instruction. (. = 0x0 means that the first addi instruction is at address 0x0. Zero is x0 which = 0).

```
. = 0x0
    addi a1, zero, 0x234
    addi a3, zero, 0x1
    jal a2, L1
    xor a3, a1, a1
    beqz a3, end
    j L2

L1: addi a2, a2, 4
    jr a2
L2: add a3, a1, a1

end:  unimp
```

Value left in a2: 0x_____

Value left in a3: 0x_____

Problem 3. RISC-V Assembly and Calling Convention (18 points)

(A) (8 points) The **getPower** function in C given below recursively computes the power of a number (e.g. **getPower**(2,3) = 2^3 = 8). Since our RISC-V processor does not have a multiply instruction, the code uses a multiply procedure, **mult**, which multiplies two signed integers. The implementation of the **mult** procedure is not provided to you, but it does obey the calling convention.

The RISC-V code given below is not behaving as expected and you are tasked with fixing the bugs so that the code works correctly and follows the RISC-V calling convention. You're allowed to add up to **5 instructions** to make it work. You are **not** allowed to modify or remove any of the given instructions, but feel free to change the order in which they appear.

```
int getPower(int b, int p){
      if (p == 0)
           return 1;
      else
           return mult(getPower(b, p-1), b);
}
```

```
getPower:
 ⟶  beqz a1, end2
      addi sp, sp, -8
      addi a1, a1, -1
      jal getPower
      jal mult
end2:
      li a0, 1
      ret
end1:
      addi sp, sp, 8
      ret
mult: ...
```

Provide the correct assembly code here. Your implementation should obey the RISC-V calling convention.

```
getPower:
```

With the correct RISC-V code, what is the value of **`sp`** at the pointed instruction (`beqz a1, end2`) during its third recursive call to **`getPower`**? Assume that the original call to getPower is **not** considered one of the recursive calls. **Assume `sp` is 0x101C (in hex) initially.** Assume p > 3.

sp (0x) = _____

(B) (10 points) The **getSSD** function given below computes the sum square distance between the elements in arrays a and b (i.e. getSSD(a[], b[], size) = $\sum_{i=0}^{size-1}(a[i]-b[i])^2$. Again, this code uses a multiply procedure, **mult**, which multiplies two signed integers. The implementation of the **mult** procedure is not provided to you, but it does obey the calling convention.

Please fix the bugs in the given RISC-V code such that it works correctly and follows the RISC-V calling convention. **Assume arrays a and b have the same size.** You're allowed to add up to **13 instructions** to make it work. Feel free to change the order of instructions, but you **are NOT allowed to modify any of the given instructions.**

```
int getSSD(int a[], int b[], int size) {
      int sum = 0;
      for (i = 0; i < size; i = i + 1) {
            sum = sum + mult((a[i] - b[i]), (a[i] - b[i]));
      }
      return sum;
}
```

Provide the correct assembly code here. Your implementation should obey the RISC-V calling convention. *Feel free to only show the changes to the code with arrows to where the changes go in the original code.*

```
// a0 -> base address for a
// a1 -> base address for b
// a2 -> size

getSSD:
      mv s1, a0 // a address
      mv s2, a1 // b address
      slli a2, a2, 2
      add a2, a0, a2 // end address of a
      li s0, 0 // s0 has a running sum
      j compare
loop:
      lw a4, 0(s1)
      lw a5, 0(s2)
      sub a0, a4, a5
      jal mult
      add s0, s0, a0
      addi s1, s1, 4
      addi s2, s2, 4

compare:
      bgt a2, s1, loop
end:
      mv a0, s0
      ret
```

```
getSSD:
```

Problem 4. Stack Detective (17 points)

Consider the C procedure below and its translation to RISC-V assembly code, shown on the right.

```
int f(int a, int b) {
  int c = b - a;
  if (c & ??? == 0)
    // c is a multiple of 4
    return 1;
  else {
    int d = f(a - 1, b + 2);
    return 3 * (d + a);
  }
}
```

```
f:    sub a2, a1, a0
      andi a2, a2, ???
      bnez a2, ELSE
      li a0, 1
      jr ra  // ret

ELSE: addi sp, sp, -8
      sw a0, 0(sp)
      sw ra, 4(sp)
      addi a0, a0, -1
      addi a1, a1, 2
      jal ra, f
      lw a1, 0(sp)
      lw ra, 4(sp)
L1:   add a0, a0, a1
      slli a1, a0, 1
      add a0, a0, a1
      addi sp, sp, 8
      jr ra  // ret
```

(A) (2 points) What value should the **???** term in the C code and the assembly be replaced with to make the if statement correctly check if the variable 'c' is a multiple of 4? *(Hint: In the collatz program in Lab 1 we learned that we can test if a number is even, or a multiple of 2, by checking if its last bit is 0.)*

Correct value of '???' term? _____

(B) (2 points) How many words will be written to the stack before the program makes each recursive call to the function f?

Number of words pushed onto stack before each recursive call? _____

The program's initial call to function **f** occurs outside of the function definition via the instruction '**jal ra, f**'. The program is interrupted at an execution (not necessarily the first) of function **f**, *just prior* to the execution of '**add a0, a0, a1**' at label **L1**. The diagram on the right shows the contents of a region of memory. All addresses and data values are shown in hex. **The current value in the SP register is 0xEB0 and points to the location shown in the diagram.**

Memory Contents

	Address	Data
	0xEA4	0x0
	0XEA8	0x1
	0XEAC	0xA4
SP→	0XEB0	0x2
	0xEB4	0xA4
	0xEB8	0x3
	0xEBC	0xA4
	0xEC0	0x4
	0xEC4	0x3B8
	0xEC8	0x12
	0xECC	0x44
	0xED0	0xCE8

(C) (4 points) What were the values of arguments a and b to the initial call to f? Write CAN'T TELL if the argument does not show up in the stack.

Initial arguments to f: a = _____ ; b = _____

(D) (4 points) What are the values in the following registers right when the execution of f is interrupted? Write CAN'T TELL if you cannot tell.

Current value of a1: 0x_____

Current value of ra: 0x_____

(E) (2 points) What is the hex address of the '**jal ra, f**' instruction that made the initial call to f?

Address of instruction that made initial call to f: 0x _____

(F) (3 points) What is the hex address of the instruction at label ELSE?

Address of instruction at label ELSE: 0x _____

Problem 5. Boolean Algebra and Combinational Logic (21 points)

You are given the truth table for a circuit that takes a 3-bit unsigned binary input (X = ABC), adds 3 mod 8 to it, to produce a 3-bit unsigned binary output (Y = A'B'C').

A	B	C	A'	B'	C'
0	0	0	0	1	1
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	0	0	1
1	1	1	0	1	0

(A) (9 points) For the above truth table, write out a **minimal sum-of-products** for each function A'(A,B,C), B'(A,B,C), and C'(A,B,C)

Minimal sum-of-products for A'(A,B,C)= _____

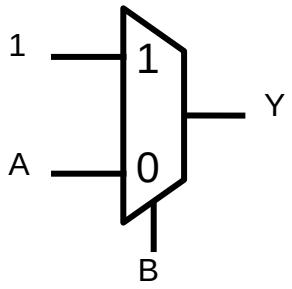
Minimal sum-of-products for B'(A,B,C)=_____

Minimal sum-of-products for C'(A,B,C)=_____

(B) (3 points) Which **one** of these functions can be used to build any boolean function (i.e., it's a universal gate)? Assume that you may tie inputs to 1 or 0 if necessary. **Explain your answer.**

A' **B'** **C'**

(C) (3 points) What function, of A and B, does the following circuit implement?



Y(A, B) = _____

(D) (4 points) Draw the circuit for **B'** using only 2-input 1-bit muxes. Make sure to label all the inputs and the output of your circuit.

(E) (2 points) You are told that the propagation delay of a 2-input 1-bit mux is 2ns. What is the propagation delay for output B' based on the circuit you drew above?

Propagation Delay of B' (ns) = _____

END OF QUIZ 1!