20F-COMSCIM151B-1 Homework 2

NEVIN LIANG

TOTAL POINTS

47 / 55

QUESTION 1

1 Assembly for simple C statement 5 / 5

✓ **+ 5 pts Correct**

+ 0 pts Incorrect

+ 2 pts Not minimal

QUESTION 2

Load immediate 10 pts

2.1 A_upper bits 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect

2.2 Leftmost bits 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect

2.3 Filled with ones? 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect

2.4 U_upper fix 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect

2.5 Alternative instruction 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect

QUESTION 3

Assembly to machine code 10 pts

3.1 lui 2.5 / 2.5

✓ **+ 2.5 pts Correct**

+ 0 pts Incorrect

+ 1 pts Correct binary

3.2 ori 2.5 / 2.5

✓ **+ 2.5 pts Correct**

+ 0 pts Incorrect

+ 1 pts Correct binary

3.3 addi 2.5 / 2.5

✓ **+ 2.5 pts Correct**

+ 0 pts Incorrect

+ 1 pts Correct binary

3.4 lw 0 / 2.5

+ 2.5 pts Correct

✓ **+ 0 pts Incorrect**

+ 1 pts Correct binary

QUESTION 4

Narrow immediate 10 pts

4.1 Load immediate 5 / 5

✓ **+ 5 pts Correct**

+ 0 pts Incorrect

- 1.5 pts Invalid assembly

+ 2.5 pts Correct idea but wrong constant loaded

- 1 pts Side effects

4.2 Load from memory 5 / 5

✓ **- 0 pts Correct**

- 1 pts Side effect

- 2 pts Issue with sign extension

- 1 pts Invalid assembly

- 5 pts Incorrect

QUESTION 5

5 Multiplication in assembly 7 / 10

- 0 pts Correct

- **10 pts** Incorrect
- **2 pts** Invalid assembly
- **8 pts** Incorrect loop construct
✓ **- 3 pts** **Wrong for some operands**
- **2 pts** Wrong load immediate
- **1 pts** Only one wrong instruction

QUESTION 6

HW/SW multipliers 10 pts

6.1 HW 4 bits 2.5 / 2.5

✓ **+ 2.5 pts** **12tu**
+ **2.5 pts** 24tu
✓ **+ 0 pts** **36tu**
+ **0 pts** 48tu
+ **0 pts** Incorrect

6.2 HW 32 bits 0 / 2.5

+ **2.5 pts** 224tu
+ **2.5 pts** 448tu
✓ **+ 0 pts** **672tu**
+ **0 pts** 896tu
+ **0 pts** Incorrect

6.3 SW 4 bits 2.5 / 2.5

+ **2.5 pts** 36tu (3 steps)
✓ **+ 2.5 pts** **48tu (4 steps)**
+ **2.5 pts** 60tu (5 steps)
+ **0 pts** Incorrect

6.4 SW 32 bits 2.5 / 2.5

+ **2.5 pts** 672tu (3 steps)
✓ **+ 2.5 pts** **896tu (4 steps)**
+ **2.5 pts** 1120tu (5 steps)
+ **0 pts** Incorrect

QUESTION 7

7 Late submission penalty 0 / 0

✓ **+ 0 pts** **On time**
- **13.75 pts** Late submission

Homework Set – II

Session: Fall 2020 **Instructor: Prof. P. Gupta**

Total Points: 55 **Due Date: 15 Oct (before class starts)**

Instructions

1. **There are two sections in this homework set. The questions under Section – I are for your practice. Solutions to these problems are already provided on the course web page. Students are encouraged to solve the problems before looking at the solution provided.**
2. **Problems under Section – II are to be submitted onto Gradescope.**

Section I

1. Given your understanding of PC-relative addressing, explain why an assembler might have problems directly implementing the branch instruction in the following code sequence:

   here: *beq x10, x2, there*

   ...

   there: *add x10, x11, x10*

Show how the assembler might rewrite this code sequence to solve these problems.

2. How would you design a multiplier if the one of the operands is always a power of 2?
3. What is the range of a 10-bit 2's complement representation?
4. What is the instruction to load register x5 with content of array data structure A[3] stored in memory? Assume that the base address of array data structure A i.e. address of A[0] is stored in register x9.

lw x5, x9, 24

lw x5, 24/2(x9)

Section II

1. Show the single RISC-V instruction or minimal sequence of instructions for this C statement: **b = 25 | a.** Assume that *a* corresponds to register x0 and *b* corresponds to register x1. Write each instruction in a separate box (for partial scoring). **[5 points]**

x1 = x0

x1 = x1 | 25

ORI x1, x0, 25

2. If A is a 32-bit address, typically an instruction sequence such as:

lui x19, A_upper

addi x19, x19, A_lower

can be used to load the word at A into a register (in this case x19).

- How many bits of A are in A_upper? **[2 points]**

lui x19, upper

20 bits

20

- If the MSB of A_upper is 1, what would be the value of the leftmost 32-bits of register x19. Provide your answer in decimal. **[2 points]**

A_upper: 10001011010010111

32 1's

all 1's

4294967295

- After the *lui* instruction, the rightmost 12-bits of x19 would be filled with all 1's. True or False? **[2 points]**

False

- If the 11th bit of *A* is 1, how should A_upper be changed? Choose (X) one of the following **[2 points]**

  ☒ A_upper = A_upper + 1 ☐ A_upper = A_upper - 1 ☐ None

- What other instruction can be used instead of *addi*? **[2 points]**

ori can be used

3. For the RISC-V statements below, show the bit-level instruction representation of each of the instructions in **hexadecimal**. **[2.5 points each: 10 points]**

a.	0x0000 0000 0000 0000 0x0000 0000 0000 0004	lui x19, 100 ori x19, x19, 40
b.	0x0000 0000 0000 0100 0x0000 0000 0000 0104	addi x19, x19, 0x000 lw x20, 0x400(x19)

lui x19,100 : imm 31:12, rd, opcode

100 = 0b1100100

00000000000001100100100110110111

ori x19, x19, 40 ; imm 11:0, rs1, funct3, rd, opcode

40 = 101000 , 19 = 10011

00000010100010011110010011 00100 11

addi x19, x19, 0x000 ; imm 11:0, rs1, funct3, rd, opcode

00000000000010011000100110010011

lw x20, 0x400 (x19);

0x000649B7

0x0289E993

0x00098993

0x4009 8A03

4. By reducing the size of the immediate fields of the I-type and the U/UJ-type instructions, we can save on the number of bits needed to represent the instructions. If the immediate field of I-type instructions were 8 bits and the immediate field of the U/UJ-type instructions were 16 bits, rewrite the RISC-V code above to reflect this change. Avoid using the *lui* instruction. **[5 + 5 points]**

100 = 1100100 bin
40 = 101000 bin

lui x19, 100
ori x19, x19, 40

1100100 _ _ _ _ _ _ _ _ _ _ 101000

addi x19, x0, 100
SLLI x19, x19, 12
addi x19, x19, 40

addi x19, x19, 0x000
lw x20, 0x400(x19)

a.

ori x19, x0, 100
SLLI x19, x19, 12
ori x19, x19, 40

b.

andi x20, x0, 1
SLLI x20, x20, 10
add x20, x20, x19
lw x20, x20, 0

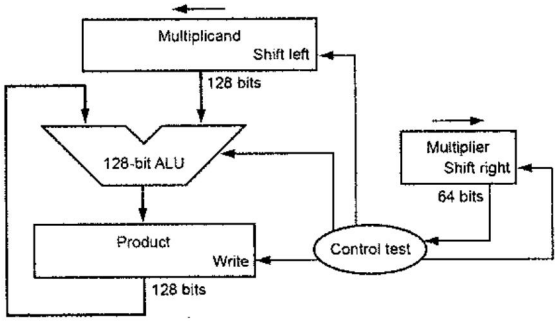
5. Assuming that *mult* operation is not available in RISC-V, implement it using add, shift, bitwise, and branching instructions. i.e. write assembly code to write product result in x21, assuming that operands are available in registers *x19* and *x20*. **[10 points]** Partial points will be provided

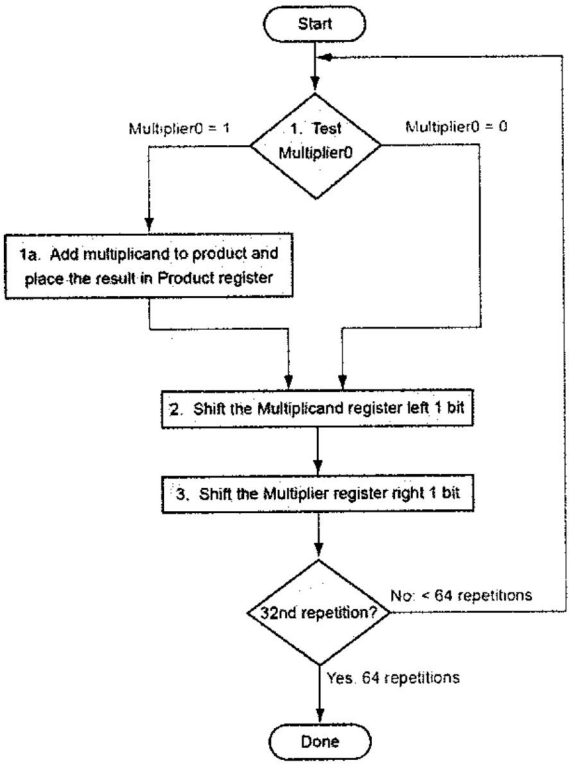
```
andi  x21, x21, 0
l1: add  x21, x21, x19
addi  x20, x20, -1
beq   x20, x0, l2
beq   x0, x0, l1
l2:
```

6. For many reasons, we would like to design multipliers that require less time. Many different approaches, have been taken to accomplish this goal. In the following table, A represents the bit width of an integer, and B represents the number of time units (tu) taken to perform a step of an operation. **[2.5x4 = 10 points]**

	A (bit width)	B (time units)
a.	4	3 tu
b.	32	7 tu

Calculate the time necessary to perform a multiply using the approach shown below if an integer is A bits wide and each step of the operation takes B time units.
Assume that in step 1a. an addition is always performed - either the multiplicand will be added, or a 0 will be. Also assume that the registers have already been initialized (you are just counting how long it takes to do the multiplication loop itself). If this is being done in hardware, the shifts of the multiplicand and the multiplier can be done simultaneously. If this is being done in software, they will have to be done one after the other. Solve for each case.





Hardware:

36 tu

672 tu

Software:

48 tu

896 tu