# NATIONAL TSING HUA UNIVERSITY
## DEPARTMENT OF COMPUTER SCIENCE
## CS 4100: Computer Architecture
## Spring 2020, Mid-term Examination

1. (20%) Consider a program with the following instruction counts on EACH CORE when running on a 1-core computer and a 4-core computer, respectively. Note that on the 4-core computer, the 4 cores run the program in parallel.

| Instruction count | Arithmetic | Load/store | Branch |
|---|---|---|---|
| 1-core | $3 \times 10^9$ | $3 \times 10^9$ | $2 \times 10^9$ |
| Each core of 4-core | $1 \times 10^9$ | $1 \times 10^9$ | $2 \times 10^9$ |

Assume that each core has a 3 GHz clock frequency and has CPIs of 2, 6, and 4 for arithmetic, load/store, and branch instructions, respectively.

(a) (10%) Find the total execution time for this program running on the 1-core computer and 4-core computer, respectively. Calculate the speedup due to the use of 4 cores.

(b) (10%) You can get the same result by using the Amdahl's law.

$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where $S_{latency}$ is the speedup of the execution of the whole program, *s* is the speedup of the part of the program that benefits from the improved system resources, and *p* is the proportion of the execution time that the part benefiting from improved resources originally occupied. What is the *p* for this program? What is the *s* for this program on the 4-core system? (Note: If you think *s* = 4, then think twice. Use the execution time of arithmetic and load/store instructions to calculate.) What is the speedup due to the use of 4 cores?

2. (5%) For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables i, and j are assigned to registers x28, and x29, respectively. Assume that the base addresses of the arrays A and B are in registers x10 and x11, respectively. Assume that each element of A or B is 8 bytes

B[100] = A[i−j];

3. (30%) Consider the RISC-V assembly code (shown below right) for the fib() shown below.

```
int fib(int n) {
   if (n == 0)
      return 0;
   else if (n == 1 || n == 2)
      return 1;
   else
      return 2*fib(n-1)+4*fib(n-2);
}
```

(a) (6%) Use ABI names to complete lines 1 and 2 in the assembly code that correspond to the C code:
if (n==0) return 0;

(b) (12%) Use ABI names to complete lines 9 and 10 in the assembly code. Why do we want to save their values? We do this as the caller or

```
1  fib: _____
2       _____
3       addi    a1,zero,1    #set return as 1
4       addi    t0,zero,1
5       beq     a2,t0,Done   #if(n==1) return 1
6       addi    t0,zero,2
7       beq     a2,t0,Done   #if(n==2) return 1
8       addi    sp,sp,-16    #space on stack
9       _____                #save return addr
10      _____                #save argument
11      addi    a2,a2,-1     #n=n-1
12      jal     ra,fib       #fib(n-1)
13      ld      a2,0(sp)     #load old n
14      sd      a1,0(sp)     #store new n
15      addi    a2,a2,-2     #n=n-2
16      jal     ra,fib       #fib(n-2)
17      ld      t0,0(sp)     #load old n
```

callee? Why?

(c) (6%) What is the binary representation of line 7?

(d) (6%) We can replace line 12 with the following sequence of code:

```
auipc t2,0  #t2←PC
addi  t2,t2,_____
jalr  ra,0(t2)
```

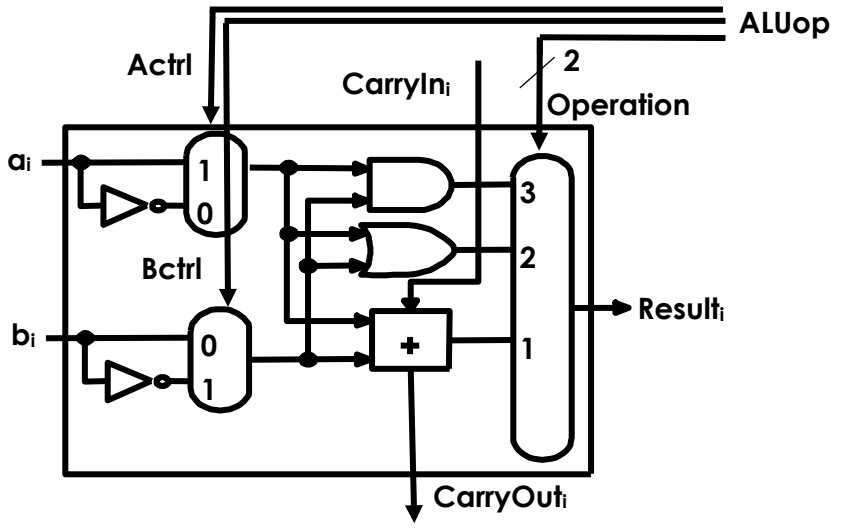
What should be put in the blank? (Be careful of the PC value!)

| | | | |
|---|---|---|---|
| 18 | slli | a1,a1,2 | #4*fib(n-2) |
| 19 | slli | t0,t0,1 | #2*fib(n-1) |
| 20 | add | a1,a1,t0 | #4*fib(n-2)+2*fib(n-1) |
| 21 | ld | ra,8(sp) | #return address |
| 22 | addi | sp,sp,16 | #pop stack |
| 23 | Done:jalr | zero, 0(ra) | #return |

4. (4%) Both **`beq x0,x0,Loop`** and **`jal x0,Loop`** implement unconditional jump. What are the differences between the two?

5. (6%) Suppose registers x1, x2, and x3 of a RISC-V processor contain 0x000000005F82C95F, 0x900FFFFFFFF000FF, and 0x35ED293045BCE0F4, respectively. What will be the result in x1 for each of the following instructions? Explain your answers.
   (a) **`slt  x1,x2,x3`**
   (b) **`srai  x1,x2,4`**
   (c) **`srli  x1,x2,4`**

6. (8%) Consider an ALU which has two $n$-bit data inputs A = $a_{n-1}a_{n-2}...a_0$, B = $b_{n-1}b_{n-2}...b_0$, and one 4-bit control input ALUop. By ignoring the overflow detection, each bit slice of the ALU is shown below, where $0 \le i \le n-1$. The ALU supports two's complement addition/subtraction by connecting $CarryIn_0$ to Bctrl.




Give the 4-bit value of ALUop (from left to right: Actrl, Bctrl, Operation) for each of the following operations:

A + B

A – B

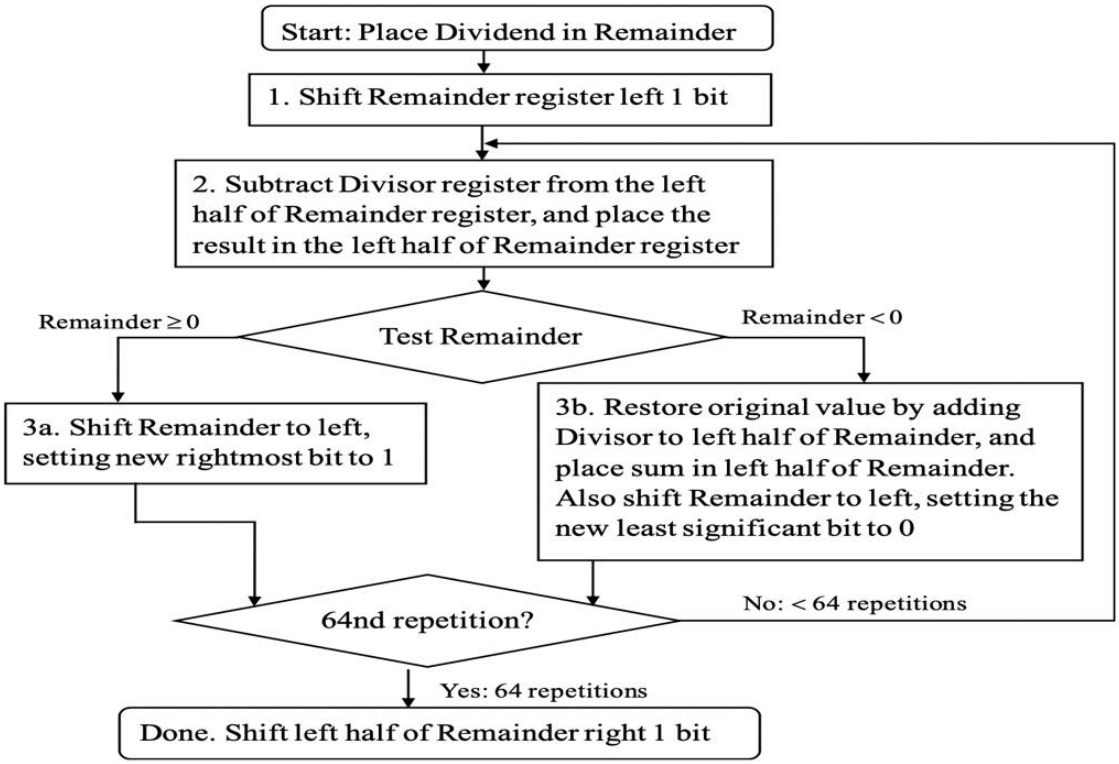
A nor B

A or B

(Note: Be careful of the data-input ordering of each multiplexer.)

7. (7%) The following divide algorithm (version 2) was introduced in class for performing 64-

bit division on positive integers. Explain how to revise the algorithm such that the last step "Done. Shift left half of Remainder right 1 bit" can be removed without losing the correctness. To simplify the writing of your answer, you only need to describe what change or no change is made for each of steps 1, 2, 3a, and 3b. (Hint: Move step 1 to be inside the loop.)




8. (15%) Consider the IEEE 754 floating-point standard and its arithmetic operations.
   (a) (6%) Represent the decimal numbers 4.75 and -10.375 in the IEEE 754 single precision format.
   (b) (6%) Calculate 4.75 + (-10.375). The result should be derived based on the IEEE 754 single precision format.
   (c) (3%) Suppose the step size of a number $x$ is defined to be a positive number denoting the difference between $x$ and the smallest number that is larger than $x$ and can be represented by the IEEE 754 single precision format. Now consider two decimal numbers 2 and 4. Which one has a larger step size? Why?

9. (5%) Assume IEEE decided to add an 8-bit floating point representation whose main characteristics are consistent with the 32-bit and 64-bit representations. Consider the following four 8-bit floating point numbers:

   A: 11100110
   B: 00111010
   C: 00001100
   D: 00011101

   -11=-1.011^2^3
   1.1101*2^-2

   They represent the following four decimal numbers <u>in no particular order</u>:

   $$1\frac{5}{8},\ -11,\ \frac{29}{64},\ \frac{3}{16}$$

   (a) (3%) Which one among A, B, C, and D represents $\frac{29}{64}$? Why?

(b) (2%) What is the smallest positive number that can be represented by this 8-bit floating point representation?