CSE4010-01 Computer Architecture and Logic Midterm

- **Exam instructions**

Answer each of the questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit. The examination is closed-book, and you may use a calculator. You cannot use any other electronic devices including your cellular phone. You will have 75 minutes to complete this exam. Budget your time and try to leave some time at the end to go over your work. You can handout your exam and leave the classroom if you complete the exam before the due time.

Name: _____

ID: _____

		Score
1. General questions	**(10)**	**_____**
2. Amdahl's law	**(10)**	**_____**
3. Pseudo-instructions	**(15)**	**_____**
4. MIPS Assembly	**(15)**	**_____**
5. Multiplication	**(15)**	**_____**
Total	**(65)**	**_____**

1. Yes/No questions [10 points]

(a) Code with the larger static instruction count can have smaller execution time than the one with the smaller instruction count. ( )

(b) MIPS(million instructions per second) is a good performance metric if two machines have the same clock frequency. ( )

(c) ALU instructions in MIPS can have both immediate and memory operands. ( )

(d) The target address of MIPS j instruction is always larger than the current PC. ( )

(e) Static power consumption depends on the workload of CPU. ( )

(f) PC is a special register in MIPS. ( )

(g) Auto variables in a function are allocated in the stack area of the memory. ( )

(h) $a0 register in MIPS is always saved in the stack area when it is used as passing an argument of a function. ( )

(i) Pseudo-instructions in MIPS may be assembled into more than one instruction. ( )

(j) Linking always happens before loading a program. ( )

1. Amdahl's law [10 points]

A program spends 60% of its execution time doing floating-point arithmetic. Of the floating-point operations in this program, 80% are executed in parallelizable loops. (For answers, write your numbers down to hundredth places after the decimals point (e.g. 1.33). [소수점 둘째 자리])

(a) **[3 points]** Find the improvement in terms of speedup if new the floating-point hardware is made three times as fast.

(b) **[3 points]** Find the improvement in terms of speedup if we use six processors to run the program's parallelizable loops six times as fast. (In this problem, you do not use the new fast floating-point hardware.)

(c) **[4 points]** Find the improvement in terms of speedup resulting from both modifications (a) and (b)

2. Pseudo-instructions [15 points]

You want to create new pseudo-instructions for easy programming.

(a) **[8 points]** You decide to extend the MIPS assembler to provide support for a new rori pseudo-instruction that performs a rotate right logical. Unlike a standard shift right logical operation, the most significant bits of the resulting value come from the least significant bits of the source register. The instruction would look like:

rori $t0, $t1, 3 # rotate right $t1 by 3 into $t0

to indicate that the value in $t1 should be rotated logically right by 3 places and stored into $t0. A sample input and output is shown below:

$t1 0000000000000**11**00000000000000**111**

$t0 **111**0000000000000**11**00000000000000

Write the MIPS instructions into which the assembler might convert the above pseudo-instruction. Use the minimum number of MIPS instructions. (Hint: shift instruction has the following format: sll, register1, register2, immediate)

(b) **[7 points]** You also decide to extend the MIPS assembler to support a method of memory addressing known as memory indirect by way of another pseudo-instruction. In memory indirect addressing, the memory address is dereferenced, as if following a pointer. The instruction load memory indirect (lmi) would look like:

lmi $t0, 8($t1) # $t0 ← Mem[Mem[$t1+8]]

In above example, 8+$t1 is the address of the memory location you want to read.

Write the MIPS instructions that the assembler might convert the above pseudo-instruction. Use the minimum number of MIPS instructions.

3. MIPS assembly [15 points]

Please consider the following C and assembly code. The following codes are the C and assembly code for Fibonacci number generator.

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

```
fib:    addi $sp,$sp, -12       # make room on stack
        [          ]            #
        sw $s0, 4($sp)          # push $s0
        sw $a0, 0($sp)          # push $a0 (N)
        bgt $a0,$0, test        # if n>0, test if n=1
        add $v0, $0, $0         # else fib(0) = 0
        [          ]            #
test:   addi $t0, $0, 1         #
        bne $t0, $a0, gen       # if n>1, gen
        add $v0, $0, $t0        # else fib(1) = 1
        j rtn
gen:    addi $a0,$a0,-1         # n-1
        jal fib                 # call fib(n-1)
        add $s0,$v0, $0         # copy fib(n-1)
        addi $a0,$a0,-1         # n-2
        [          ]            #
        add $v0, $v0, $s0       # fib(n-1)+fib(n-2)
rtn:    lw $a0, 0($sp)          # pop $a0
```

```
lw $s0, 4($sp)          # pop $s0
lw $ra, 8($sp)          # pop $ra
                        # restore sp
jr $ra
```

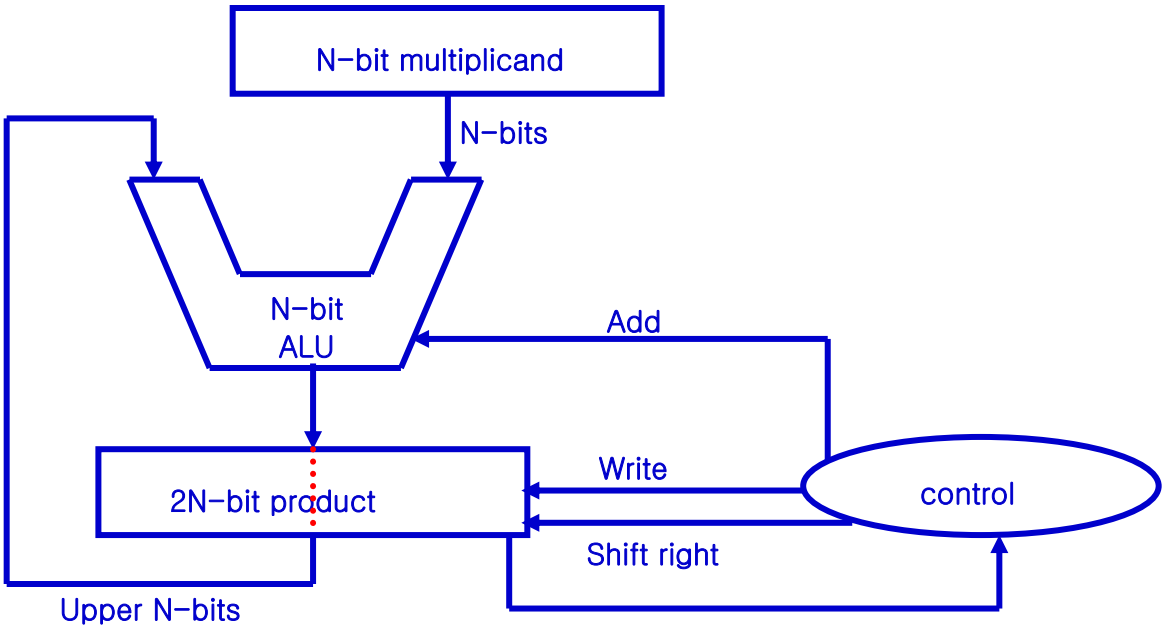
Note that bgt is a pseudo-instruction performing branch if one number is greater than the other.

(a) [8 points] fill in the blanks with necessary codes.

(b) [5 points] The address of the label ‘fib’ 0x00001000. Please write down the bit encoding of the immediate operand (branch offset) for ‘bne $t0, $a0, gen’ in hex decimal number.

(c) [7 points] The address of the label ‘fib’ 0x00001000. Please write down the all possible value for $ra while the code is being executed. (Please exclude the return address of the original caller function of fib.)

4. Multiplication [15 points]



(a) **[5 points]** Determine the minimum number N that can compute two number multiplication for 5*-9.

(b) **[10 points]** Show how the above multiplier computes 5*-9. Show the contents of 2N-bit product register each cycle using the number you found for N in (a). Each cycle consists of an addition step and a shift step. Show only the content of the register after the shift is completed. Assume the multiplicand is -9 and the multiplier is 5. Cycle 0 is initial state. Fill as many blanks as necessary.

	Content of product register
Cycle 0 (initial state)	
Cycle 1	
Cycle 2	
Cycle 3	
Cycle 4	
Cycle 5	
Cycle 6	