Indian Institute of Technology Kharagpur

Department of Computer Science and Engineering

Quiz-1, Autumn 2019-20

Computer Organization and Architecture (CS31007)

Students: 108 **Date:** 13-August-2019

Full marks: 25 **Time:** 75 minutes

Name	Roll No.

INSTRUCTIONS: This quiz is open book and open notes, but access to the internet via use of smartphones/portable computers is not allowed. Answer in the space provided only. Perform all rough work in blank sheets provided, your rough work will not be evaluated. Use of calculators is allowed. ANSWER ALL QUESTIONS.

1. Consider the following MIPS code running on machine M:

```
lw $t1, 1000($t2)
lw $t2, 1000($t2)
addu $t2, $t2, $t2
Loop:  addu $t1, $t1, $t1
beq $t1, $t2, Loop
sw $t2, 1000($t3)
```

Assume M has a clock frequency is 1 GHz. Also, assume that `lw` needs 5 clock cycles, `sw` 4 clock cycles, `addu` 3 clock cycles, and `beq` needs 2 clock cycles. Calculate the total amount of CPU-time required to execute the code. [5]

The program executes the loop only once. Hence, execution of the following instructions proceeds as stated below:
lw – 2 times = $2 \times 5 = 10$ clock cycles; sw – 1 time = 4 clock cycles
addu – 3 times = $3 \times 3 = 9$ clock cycles; beq – 2 times = $2 \times 2 = 4$ clock cycles
CC = $10+4+9+4 = 27$ clock cycles; $1\ CC = \frac{1}{10^9} = 1\ ns$. Hence
CPU-time = 27 ns.

2. Consider the following MIPS code:

```
lui  $ t1, 0x7FFF
ori $t1, $t1, 0xFFFF
addu $t1, $t1, $t1
sll $t1,$t1, 2
addi $t1, $t1, 9
```

The content of the register `$t1` after execution of the above code is (choose one): (i) 0, (ii) 1, (iii) −1, (iv) a number causing overflow, (v) none of these. Justify your argument. [5]

After execution of first two instructions, the content of t1 is 0x7FFF FFFF; following addu and sll, the content of t1 becomes −8. Hence, after execution of addi, the content of t1 will become +1; hence, the choice is (ii), i.e, +1.

3. Study the following recursive C function to calculate and return the value of an argument incremented by one, and write an equivalent recursive MIPS function. Note that the function can handle both positive and negative integers as argument. You are allowed to use pseudoinstructions. NO CREDIT WILL BE GIVEN FOR A NON-RECURSIVE IMPLEMENTATION. [10]

```
/* The following function returns y+1 */
int increment (int y)
{
   if (y == 0) return 1;
   else if (y % 2 == 1) return (2 * increment(y/2));
   else return (y+1);
}
```

4. Two enhancements with the following speedup factors, and fraction of usage in a benchmark program are proposed for designing a new architecture. If only one enhancement has to be chosen, which one do you implement for maximizing performance? Use *Amdahl's Law* to justify your answer. [5]

Table 1: Enhancement Characteristics

Enhancement Type	Speedup Factor	Usage
Enhancement-1	20	15%
Enhancement-2	10	70%

For enhancement 1,

$$\text{Speed-up} = \frac{1}{(1-0.15)+\frac{.15}{20}} = \frac{1}{.85+.0075} = \frac{1}{.8575} = 1.166$$

For enhancement 2,

$$\text{Speed-up} = \frac{1}{(1-0.70)+\frac{.70}{10}} = \frac{1}{0.30+.07} = \frac{1}{0.37} \approx 2.70$$

Hence, enhancement 2 is to be choosen.

```
####################### Data segment ############################
        .data
msg_input:   .asciiz "Enter the argument: "
msg_arg:   .asciiz "The argument is: "
msg_result:   .asciiz "The incremented value is: "
newline:   .asciiz "\n"
####################### Data segment ###########################

####################### Text segment ###############################
.text
.globl main
     main:
          la $a0, msg_input # message string in $a0
          li $v0, 4 # Prepare to print the message
          syscall  # print the message

          li $v0, 5 # for read_int
          syscall # argument in $v0
          move $a0, $v0 # argument in $a0

          # Print argument to make sure....debug step
          move $t0, $a0 # register $t0 contains the argument for now
          li  $v0, 4 # for print_str
          la  $a0, msg_arg  # preparing to print the message
          syscall  # print the string
          li  $v0, 1 # for print_int
          move $a0, $t0 # get argument back in $a0
          syscall  # print the argument

          jal function_increment # call the function

          # Have returned from the function
          move $t0, $v0 # copy result in $t0 temporarily

          # Print a newline
          li  $v0, 4 # for print_str
          la  $a0, newline # preparing to print the newline
          syscall  # print the newline

          # Print result
          li  $v0, 4 # for print_str
          la  $a0, msg_result  # preparing to print the message
          syscall  # print the string


       move $a0, $t0 # get result in $a0
          li  $v0, 1 # for print_int
          syscall  # print the result
     Exit:
          li $v0, 10
          syscall # exit


      # Start of recursive function
      function_increment:
          addi $sp, $sp, -8 # adjust stack pointer
          sw   $ra, 4($sp) # save return address
          sw   $a0, 0($sp) # save argument
          li   $v0,  1  # Initialize return value (pseudoinstruction)

          bne  $a0, $zero, L1 # If argument is non-zero then continue
```

```
      # Return if argument is zero
      # $v0 already contains the required value, i.e. 1
       return_if_zero_arg:
          j return

       L1:
          # Argument is non-zero
          # Prepare mask to check LSB
          # $t0 used as mask
          lui $t0, 0
          ori $t0, 1 #$t0 now contains 0x00000001
          and $t1, $t0, $a0 # $t1 <--- $t0 & $a0
          # beq succeds if $a0 is even
          beq  $t1, $zero, even_arg # branch to handle even case

          # The following two instructions handle when $a0 is odd
          div  $a0, $a0, 2 # $a0 <--- $a0 / 2 (pseudoinstruction)
          jal  function_increment # recursive function call

          # Have returned from function
          mul  $v0, $v0, 2 # modify $v0 (pseudoinstruction)
          lw   $ra, 4($sp) # restore return address
          j return


      # The next instruction is for even argument
      even_arg:
        addi  $v0, $a0, 1 # add with current argument

      return:
          addi $sp, $sp, 8 # restore stack pointer
          jr   $ra         # return to caller

####################### Text segment ##############################
```