Indian Institute of Technology Kharagpur

Department of Computer Science and Engineering

Class Test-2, Autumn 2020-21

Computer Organization and Architecture (CS31007)

Students: 118	**Date**: 03-October-2020
Full marks: 30	**Time**: 75 minutes
Credit: 20%	

INSTRUCTIONS: This is an OPEN-BOOK, OPEN-NOTES test. The questions are such that they require either numerical answers or very short answers (in one/two sentences or a few lines of code). Please write ONLY THE ANSWERS on a sheet of paper, scan it (or take a photo), and submit the image of the solution sheet on Moodle. DO NOT FORGET TO WRITE YOUR NAME AND ROLL NUMBER AT THE TOP OF YOUR ANSWER SHEET. You may use calculators if required. ANSWER ALL QUESTIONS.

1. Consider the following MIPS code segment being executed 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 2.5 GHz. Also, assume that `lw` needs 5 clock cycles, `sw` 4 clock cycles, `addu` 3 clock cycles, and `beq` needs 2 clock cycles to execute. Calculate the total amount of CPU-time (in nanoseconds) required to execute the above code. [**5**]

2. Consider the following MIPS code segment:

```
lw $t1, 1000($t2)
lw $t3, 1200($t2)
add $ t2, $t1, $t3
slt $t1, $t2, $t3
addi $t2, $t2, -1000
sw $t2, 1000($t3)
```

Calculate the total number of data transferred (in bytes) between CPU and the main memory (both directions combined). [**5**]

3. Consider the following MIPS code segment:

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

What is the content of the register `$t1` after execution of the above code? [**5**]

4. Write a MIPS code segment to initialize the `$t1` register to the value 75000_{10}. You should use maximum two MIPS instructions, and you are not allowed to use any pseudoinstruction. [**5**]

5. Suppose, the register `$t1` has been initialized to a 32-bit 2's complement value. Now, the following MIPS code is executed:

```
move $t2, $t1 #pseudoinstruction, to copy the contents of $t1 into $t2
bge $t1, $zero, Exit #pseudoinstruction, branch to label Exit if $t1 >= 0
not $t2, $t2 # pseudoinstruction for bitwise negation
addi $t2, $t2, 1
Exit: ...
```

What is the relationship between the values in registers `$t1` and `$t2` after this code segment is executed? [**4**]

6. Consider the MIPS procedure `function_increment` given below, to recursively compute and return in register `$v0`, the incremented (by 1) value of an integer argument which is present in register `$a0`. Write the body of the equivalent C function `int function_increment (int x)` corresponding to the given MIPS procedure. [**6**]

```
###################################################################################
# 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
###################################################################################
```