Queen Mary
University of London

Coursework (15% contribution) 2020-21(Semester 1)

Released: Monday 23rd November
Submission Deadline: Friday 4th December, 10:00 AM (UK Time)

ECS404U: COMPUTER SYSTEMS AND NETWORKS

► Late submissions will incur penalty. No other means of submission (like email attachments) is acceptable. Allow enough time for submission as servers may be busy right at the deadline. You can only submit once: so be careful to submit the correct document to the correct link.
► You can refer to any textbooks, notes and online materials, but you **must** cite your sources, irrespective of their nature, except if you are using the lecture slides, course-notes, or the lab manuals of the module.

► **You MUST complete the exam individually on your own, without consulting others. No collaboration is permitted. There is zero-tolerance policy on plagiarism.**

► You **must** type your answers, **then create a single PDF file**. For this submission, **no handwritten answers are accepted** except with prior permission and with strong justification.

► In your submission, start answering each of the 4 questions on a new page (but continuing the parts of the same question on the same page is fine).
► In your submission, you **MUST NOT** repeat the statement of the question. Just clearly state the question number (e.g. Q1-a-i) and then only write down your answer. Similarly, you **MUST NOT** include this cover page in your answer file. Including the question statements or the cover page can lead to a high `Turnitin` similarity score.
► Try to write down your answers sequentially and well-marked (e.g. Q1-a-i), even if you wish to leave a question blank. Otherwise, you risk losing marks (e.g. if the grader cannot associate an answer to its question).

Question:	1	2	3	Total
Points:	30	40	30	100
Score:				

Examiners: Prof Edmund Robinson and Dr Arman Khouzani

Question 1

A programme in 32-bit MIPS Assembler is shown in the following:

```
.data
msg1: .asciiz "\nEnter the first integer: "
msg2: .asciiz "Enter the second integer: "
msg3: .asciiz "Result: "

.text
li $v0, 4 #syscall to print a string
la $a0, msg1
syscall

li $v0, 5 #syscall to read an integer
syscall
add $t1, $zero, $v0

li $v0, 4 #syscall to print a string
la $a0, msg2
syscall

li $v0, 5 #syscall to read an integer
syscall
add $t2, $zero, $v0

add $t0, $zero, $zero

LOOP:
slt $t3, $t1, $t2
bne $t3, $zero, DONE
sub $t1, $t1, $t2
addi $t0, $t0, 1
j LOOP

DONE:
li $v0, 4 #syscall to print a string
la $a0, msg3
syscall

li $v0, 1 #syscall to print an integer
add $a0, $t0, $zero
syscall

li $v0, 10     # syscall code to exit
syscall
```

(a) Suppose when prompted at the console, **we input numbers 27 and 4** . Provide the trace of the values in registers **`$t0`**, **`$t1`** and **`$t3`** (as a triple) when this code is executed until

it finishes. That is, write down the new values in all three of `$t0`, `$t1` and `$t3` each time either one of them changes as you step through the programme until the programme finishes execution. You can choose to present the values either in decimal, hex, or binary, as long as you clearly specify which. Your answer should have a correct sequence of changes. Suppose that initially, the value in these registers is **unknown** to us (which you can show by a question mark **?**). Provide a very brief explanation along with your trace table.

[15 marks]

(b) Explain what happens if one of the integers (or both of them) entered by the user is negative.

[5 marks]

(c) A close investigation of this code should reveal that this programme takes two positive integers from the user and computes the quotient (i.e., the integer part of division). Modify the programme such that both quotient and the remainder are printed.

[10 marks]

Question 2

MIPS, memory access and Arrays

In all of the questions of this part, you are still only allowed to use **`beq`** or **`bne`** for branching, and **`slt`** for comparison. Also, try to reuse the given code as much as possible. In particular, try to keep the name of the variables and labels the same. For instance, use register **`$s0`** to hold the value of the requested quantity.

(a) The following code computes and prints the sum of the entries in the array *A*:

```
.text
    la   $t0, A_LENGTH
    lw   $t0, 0($t0)              # t0 <- A_LENGTH
    la   $t1, A                   # t1: to hold the "address" of the next
                                  # array element, initialised to the
                                  # address of the first byte of the array
    addi $s0, $zero, 0            # s0: will hold the total sum,
                                  # initialised to zero

    NEXT_ARRAY_ELEMENT:

    slt $t3, $zero, $t0             # t3<-(0<t0), t3 will be 0 if t0<=0
    beq $t3, $zero, DONE

    lw $t2, 0($t1)                # t2 <- the current array element
    add $s0, $s0, $t2             # s0 <- s0 + t2

    addiu $t1, $t1, 4             # t1<-t1+4, to get the address of
                                  # the the next element
    addiu $t0, $t0, -1            # decrementing t0 by 1


    j NEXT_ARRAY_ELEMENT          # jump to NEXT_ARRAY_ELEMENT (for loop)

    DONE:
    addi $v0, $zero, 1          # set v0 to "1" to select
                                # "print integer" syscall
    add  $a0, $zero, $s0        # a0 <- s0 (the total sum) to be printed
    syscall                     # invoking the syscall to print the integer

    addi $v0, $zero, 10         # set v0 to "10" to select exit syscall
    syscall                     # invoking the syscall to actually exit!

.data
    A:                           # our integer array
        .word   -1
        .word   4
        .word   -16
        .word   0
        .word   -2
```

```
41          .word   5
42          .word   13
43          .word   2
44      A_LENGTH:   .word   8          # the length of the array
```

Codes/array_sum.asm

Modify the code so that it prints the maximum of the elements (the largest element). So say for the array in our example code, it should return 13.

[10 marks]

(b) Modify the code so that it prints the maximum absolute value among the elements (the largest element in terms of just magnitude without considering sign). For instance, it should return 16.

[10 marks]

(c) (i) Suppose there is an integer in register **`$t1`**. In one sentence, describe (along with a brief explanation) how the result of the following instruction can be used to determine whether the number is divisible by 8 or not.

```
1    andi $t0, $t1, 0x0007
```

Incidentally, this is a simple example of a technique called “masking”.

[10 marks]

(ii) Now, using this observation, modify the code in part (a), to print how many of the numbers in the array are divisible by 8. So in our example, for instance, it should print 2, since both -16 and 0 are perfect multiples of 8.

[10 marks]

Question 3

Simple arithmetic with bit shifting

(a) **`sll`** (**shift left logical**) is a MIPS instruction with the following description:

syntax : **`sll $rd, $rt, h`**
operation : Shifts the value in register **`rt`** left by the shift amount **`h`** (zeroes are shifted in for the new **`h`** rightmost bits) and places the result in register **`rd`**.

Suppose that **`$rt`** is a register that contains an integer (for the purposes of this question, you can take this to be unsigned, but it could also be signed). Describe the relation of the integer in **`$rd`** with respect to the integer **`$rt`** (in terms of *h*) after execution of instruction `sll $rd, $rt, h`. Support your answer with a brief explanation.

[10 marks]

(b) Provide the 32-bit machine code of the instruction **`sll $t0, $t1, 2`**. Make sure to deconstruct and specify each component.

(c) The following examples show how to compute multiplications with some (small) constants using shifts and additions. Provide a short sequence of MIPS instructions that will implement the multiplication operation given. Note that you are allowed to use the **`add`**, **`sub`** and **`sll`** instructions, however, you must at most use as many instructions as requested (and of course, you are not allowed to use any "multiply" instructions.)

(i) **`$t1`** ← **`4`** × **`$t1`** (1 instruction)
solved example 1:
```
sll $t1, $t1, 2
```

(ii) **`$t1`** ← **`5`** × **`$t1`** (2 instructions)
solved example 2:
```
sll $t0, $t1, 2
add $t1, $t0, $t1
```

(iii) **`$t1`** ← **`8`** × **`$t1`** (1 instruction) **[2 marks]**

(iv) **`$t1`** ← **`24`** × **`$t1`** (3 instructions) **[3 marks]**

(v) **`$t1`** ← **`28`** × **`$t1`** (2 instructions) **[3 marks]**

(vi) **`$t1`** ← **`60`** × **`$t1`** (2 instructions) **[2 marks]**