# University of Algiers 1

Faculty of Sciences
Department of Computer Science

# Test L2-ADO

02-Dec-2025

Duration : 90 minutes

| | |
|---|---|
| Surname:<br>First name:<br>Section and Group:<br>Student ID No: | / 20 |

## Exercise 1: (10 points)

Complete the missing information in the table below. If a field doesn't exist in the target instruction, put not applicable (N/A). (4 points, missing details = no points)

| Instruction Type | Opcode (hex) | Function (hex) | Assembly Instruction | Register Fields (rs, rt, rd) / Immediate / Target | Machine Code (Hexadecimal) |
|---|---|---|---|---|---|
| R-Type – 0.25pt | 00 | 23 | subu $t0, $s1, $s2 –0.5pt | rs: $s1, rt: $s2, rd: $t0 | 0x02324023 –1pt |
| I-Type – 0.25pt | 0D – 0.25pt | N/A – 0.25pt | ori $t0, $s0, 0x3247 –0.5pt | rs: $s0, rt: $t0, immediate: 0x3247 –1pt | 0x36083247 |

The MIPS slt (set on less than) instruction sets a register to 1 if one value is less than another; otherwise, it sets it to 0. Using slt with combinations of branch instructions and the zero register $0 lets you implement all common comparisons. Assume i is at $s1 and j is at register $s2 and destination register is $t0 (i.e. for slt). The MIPS assembly code for the (i<j) case is provided for you along with its explanation; apply the necessary modifications on the same code to get the (i>j) and (i≥j) cases. (4 points, missing details = no points)

| Condition | Assembly instructions | explanation |
|---|---|---|
| If (i<j)<br>i++; | slt $t0, $s1, $s2<br>beq $t0, $0, exit<br>addi $s1, $s1, 1<br>exit: | When **i < j**, after we check with slt, $t0 = 1, so the branch is **not taken** and the loop continues.<br>When **i ≥ j**, $t0 = 0, so the branch **is taken**, and the loop exits. |
| If (i>j)<br>i++; | slt $t0, $s2, $s1<br>beq $t0, $0, exit<br>addi $s1, $s1, 1<br>exit: | When j **< i (i>j)**, after we check with slt, $t0 = 1, so the branch is **not taken** and the loop continues to i++.<br>When **j≥i (else case)** , $t0 = 0, so the branch **is taken**, and the loop exits without i++. |
| If (i≥j)<br>i++; | slt $t0, $s1, $s2<br>bne $t0, $0, exit<br>addi $s1, $s1, 1<br>exit: | When i **< j (j>i)**, after we check with slt, $t0 = 1, so the branch is **taken** and the loop exits.<br>When i**≥j** , $t0 = 0, so the branch **is not taken**, and the loop continues. |

Consider the following code:

```
lbu $t0, 1($t1)
sw  $t0, 0($t2)
```

Let register **$t1** contain the address 0x1000 0000 and register **$t2** contain the address 0x1000 0010. Assume as well, a big-endian addressing scheme and the data at memory address 0x1000 0000 is 0x11223344. What value is stored at the address pointed to by register **$t2**.

**(2pt. missing details = no mark)**

Big Endian memory layout means the most significant byte is stored at the smallest address.
Memory at 0x10000000 stores the word 0x11223344:

- Address 0x10000000: 0x11 (most significant byte)
- Address 0x10000001: 0x22
- Address 0x10000002: 0x33
- Address 0x10000003: 0x44 (least significant byte)

Step 1: lbu $t0, 1($t1) loads 1 byte unsigned from 0x10000000 + 1 = 0x10000001.

- Loads byte 0x22 into $t0 (upper 24 bits zeroed since it's unsigned), So, $t0 = 0x00000022. –1pt

Step 2: sw $t0, 0($t2) stores the entire 32-bit word from $t0 into memory at 0x10000010.
So memory at 0x10000010 to 0x10000013 will be updated with bytes of 0x00000022. –1pt

## Exercise 2: (10 points)

Given the MIPS assembly code below:

```
.data
array:      .word 1, 5, 3, 7, 2, 4  # Assume at address 0x10010000
array_size: .word 6                 # 0x10010018 (6 words * 4 bytes after array start)

.text
.globl main

main:
0x00400000:  la $a0, array          # 1 load address of array into $a0
0x00400008:  lw $a1, array_size     # load array size (6) into $a1
0x0040000c:  jal process            # 2 jump and link to process, $ra will contain 0x00400010
0x00400010:  li $v0, 0              # $v0 = 0
0x00400014:  jr $ra                 # exit main program

process:
0x00400018:  li $t2, 0              # i = 0

loop:
0x0040001c:   beq $t2, $a1, end     # 3 if i == size, branch to end
0x00400020:   sll $t3, $t2, 2       # offset = i * 4 (word size)
0x00400024:   add $t4, $a0, $t3     # $t4 = base address + offset (array[i] address)
0x00400028:   lw  $s0, 0($t4)       # load array[i] value
0x0040002c:   li  $t6, 4            # load immediate 4
0x00400030:   bge $s0, $t6, skip    # if array[i] >= 4, skip doubling
0x00400038:   sll $s0, $s0, 1       # double the value
0x0040003c:   sw  $s0, 0($t4)       # store back doubled value
skip:
0x00400040:   addi $t2, $t2, 1      #  i++
0x00400044:   j loop                #  jump back to loop (0x0040001c)
end:
0x00400048:   jr $ra                # return to caller (main)
```

Answer the following questions (check the instruction comments for the numbers in the questions):

1. Convert the pseudo instruction in **1** to regular MIPS instructions and add a description comment

**(2pt. missing details = no mark)**

lui $at, 0x1001
ori $a0, $at, 0x0000

2. Why is there 8 bytes jump between the addresses of some successive instructions instead of the usual 4 bytes skip.

pseudo instructions will be converted into one or two instructions, hence the addresses shown. **(1pt. missing details = no mark)**

3. Give the machine code (in hexadecimals) for the instruction in **2** and add a description comment. What will be the value of **$ra** register after this instruction?

0x0C100006 –1pt **(2.5pt. missing details = no mark)**
Type: j, Opcode: 3, destination: bits 27-2 from process address –0.5pt / comment –0.5pt
$ra: 0x00400010 --0.5pt

4. Give the machine code for the instruction in **3.** And add a description comment. What is the addressing mode used in this instruction?

0x10AA000A –1pt : type I, opcode: 4, rs: $a1 ($5), rt: $t2 ($10), --0.5pt **(2.5pt. missing details = no mark)**
imm=offset= 10 (0x000A) →(0x00400048 – ( 0x0040001c + 0x4) ) /0x4 = 0xA –0.25pt , comment —0.25pt
Memory addressing mode: PC indexed –0.5pt

5. Correct the code by adding the necessary instructions (prologue/epilogue)

**before 0x0040000c: jal process --1pt** **(2pt. missing details = no mark)**
addi $sp,$sp,-4 #prologue
sw $ra, 0($sp) # if we think about main as a callee that called process and we need to keep $ra in mind
**after 0x0040000c: jal process**
lw $ra, 0($sp) #epiloque
addi $sp,$sp,4

**At the start of process: --1pt**
addi $sp,$sp,-4 #prologue
sw $s0, 0($sp)
**before 0x00400048: jr $ra**
lw $s0, 0($sp) #epiloque
addi $sp,$sp,4