Queen Mary
University of London

ECS404U: COMPUTER SYSTEMS & NETWORKS

2020/21 – Semester 1
Prof. Edmund Robinson, Dr. Arman Khouzani

Lab Week 10: *Grand Finale with (MIPS) Assembly array manipulation, comparison, branching*

November 23/25, 2020

Deadline for submitting your proof of work: ***Next week's Thursday, at 10:00 AM UK time***

Student Name:

Yujeong JUNG

Student ID:

200699145

Summary and Learning Objectives

This is the third and final lab on MIPS architecture and assembly. The learning objectives of today's lab are the following:

- Practising how to manipulate the contents of an array in the memory (noting that an array is just a group of data of the same type stored consecutively in the memory).
- Learn how to use the comparison instruction **`slt`** in combination with branching instructions **`beq`** and **`bne`** to achieve any complex program flow control;
- Combining all we have learned in these three labs to carry out some non-trivial tasks.

1 Array Manipulation

Recall that an array in MIPS assembly is nothing but a sequence of data of the same type stored sequentially in the memory; In this lab, we will use this fact to be able to “index” through an array to be able to access its elements and manipulate them.
Also recall from the previous lab the instructions to load data from the main memory into the (CPU’s) registers and store data back from registers to main memory:

Load word: `lw rt, offset(rs)`
This puts 4 bytes (a word) from the memory into register `rt`, where the address of *the starting byte* is `$rs + offset`, i.e., whatever address we have loaded in register `rs` plus whatever `offset` we provide as an immediate value (as a signed integer).

Store word: `sw rt, offset(rs)`
The content of register `rt` (which is 4 bytes, or a “word” long) is stored in 4 consecutive bytes of memory, where the address of its *starting byte* is `$rs + offset`. Keep in mind that this is the only one of our MIPS assembly instructions where the destination is written last.

Consider the following MIPS Assembly program (you can download it from QM+). The purpose of this program is to print the elements of an array to the console, separated by a “comma” character. *Note:* The code is intentionally wrong! Your first exercise is exactly to identify the problem in the code and rectify it:

```
    # All program code is placed after the
    # .text assembler directive

.text
    la   $s0, MY_INT_ARRAY_LEN
    lw   $s0, 0($s0)               # s0: MY_INT_ARRAY_LEN
    addi  $t0, $zero, 0            # t0: to hold the "index" of the next array entry
    la   $t1, MY_INT_ARRAY         # t1: to hold the "address" of the array entries
                                   #     initialised to the address of the first
                                   #     byte of the array

    NEXT_ARRAY_PRINT:
    beq $t0, $s0, DONE_PRINTING    # jump to DONE_PRINTING when reached array's end
    addi $v0, $zero, 1             # set v0 to invoke "print integer" syscall
    addu $a0, $zero, $t1           # a0 <- the integer to be printed
    syscall                        # invoking the syscall to print the integer

    addi  $v0, $zero, 4            # set v0 to invoke "print string" syscall
    la $a0, MSG_SPACER             # a0 <- the address of the string's starting byte
    syscall                        # invoking the syscall to print the string

    addi $t1, $t1, 4              # incrementing t1 by 4 (why?)
    addi $t0, $t0, 1              # incrementing t0 (the array index) by 1

    j NEXT_ARRAY_PRINT             # jump to NEXT_ARRAY_PRINT (our for loop)

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

    # All memory structures are placed after the
    # .data assembler directive
```

```
.data
    MSG_SPACER:   .asciiz ", "    # the comma (and space) string
    MY_INT_ARRAY:                 # our integer array
        .word    -1
        .word     4
        .word     5
        .word     0
        .word    -2
        .word    -5
        .word     3
        .word     1
    MY_INT_ARRAY_LEN:    .word    8 # the length of the array
```

./Codes/week10_array_example.asm

Q1. Identify the problem in the code and fix it.

- *Hint 1:* if it helps, run the programme in **QtSpim** (Recall: always use "`Reinitialize and Load File`"). What is being printed to the console? (Reminder: if you do not see the console, make sure it is ticked under "`Window`").
- *Hint 2:* The main problem is only in one line of instructions, so the main fix is to modify only that line.

Fix the code in your favourite text editor (**Atom** ?!), and then write down only the fix here.

Error: The code will output the address of all the integers in the array rather than the integers themselves.
Line 6: lw $t1, 0($t1)   # t1 : MY_INT_ARRAY_LEN

Q2. (*Optional*) Even after you "fix" the "bug" in the previous code, there is still something that can be improved: it prints one extra comma at the end (after the last integer in the array is printed). Fix the code so that the last extra comma is not printed.

- *Hint:* You can achieve this by modifying (re-arranging) the control flow of the programme. It helps to draw the flow-chart of the programme first.

Write here only the lines that you changed/added.

2 Comparison, more complex branching

A useful MIPS Assembly instruction that we have not covered yet is "**set on less than**" **`slt`**:

syntax : **`slt $rd`**, **`$rs`**, **`$rt`**

operation : compares the contents of registers **`rs`** and **`rt`**, if **`$rs`**<**`$rt`**, then set the content of register **`rd`** to one, otherwise (that is, if **`$rs`**≥**`$rt`**), set the content of register **`rd`** to zero. In short: **`rd`**←(**`$rs`**<**`$rt`**).

Note in particular that **`slt`** on its own is **not** a branching instruction. However, in combination with **`beq`** and **`bne`**, it can achieve complex branching rules, as we see next. Bear in mind that the values in the registers **`$rs`** and **`$rt`** are interpreted as signed (2's complement).

Q3. Suppose we have loaded our variables in registers **`s0`**, **`s1`** and **`s2`**. Write a MIPS Assembly code that prints "`checked!`" if the given condition is satisfied.

Assume you already have the following in your `.data` section:

```
.data
CHECKED: .asciiz "checked!"
```

So you should only provide the `.text` part in the provided box.

Comments in your codes are optional, but recommended.

Note: you are not supposed to use any instructions for branching other than **`beq`** or **`bne`**, and for comparison other than **`slt`** (so no "pseudo-instructions" allowed here!). You can try your code in **`QtSpim`** to ensure its correctness with some values of your own in the **`$s0, $s1, $s2`** registers.

(a) **$s0<$s1**

(b) **s0≤s1**

(c) **s0<s1<s2**

(d) **s1>s0 OR s1<s2**

3 Main Tasks

The following two questions require you to put a lot of what we have learned in the past three labs together.

Q4. Modify your (fixed) code in section 1 (as a new file, name it e.g. `week10_task1.asm`) that does the following: replaces each of the elements of the array in the main memory with triple its value. That is, implement the following:

```
for(int i = 0 ; i < MY_INT_ARRAY_LEN ; i++){
    MY_INT_ARRAY[i] = 3*MY_INT_ARRAY[i];
}
```

Note: print the final array (using the code that you previously developed) to ensure your code is working properly (using **`QtSpim`** , of course!).

Q5. Write another code that finds and prints the "minimum" (the smallest number) in a given array of integers. Again, you are not allowed to use any instructions that we have not covered. In particular, you should only use **`beq`** or **`bne`** for branching, and **`slt`** for comparison. As before, try your code in **`QtSpim`** to ascertain its correctness.