Lab 2: MIPS assembler and introduction to PCSpim

Objectives of this lab

The objective of this lab is to get familiar with the development of assembly programs and the representation of data types using a MIPS 32 architecture. For this lab we will use the PCSpim simulator that can be downloaded from its web page: `http://spimsimulator.sourceforge.net/`. You can employ whatever version of the simulator you prefer (Windows, Linux or Mac).

In Aula Global you will find the zip file *Exercises.zip*. This zip contains the materials to be used during the lab.

Exercise 1

Uncompress the file *exercise1.s* from the lab materials and answer the following questions:

1. Indicate the memory address where we can find the first main instruction.
2. Indicate the memory address that contains the register $ra after running the program. Indicate the instruction that contains the mentioned address.
3. The address 0×10010043 contains a character that belongs to the string declared in the data segment. Indicate the value of the mentioned address, and the corresponding ASCII character.
4. Indicate in what address is the second 'i' character of the string declared in the data segment.
5. Indicate the address obtained when using: $gap + 2$.
6. Indicate the address obtained when using: $string + 4$.
7. Write down the instructions to get the sixth byte of the declared string.

Exercise 2

In the lab materials you will find the file called *exercise2.s*. This file contains the code to run an unknown algorithm. Load, run the program, and answer the following questions:

- Analyze the code and indicate what is it doing. Indicate what is the printed value after running the execution.
- Indicate what are registers $t0 and $t1 used for. What is the difference between them? What is the reason for using both of them?
- Given the code below. Explain what the code does and why it is used.

```
li $v0, 9
li $t7, 4
mul $a0, $t7, $s0
syscall
move $s1, $v0
```

- Complete the value of the registers before the execution of the next instruction in the table below:

```
bgeu $t0, $s0, end
```

REGISTERS	Iteration 1	Iteration 2	⋯	Iteration n
R8 ($t0)				
R16 ($s0)				
R18 ($s2)				
PC				

Exercise 3

Using a file with name *exercise3.s* write a program to calculate the length of a string declared in the data setction. This function (typically known as strlen) counts the total number of characters in the string ignoring the end of string character. The function shall print the number of characters using the output console.

The string must be declared in the data section as follows:

```
.data
stringToLength:    .asciiz    "This is a test"
```

For the example above the program will print 14.

⚠**The program must work independentely of the declared string**.

Exercise 4

Using a file with name *exercise4.s* write a program that given a string converts all the characters to lowercase.

As in the previous exercise the string must be delcared in the data section as follows:

```
.data
stringToLower:    .asciiz    "THIS IS A TEST"
```

After running the program with the string example above, the console must print *this is a test*.

⚠**The program must work independentely of the declared string**.

Exercise 5

Write a program called *exercise5.s* that given an array of integer numbers returns the greatest number and its position in the array. The program will read the array declared in the data section, check which is the greatest number, its position and print both values.

The input array must be declared as indicated below:

```
.data
array:    .word    60, 20, 80, 40, 120
n:        .word    5
```

Where array is the array of integer values and n its length. For the example above the output must be 120 (greatest value) and 4 its position in the array.

⚠**The program must work independently of the given array and length**.

Exercise 6

Write a program called *exercise6.s* that indicates if one string is contained in other string. This program will read two strings: one indicating what to search and other indicating where to search. The program will print 1 if the searched string was found, and -1 otherwise.

The string must be declared as follows:

```
.data
baseString:    .asciiz    "All work and no play makes Jimmy a dull boy"
searchString:  .asciiz    " play makes Jimmy "
```

For the example above the program must print 1.

⚠**The program must work independently of the declared strings**.

Exercise 7

Write a program called *exercise7.s* that emulates the C function printf. In this case, we will declare a string indicating the positions to be replaced by a number. These numbers will be taken from an array declared in the data section.

The input parameters must be declared as follows:

```
.data
toPrint:    .asciiz   "It is known that %i percent of the work requires
                      %i percent of the time."
numbers:    .word     20, 80
```

The output for the example above will be *It is known that 20 percent of the work requires 80 percent of the time.* The program must replace the %i symbols by the numbers contained in the array in the order they are found in the array. Only integer numbers will be printed. The % symbol will always be followed by i. The size of the array and the number of replacements will always be the same.

⚠**The program must work independently of the string and the number of elements declared in the data section**.

Evaluation

This lab will be evaluated in three steps:

1. **Presential part (2 points)**. This submission will be done during the lab session. Only exercise 1 must be submitted in this part.
2. **Mandatory part (4 points)**. This part includes exercises 2, 3, 4 and 5 and the final document. The exercises are graded as follows:
   (a) Exercise 2 (0.75 points)
   (b) Exercise 3 (0.75 points)
   (c) Exercise 4 (0.75 points)
   (d) Exercise 5 (0.75 points)
   (e) Final document (1 point)
3. **Optional part (2 points)**. Exercise 6 is optional and will be grades as follows:
   (a) Exercise 6 (1.75 points)
   (b) Explanations in the final document (0.25 points)

Submission

The submission will be as follows:

- **Exercise 1** will be submitted through aula global during the lab session. JUST ONE ZIPPED file with name **ec_p2_e1_A...A_B...B.pdf** with A...A and B...B the student identifiers of the group members. This file must contain ALL the answers for the questions formulated in the exercise 1.
- The remainder exercises and questions will be submitted a zip file with name **ec_p2_e2_A...A_B...B.zip** before **November 11**th at 23:55 through Aula Global. This zipped file must contain the assembly source code (*exercise3.s*, *exercise4.s*, *exercise5.s*, *exercise6.s* and *exercise7.s*) and the Document.pdf file with the answers for the formulated questions, the explanation of your code with the explanations you may consider useful (flow diagrams, algorithms, etc) and the taken design decisions. NO DOCUMENT WILL BE REVIEWED if it does not include at least:
  1. Front page indicating authors and their student numbers.
  2. Index of contents.
  3. Description of the implemented methods.
  4. Done tests and their results.
  5. Conclusions.
  6. Justified margins and numerated pages.

The final document is fundamental to get a good mark in your lab work. It is also necessary to submit the correct code, with the requested method names. The length of the final document MUST NOT exceed 10 pages (index and front page included). If the final document does not fulfill the mentioned requirements, the lab will be failed.

- The submission will ONLY be done through AULA GLOBAL.
- The version to be reviews is the last one submitted.

Rules

1. The code MUST compile and MUST work as expected.
2. Non-commented code is considered void.
3. The submission MUST be done through AULA GLOBAL. No alternative methods can be used.
4. The submitted code and final document MUST be original work. If cheating is detected, all the students involved in the copy will fail the lab.