COMPUTER STRUCTURE

GRADO EN INGENIERÍA INFORMÁTICA

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN DE EMPRESAS

uc3m | Universidad Carlos III de Madrid

Grupo de Arquitectura de Computadores

Assignment 1
Introduction to assembly programming

Course 2022/2023

Content

Objectives

The objective of the lab is to understand the concepts related to assembly programming. For this purpose, the RISC-V assembler and the CREATOR simulator, which is available at https://creatorsim.github.io/creator/ , will be used as a basis.

In order to become familiar with assembly programming and the previous simulator, it is recommended to solve **the exercises available in Aula Global** in the section corresponding to the first assignment.

The assignment consists of 4 exercises.

Exercise 1

The objective of this exercise is to develop in RV32IMF assembler a function (called `string_compare`) to work with strings:

```
Function string_compare ( char A[], char B[] )
```

This function allows to compare two strings so that it is possible to know if the strings stored in memory are the same or not. The string `A` stores the memory address of the first string and `B` the address of the second string.

This function receives the following arguments in the order given:

- Argument 1: (**A**) starting address of a string terminated in end of string (ASCII code 0).
- Argument 2: (**B**) starting address of a string ending in end of string (ASCII code 0).

The function returns a single value that takes one of these three possible values:

- On error, it returns the integer -1.
- If the two strings are the same, it returns the integer 1 (one).
- If the two strings are different, it returns the integer 0 (zero).

The following figure illustrates an example when the content of two strings is different:

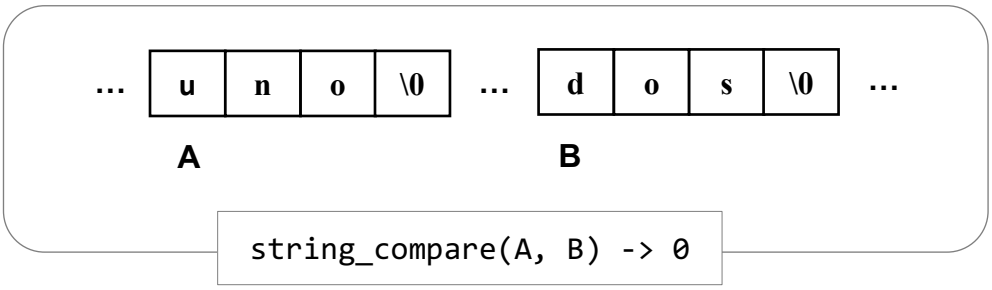

Figure 1.- Different strings.

The following figure illustrates an example when the content of two strings is equal:

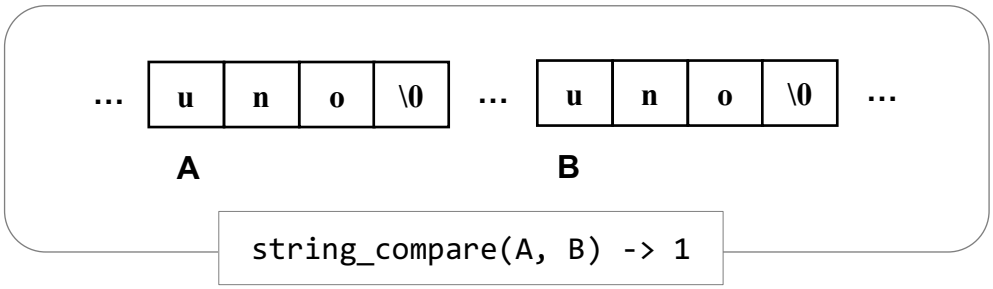

Figure 2.- Equal Strings

The two possible errors to be considered in this function are that address A is null (zero value) and that address B is null (zero value).

In the `string_compare` function, in case of finding an error, it must return only the value -1 without doing anything else. In the same way, the function must compare character by character and as soon as it is a different one it must return 0.

It is mandatory to correctly follow the parameter passing convention described in the course for RISC-V, as well as to respect the function signature (name, number of parameters, order of parameters and value to return).

It will not be considered valid to develop the code of the functions directly in the `main` function. You must write the code corresponding to the requested functions outside the main function, as separate functions.

Exercise 2

A computer security company wants to use the `string_compare` function of exercise 1 to check if a user's password, stored in memory, is the same as the password that the user enters by keyboard to authenticate.

As a preliminary step, the objective of this exercise is to study the impact of power consumption when using the `string_compare` function, for a possible side channel attack. For this purpose, the following function will be developed in this exercise:

`Function` **`study_energy`** `(char password[])`

This function receives the following argument:

- `A`: starting address of a string. Like all strings, it ends with the ASCII code 0 used as the end of the string.

This function must print on each line, for each of the letters ('a' to 'z') the following information:

```
letter: number of cycles.
```

For example, a possible output would be this:

```
a: 10
b: 10
c: 10
d: 20
e: 10
.
.
z: 10
```

where number of cycles represents the cycles used by the processor to invoke the `string_compare` function. Thus, the number of cycles obtained for the letter `a` is the number of cycles required to invoke the `string_co`mpare function with the following two arguments:

- `Password`: string passed to the `study_energy` function.
- String "`a`". This refers to the string "`a`", not the character '`a`'.

The function does not return any value, it simply prints to the console one line for each letter of the alphabet in the format indicated above:

```
letter: number of cycles
```

To create the different strings to be used inside the function ("a", "b", "c") you can use a program fragment similar to this one:

```
Addi   sp, sp -4 # space is left to store a string.
li     t0, 'k
sb     t0, 0(sp)
sb     zero, 1(sp)
```

Thus, the stack pointer `sp` points to a memory area where the string 'k' is stored. Note that we store the character 'k' and then the ASCII code 0.

It is mandatory to correctly follow the parameter passing convention described in the course for RISC-V, as well as to respect the function signature (name, number of parameters, order of parameters and value to be returned). Similarly, the `study_energy` function must use the `string_compare` function of the first exercise (it must call this function).

It will not be considered valid to develop the code of the functions directly in the main function. You must write the code corresponding to the requested function as a separate function.

The following RISC-V pseudo-instruction will be used to calculate the number of cycles:

- `rdcycle rd`. This instruction stores in `rd` the number of cycles that have been executed so far.

Thus, when you want, for example, to calculate the number of cycles required to call `string_compare` and compare the string `password` with a certain letter, you can use, as an idea, the following:

```
.data
    possible_key:     .string "k"

...

       # he cycles executed up to now are obtained
       rdcycle  t0

       # call string_compare (password, possible_key)
       ...
       # prepare arguments for string_compare
       jal ra string_compare

       # get cycles executed so far
       rdcycle  t2
       sub t0 t2 t0           # t0: number of cycles consumed

       # print the number of cycles executed
       mv a0 t0
       li a7 1
       ecall
       ...
```

Exercise 3

The objective of this exercise is to implement a simple function that performs a side-channel attack.

Consider that the keys are at most 8 characters for the study.

Through a brute-force attack, we would have to test up to 27^8 (282,429,536,481) possible keys. But with a side-channel attack, it would be possible to detect by using the energy consumption (number of cycles executed) the first character of the key, and once this character is fixed, it would be possible to study the 27 possible characters for the second letter and so on. In other words, possible combinations are reduced to 27*8 (216 possibilities), in case the key is just 8 characters long.

In Exercise 3, we have to implement in assembler RV32IMF a function called attack to discover a possible key of up to 8 characters:

```
Function attack ( char password[], char dummy[] )
```

This function receives the following arguments in the provided order:

- Argument 1: (A) starting address of the user key to be discovered. This string ends with the ASCII code 0 used as the end of the string.
- Argument 2: (B) starting address of a string in memory where the detected key will be stored. This string must have space for 9 bytes (8 of the string plus the end of the string).

The function does not return any value, it simply stores in dummy memory address the value of the discovered key.

For the development of this exercise, use as a basis the idea of Exercise 2 and add everything necessary to perform the indicated attack.

Note that this is an execise, in a real scenario the contents of the user's password would never be known. However, the principle used (side-channel attack) is the same that has inspired attacks on current processors such as spectre.

Exercise 4

How should the `string_compare` function be implemented to avoid attacks of this type?

Important aspect to consider

General rules

1) The names of the functions (`string_compare, study_energy` and `attack`) must be written in lower case.
2) The assignment will be done in groups of two.
3) The submission of the assignment will be done through the enabled deliverers. Delivery via e-mail is not allowed.
4) The delivery shall be made within the deadline given by the deliverers.
5) Special attention will be paid to detect functionality copied between two assignments. In case of finding common implementations in two assignments (or similar contents in the report), both will get a grade of 0 (zero).
6) The parameter passing convention described in class must be followed. Those functions that do not correctly follow the parameter passing convention will also be graded 0 (zero).
7) Both required exercises must be handed in. If an exercise is not handed in, the assignment will be considered as not handed in.
8) Exercises that do not compile or do not conform to the functionality and requirements will be graded 0 (zero).
9) An uncommented program will get a grade of 0 (zero).

Report

1) The report (a single document) must contain at least the following sections:
   - Title page listing the authors (including full name, NIA, class group to which they belong and e-mail address).
   - Table of contents.
   - Contents requested in the different exercises (one section per exercise).
   - Conclusions and problems encountered.
2) **The length of the report should not exceed 10 pages** (cover page and table of contents included).
3) Regarding the possible description of the requested programs:
   - The report should describe the behavior of the programs, as well as the main design decisions. **The pseudocode corresponding to each of the exercises of this assignment must be included**.
   - The test battery (as defined in the following section) used to validate the functionality of the requested functions and the results obtained must be included. Higher scores will be given to advanced tests, extreme cases, and in general to those tests that guarantee the correct functioning of the assignment in all cases.

- Avoid duplicate tests that evaluate the same program execution flows. Scoring in this section is not measured by the number of tests, but by the degree of coverage of the tests. Few tests that evaluate different cases are better than many tests that always evaluate the same case.

NOTE: DO NOT NEGLECT THE QUALITY OF THE REPORT OF YOUR ASSIGNMENT.

Passing the report is as essential to pass the assignment as the correct functioning. If, when evaluating your report, it is considered that it does not reach the minimum admissible, your assignment will be failed.

Tests definition

For the definition of a test battery for each of the exercises to be included in the report, include a table with the following format:

Input data:	Test description:	Expected output:	Actual output:

Submission procedure

The submission of assignment 1 will be done electronically through Aula Global,

The assignment will be done in groups of two students.

The deadline for both **is November 2, 2022 at 23:55 hours.**

It is possible to submit as many times as you want within the given deadline, the only registered version of your assignment is the last one submitted. The evaluation of the assignment is the evaluation of the content of this last submission. Always check what you submit.

Submission: You must submit a single compressed file in zip format with the name ec_p1_AAAAAAAAAAAA_BBBBBBBBBBBB.zip where A...A and B...B are the NIA of the group members.

The **zip** file must contain only the following files:

- **exercise1.s**. This file will only contain the code of the functions. It will have neither main subroutine nor data segment. Therefore, the functions you deliver must work without depending on a specific data segment.
- **exercise2.s**. This file will only contain the code of the requested function. It will have neither main subroutine nor data segment. Therefore, the functions you deliver must work without depending on a specific data segment.
- **exercise3.s**. This file will only contain the code of the requested function. It will have neither main subroutine nor data segment. Therefore, the functions you deliver must work without depending on a specific data segment.
- **exercise4.txt**. This text file will only contain the answer to exercise 4.
- **report.pdf** This file will contain the contents of the report, and will be in PDF format (it is not valid to rename a text file or similar with .pdf extension).
- **authors.txt** This file will contain a line for each member of the group with the NIA and the group to which he/she belongs.

Evaluation and scoring

The evaluation will be performed as follows:

	Code	Memory
Exercise 1	0,5	0,5
Exercise 2	3,0	1,0
Exercise 3	3,5	1,0
Exercise 4	0,0	0,5

Please note that, in order to follow the continuous evaluation process, the minimum grade obtained in each assignment must be 2 out of 10 and the average of the two assignment exercises must be 4 out of 10. **The two exercises must be delivered to be considered as having been delivered.**

NOTE:

1. **If a serious misconception is detected in the assignment (in any section of any exercise), the overall evaluation of the whole assignment will be zero points (0 points).**
2. **If the format of delivery is not respected (for example, not respecting the name of the requested files, delivering a .rar file, delivering the files in a directory, etc.), the grade will be significantly reduced.**
3. **In case of finding common implementations in two assignments (or similar contents in the report), it will be understood that the practice has been copied and both will get a grade of 0.**
4. **In case of finding code fragments obtained directly from the Internet, it will be understood that this content has been copied and the assignment will be graded 0.**