COMPUTER STRUCTURE

BACHELOR IN COMPUTER SCIENCE AND ENGINEERING

DUAL BACHELOR IN COMPUTER SCIENCE AND ENGINEERING AND BUSINESS ADMINISTRATION

BACHELOR IN APPLIED MATHEMATICS AND COMPUTING

UNIVERSIDAD CARLOS III DE MADRID

ARCOS Group

Assignment 2

Introduction to microprogramming

Course 2022/2023

Version 2.2

Contents

Goals of the practice

The **main goal** of this assignment is to learn how to design an instruction set for a computer.

The main knowledge that will be practiced are those of microprogramming, assembly programming and information representation. You will also learn to evaluate different design alternatives and to work in a group.

For the development of the assignment, it is necessary to review the following concepts:

- The representation of integers, strings, etc.
- The main aspects of assembly language.
- The instruction format and the addressing modes.
- The operation of a processor, including the execution stages, microprogramming, etc.

The student **will use** the WepSIM simulator to be able to exercise in an interactive way the concepts and knowledge indicated above, thus completing the exercises of the subject and also improving the knowledge of the overall functioning of a computer.

The simulator is available in https://wepsim.github.io/wepsim and the initial documentation is accessible from https://wepsim.github.io.

Exercise 1

The company we work for requests that we **design, implement and test** an instruction set similar to that of the RISC-V processor for the WepSIM simulator shown in Table 1..

Instruction	**Format**	**Associated functionality**	**Status Register (SR)**
lui R_{RE1}, U32	CO (31-26): 010010 R_{RE1} (25-21) U32 (63-32)	BR[R_{RE1}] ← U32	Not updated
sw R_{RE1}, (R_{RE2})	CO (31-26): 010000 R_{RE1} (25-21) R_{RE2} (20-16)	Memory [R_{RE1}] ← BR[R_{RE2}]	Not updated
lw R_{RE1}, (R_{RE2})	CO (31-26): 010011 R_{RE1} (25-21) R_{RE2} (20-16)	BR[R_{RE1}] ← Memory [R_{RE2}]	Not updated
add R_{RE1}, R_{RE2}, R_{RE3}	CO (31-26): 011000 R_{RE1} (25-21) R_{RE2} (20-16) R_{RE3} (15-11)	BR[R_{RE1}] ← BR[R_{RE2}] + BR[R_{RE3}]	Updated
mul_add R_{RE1}, R_{RE2}, R_{RE3}, R_{RE3}	CO (31-26): 011001 R_{RE1} (25-21) R_{RE2} (20-16) R_{RE3} (15-11) R_{RE4} (10-6)	BR[R_{RE1}] ← BR[R_{RE2}] * BR[R_{RE3}] + BR[R_{RE4}]	Updated **(by add)**
beq R_{RE1}, R_{RE2}, S10	CO (31-26): 110100 R_{RE1} (25-21) R_{RE2} (20-16) S10 (9-0)	Si (R_{RE1} == R_{RE2}): PC ← PC + S10	Not updated
jal U16	CO (31-26): 100001 U16 (15-0)	BR[ra] ← PC PC ← U16	Not updated
jr_ra	CO (31-26): 100010	PC ← BR[ra]	Not updated
halt	CO (31-26): 100011	PC ← 0x00 SR ← 0x00	Not updated
xchb (R_{RE1}), (R_{RE2})	CO (31-26): 011010 R_{RE1} (25-21) R_{RE2} (20-16)	RT1 ← Memory[R_{RE1}] Memory [R_{RE1}] ← Memory [R_{RE2}] Memory [R_{RE2}] ← RT1	Not updated

Table 1.- Extended RISC-V instruction set

All instructions will be encoded using 32 bits (except "lui" which is encoded in 64 bits).

The notation used in Table 1 for immediate values is as follows:

- U32 refers to a 32-bit unsigned integer.
- U16 refers to a 16-bit unsigned integer.
- S16 refers to a 16-bit signed integer in two's complement.
- S10 refers to a 10-bit signed integer value in two's complement.

For the values "S16/S10" you must perform a sign extension while in the "U16" no sign extension must be done (it is filled with zeros on the left, on the most significant part).

MEMORY[R] refers to the content of the memory position whose address is stored in the R register.

The notation used in Table 1 for registers is as follows:

- R_{RE*} will be used to denote the general-purpose registers of RISC-V, which in this 32-bit version are 32 registers of 32-bit.
- BR will be used to reference the register file.
- PC will be used to reference the PC register (Program Counter).
- SR will be used to reference the SR register (Status Register).
- $BR[R_{RE1}]$ shall be used to indicate the contents of the R_{RE1} register.

The integers stored in registers are 32 bits using two's complement.

The following is the mapping between RISC-V registers and WepSIM registers. This mapping must be indicated in the register section of the microcode.

RISC-V Registers		**WepSIM Registers**	**Meaning**
x0	zero	R0	Contains a zero
x1	ra	R1	Return address of a function call
x2	sp	R2	Stack pointer
x3	gp	R3	Global pointer
x4	tp	R4	Thread pointer
x5…x7	t0…t2	R5…R7	Temporary registers (1/2)
x8	fp	R8	Stack frame
x9	s1	R9	To be saved register
x10…x11	a0…a1	R10…R11	Argument for functions (1/2) and returned values
x12…x17	a2…a7	R12…R17	Argument for functions (2/2)
x18…x27	s2…s11	R18…R27	To be saved register
x28…x31	t3…t6	R28…R31	Temporary records (2/2)

Table 2.- RISC-V registers mapping

Each register has a name, which is indicated in the RISC-V registers' column of Table 2. There are 4 registers of special interest:

- The stack pointer register or sp register is the R2 on the WepSIM elementary processor.
- The return address register (ra) is the R1 register in the elementary processor.
- The PC register or program counter is the PC register of the elementary processor.
- The status register (SR) is the SR register on the WepSIM processor.

The RT1, RT2 and RT3 registers are transparent to the assembly programmer.

When invoking a function, consider for this assignment the following:

- **Argument passing**: To pass parameters to a function in the RISC-V assembly, the a0...a7 registers are used. In case of passing more than 8 arguments to a function, from the eighth to the last arguments should be passed using the stack.
- **For return of value**: To return a result, the a0 and a1registers are used.
- **Consider that** a function must preserve the values of all registers s0…s11, sp, fp y ra that it modifies.

A valid implementation that minimizes the number of clock cycles will be considered positively. Please justify briefly in the report the design decisions that have been made to achieve it.

The results of this exercise are related to the design and implementation of the microcode and will be both indicated in the corresponding part of the report and in the file associated with the requested functionality.

The section in the report for Exercise 1 should contain:

- A table with four columns. The first column includes the name of the instruction, the second the design of the instructions requested in RT language (transfer between registers language). In this second column it is necessary to indicate, for each cycle, the elementary operations of transfer between registers, necessary for the execution of the instruction. The third column includes the control signals to be activated in each cycle of each instruction. The fourth includes the design decisions you have made for this instruction. This table will have as many rows as instructions have been requested in the statement.

The file with the requested functionality is:

- **e1_checkpoint.txt**:
  A *checkpoint* containing the microcode of the requested instructions (see Appendix 1).
  You only have to include **the correct microcode for** the **requested instructions (without fetch, and without register section)** according to the given requirements and being in the correct format for the WepSIM simulator.

Exercise 2

To test the instructions, you can code the programs you consider appropriate. However, in order to perform a demonstration, you have to implement in assembly, using the RISC-V instruction set developed in exercise 1, the following program:

The program to be encoded will have to work with a 24x24 pixel image in memory using the RISC-V instructions designed in the previous section.
Each pixel is encoded with one byte and the 24x24 pixels are stored consecutively in memory as the following example shows:

```
.data
  msg: .byte 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,1,0,1,0,1,1,1, 0,1,1,1,0,1,1,1, 0,1,0,1,0,0,0,0,
             0,1,0,1,0,1,0,1, 0,1,0,1,0,1,0,1, 0,1,0,1,0,0,0,0,
             0,1,1,1,0,1,1,1, 0,1,1,1,0,1,1,1, 0,1,1,1,0,0,0,0,
             0,1,0,1,0,1,0,1, 0,1,0,0,0,1,0,0, 0,0,0,1,0,0,0,0,
             0,1,0,1,0,1,0,1, 0,1,0,0,0,1,0,0, 0,1,1,1,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,1,0,1,0,1,1,1, 0,1,1,1,0,1,1,1, 0,0,0,0,0,0,0,0,
             0,1,0,1,0,1,0,0, 0,1,0,1,0,1,0,1, 0,0,0,0,0,0,0,0,
             0,1,1,1,0,1,1,1, 0,1,1,1,0,1,0,1, 0,0,0,0,0,0,0,0,
             0,0,0,1,0,1,0,0, 0,1,0,1,0,1,1,0, 0,0,0,0,0,0,0,0,
             0,1,1,1,0,1,1,1, 0,1,0,1,0,1,0,1, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,1,1,1,0,1,1,1, 0,1,1,1,0,1,1,1, 0,0,1,0,0,0,0,0,
             0,0,0,1,0,1,0,1, 0,0,0,1,0,0,0,1, 0,0,1,0,0,0,0,0,
             0,1,1,1,0,1,0,1, 0,1,1,1,0,1,1,1, 0,0,1,0,0,0,0,0,
             0,1,0,0,0,1,0,1, 0,1,0,0,0,1,0,0, 0,0,0,0,0,0,0,0,
             0,1,1,1,0,1,1,1, 0,1,1,1,0,1,1,1, 0,0,1,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0
```

To display the saved image from the "msg:" tag, the following code fragment, which uses input/output instructions, must be used:

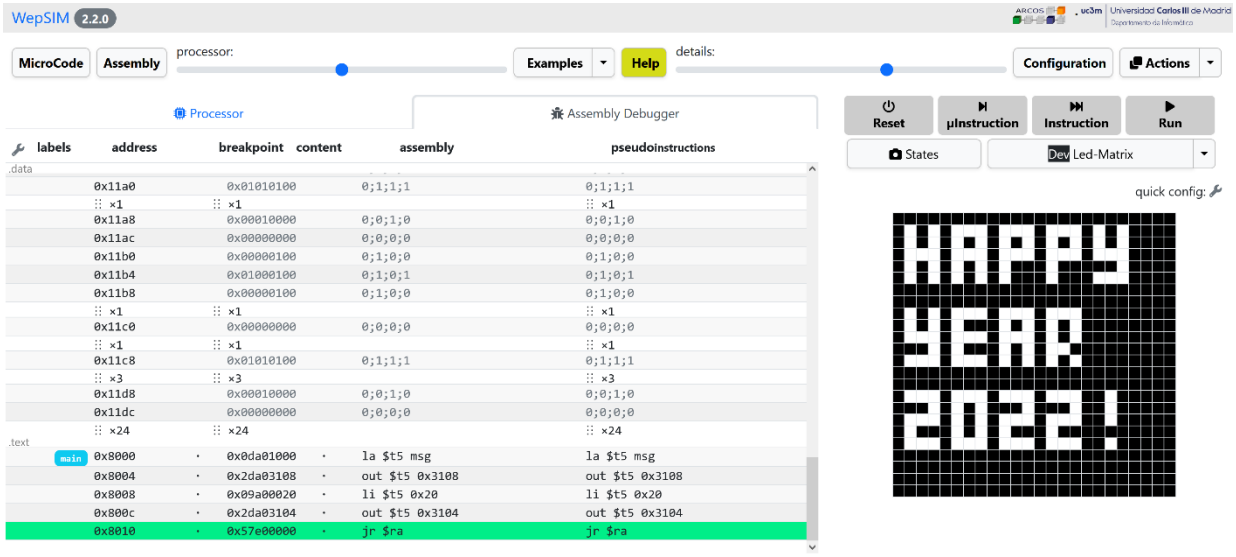
```
# send address of msg to 0x3108 (data I/O register)
lui  t5 msg    # lui load in t5 the msg address
out  t5 0x3108 # out send t5 value to 0x3108 I/O address

# send show command to 0x3104 (control I/O register)
lui  t5 0x20   # 0x20 means "show image" command
out  t5 0x3104 # out send t5 value to 0x3104 I/O address
```

Code 1.- Code fragment for displaying 24x24 bytes image (saved from "msg:" tag) on led matrix device.

Figure 1 shows the result obtained:

Figure 1.- Example showing a 24x24 image



The program to be coded to demonstrate the use of the instructions is composed of two functions: **`demo`** and **`main`**.

The main function or routine is in charge of:

1. Call the **`demo`** routine that receives one parameter: image start address. The routine returns in a0 an integer with value one to indicate that everything has been executed without problems.
2. Terminate program execution (halt instruction).

Note the parameter passing (and register usage) convention for RISC-V that is described at the top of page 6 of this statement (including the appropriate use of registers).

The **`demo`** function must do two things:

1. Rotate the image so that the byte representing the color of pixel image[i][j] is in image[j][i] and vice versa (where i and j are natural values from zero up to 23).
2. Display the rotated image using the Led-Matrix device as described above (in the section shown in Code *1*).

The results of this exercise must be indicated both in the part of the report corresponding to this exercise and in the file with the source code associated with the requested functionality.

L The section of the report for Exercise 2 should contain:

- After implementing the requested program with the RISC-V instructions for Exercise 1, you have to compare both instruction sets (original and the extended instructions of the exercise 1) indicating: differences in the types of instructions, advantages and disadvantages, and possible improvements.

E The file with the requested functionality is:

- **e2_checkpoint.txt**:
  You have to save a checkpoint containing the program **in assembly** for the test requested in exercise 2 **including the microcode of exercise 1 with fetch and register section,** according to the requirements given and being in the correct format for the WepSIM simulator.
  Appendix 1 summarizes the steps to save a *checkpoint*.

Submission procedure

The delivery of assignment 2 will be done electronically through Aula Global, for which two links associated with said practice will be enabled.

Generally speaking, please note that:

- The assignment can be done in groups of **up to two students** (no more).
- The deadline for delivery is **December 4th, 2022 at 23:50 (Aula Global).**
- The content of this submission is the one evaluated. Please review (all members of the team) your files before submitting them.

To be submitted:

Two links will be used. A deliverer is enabled to deliver memory via turnitin. A file must be delivered in PDF format with the name AAA_BBB. pdf where AA and BBB are the NIAs of the members of the group. **Report can only be delivered once via turnitin.**

The other link will be used to submit all in a single compressed file in .**zip** format with the name AAA_BBB.zip where AAA and BBB are the NIA of the members of the group.

The **zip** file should contain only the following files (without subdirectories):

- **e1_checkpoint.txt**
- **e2_checkpoint.txt**
- **report.pdf**
- **authors.txt**

Where the file "authors.txt" will contain one line per author with only its corresponding NIA.

All files will be ASCII type text except for the **report that will be in PDF format**. The report is the same as that delivered through the Turnitin deliverer enabled for the delivery of the memory only. Check that you have used each deliverer correctly to prevent the delivery from being null (and therefore of equal value as not having delivered it).

It is possible to deliver as many times as you want within the given time frame **the zip file**, the only recorded version of your practice is the last one delivered. The content of this latest submission is the one evaluated.

Important aspects to keep in mind

It must carefully check that all the requirements indicated in the statement are met. It is recommended from the statement to generate a checklist that helps the team to review everything.

It is important to remember that the name of the functions/subroutines, files, etc. must be those indicated in this statement. Not respecting these names will mean a zero of note in the corresponding section so it is recommended to add to the list of checks of the names.

General rules

1) Submission will be made through the authorized deliverers. Delivery via email is not allowed.
2) Submission will be made within the period given by the deliverers. It is possible that for a Aula Global deliverer the end of the deadline for a delivery at 24:00 ends 10 minutes earlier. Review Aula Global support to confirm the deadline.
3) Particular attention will be paid to detecting functionalities copied between assignment. In case of detecting plagiarism between two assignments (or reports), all the team involved (copied and copiers) will obtain a rating of 0 (zero), and a plagiarism file can be opened depending on the severity. The copying of portions of the Internet or practices of other courses is also considered a plagiarism, and all team involved may be issued.

Source code considerations

1) It will be valued that a valid implementation has been made that minimizes the number of clock cycles.
2) A program not properly commented on will obtain a rating of 0. Comments should seek to describe the steps that are intended to be implemented before the block of code that implements it.
3) Keep in mind that a program that compiles correctly is no guarantee that it will work correctly. Therefore, you will have to carry out those tests that guarantee the correct functioning of the practice.
4) The source code submitted must **not print messages** on the screen or contain any additional code used for diagnostics other than that **described in the** statement.
5) All exercises must always work with the version of the WepSIM simulator given in the URL: **https://wepsim.github.io/wepsim/**

Report

1) **The extension of the report must not exceed 12 pages** (cover page and index included).
2) **It will be delivered in PDF format with selectable text (it must allow turnitin to perform the copy control in** memory).
3) The report (a single document) must contain at least the following sections:
    - Cover where the authors appear (including full name, NIA and group of each author), degree, subject and practice.
    - Table of Contents.
    - Section for exercise 1 where the request for that exercise is included.
    - Section for exercise 2 where the request for that exercise is included.
    - Conclusions and problems found. Include a summary of the number of hours allocated to the practice.

NOTE: DO NOT NEGLECT THE QUALITY OF THE MEMORY OF YOUR PRACTICE.

Passing the report is as essential to approve the practice, as the correct functioning of the practice. If, when evaluating the memory of your practice, it is considered that it does not reach the minimum admissible, your practice will be suspended.

Scoring

The score of the practice will be distributed between the code and memory delivered as follows:

- **Microcode and code (7 points)**
- **Report (3 points)**

VERY IMPORTANT:

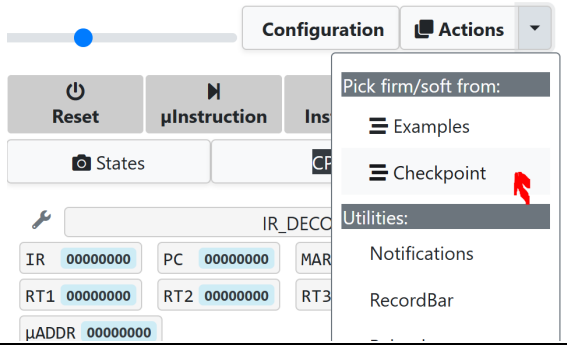
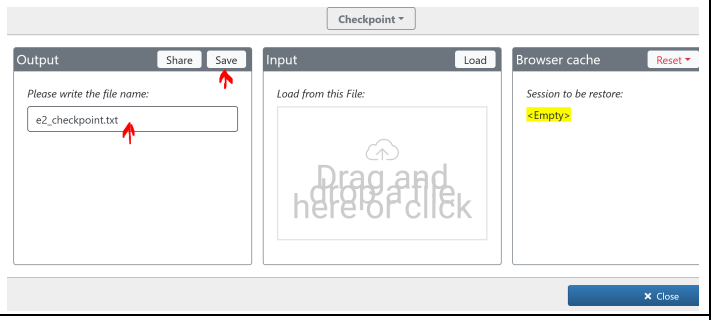
1. **Although the score can be divided into sections, the correction of the assignment will be carried out at a global level, that is, this includes:**
   a. **If an exercise is not delivered the grade of the entire assignment will be zero.**
   b. **If a serious misconception is detected in practice (in any section of any exercise), the overall assessment of the entire practice will be zero points (0 points).**
2. **Checks will be carried out to avoid full or partial plagiarism in delivered work, i.e. this includes:**
   a. **In case of finding common implementations in two practices (or similar contents in the memory), both will obtain a grade of 0.**
   b. **If un-referenced snippets of code obtained directly from the Internet are found, the assignment will have a rating of zero.**
3. **You must respect the format and names requested, this includes:**
   a. **It is essential to respect the name and format of the files since otherwise it will mean a note of 0 (zero). This includes not respecting lowercase letters, delivering a .rar instead of .zip, a .docx instead of pdf (not worth renaming), etc.**
   b. **The text of the file with the report (report.pdf) must be selectable and copyable. That is, if a PDF file is generated based on one image per page (to avoid its treatment by copy control tools) then the rating will be a zero for the whole assignment.**

Appendix 1: Storing a checkpoint with WepSIM

WepSIM enables to store the entire work session in a single file (it is called *checkpoint*). This session can include the requested microcode, assembly code, states at different execution points, and a recording of the work session. In this way it is more agile to continue the work or share this work among members of the practice group.

Below are the steps to save a *checkpoint*.

Save a *checkpoint*

In the run mode menu, please select the *"Checkpoint"* option.	
Please enter the name of the file in the "File name:" field and then press the "Save" button to save the file.	
Please verify that the file has been saved correctly and contains everything requested in the statement.