

COMPUTER STRUCTURE  
DEGREE IN COMPUTER ENGINEERING  
DOUBLE DREGEE IN COMPUTER  
ENGINEERING AND BUSSINESS  
ADMINISTRATION

**Practice 2**

**MICROPROGRAMMING  
INTRODUCTION**



UNIVERSIDAD CARLOS III DE MADRID  
ARCOS GROUP

**Course 2019/2020**

**Version 2.2**

# Content

|                                     |    |
|-------------------------------------|----|
| Practice's objectives.....          | 3  |
| Exercise 1 .....                    | 4  |
| Exercise 2 .....                    | 6  |
| Practice delivery procedure.....    | 8  |
| General rules .....                 | 9  |
| Practice's code.....                | 9  |
| Report of the practice .....        | 10 |
| Practice evaluation .....           | 10 |
| Appendix 1: proof record .....      | 12 |
| Appendix 2: Save a checkpoint ..... | 14 |

## Practice's objectives

The **main objective** of this practice is to design and use an instruction set for a computer. The main knowledge that will be put on practice are microprogramming, assembly programming and information representation. You will also learn to evaluate different design alternatives and work in groups.

It is necessary to review the following concepts of the Computer Structure course:

- Data representation.
- The main aspects of assembly programming.
- Format of instructions and types of addressing.
- The operation of a processor, including execution stages, microprogramming, etc.

The student will use the WepSim simulator, which will allow him to perform, in an interactive way, all the concepts learned during the course and improve the knowledge of how a processor works. The simulator is available at <https://wepsim.github.io/wepsim> and the initial documentation is accessible from <https://wepsim.github.io>.

## Exercise 1

Let's assume that the company we are working for, asks us to design, implement and test a set of instructions adapted from the **Z80 processor for WepSim** simulator. **The instructions are coded in 32 bits.** The following table describes the required instructions.

| Instruction  | Format  | Associated functionality  | Status register     |
|--|---|---|---------------------|
| <b>ld</b> <b>R<sub>TARGET</sub></b> , <b>R<sub>SRC</sub></b>   | CO (31-26): 010000<br>R <sub>TARGET</sub> (25-21)<br>R <sub>SRC</sub> (20-16) | $BR[R_{TARGET}] \leftarrow BR[R_{SRC}]$                                     | Must not be updated |
| <b>ldi</b> <b>R<sub>TARGET</sub></b> , <b>U16</b>              | CO (31-26): 010010<br>R <sub>TARGET</sub> (25-21)<br>U16 (15-0)               | $BR[R_{TARGET}] \leftarrow U16$   | Must not be updated |
| <b>ld</b> <b>R<sub>TARGET</sub></b> , <b>(R<sub>SRC</sub>)</b> | CO (31-26): 010011<br>R <sub>TARGET</sub> (25-21)<br>R <sub>SRC</sub> (20-16) | $BR[R_{TARGET}] \leftarrow Memory[R_{SRC}]$                                 | Must not be updated |
| <b>add_a</b> <b>R<sub>SRC</sub></b>                            | CO (31-26): 011000<br>R <sub>SRC</sub> (25-21)                                | $A \leftarrow A + BR[R_{SRC}]$  | Must be updated     |
| <b>addi_a</b> <b>S16</b>                                       | CO (31-26): 011010<br>S16 (15-0)  | $A \leftarrow A + S16$  | Must be updated     |
| <b>inc</b> <b>R<sub>SRC</sub></b>                              | CO (31-26): 011100<br>R <sub>SRC</sub> (25-21)                                | $BR[R_{TARGET}] \leftarrow BR[R_{SRC}] + 1$                                 | Must be updated     |
| <b>dec</b> <b>R<sub>SRC</sub></b>                              | CO (31-26): 011101<br>R <sub>SRC</sub> (25-21)                                | $BR[R_{TARGET}] \leftarrow BR[R_{SRC}] - 1$                                 | Must be updated     |
| <b>jp</b> <b>S16</b>   | CO (31-26): 110000<br>S16 (15-0)  | $PC \leftarrow PC + S16$  | Must not be updated |
| <b>jpz</b> <b>S16</b>  | CO (31-26): 110011<br>S16 (15-0)  | IF (Flag.Zero == 1)<br>$PC \leftarrow PC + S16$                             | Must not be updated |
| <b>call</b> <b>U16</b>   | CO (31-26): 100001<br>U16 (15-0)  | $SP \leftarrow SP - 4$<br>$Memory[SP] \leftarrow PC$<br>$PC \leftarrow U16$ | Must not be updated |
| <b>Ret</b>   | CO (31-26): 100010  | $PC \leftarrow Memory[SP]$<br>$SP \leftarrow SP + 4$                        | Must not be updated |
| <b>Halt</b>  | CO (31-26): 100011  | $PC \leftarrow 0x00$ (halt execution)                                       | Must not be updated |
| <b>push</b> <b>R<sub>SRC</sub></b>                             | CO (31-26): 100100<br>R <sub>SRC</sub> (25-21)                                | $SP \leftarrow SP - 4$<br>$Memory[SP] \leftarrow R_{SRC}$                   | Must not be updated |
| <b>pop</b> <b>R<sub>TARGET</sub></b>                           | CO (31-26): 100101<br>R <sub>TARGET</sub> (25-21)                             | $R_{TARGET} \leftarrow Memory[SP]$<br>$SP \leftarrow SP + 4$                | Must not be updated |

In this table **BR** stands for **Bank of Registers**. Thus, **BR[R<sub>TARGET</sub>]** or **BR[R<sub>SRC</sub>]** indicate the **contents** of one of the general-purpose Z80 registers, which are **A, BC, DE, HL, IX, IY**. In this version, **all the registers are of 32 bits**.

**S16** represents a 16-bit value with the **Sign Extension flag activated**, which means that for values that have less than 16 bits, the left-part of the value is filled with the value of the most significant bit (0 or 1). **U16** represent a 16-bit value with the **Sign Extension flag deactivated**, which means that the left-part of the value is filled with zeros. Wepsim

allows to active Sign Extension by using the SE flag. The integers numbers stored in registers are of 32 bits and are represented in two's complement form.

All instructions work with integer numbers, represented by two's complement. Registers involved in these instructions ( $R_{TARGET}$  and  $R_{SRC}$ ) refers to Z80 registers. The following table indicates the association between Z80 and Wepsim registers. This association must be included in the microcode developed for the Z80 instructions.

| Z80 register | WepSIM register | Meaning            |
|--------------|-----------------|--------------------|
| A            | R4              | Accumulator        |
| BC           | R5              | Paired register BC |
| DE           | R6              | Paired register DE |
| HL           | R7              | Paired register HL |
| IX           | R8              | Paired register IX |
| IY           | R9              | Paired register IY |

The stack-pointer register (SP) corresponds to register 29 of the WepSim bank of registers, and the *Flag* register, mentioned in the Z80 instruction jpz, corresponds to the bit Z of the SR register in the Wepsim simulator. **To pass arguments to a subroutine you must use IX and IY (equivalent to \$a0 and \$a1 in MIPS). To return a value from a subroutine you must use the HL register (equivalent to \$v0 in MIPS).** BC and DE are general purpose registers that could be similar to \$t0 and \$t1 registers in MIPS.

The expected results of Exercise 1 are the **microcode and the justification of decision made in each instruction implementation**, which must be indicated in the report. Minimize the number of clock cycles in each instruction implementation will be appreciated.

Exercise 1 section of the report must include:

- A table with four columns. The first column contains the instruction name, the second one the instruction design in RT language (this column must include the elementary register-transfer operations for each clock cycle). The third column must include the control signals required for each clock cycle. The last column must contain a brief description of the design decisions you have made for implementing each instruction. This table will have as many rows as instructions are requested in the statement.

The name of the file with the requested functionality will be:

- **e1\_checkpoint.txt:**  
This file represents a checkpoint file created with Wepsim, which contains the microcode with the requested instructions (**see Appendix 2, which explains how to create a checkpoint file**). **It only must include the correct microcode for the requested instructions (without fetch)** 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 to test your instruction separately. However, in order to be able to demonstrate to the company the correct functionality of our instructions, we are required to make an assembly program, which must use the set of microprogrammed instructions in Exercise 1.

The program to be coded is the equivalent in Z80 (using the instructions requested in Exercise 1 and with the same functionality) to the next one indicated in MIPS32:

```
.data
    vector: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

.text
addv: # push $a0 and $a1
        addi $sp $sp -8
        sw $a0 0($sp)
        sw $a1 4($sp)
        # $v0 will contain the addition of the vector elements
        li $v0 0
b1:    beq $a0 $0 f1
        lw $t0 ($a1)
        add $v0 $v0 $t0
        addi $a1 $a1 4
        addi $a0 $a0 -1
        b b1
        # pop $a1 and $a0
f1:    lw $a0 0($sp)
        lw $a1 4($sp)
        addi $sp $sp 8
        # return
        jr $ra

main: # call addv subroutine
        li $a0 10
        la $a1 vector
        jal addv
        # Finish the program
        li $v0 10
        syscall
```

This program has the **addv** subroutine that receives two parameters: the number of elements of an integer vector and the start direction of that integer vector. The routine returns in \$v0 the sum of the numbers contained in the vector passed by parameter. There is also a **main** routine that calls the **addv** routine and ends the program. Please consider the Z80 registers that must be used as parameters and returning values in a subroutine call, explained in Exercise 1.

You must include the result obtained of this program in both the file in which you make the solution and in the report.

The Exercise 2 section of the report must contain:

- After coding the previous MIPS program using the Z80 instructions implemented in Exercise 1, using the same names for the routines and the vector, you must compare both sets of instructions indicating: differences in the types of instructions, advantages and disadvantages presented by each of these instruction sets.

The name of the file with the requested functionality will be:

- **e2\_checkpoint.txt:**  
Similar to Exercise 1, a checkpoint file will be created containing: a) the required **assembly program**, according to the given requirements and in the correct format for the WepSIM simulator; b) A Wepsim **recording** to prove that the **microcode** of exercise 1 is correct. The expected results before calling the function **addv** must be indicated, and after the call the obtained results must be explained.

**Appendix 1** summarizes the steps for making a recording in Wepsim.

**Appendix 2** summarizes the steps to create a checkpoint file.

## Practice delivery procedure

The delivery of practice two will be made through the authorized delivery points in Aula Global. Delivery via email is not allowed.

The deadline for delivery is **December 2st, 2019 at 11:50 p.m.**

**It is possible to deliver as many times as you want within the given deadline, the registered version of your practice is the last one delivered.** The final assessment of the practice will be the assessment of the content of the latest submitted. Always check what you are delivering.

### Delivery format:

A deliverer point will be enabled to deliver the memory through Turnitin (in Aula Global). A pdf file must be delivered. The name of this file must have the next format **AAAAAAAAA\_BBBBBBBBBB.pdf** where A...A and B...B are the NIAs of the group members.

A second deliverer point will be enabled to deliver a single compressed file (zip format). The name of this file must have the following format **AAAAAAAAA\_BBBBBBBBBB.zip** where A...A and B...B are the NIAs of the group members.

The **zip** file must contain only the following files (without subdirectory):

- **e1\_checkpoint.txt**
- **e2\_checkpoint.txt**
- **report.pdf**
- **authors.txt**

The authors.txt file will contain a line for each member of the group with the NIA and the group to which he/she belongs.

**All files will be ASCII text type except the report, which will be in pdf format.** The memory you deliver in this section must be the same as that delivered through Turnitin. Checks that each file is delivered correctly.



## Important aspects to considered

You should carefully check that all the requirements indicated in this statement are fulfilled.

**The names of the files and subroutines must be like they are indicated in this statement.** If you do not respect these names your practice will **have a score of 0**.

### ***General rules***

- 1) The delivery of the practice will be made through the authorized points in Aula Global. Delivery via email is not allowed.
- 2) The delivery will be made within the period given in delivery points in AulaGlobal. It is possible that for a delivery point in Aula Global the deadline for a delivery at 24:00 ends 10 minutes before. Please check the support of Aula Global.
- 3) Special attention will be paid to detect features copied between two practices. In case of finding common implementations in two practices (or similar contents in memory), both will obtain a grade of 0 (zero).

### ***Practice´s code***

- 1) A valid implementation that minimizes the number of clock cycles will be assessed.
- 2) A program not properly commented will get a score of 0. Comments should seek to describe each step that is 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 properly. For this reason, you will have to carry out those tests that guarantee the correct functioning of the practice.
- 4) The code to be delivered must not contain messages printed on the screen or 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 of the practice***

- 1) **The report length must not exceed 12 pages** (cover and index included).
- 2) **The report will be delivered in pdf format with selectable text (the report must allow Turnitin to perform copy control in memory).**
- 3) The report (a single document) must contain at least the following sections:
  - Cover page, where the authors appear (including full name, NIA, class group to which they belong and email address).
  - Contents index.
  - Contents requested in different exercises (one section per exercise)
  - Conclusions and problems encountered. Include a summary with the number of hours dedicated to practice.

**NOTE: DO NOT NEGLECT QUALITY OF THE REPORT OF YOUR PRACTICE.**

Approving the report is as essential to approve the practice. If when we evaluate the report of your practice, it is considered that it does not reach a minimum quality, your practice will be suspended

## ***Practice evaluation***

The evaluation of the practice will be divided into two parts:

- **Code (7 puntos)**
- **Report (3 puntos)**

### **VERY IMPORTANT:**

- 1. Although the score can be divided into sections, the correction of the practice will be carried out at a global level. this includes:**
  - a. If an exercise is not delivered your score in all practice will be 0.**
  - 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 made to avoid total or partial copies in delivered work, this includes**
  - a. In case of finding common implementations in two practices (or similar contents in the report) it will be understood that the practice has been copied and both will obtain a grade of 0.**
  - b. If code fragments obtained directly from the Internet are found, it will be understood that the content has been copied and the practice will have a rating of 0.**
- 3. You must respect the format and names requested; this includes:**
  - a. If you do not respect the delivery format (for example, do not respect the name of the requested files, deliver a .rar file, deliver the files in a directory, etc.) the note will be significantly reduced.**
  - b. The text of the memory file (memory.pdf) must be selectable and copiable. 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 zero for all practice.**

## Appendix 1: Recording a session in Wepsim

In order to verify that the code works correctly with the given specifications, it is necessary to carry out different tests. Because reprogramming millions of processors is very expensive, it is necessary to run a good battery of tests.

The first tests must be carried out step by step to know how circuitry works. A test consists of executing a test code and checking that at different execution points the state is correct.

Once the operation of the different elements in a microprogram is understood, it is possible to continue testing in a more automated way. WepSIM allows to record most of the interactions with the interface, as well as to add comments in different points of the execution. Before making a recording, it is advisable to first rehearse the steps you want to record.



### Recording the working session

|   |  |
|---|--|
| To work with recordings, press the 'RecordBar' button at the top of the screen. |  |
| The toolbar for recordings will appear below the screen.                        |  |
| To start recording a session, press the 'Record' button.                        |  |
| When you finish recording the work session you should press the stop button     |  |

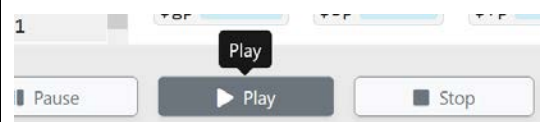

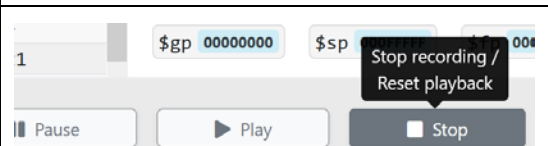
### Remove a session work

|  |  |
|--|--|
| If you want to delete the recording (to make another one) press the 'Reset' button. A dialog box will appear asking you to confirm that you want to delete ('Reset' button again). |  |
|--|--|

## Add a comment to a record

|   |  |
|---|--|
| While recording, it is possible to add a comment to the recording. To do this, press the 'Comment' button.                                    |  |
| A dialog box appears to fill in the data associated with the message: title, content of the message and duration (in seconds) of the message. |  |

## Working Session Playback

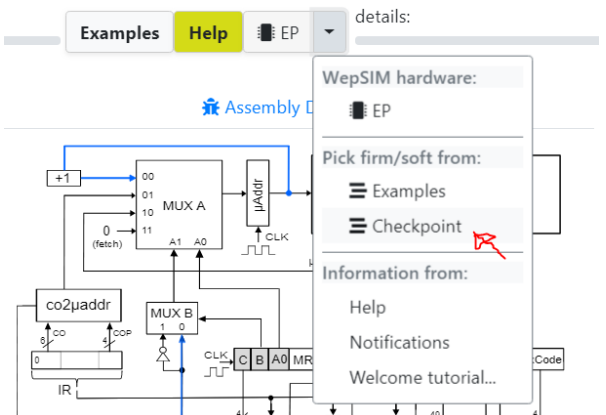
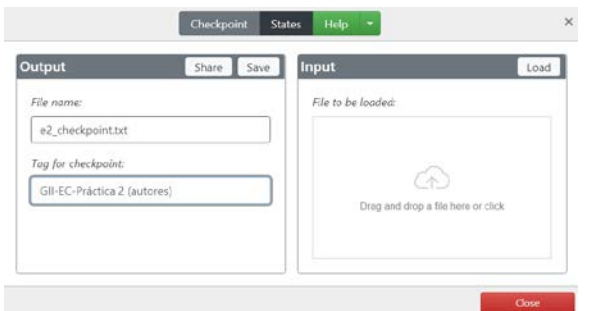

|   |  |
|---|--|
| To play back a recording, press the 'Play' button.  |    |
| You can pause session playback with the 'Pause' button. When you press the 'Pause' button again, the execution continues. |   |
| To stop and return to the initial step, press the 'Stop' button.  |  |

## Appendix 2: Creating a checkpoint file in Wepsim

WepSIM allows you to save the entire work session in a single file. This session can include the requested microcode, the assembly code, the states at different execution points and a recording of the working session. In this way it is more agile to continue the work or to share this work between members of the practice group.

Checkpoints must be used for this. The steps for saving a checkpoint are shown below.

### Save a *checkpoint*

|  |  |
|--|--|
| <p>In the run mode menu, select the "Checkpoint" option.</p>   |  The screenshot shows the WepSIM software interface. At the top, there are tabs for 'Examples', 'Help', and 'EP'. The 'Help' tab is active, and a dropdown menu is open, showing options: 'WepSIM hardware: EP', 'Pick firm/soft from: Examples, Checkpoint', and 'Information from: Help, Notifications, Welcome tutorial...'. A red arrow points to the 'Checkpoint' option in the 'Pick firm/soft from' list. In the background, a circuit diagram is visible with components like 'MUX A', 'MUX B', 'co2uaddr', and 'IR'. |
| <p>Enter the name of the file in the "File name:" field and then press the "Save" button to save the file.</p> |  The screenshot shows a 'Checkpoint' dialog box. It has two main sections: 'Output' and 'Input'. In the 'Output' section, the 'File name:' field contains 'e2_checkpoint.txt' and the 'Tag for checkpoint:' field contains 'GII-EC-Práctica 2 (autores)'. There are 'Share' and 'Save' buttons. In the 'Input' section, there is a 'File to be loaded:' field and a 'Load' button. A red arrow points to the 'Save' button. The dialog box has a 'Close' button at the bottom right.   |
| <p>Check that the file has been saved correctly and contains everything requested in the statement.</p>        |  The screenshot shows the WepSIM software interface. At the top, there are tabs for 'Examples', 'Help', and 'EP'. The 'EP' tab is active, and a dropdown menu is open, showing options: 'WepSIM hardware: EP', 'Pick firm/soft from: Examples, Checkpoint', and 'Information from: Help, Notifications, Welcome tutorial...'. A red arrow points to the 'Checkpoint' option in the 'Pick firm/soft from' list. In the background, a circuit diagram is visible with components like 'MUX A', 'MUX B', 'co2uaddr', and 'IR'.  |