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 2023/2024

Version 2.2

Contents

Goals of the assignment

The **main goal** of this assignment is to learn how to design an instruction set for a computer (according to given requirements).

The primary bits of knowledge that will be practiced are microprogramming, assembly programming, and information representation. You will also learn to evaluate design alternatives and work in a team.

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 interact with the concepts and knowledge indicated above, thus completing the exercises of the subject and improving the knowledge of the overall functioning of a computer.

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

Exercise 1

We are involved in a project of our company that deals with complex numbers where the real and imaginary parts of the numbers will be considered as integers represented in two's complements.

The company uses a subset of RISC-V instructions for that project to work with complex numbers. Still, it is considering whether to include a minimal extension to RISC-V to cope with complex numbers. Complex numbers will be represented by either two registers (one for the real part and another for the imaginary part) or two consecutive memory locations (the first location will store the real part and the next consecutive one the imaginary part of the number).

Instruction	**Format**	**Associated functionality**	**SR Register**
la R_{R1}, U32	CO (31-26): 010001 R_{R1} (25-21) U32 (63-32)	BR[R_{R1}] ← U32	No updated
sc R_{R1}, R_{R2}, (R_{R3})	CO (31-26): 010010 R_{R1} (25-21) R_{R2} (20-16) R_{R3} (15-11)	MEM[R_{R3}+0] ← BR[R_{R1}] MEM[R_{R3}+4] ← BR[R_{R2}] **R_{R3} does not change during sc**	No updated
lc R_{R1}, R_{R2}, (R_{R3})	CO (31-26): 010011 R_{R1} (25-21) R_{R2} (20-16) R_{R3} (15-11)	BR[R_{R1}] ← MEM[R_{R3}+0] BR[R_{R2}] ← MEM[R_{R3}+4] **R_{R3} does not change during lc**	No updated
addc R_{R1}, R_{R2}, R_{R3}, R_{R4}	CO (31-26): 010100 R_{R1} (25-21) R_{R2} (20-16) R_{R3} (15-11) R_{R4} (10-6)	BR[R_{R1}] ← BR[R_{R1}] + BR[R_{R3}] BR[R_{R2}] ← BR[R_{R2}] + BR[R_{R4}]	Updated (real part)
mulc R_{R1}, R_{R2}, R_{R3}, R_{R4}	CO (31-26): 010101 R_{R1} (25-21) R_{R2} (20-16) R_{R3} (15-11) R_{R4} (10-6)	BR[R_{R1}] ← BR[R_{R1}] * BR[R_{R3}] **-** BR[R_{R2}] * BR[R_{R4}] BR[R_{R2}] ← BR[R_{R1}] * BR[R_{R4}] **+** BR[R_{R2}] * BR[R_{R3}]	Updated (real part)

beqc R_{R1}, R_{R2}, R_{R3}, R_{R4}, S6	CO (31-26): 110100
	R_{R1} (25-21)
	R_{R2} (20-16)
	R_{R3} (15-11)
	R_{R4} (10-6)
	S6 (5-0)
		IF (R_{R1} == R_{R3}) AND (R_{R2} == R_{R4}):
		PC ← PC + S6
			No updated

call U20	CO (31-26): 100001
	U20 (19-0)
		BR[ra] ← PC
		PC ← U20
			No updated

ret	CO (31-26): 100010	PC ← BR[ra]	No updated

hcf	CO (31-26): 100011	PC ← 0x00
		SR ← 0x00
			Updated to zero

Table 1.- Extended RISC-V instruction set.

We are asked to **design, implement, and test** the instruction set shown in Table 1 for the WepSIM simulator to evaluate a possible extension of RISC-V for complex numbers.

All instructions will be encoded using 32 bits (except "la" which is 64 bits).
The notation used in Table 1 for immediate values is as follows:

- U32 refers to a 32-bit unsigned integer; U20 refers to a 20-bit unsigned integer.
- S6 refers to a 6-bit signed integer value in two's complement.

For the "S6" value, you must perform a sign extension, while in the "U32/U20," no sign extension must be done (it is filled with zeros on the left, on the most significant part).

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

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

- R_{R*} will 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_{R1}] shall be used to indicate the contents of the R_{R1} register.

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

The following table describes 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 at least two names, indicated in the RISC-V registers column of Table 2. There are four registers of particular interest:

- The stack pointer 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.

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

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

The section in the report for Exercise 1 should contain:

- A table with four columns (and as many rows as instructions have been requested in the statement):
    - The first column includes the name of the instruction.
    - The second includes 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, all the elementary operations necessary for the execution of the instruction (in RT language).
    - The third column includes the control signals to be activated in each instruction cycle.
    - The fourth includes the design decisions your team has made for this instruction.

The file with the requested functionality is:

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

Exercise 2

To test the instructions, you can code the programs you consider appropriate.

However, the company requests us to make a specific assembler program to demonstrate the advantages of using the extension requested in Exercise 1, which allows working with complex numbers based on integers in two's complement.

For the demonstration, we are going to use a program whose skeleton is the following:

```
.data
  a: .word 35, 15
  b: .word 10, 20

.text
 no_ext:
          # To implement with RISC-V instructions (without extension)
          #  if (a == b):
          #      return a * b;
          #  else
          #      return a + b;
           ret
with_ext:
          # To implement with RISC-V instructions (with extension)
          #  if (a == b):
          #      return a * b;
          #  else
          #      return a + b;
           ret

main:    rdcycle s0
         la a0, a
         la a1, b
         call with_ext
         rdcycle s1
         sub s1 s1 s0

         rdcycle s0
         la a0, a
         la a1, b
         call no_ext
         rdcycle s2
         sub s2 s2 s0

         hcf  # https://en.wikipedia.org/wiki/Halt_and_Catch_Fire_(computing)
```

This program has three subroutines or functions:

- "no_ext" performs the steps requested in the comments using only the RISC-V instructions (except ret to return to the caller).
- “with_ext" instead uses only the instructions of the extension developed in Exercise 1 to do the same functionality.
- "main" is in charge of calling both subroutines. Before calling, it stores the number of clock cycles used since the last reset in register s0 and saves the number of clock cycles again at the return of the corresponding subroutine. It subtracts both values (before calling and after calling) to know the number of clock cycles consumed and stores this value in register s1 for the first case (with_ext) and in register s2 for the second (no_ext).

To prepare for the demonstration, you must:

a) Code the above test program completely.

b) Measure and compare the number of clock cycles required in the different cases by completing the following table:

	Clock cycles without extension	**Clock cycles with extension**	**Improvements (%)**
A == B	¿?	¿?	¿?
A != B	¿?	¿?	¿?

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

The section of the report for Exercise 2 should contain:

- After coding the functions requested in the RISC-V instructions of Exercise 1, you have to compare the original RISC-V instruction set with the extensions proposed in Exercise 1: differences, advantages, and disadvantages and proposed improvements.
  You must include the table with the measurement results of the clock cycles requested.
  You should also answer the following question with a brief reasoning of your answer: If you are going to work with complex numbers, is it worth having the extension of Exercise 1?

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, **<u>including the microcode of Exercise 1 with fetch and register section,</u>** 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

Assignment 2 will be submitted electronically through Aula Global, for which two links associated with said practice will be enabled.

Please remember that:

- The assignment can be done in groups of **up to two students** (no more).
- The deadline for submission is **December 3rd, 2023, 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 submit a report 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 group members. **<u>The report can only be delivered once via Turnitin.</u>**

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 group members.

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 <u>only</u> its corresponding NIA.

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

Submitting the zip file as many times as you want within the given time frame **is possible**. The only recorded version of your assignment is the latest submitted. The content of this latest submission is the one evaluated.

Important aspects to keep in mind

It must carefully check that all the requirements 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 names of the functions/subroutines, files, etc., must be those indicated in this statement. Not respecting these names will mean a zero in the grade of the corresponding section, so adding to the list of checks of the names is recommended.

General rules

1) Submission will be made through the authorized deliverers. Submissions via email are not allowed.
2) Submission will be made within the period given by the deliverers. It is possible that for an 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 assignments. In case of detecting plagiarism between two assignments (or reports), all the teams involved (copied and copiers) will obtain a rating of 0 (zero) and a plagiarism file can be opened depending on the severity. Copying portions of the Internet or practices of other courses is also considered plagiarism, and all teams 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 TOC 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 employed to complete the assignment.

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

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

Scoring

The score of the assignment will be distributed between the code and report delivered as follows:

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

VERY IMPORTANT:

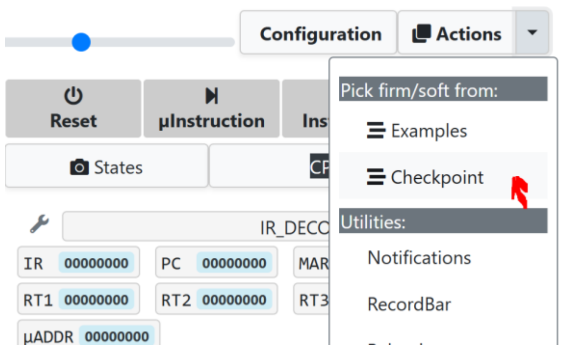
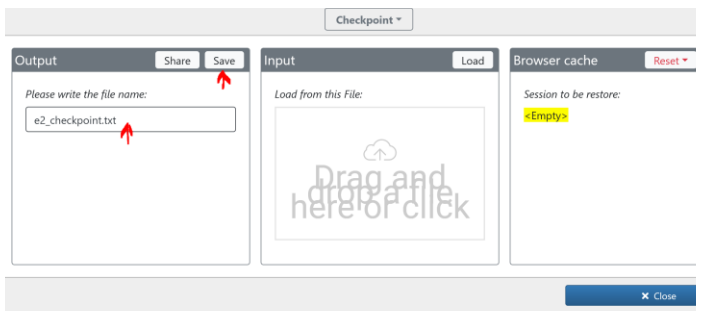
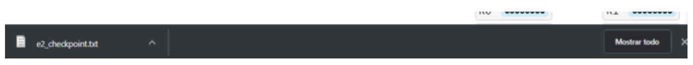
1. **Although the score can be divided into sections, the correction of the assignment will be carried out at a global level, that is, this includes:**
   a. **If an exercise is not submitted, the grade of the entire assignment will be zero.**
   b. **If a serious misconception is detected in practice (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 zero rating (grade).**
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 file's text 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 you 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 team.

Below are the steps to save a *checkpoint*.

Save a *checkpoint*

In the run mode menu, please select the *"Checkpoint"* option.	 Configuration Actions Reset µInstruction States IR_DECO IR 00000000 PC 00000000 RT1 00000000 RT2 00000000 µADDR 00000000 Pick firm/soft from: Examples Checkpoint Utilities: Notifications RecordBar
Please enter the name of the file in the "File name:" field and then press the "Save" button to save the file.	 Checkpoint Output Share Save Please write the file name: e2_checkpoint.txt Input Load Load from this File: Drag and drop or click here Browser cache Reset Session to be restore: <Empty> Close
Please verify that the file has been saved correctly and contains everything requested in the statement.	 e2_checkpoint.txt Mostrar todo