Presentation guide for Unit 3
Fundamentals of Assembly Programming

The objective of this unit is to understand the main aspects related to assembly programming. Assembly language is a programmer-readable language that constitutes the most direct representation of the specific machine code of an architecture or computer.

It is also pursued that the student learns to make small programs written in assembly. This unit is composed by the following blocks:

1. Basic fundamentals of the assembly programming.
2. RISC-V 32 assembler, memory model and data representation.
3. Instructions format and addressing modes.
4. Procedure calls and battery use.

As an example of assembly language, we will use the RISC-V 32 assembler, a 32 bits RISC processor, which has a very easy to use instruction set. It is a very didactic language that does not use complicated instructions. Besides, there are several simulators that can be used to make programs in this assembly language.

1. Assembly Programming Basics

This first block describes the main features included in an instruction set:

- Operands that can reside in registers, in memory or in the instruction itself.
- Memory addressing. How are the mechanisms used to access the data stored in the memory through the machine instructions.
- The addressing modes, which specify the place and the way to access the operands referred to in the machine instructions.
- The type and size of the operands.
- Different operations they perform: arithmetic, logic, transfer, etc.
- Flow control instructions, which allow conditional and unconditional jumps and calls to functions and procedures.
- The format and codification of the instruction set.

Then, we describe the main types of instructions, showing examples from the RISC-V 32 assembly: transfer instructions, arithmetic, logic, shifts, comparison, flow control, conversion, input/output and system calls.

Throughout the unit, it is shown how the translation of small fragments written in a high-level language is done to the RISC-V 32 assembly language. Specifically, it is described how the main flow control structures of high-level languages like C or Java are translated to assembler: while, do-while loops and if-else statements.

2. Memory model and data representation

This second part presents the memory model of a computer and the instructions offered by the RISC-V 32 to access memory to different types of data: bytes, half words and words. Next, it is shown how to represent in assembler different types of data from high level languages: boolean, characters, integers, real, vectors, matrices and strings. It is also shown the RISC-V 32 instructions that allow working with floating point numbers, both single and double precision.

3. Instruction format and addressing models

This part starts introducing the difference between an assembly instruction and a pseudo-assembly instruction. An assembly instruction corresponds to a machine instruction, while a pseudo instruction is an instruction that can be used in programs written in assembly, but it doesn't have a direct correspondence with a machine instruction. The concept of addressing mode is presented below. The addressing mode is the procedure to determine the location of an operand, a result or an instruction.

In this part of the unit, we also present the way to design different sets of instructions, the difference between CISC and RISC computers. Finally, the different execution models of a computer are presented. The execution model indicates the number of addresses and type of operands that can be specified in a machine instruction.

4. Procedure calls and stack usage

This last block is dedicated to procedure calls and stack usage in assembly. This part starts illustrating the process of calling functions or procedures in the RISC-V 32 and the machine instructions available to invoke functions and return from them. It then describes the use of the stack on the RISC-V 32 and the importance it has in the whole function call process. It also describes the convention in the use of registers in the RISC-V 32 and the convention of parameter passing in this RISC-V32. It is described the stack frame or activation register, which is the mechanism that compilers use to activate procedures or functions in high-level languages. It is illustrated in a detailed way all the necessary process to create the stack frame associated to a function, indicating the steps that the subroutine that makes the call must do and the steps that the called subroutine must do. Finally, it is described all the process of translation and execution of programs, describing in a summarized way the functions that the compiler, the assembler, the linker and the loader do.

Material

As material associated with this unit is included the theory material and a collection of exercises proposed and resolved on the aspects discussed in the unit.

Recommended bibliography

- “Problemas resueltos de estructuras de computadores” (GARCIA CARBALLEIRA, Félix et al.).
- “Computer organization and design. The hardware/software interface” (PATTERSON, David, et al).
- “Computer Organization and Architecture” (STALLINGS, William).