# Introducing computer architecture education in the first course of computer science career

José R. Arias, Daniel F. García

*Abstract*— **The introduction of computer architecture in computer science studies has created a demand for a highly simplified architecture and graphical tools to illustrate its operation. This paper presents the approach followed at the University of Oviedo. The main concepts introduced to the students are relative to RISC instruction sets, data-path architecture, control steps within instructions and microinstructions.**

*Keywords*—**Computer architecture education, data-path control signals, CPU operation simulation**

## I. INTRODUCTION

TO introduce the basic concepts concerning the internal architecture and operation of a CPU to first-year students of computer science, the utilization of a simplified model of a CPU, jointly with proper simulation and visualization tools is essential. In this way, students master the basic concepts of computer architecture easily.

The approach to introduce computer architecture education presented in this paper is conditioned by the prior knowledge of the students. When the students are faced with computer architecture in the first year, they have only received a 14-hour elementary course on digital systems. So, the complexity of the internal architecture of the CPU must be simplified to a maximum in order to concentrate the efforts of teachers and students on the aspects relative to the operation of the architecture. The *Simple CPU* and its simulator are used in the first term in the course called ”Computer Structure” in the Computer Science Degree of the University of Oviedo.

To fulfill the global or general objective of teaching the architecture and operation of a CPU, several stages are followed. Firstly, the students study the very simple instruction set of Simple CPU, of RISC type. The concept of binary codification of the instructions is also illustrated. The introduction of the concepts related to codification is facilitated using a fixed-length instruction format, each instruction occupying one word in the memory. Of particular importance to the students is the codification of the jump instructions. In this first stage the students learn the external vision of the architecture, that is, from the point of view of a low-level programmer or a compiler designer.

In the second stage the simplest architecture capable of supporting the instruction set is presented. In this stage the students learn the internal view of the architecture, in particular a static internal view. This is the focus of the designer of the CPU.

The third stage is devoted to illustrating the global operation of the architecture, with special emphasis on internal operation. The break-down of instructions into several steps, as conditioned by the internal architecture selected, is explained. This introduces the students to the concept of clock-cycle and the sequences of control signals for each instruction. The objective is to show the operation of the data-path, without explaining the construction of related elements such as the arithmetic-logic operators, the registers or the control unit. At this stage students learn about the operation of the architecture, obtaining a dynamic internal view.

Both authors are with the Universidad de Oviedo - Departamento de Informática Campus de Viesques, 33205 Gijón SPAIN e-mail: arias, daniel@atc01.etsiig.uniovi.es

TABLE I
REGISTER SET OF THE *Simple CPU*

| Reg. | Type | Description |
|---|---|---|
| R0-R7 | A | General Purpose Registers |
| PC | NA | Program Counter |
| SR | NA | Status Register |
| TMPI | NA | ALU Temporal Input Register |
| TMPO | NA | ALU Temporal Output Register |
| MDR | NA | Memory Data Register |
| MAR | NA | Memory Address Register |
| IR | NA | Instruction Register |

SPIM and XPIM for MIPS [1] and Simplez/Algoritmez [2] can be considered as good examples of simulators, but these kinds of simulators only simulate the operation of the CPU at an external level. The simulation of the sequence of control signals for each instruction is accomplished by only some of the tools, and then in a very restricted manner [3]; or by focusing on the implementation of the control unit using microprogramming techniques [4]. Other software packages have been designed to explain the behavior of computer architecture from both internal and external points of view, but they use specific microprocessors, such as Z80 [5] or 68000 [6] and require specific graphic libraries. This lack of simulation tools of the internal operation of a CPU using a windows environment has motivated the construction of the CPU simulator presented in this work.

## II. THE *Simple CPU*

The *Simple CPU* is a CPU of 16 bits (for data and addresses). This format allows the operation of a complete CPU to be fully illustrated, while maintaining the manageability of the length of the numbers. It is based on a single internal bus which connects 8 general purpose registers (R0-R7) to the arithmetic-logic unit (ALU). The registers of the *Simple CPU* are classified as being available (A) or not available (NA) to the low-level programmer and are summarized in table I.

The *Simple CPU* is seen to lack elemental instructions, such as multiply or divide, as well as all the mechanisms relative to subroutines and stack management. These instructions have been deliberately omitted in order to maximize the simplicity

TABLE II

CONTROL SIGNALS FOR INSTRUCTION FETCHING AND PC UPDATING

| Step | Control signals |
|---|---|
| 1 | PC-IB, IB-MAR, READ, TMPI_CLR, CARRY_IN, ADD, ALU-TMPO |
| 2 | TMPO-IB, IB-PC |
| 3 | MDR-IB, IB-IR |

TABLE III

SPECIFIC CONTROL SIGNALS FOR INSTRUCTION ADD R0,R1,R2

| Step | Control signals |
|---|---|
| 4 | R1-IB, IB-TMPI |
| 5 | R2-IB, ADD, ALU-TMPO |
| 6 | TMPO-IB, IB-R0, END |

of the CPU. Furthermore, in the first term of the course, the students do not have the concept of subroutine in high-level language, so there is little point in explaining how the architecture supports the subroutines. The *Simple CPU* uses only three addressing modes: the immediate mode, which allows an 8-bit constant to be loaded in a register, the register mode and the indirect memory addressing through a register. The immediate and memory addressing are supported only by the MOV instruction.

To execute each instruction, the *Simple CPU* requires several steps, using one clock cycle per step. The first three steps are devoted to fetching the instruction from memory and updating the PC. They are identical for all the instructions. The fourth and successive steps are devoted to carrying out the operations defined by the instructions, so they are different for each instruction. In table II, the control signals of the first three steps are presented, and in the table III the specific control signals for the `ADD R0,R1,R2` instruction is shown. In the tables of control signals for each instruction, only the signals that are activated in each step appear. The following nomenclature is used:

- The signals that control the actions to be carried out by one element of the CPU, or by an external device have a single name. For example, ALU_OP to define the operation to perform in the ALU, TMPI_CLR to clear the register TMPI, or READ to start a memory access cycle to read data.
- The signals that control the input and output of data in registers have a name composed of two parts separated by a hyphen. The first part indicates the name of the element that provides the data, and the second part the element that receives it. For example, PC-IB enables the connection of the register PC to the internal bus, whereas when IB-MAR is activated, the register MAR captures the data present in the internal bus.

The reduced number of instructions of the *Simple CPU* makes the design of a hardwired or microprogrammed control unit to implement the sequences of control signals for each instruction, an affordable educational project.

The *Simple CPU* follows the ”load-store” operation principle and never operates directly with memory operands. The data must be loaded in registers, operate within them and store the results in the memory. This operational design of the *Simple CPU* has been selected to introduce the students to modern architectures. Another concept introduced with the *Simple CPU* is the orthogonality of the architecture, in which all the registers can be used with any instruction and for any purpose.

TABLE IV

TRANSLATION AND CODIFICATION PHASES

| **Code** | **Assembly Language** | **High-level Language** |
|---|---|---|
| 5800 | CLR R0 | Tot := 0 |
| 2106 | MOVL R1, 06 | NItems := 6 |
| 2900 | MOVH R1, 00 | |
| 0F40 | M: MOV R7, [R2] | REPEAT |
| 4007 | ADD R0, R0, R7 | Tot := Tot + List[I] |
| 5280 | INC R2 | I := I + 1 |
| 5440 | DEC R1 | NItems := NItems - 1 |
| EFFB | BR_NZ M | UNTIL (NItems = 0) |
| 1300 | MOV [R3], R0 | Result := Tot |

⇐ **Codification** ⇐ **Translation**

## III. TYPICAL PRACTICE USING A SIMULATOR OF THE *Simple CPU*

In order that the students learn the operation of the *Simple CPU*, the design and execution of a small program is proposed. The proposed program accumulates the sum of a list of values storing the result in one word of memory.

### *A. Compilation or Translation Phase*

The students start the practice by representing the solution to the problem in a high-level structured language, such as PASCAL. Next, they translate the algorithm into a program in assembler language using the instruction set of the *Simple CPU*. The objective of this task is that the students perceive the semantic gap between the language used to describe the problems to the CPU and the native language used by the CPU itself. In this phase of the practice the students operate as human compilers.

### *B. Assembly or Codification Phase*

At this point the codification phase starts. In this phase the students work manually in the same way an assembler program would perform automatically. One of the main objectives during this phase is the consolidation of the concept of symbol, mainly used as address labels, working on their manual resolution when they are used in branch and jump instructions. Translation and codification phases are shown in table IV.

### C. Load Phase

After the program has been codified as hexadecimal numbers, the students write it in a text file, specifying the initial address in which to store the program in memory and the address of the first executable instruction in the header of the file.

In the *Simple CPU* all the jumps and branches are relative to the program counter. Furthermore, all the references to the data stored in the memory are carried out through registers. There is no absolute addressing, neither for instructions nor for data. This allows the direct loading of data and code in any part of the memory without recalculating addresses. This simplicity in the design of the Simple CPU avoids the introduction of complex concepts, such as code relocation, which would be incomprehensible in an introductory course in computer architecture.

### D. Execution Phase

At this point of the practice the students start the execution phase of the program, learning how the architecture executes the instructions.

The execution can be traced in two ways: instruction to instruction, or step to step within each instruction. In Figures 1 to 4, the sequence of steps carried out by the *Simple CPU* to execute the instruction `MOV R2, R5` is shown. The main goal of this phase is to introduce the students to the concepts of control step and microinstruction.

The first three steps are the same for all the instructions of the *Simple CPU*. They implement the tasks of instruction fetch and update the program counter. The next steps are specific for each instruction. In the simulator, the active elements or modified data in each step are remarked with a black background and the signals activated by the Control Unit (CU) of the *Simple CPU* in each step are presented in the white box of the CU.

## IV. CONCLUSIONS

In this paper a simple but effective approach with its supporting tool to introduce the students of computer science to the area of computer architecture has been presented.

The method is highly effective, because it allows students to assimilate the concepts of instruction set, addressing modes, the internal architecture of a CPU, the sequence of steps of the instructions, etc., in only 14 hours of theory and 6 of practice.

## REFERENCES


[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design. The Hardware/Software Interface*, Morgan Kaufmann, 1994.

[2] G. Fernandez, *Conceptos basicos de Arquitectura y Sistemas Operativos*, Sistemas y Servicios de Telecomunicacion, 1994.

[3] S. Scott, "Tisc, tiny instruction set computer," Tech. Rep., University of Arkansas, 1995.

[4] H. B. Gumm and M. Perner, "Microcode simulator," ASK (Akademische Software Kooperation) of the University of Karlsruhe, 1995.

[5] H. B. Diab and I. Demashkieh, "A computer-aided teaching package for microprocessor systems education," *IEEE Transactions on Education*, vol. 34, no. 2, 1991.

[6] W. D. Henderson, "Animated models for teaching aspects of computer systems organization," *IEEE Transactions on Education*, vol. 37, no. 3, 1994.


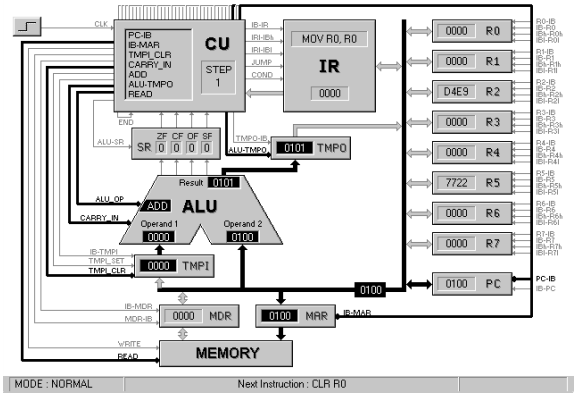


Fig. 1. Step 1: Address the new instruction

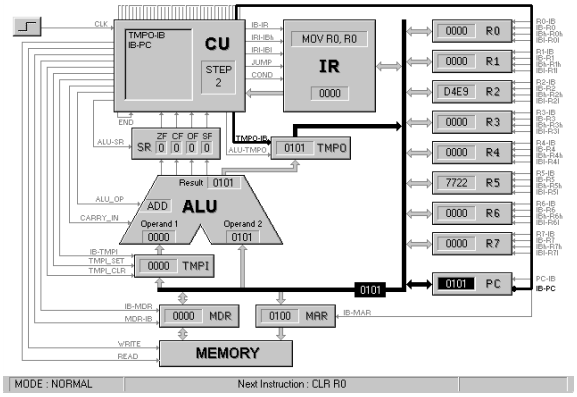


Fig. 2. Step 2: Increment the PC

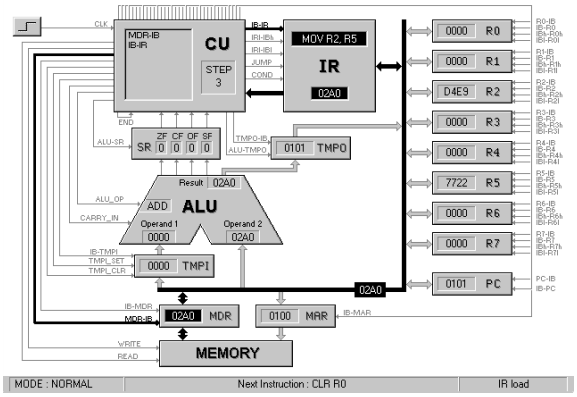


Fig. 3. Step 3: Load new instruction in register IR

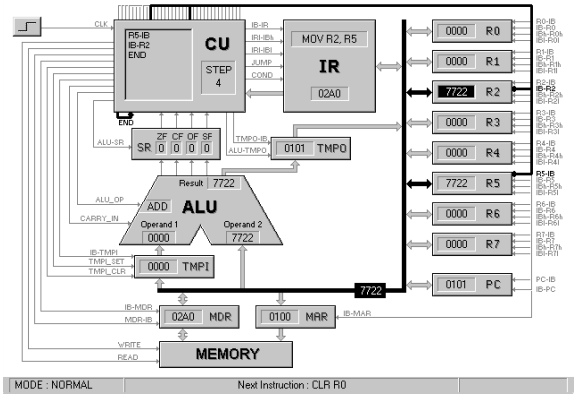


Fig. 4. Step 4: Transfer R2 content to register R5