ISSN 2007-9737

Design of a General Purpose 8-bit RISC Processor for Computer Architecture Learning

Antonio Hernández Zavala[1], Oscar Camacho Nieto[3], Jorge A. Huerta Ruelas[1], Arodí R. Carvallo Domínguez[2]

[1] Instituto Politécnico Nacional, Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada, Mechatronics Department, Querétaro, Mexico

[2] Instituto Politécnico Nacional, Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas, Engineering Department, Mexico City, Mexico

[3] Instituto Politécnico Nacional, Centro de Innovación y Desarrollo Tecnológico en Cómputo, Mexico City, Mexico

{anhernandezz, ocamacho, jhuertar, acarvallo}@ipn.mx

Abstract. Computers are becoming indispensable for manipulating most everyday consumer products, ranging from communications and domestic electronics to industrial processes monitoring and control. High performance computer design is not only subject to the technology used for its implementation, it is also a matter of efficient training. The skills that must prevail in a good computer designer come from the type of courses taken and the tools employed during them. This work shows the design of an 8-bit RISC soft-core processor dedicated to a complete understanding of computer architecture. We consider this Processor an effective hands-on training solution for the comprehension of a computer from its lowest level up to testing.

Keywords. Computer architecture, digital design, digital logic, microprocessor, programmable logic devices, training system.

1 Introduction

Nowadays, computers perform required calculations for almost all electronic devices in the market. Computer hardware is build based upon binary operations to satisfy a set of mathematical, data transfer, and control instructions. The basic instructions realized directly by hardware are known as the processor instruction set. It is used for the creation of structured software programs for data manipulation in problem solving.

Computer types differ in the amount of information they can process. As information is grouped into 8-bit data packages, there are 8, 16, 32, and 64-bit computers. Today's devices such as desktops, laptops, notebooks, or tablets are 32 and 64-bit computers. For specific tasks in communication, automotive, and industrial appliances, 16-bit computers are commonly used. However, 8-bit computers are the number one sales processors in the world as they are used as controllers for simple computational tasks. According to the divide and conquer principle, a common personal computer is divided into smaller ones (commonly 8-bit) which share information with the main computer (32 or 64-bit). An 8-bit processor commonly manipulates the drivers for almost every component card inside a computer. It is noteworthy to mention that 8-bit type processors are widely used in home appliances and industrial specific systems.

In order to train engineering students in the computer architecture field, it becomes necessary to have the correct tools. In this sense, there have been different approaches in which students use hardware or software tools with the purpose of understanding the processor architecture.

Our purpose is to provide students with a flexible but consistent tool for understanding computer architecture from its basics. This work

presents the design of an 8-bit data width processor with the Harvard architecture and the Reduced Instruction Set (RISC) for educational purposes. It includes a basic instruction set and all the functional blocks such as program and data memories, general-purpose registers, and a simple Arithmetical Logical Unit (ALU) for basic operations. Its operating mode is multi-cycle execution due to its easy visualization and effectiveness. The processor implementation is in a Xilinx Spartan-3E FPGA.

The organization of the paper is as follows: Section 2 explains current methodologies used in computer architecture courses. Section 3 introduces some general and basic concepts concerning computer architecture. Section 4 presents the processor development, describing the operation of each functional unit. Finally, Section 5 presents the results from tests and compilations, along with a comparative discussion.

2 Computer Architecture Education

It is noteworthy that the educational processors theme has been widely explored since Edmund Berkeley and Robert Jensen made the first 2-bit desktop computer implementation using relays in 1950 [1]. Since then, there have been five different approaches used to explain computer architecture:

1. **Paper.** This was the first approach to teach computer architecture because of the limited computer availability. It was based on the use of pencil and paper to understand the instruction execution flow by means of drawings. Finally, with high limitations, students were able to understand the general functioning of a computer.
2. **Simple Hardware.** This type of computer is constructed using available technology. There are relay and TTL versions that can show the physical interaction among the computer modules. Its main disadvantage is the time invested in the wiring process.
3. **Simulator.** This is a software-based simulator that allows the execution of a code to see how each functional unit works in a computer. The disadvantage is that it does not allow interaction with physical devices.
4. **Hardware Description Languages HDL.** Each functional unit is programmed using an HDL (Verilog or VHDL) which is compiled to create a logical array for the processor description. Next step is to download it into a device. This approach is fast to carry out and implement.
5. **Logic blocks.** This approach is also intended for programmable logic devices based in the use of basic logical gates. The construction of each functional unit is by means of the union of pre-constructed blocks. This approach allows students to fully understand a processor from its basics along with its integration and the synchronization of the functional blocks.

In the paper category, Stuart Madnick designed the “Little Man Computer LMC” in 1965 [2] to teach students. Nowadays, there is a modern version of his 3-digit Von Neumann computer which has all the basic computer features and it can be programmed either in machine code or in assembly code. These new versions include a graphical user interface and a compiler. Another paper approach is the “CARDIAC” computer developed by David Hagelbarger and Saul Fingerman from Bell Laboratories in 1968 [3], it is a processor made of cardboard which nowadays is implemented in a simulator software. Finally, the “Paper Processor” developed by Saito Yutaka in 2010 [4] is another tool to understand processor behavior.

In the hardware approach there are many proposals starting with [1] which is a 2-bit relay based computer. In this approach it is common to use MSI digital components such as TTL’s to construct a computer as in the 16-bit Glen G. Langdon “SC-16” processor in 1982 [5], the 32-bit Hennessy-Patterson “DLX” processor in 1990 [6] which was based on the MIPS and is still used with TTL’s by ElAarag in [7]. The 16-bit Bradford Rodriguez “PISC” Processor in 1994 [8] and the 32-bit MIT “Beta” processor in 1997 [9] are based on the DEC computer. The hardware approach for computer architecture courses is a very good exercise to synchronize data transfer between functional blocks, although using wires to build those complex connections is extremely time-consuming.

There are several simulation tools for the computer architecture teaching approach [10-16].

Most of these are graphical environments in which the student can see all the functional modules of a processor and data transfer among them. It also allows the interaction to make variations in the testing code, registers, or memories. These characteristics are advantageous given that there is no need to have physical processors. At the same time it is its weakness, since there is no interaction with physical devices.

For the case of the HDL language approach implementations, there are also many types of didactical processors in 8, 16, and 32-bit versions [17, 23]. This option allows the physical construction of a simple computer by using FPGA devices programmed by HDL language. The advantage is that the time required to develop a processor is short, but the main disadvantage is the lack of understanding of the physical data flow induced by programming.

In the logic blocks approach, the option is to build a simple computer by using schematic mode. A graphical user interface allows gate level design by placing components in a spreadsheet. It is easy to use, and data flow between the components is straightforward. It is even more comprehensive if we can create all of the functional blocks out of logical gates. Numerous authors prefer this approach for their courses, presenting their different processor versions [24, 30]. This is the best option since it allows the creation of each functional block and testing it for further integration.

3 Computer Architecture Primer

In order to introduce some basic concepts used in the computer architecture field, this section explains the computer classification and the instruction execution.

3.1 Computer Classification

The different types of computers are classified according to the amount of information they process, the type of operations they execute, or their architecture.

The information is grouped into 8-bit data packages called bytes. A pair of bytes is called a word (16-bit); a pair of words is called a double word (32-bit); and four words are called a quad word (64-bit). This is how computers are classified according to the amount of information they can process (i.e. 8, 16, 32, 64-bit).

All computers perform data transfer instructions in order to interchange data among the different memories and peripherals. There are integer arithmetic processors, floating point processors, digital signal processors, and application specific processors. Integer arithmetic processors are best suited for general-purpose applications. Thus, it is possible to find byte, word, dword, and qword sized general-purpose processors.

According to the architecture of the functional elements, there are two architectures: the Von Neumann architecture and the Harvard architecture. The main difference is in the memory elements, given that the Von Neumann architecture shares the program and data memory in a single bus connection. The Harvard architecture has two separate memories for program and data, each with an independent bus connection. There are also architectural techniques applied to any case in order to increase the performance such as pipelining and parallelism.

According to the technology used for the construction of a processor, it can be fully personalized at the factory as an Application Specific Integrated Circuit (ASIC). It can also be implemented as a general purpose Programmable Logic Device (PLD), or it can be built as a soft core library for use among Field Programmable Gate Arrays (FPGA).

FPGA technology uses sophisticated simulation and design verification tools which enable engineers to reach new levels of complexity and robustness, while greatly reducing the time between development and utilization. It also allows integration of multiple elements into a single chip, with the capacity to add or remove modules according to future requirements.

Beside the previous classifications, there are others based on the type of instructions computers execute and the number of instructions they are able to execute. Based on the instruction types, computers are classified into three kinds: the Reduced Instruction Set Computer (RISC), the Complex Instruction Set Computer (CISC), and the Specific instruction Set Computer (SISC). General-purpose processors are in the RISC or CISC

category, while the special purpose processors are in the SISC category.

Depending on the number of instructions computers can execute, they are Single Instruction-Single Data (SISD), Single Instruction-Multiple Data (SIMD), Multiple Instruction-Single Data (MISD), and Multiple Instruction-Multiple Data (MIMD). SISD and SIMD are the classifications for the most common processors as they perform one instruction at a time. For the case of big processors like those in servers or scientific computing, the use of MISD or MIMD is preferred.

3.2 Instruction Execution

The processor, following a predefined sequence which corresponds to the task to solve, must execute a source code program. Each instruction delivers the result needed by the next instructions. The execution order of the instructions in the program memory must be sequential, and each instruction must finalize prior to loading the next instruction.

To complete an instruction it is necessary to divide the processing time into basic tasks applicable to every instruction. The first task to do with an instruction is to retrieve it from the program memory and to load it into the instruction registers. This task is called FETCH. The second task is to decompose the instruction into sub-instructions useful for each of the following functional units. This stage is called DECODE.

Once the instruction is decoded, the processor functional units know what to do with the resultant data. This is the EXECUTE stage, in which the processor obtains a result from the instruction. The final task for an instruction is to save the data generated in the previous stage. This task is called WRITEBACK. Note that not all the instructions require writing back their result, as they do not give any result. In this case, the instructions that do not require storing a result end at the previous stage.

Modern processors use much more than these basic stages as they decompose them into more sub-stages.

4 Definition of the Didactic RISC Soft Processor

As the purpose of this processor design is to serve as a tool for computer architecture understanding, it is desirable to develop a fully functional processor that can be used in any type of general-purpose problem. It must also achieve a fast enough performance to be used in real tasks. Considering the previous requirements, the resulting processor must have the following characteristics:

1. Based on the Harvard architecture,
2. RISC instruction set with 29 instructions,
3. Single Instruction – Single Data (SISD) execution order,
4. Eight 8-bit general-purpose registers,
5. 256 allocations of 16-bit wide ROM program memory,
6. 256 allocations of 8-bit wide RAM data memory,
7. ALU with two basic arithmetic and six logical operations.

To accomplish the goal of obtaining a functional processor, the first activity is to define the memory structure and the instruction set for the processor. The second activity is to define the different instruction formats, along with the addressing modes. Once the processor structure is defined, the third task is to design and construct all the functional blocks that comprise the processor.

4.1 Memory Organization

The processor is based on the Harvard architecture; it uses separate memories for program and data. In addition, it is necessary to have other useful memories for processing such as general-purpose registers and a stack. This section discusses the memories in the processor.

a. Program Memory

The program memory is a ROM type memory segment used to store the sequences of instructions in machine language. In other words, it is used to store the program.

It is organized as a linear sequence of 256 deep x 16-bit wide memory locations. It uses an 8-bit address bus to address all the locations. The 8-bit

Program Counter (PC) indicates the address of the next instruction to execute. Each address of the program memory is a sequence from location 0x00 to 0xFF.

This is a non-volatile memory, which holds the program even if the processor powers down. It has two operation modes: reading and writing. The writing mode is a first step for loading the memory content (the program). In the reading mode, the address is set at address pins and the content of that location is available immediately at the data output bus.

b. Data Memory

The data memory is a RAM type memory segment used to store the data generated by the main program. The data stored could be a variable value or a constant used by the program to perform calculations. It is organized as a set of 256 allocations, each 8-bit wide. The data memory address bus is 8-bit wide allowing the possibility to access all locations, from 0x00 to 0xFF. As this memory is volatile, when the processor powers down, the data is lost. This memory has two operation modes: write and read. The write operation must follow these steps:

1. Set the desired address at the address bus,
2. Set the desired data in the input data bus,
3. Introduce a clock pulse as the control signal,
4. The data is stored at the desired location and is available at the output bus.

The read operation is much simpler. It only needs to set the desired address at the address bus, and the content of the location transfers immediately to the corresponding data output bus.

c. General Purpose Registers

The general-purpose registers are RAM type memories which are made of conventional Flip-Flops. The main advantage of these registers is the closeness to the arithmetic and logic unit with the purpose of achieving faster calculations than with the content of Data Memory. Almost every instruction uses registers as parameters.

There are two possible operations: read and write. In the read mode, two registers are selected simultaneously. In the write mode, only the destination register can be written. The write operation must follow these steps:

1. Set the desired register address at the destination address bus,
2. Set the desired data in the input data bus,
3. Introduce a pulse as the control signal,
4. The new data is stored at the desired register and is available at the output.

The reading operation is much simpler. The only condition is to set the desired address for source and/or destination registers, and the content is transferred to the corresponding outputs. In the GPR, each register is made of 8 D-type flip-flops.

4.2 Instruction Set

A processor must have its own specific instruction set which comprises the assembly code and the machine language binary format. The instruction set must satisfy two very important concerns: it must be simple and robust. These are accomplished by selecting the simplest instructions to make the processor capable of executing any operation or routine in fewest steps.

The instructions are classified according to their purpose in three groups:

- **Operations*:*** instructions that affect the register values,
- **Program Control:** instructions that affect the execution order,
- **Data Transferring:** instructions that affect the memory contents.

For the operations case, two arithmetical ones were selected. Addition and subtraction between two registers and between one register and an immediate data were chosen. More complex arithmetical operations are build based upon these two operations. In the case of logical operations, the selected operations were AND, OR between two registers, and NOT, shift right, shift left, and SWAP with one register.

As for the program control instructions, such branch instructions as an immediate jump, an indirect jump to a location in a register, conditional branches for each status flag (Z, C, and H), a jump to subroutine, and a return from subroutine were chosen.

For the case of data transferring instructions, the load and storage instructions require

addressing data at different memory sources. In order to show the complete instruction set, it is necessary to first define the addressing modes for the different memories and their instruction format.

4.3 Addressing Modes

In processors, there must be a form to connect the core processor to the different memories with the purpose of information interchange called addressing modes. While more addressing modes are incorporated in a processor, the number of instructions in a program can be reduced. In our proposed processor, since it is didactic, it includes six addressing modes for data transfer between the GPR, the data, and program memories. It supports the following addressing modes:

- **Program Memory Direct.** This mode occurs when an 8-bit constant data is the new value for the PC causing it to change its address value. The branch instructions 1-5 from the instruction set belong to this type.
- **Immediate.** Instructions of this type use a constant value to affect the content of a register in the GPR. Instructions 6-10 from the instruction set are of this type.
- **Data Memory Direct.** In this mode, the instructions use a constant value as address to the data memory for read or write operations. Instructions 11-12 from the instruction set are of this type.
- **Register Direct, Two Registers.** This mode uses two registers from GPR and affects the content of the destination register by means of an arithmetic or logical operation using the source register value. Instructions 13-16 from the instruction set are of this type.
- **Data Memory Indirect Through Register.** In this mode, the data memory is addressed by the content of a register in the GPR to execute read or write operations with a second register. Instructions 17-18 from the instruction set are of this type.
- **Register Direct, Single Register.** This mode uses one register from GPR to affect its content according to an arithmetic or logical operation. Instructions 19-24 from the instruction set are of this type.

4.4 Instruction Format

The instruction format refers to the physical order in which the bits of an instruction are placed according to the parameters it uses. Its development is restricted to the CPU architecture, the number of instructions it involves, and the operands handled.

Each instruction must have a unique identifier called the operation code, commonly abbreviated as the OPCODE. The bit length for the OPCODE depends on the total number of instructions. As our processor complete instruction set contains 25 instructions, we only need 5 bits to represent all of them.

As the operands in an instruction are the most important factor, it is necessary to classify the instructions according to their format type as follows [31]:

- **Type J** – instructions used for jumps,
- **Type I** – instructions that use a register and an immediate data,
- **Type R** – instructions that perform operations using registers,
- **Type D** – instructions that carry out operations without parameters.

The format for an instruction requires 3 bits used to specify any of the eight registers in the GPR (source *Rs* or destination *Rd*); 8 bits used to specify a constant or address value *k*; finally, 5 bits are required for the *OPCODE*. By noting that not all the instructions require all the fields and that it is desirable to have a standard length for all the instructions, their format length was fixed to 16 bits according to the following.

The instructions grouped into Type J use the 5-bit OPCODE, and an 8-bit constant value (k), resulting in 13 useful bits leaving 3 bits unused according to Fig. 1.

The instructions grouped into Type I use the 5-bit OPCODE, a destination register (Rd) or source register (Rs), and an 8-bit constant value (k), resulting in 16 useful bits ordered as in Fig. 2.

The instructions grouped into Type R are divided into R2 and R1 instructions indicating which instructions use two registers and which use one register. R2 instructions require a destination register (Rd), a source register (Rs), and the

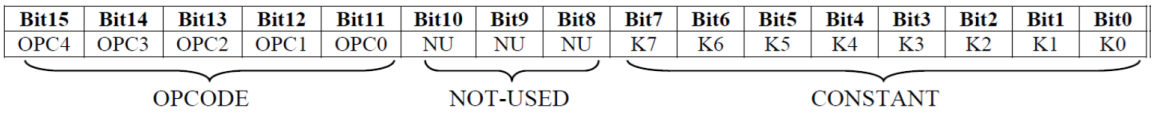

Fig 1. Type J instruction format

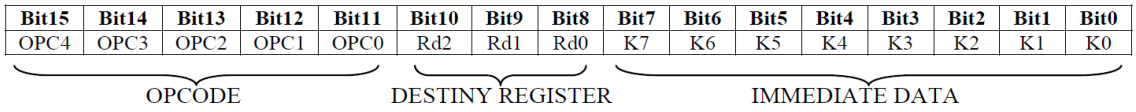

Fig. 2. Type I instruction format

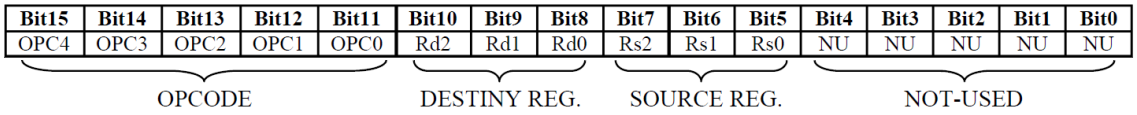

Fig. 3. Type R2 instruction format

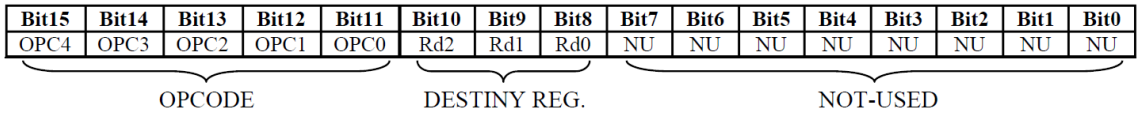

Fig. 4. Type R1 instruction format

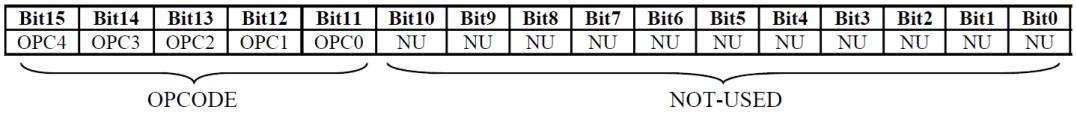

Fig. 5. Type D instruction format

OPCODE for 11 useful bits as in Fig. 3. R1 type instructions use a single register (Rd or Rs) and the OPCODE for 8 useful bits as in Fig. 4.

The instructions grouped into Type D require no more than the OPCODE for 5 useful bits as in Fig. 5.

Table 1 describes the instruction set for the processor, showing for each instruction its mnemonic, its description, the suggested syntax, the micro-operation it performs, its 16-bit instruction format, and the required clock cycles to complete the instruction.

4.5 Functional Units

Physically, a processor is made of a set of hardware blocks called functional units which are the basis for binary data processing. Each functional unit must satisfy certain logical design criteria in order to effectively accomplish its task [32]-[35]. Some of these units must be designed entirely; some others are common construction blocks which can be used from libraries.

The functional units are described below with the exception of the program and data memories which were considered previously. It is noteworthy that the control unit uses the control signals of all the functional units to manipulate the processor functioning.

It is suggested to analyze theoretically each functional unit in one day and in the next class realize its design in the lab.

a. Program Counter (PC)

The program counter PC is a binary counter which produces the address to read an instruction from the program memory. It must be capable of loading a pre-defined address if the program requires it, as in the cases of loops or branches. To construct the program counter, a common 8-bit binary counter with parallel-load is used. Note that

there are control signals used to manipulate its behavior.

Nomenclature	
Rd	Destination register
Rs	Source register
K	Constant
A	Address
P	Port Address
-	Not Used

b. Instruction Register

The instruction register is divided into two 8-bit registers named the Instruction Register (IR) and the Instruction Data Register (IDR). The IR stores the upper 8 bits of the instruction read from program memory. It usually contains the OPCODE and a register parameter. The IDR commonly contains the 8-bit constant or immediate data used by the instruction. Both registers maintain their data while an instruction is in execute and write back stages, then updates the data when a new instruction fetches. Two 8-bit registers made of a parallel array of D-type flip-flops are used as IR and IDR.

There is also an 8-bit address register AR that is dedicated to store the program counter value while the instruction is executed. This register is synchronized with both the IR and the IDR, and it has the same construction scheme.

c. Instruction Decoder

The instruction decoder is in charge of decoding the data stored in the instruction registers, i.e., it splits the data into its fundamental parts as follows: 5 bits for the OPCODE, 3 bits for destination register (Rd), 3 bits for source register (Rs), and 8 bits for the constant data (A/K).

The purpose of splitting the data is to send it to the fundamental unit that requires it (like GPR, ALU, or Memory). The design strategy for the decoder is achieved simply by using a set of buffers inside a block to sort the signals to separate buses.

d. General Purpose Registers (GPR)

General Purpose Registers (GPRs) are sets of registers used to store and save operands or results during the program execution. Their main advantage is that they can share data directly with the ALU and the data memory, resulting in high-speed calculations.

The control unit allows the ALU and the data memory to be able to read or write in those registers. A GPR consists of a set of eight 8-bit registers, a pair of eight 8-bit input multiplexers, and an 8-bit output decoder to control which register is read or written. Its operation mode defines that it reads two registers at a time (Rd, Rs) but only writes one register at a time (Rd) as in Fig. 6.

e. Arithmetic-Logic Unit (ALU)

The Arithmetic-Logic Unit is the functional unit dedicated to execution of arithmetical and logical calculations. As this is a didactic processor, it uses the simplest operations. Anyway, it is possible to construct more complex operations represented in terms of these simplest ones.

Each operation is designed separately and becomes a symbol. A multiplexor is used in order to have a 3-bit operation selector as illustrated in Table 2.

Additionally, any ALU operation activates status flags according to its result. For this purpose, the basic flags included are the carry flag (C), the half carry flag (H), and the zero flag (Z). The resulting schematic diagram for the ALU is presented in Fig. 7.

f. Control Unit

The control unit synchronizes the operation of all the previous functional units. It is a state machine which sets the functioning order for each instruction according to the OPCODE in order to satisfy the instruction execution stages presented in Section 2. The control unit design is realized by following a state diagram, in which the following considerations must be made:

- There must be a reset state present at start up, which must consider the initial conditions for all the functional units.
- The second state should represent the FETCH stage, where the instruction is retrieved from the program memory and loaded into the IR and IDR registers; the program address is also loaded into the AR register.

Table 1. Instruction set for the IPN-8 didactic soft processor

						16-bit Instruction Format																
						OPCODE						PARAMETERS										
No.	**Mnemonic**	**Description**	**Syntax**	**Micro-operation**		**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**CLK**
J - Type Instructions																						
1	JMP	Direct Jump	JMP k	PC ← k		0	0	0	0	0	-	-	-	k	k	k	k	k	k	k	k	2
2	BRC	Branch if carry	BRC k	if (C = 1) then PC ← k Else if (C=0) then PC←PC+1		0	0	0	0	1	-	-	-	k	k	k	k	k	k	k	k	2
3	BRZ	Branch if zero	BRZ k	if (Z = 1) then PC ← k Else if (Z=0) then PC←PC+1		0	0	0	1	0	-	-	-	k	k	k	k	k	k	k	k	2
4	BRV	Branch if overflow	BRV k	if (V = 1) then PC ← k Else if (V=0) then PC←PC+1		0	0	0	1	1	-	-	-	k	k	k	k	k	k	k	k	2
5	JSR	Jump to Subroutine	JSR k	[SP] ←PC, SP←SP-1, PC←k		0	0	1	0	0	-	-	-	k	k	k	k	k	k	k	k	2
I - Type Instructions																						
6	ADDI	Immediate addition	ADDI Rd, k	Rd ← Rd + k		0	1	0	0	0	Rd	Rd	Rd	k	k	k	k	k	k	k	k	3
7	SUBI	Immediate subtraction	SUBI Rd, k	Rd ← Rd – k		0	1	0	0	1	Rd	Rd	Rd	k	k	k	k	k	k	k	k	3
8	ANDI	Immediate AND	ANDI Rd, k	Rd ←Rd AND k		0	1	0	1	0	Rd	Rd	Rd	k	k	k	k	k	k	k	k	3
9	ORI	Immediate OR	ORI Rd, k	Rd ←Rd OR k		0	1	0	1	1	Rd	Rd	Rd	k	k	k	k	k	k	k	k	3
10	LDI	Load immediate	LDI Rd, k	Rd ← k		0	1	1	0	0	Rd	Rd	Rd	k	k	k	k	k	k	k	k	2
11	LDD	Load direct from data memory	LDD Rd,[A]	Rd ← [A]		0	1	1	0	1	Rd	Rd	Rd	A	A	A	A	A	A	A	A	2
12	STD	Store direct to data memory	STD [A],Rs	[A] ← Rs		0	1	1	1	0	Rs	Rs	Rs	A	A	A	A	A	A	A	A	2
R2 – Type Instructions																						
13	ADD	Addition	ADD Rd, Rs	Rd ← Rd + Rs		1	0	0	0	0	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
14	SUB	Subtraction	SUB Rd, Rs	Rd ← Rd - Rs		1	0	0	0	1	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
15	AND	Logic AND	AND Rd, Rs	Rd ← Rd AND Rs		1	0	0	1	0	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
16	OR	Logic OR	OR Rd, Rs	Rd ← Rd OR Rs		1	0	0	1	1	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
17	LDX	Indirect load from memory	LDX Rd,[Rs]	Rd ← [Rs]		1	0	1	0	0	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
18	STX	Indirect storage to memory	STX [Rd],Rs	[Rd] ←Rs		1	0	1	0	1	Rd	Rd	Rd	Rs	Rs	Rs	-	-	-	-	-	2
R1 – Type Instructions																						
19	NOT	Logic NOT	NOT Rd	Rd←NOT Rd		1	1	1	0	0	Rd	Rd	Rd	-	-	-	-	-	-	-	-	2
20	SHL	Shift register left	SHL Rd	Rd(n+1)←Rd(n), Rd(0) ← 0		1	1	1	0	1	Rd	Rd	Rd	-	-	-	-	-	-	-	-	2
21	SHR	Shift register right	SHR Rd	Rd(n)←Rd(n+1), Rd(7) ← 0		1	1	1	1	0	Rd	Rd	Rd	-	-	-	-	-	-	-	-	2
22	SWAP	Swap nibbles	SWAP Rd	Rd7←Rd3, Rd6←Rd2, Rd5←Rd1, Rd4←Rd0		1	1	1	1	1	Rd	Rd	Rd	-	-	-	-	-	-	-	-	2
23	PUSH	Push data to stack	PUSH Rs	[SP] ←Rs, SP←SP-1		1	1	0	0	0	Rs	Rs	Rs	-	-	-	-	-	-	-	-	2
24	POP	Pop data from stack	POP Rd	Rd←[SP], SP←SP+1		1	1	0	0	1	Rd	Rd	Rd	-	-	-	-	-	-	-	-	3
D – Type Instructions																						
25	RET	Return from Routine	RET	PC←[SP]+1, SP←SP+1		0	0	1	0	1	-	-	-	-	-	-	-	-	-	-	-	2

ISSN 2007-9737

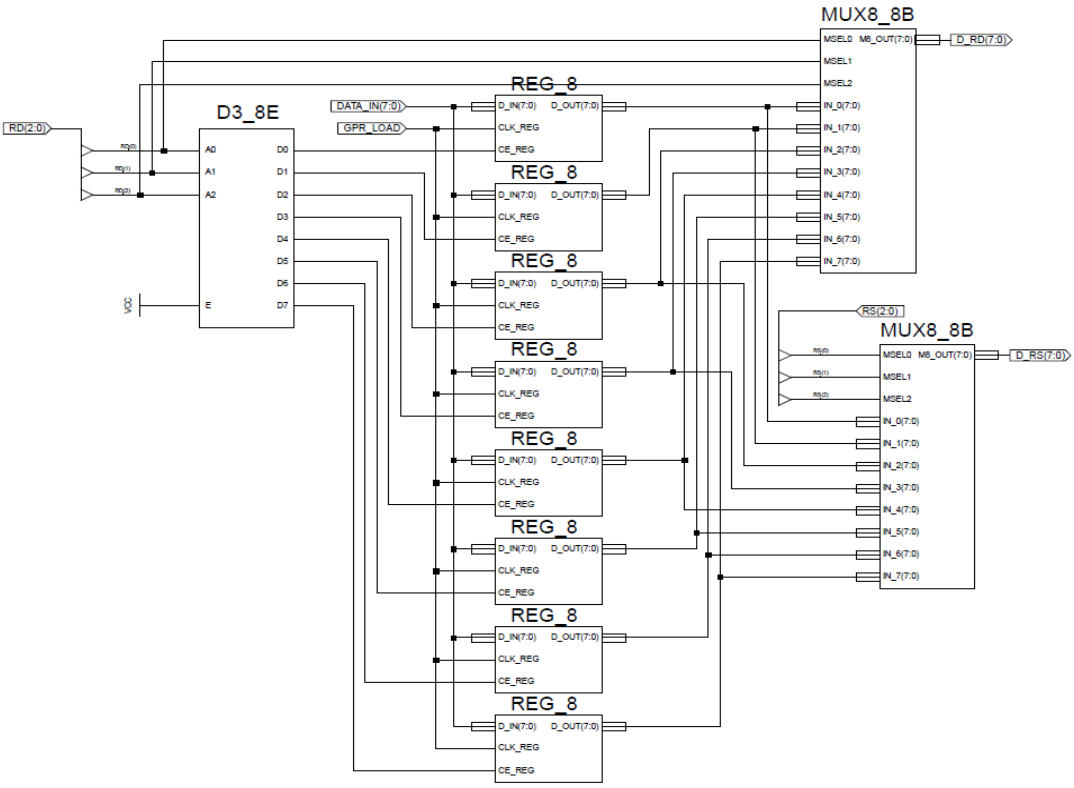

Fig. 6. General Purpose Registers schematic diagram

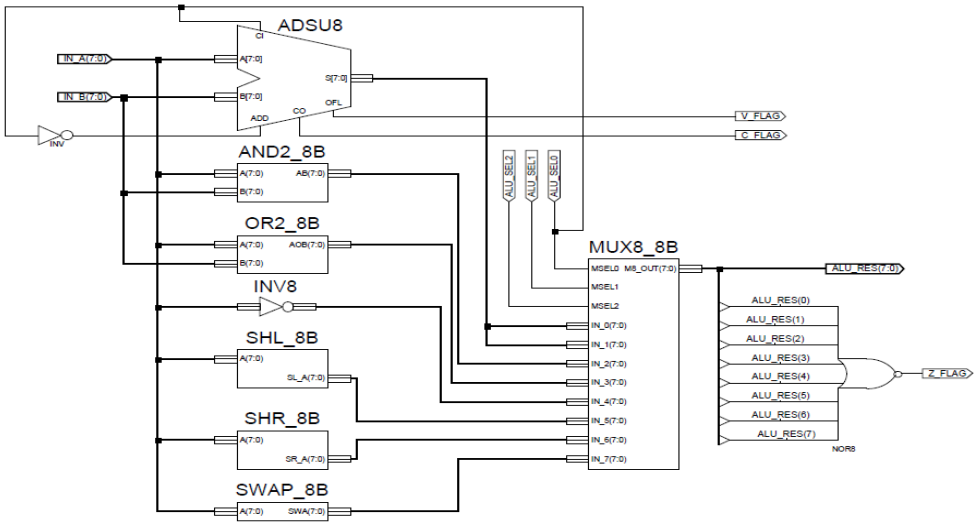

Fig. 7. Arithmetic and Logical Unit schematic diagram

- The decoding of the instruction goes straightforward once the instruction is fetched considering that the instruction registers are a kind of gate that let the data go through.

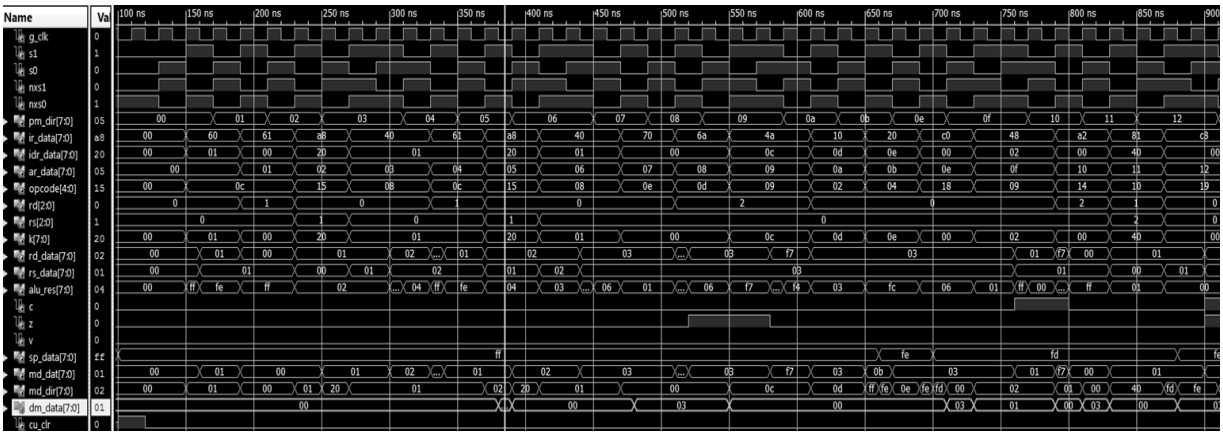

Fig. 8. Execution of the Fibonacci series program

- The next state corresponds to the EXECUTE stage, when the instruction executes its fundamental operation.
- It is noteworthy that some instructions do not require a write back stage, such as branch instructions.
- In the final state, the result of the operations is stored onto its respective functional unit, and the program counter is incremented to address the next instruction.

The design of the control unit is the most important challenge in the processor design, as it must make everything work correctly. The control unit is designed with the use of exhaustive behavioral analysis in which the control signals of each functional unit involved for each instruction must be considered. It is necessary to utilize a truth table in which the inputs are the OPCODE, the actual state, and the status flags; the corresponding outputs are the control signals for each functional unit and the next state.

The design of the control unit is a good practice for students to approve the course by making the processor work.

5 Results

In order to validate the functionality of our processor, it was implemented in a Xilinx Spartan3 XC3S500-4FG320 FPGA. The design was tested using the XilinxISE tools version 14.2 in schematic mode to allow direct gate level designs. The simulations were performed using the included XilinxISE Simulator. The main results are presented as two separate parts: the testing program and the resource consumption. Finally, we present a discussion.

5.1 Testing Program

For evaluating the performance of the CPU, it was necessary to choose a program that uses different types of instructions. The program must execute simple calculations, manipulate the stack, perform conditional and unconditional branches, execute subroutines, and access RAM memory. A program which calculates the first 10 elements of the Fibonacci series and stores them into consecutive data memory locations was used.

The simulation for the Fibonacci program is given in Fig. 8. The instruction address is shown at the pm_dir signal where the jumps can be viewed as discontinuous addresses. The dm_data signal corresponds to the data memory content given by the address of the md_dir signal. As shown, the Fibonacci sequence changes at instruction 5.

5.2 Resource Consumption

After file synthesis with the Xilinx XST tool, some numerical results were obtained reflecting the hardware resource utilization of the processor. As the processor was implemented in a Xilinx

Table 2. Arithmetic and logical operations in ALU

Selection Bits			ALU Operation
0	0	0	Addition
0	0	1	Subtraction
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Shift Left
1	1	0	Shift Right
1	1	1	Swap nibbles

Spartan3 XC3S500-4FG320 FPGA, the device utilization is shown in Table 3. IO's have the highest utilization percentage as they are required to interconnect the processor with the external pins, but an important consideration is that most of the signals were used only for monitoring purpose so they can be eliminated. The number of logical elements such as slices, flip-flops, and LUT's is minimal due to the combinational nature of the processor being capable of executing an instruction in a few clock cycles.

5.3 Discussion

It has been more than 60 years of computer architecture teaching evolution. The technological advances have directly affected the way computer architecture is taught.

According to Section 2, the different approaches to teach computer architecture were classified into five groups: paper, simple hardware, simulator, HDL, and logic blocks. These groups could be regrouped in two main branches: simulators and hardware constructs.

In the simulator branch, the paper and the software simulators can be included. They serve only to simulate the processor functioning either manually (paper) or automatically (software). This branch is not useful for evaluating our proposal.

In the hardware constructs branch, it includes the simple hardware, HDL, and logic blocks approaches. Any of these can be compared with our proposal since, as mentioned, the technological advances affects computer implementation.

Simple hardware is disappearing since it requires a great amount of time in constructions and a larger amount of money to get the components. In the case of the HDL and logic blocks, both approaches take advantage of the versatility of programmable logic devices. The main difference is the use of code versus diagramming.

Table 3. Hardware Resources Consumed

Logic Utilization	Used	Available	Utilization
Slices	202	4656	4%
Slice Flip Flops	109	9312	1%
4 input LUTs	404	9312	4%
Bonded IOBs	116	232	50%
GCLKs	2	24	8%

In this sense, the use of code leads to a shorter development time but adds uncertainty to the data path. It also depends on how efficiently the code is translated by a compiler.

On the other hand, the use of logical blocks serves to enhance the designer interconnection ability. It also leads to non-redundant circuits as they are designed only with the required gates.

Given the previous comments, the logic blocks approach is the best option since it exploits the versatility of programmable devices. Besides, it can be seen as an improved solution for the simple hardware approach.

When comparing to other educational processors, there are some advantages in our proposal. In the case of [25], it employs a Von Neumann architecture that has the disadvantage of using the same memory for program and data. The Harvard architecture in our proposal reduces memory access leading to better performance. It uses three programmable registers compared to the eight fully accessible registers in our proposal.

In the case of [26], it uses 16-bit data, and our proposal uses 8-bit data. Instructions in the former use at most three operands including an accumulator while our proposal uses at most two operands, and any general-purpose register can serve as accumulator. The same is true for [28].

Comparing our proposal with the processor in [27], the main disadvantage of the latter is that it uses 16-bit data only. When processing, it is necessary to use different data widths, as it is possible in our proposal.

Concerning [29], it does not include a register file, making direct access to the data memory. Its

construction is based on a monocycle architecture which is not most suitable for a functional processor, while our proposal is multi-cycle, resulting in a more efficient processing.

Finally, when compared to the processor presented in [30], our processor includes five new instructions, three new addressing modes, and an address register.

6 Conclusions

Applying our methodology in computer architecture courses allows students to understand how each part of a modern computer works, how the parts interact, and how to synchronize the different functional units. It also allows visualizing the relation of theoretical concepts with physical devices.

Our processor is fully functional and can operate executable programs going beyond theory, unlike most of the revised works. In addition, our processor could be used as embedded soft-core processor for FPGA designs that may benefit from the incorporation of a Central Processing Unit (CPU) for peripherals or other devices.

The processor presented in this paper was used to teach computer architecture for about 4 years having good acceptance among students and contributing effectively to learning process. It is left to the instructor to choose how to divide the theoretical and practical classes. It is suggested to review theory in one day and to make a practical session in the next class.

Acknowledgements

Authors would like to thank Instituto Politécnico Nacional and CONACYT who funded this work.

References

1. **Berkeley, E.C. & Jensen, R.A. (1950).** *World's Smallest Electric Brain*. Radio-Electronics, Oct. 1950.
2. **Illinois State University (2000).** *Little Man Computer*. http://www.acs.ilstu.edu/faculty/javila/lmc/
3. **Bell Laboratories Record (1969).** *Cardboard "Computer" Helps Students*, 216.
4. **Yutaka, S.** *Paper Processor*. Available: https://sites.google.com/site/kotukotuzimiti/
5. **Langdon, G.G. (1982).** *Computer Design*. Computeach press, California.
6. **Hennessy, J.L. & Patterson, D.A. (1990).** *Computer Architecture: A quantitative Approach*. Morgan Kaufmann San Mateo CA.
7. **El Aarag, H. (2009).** A complete design of a RISC processor for pedagogical purposes. *Journal of Computing Sciences in Colleges*, Vol. 25, No.2, pp. 205–213.
8. **Rodriguez, B.J. (1994).** A minimal TTL processor for architecture exploration. *Proceedings of the 1994 ACM symposium on applied computing*, pp. 338–340.
9. **Massachussetts Institute of Technology.** *MIT 6.004 Beta Architecture Information*. Available: http://6004.csail.mit.edu/Spring98/Beta.
10. **Verplaetse, P. & Campenhout, J. (1999).** ESCAPE: Environment for the Simulation of Computer Architecture for the Purpose of Education. *IEEE Technical Committee on Computer Architecture Newsletter*, pp. 57–59.
11. **Djordjevic, J., Milenkovic, A., & Grbanovic, N. (2000).** An Integrated Environment for Teaching Computer Architecture. *IEEE Micro*, Vol. 20, No. 3, pp. 66–74.
12. **Wainer, G., Daicz, S., De Simoni, L., & Wassermann, D. (2001).** Using the Alfa-1 simulated processor for educational purposes. *Journal on Educational Resources in Computing*, Vol. 1, No.4, pp. 111–151.
13. **Martins, C.A., Correa, J.B., Goes, L.F., Ramos, L.E., & Medeiros, T.H. (2002).** A new learning method of microprocessor architecture. *Frontiers in Education,* 3, S1F16-S1F21.
14. **Pizzutilo, S. & Tangorra, F. (2003).** A learning environment to teach computer architecture. *Proceedings of the WSEAS International Conference on Information Science and Application*, pp. 770–776.
15. **Jaumain, M., Osee, M., Richard, A., Vander Biest, A., & Mathys, P. (2007).** Educational simulation of the RISC processor. *International Conference on Engineering Education*.
16. **Garcia, M.I., Rodriguez, S., Perez A., & Garcia, A. (2009).** p88110: A Graphical Simulator for Computer Architecture and Organization Courses. *IEEE Transactions on Education*, Vol. 52, No. 2, pp. 248–256. DOI: 10.1109/TE.2008.927690

17. **Hsiao-Ping, Holmes, N.D., Bakshi, S., & Gajski, D.D. (1993).** Top-down modeling of RISC processors in VHDL. *Design Automation Conference 1993 with EURO-VHDL*, pp. 454–459.
18. **Li, Y. & Chu, W. (1996).** Using FPGA for computer architecture/organization education. *Proceedings of the 1996 workshop on Computer architecture education*, pp. 31–35.
19. **Gray, J. (2000).** Hands-on computer architecture: teaching processor and integrated systems design with FPGAs. *Proceedings of the workshop on Computer architecture education*, article 17. DOI: 10.1145/1275240.1275262
20. **Becvar, M., Pluhacek, A., & Danecek, J. (2003).** DOP: a CPU core for teaching basics of computer architecture. *Workshop on Computer architecture education*, article 4.
21. **Muñoz, A.S., & Rodríguez-Morcillo, J.D. (2008).** Microprocesador RISC sintetizable en FPGA para fines docentes. *VIII Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica*, España.
22. **Mandalidis, D., Kenterlis, P., & Ellinas, J. (2008).** A computer architecture educational system based on a 32-bit RISC processor. *International Review on computers and Software*, pp. 114–119.
23. **Wadhankar, V.R. & Tehre, V. (2012).** A FPGA Implementation of a RISC Processor for Computer Architecture. *Proceedings on National Conference on Innovative Paradigms in Engineering and Technology*, Vol. 1, pp. 24–28.
24. **Calazans, N.L.V., & Moraes, F.G. (2001).** Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses. *IEEE Transactions on Education*, Vol. 44, No. 2, pp. 109–119. DOI: 10.1109/13.925805
25. **Becvar, M., Pluhacek A., & Danecek, J. (2003).** DOP: a CPU core for teaching basics of computer architecture. *Proceedings of the workshop on Computer architecture education*, article 4. DOI: 10.1145/1275521.1275527
26. **Angelov, V. & Lindenstruth, V. (2009).** The educational processor Sweet-16. *International Conference on Field Programmable Logic and Applications*, pp. 555–559. DOI: 10.1109/FPL.2009.5272412
27. **Presa, J.L. & Calle, E.P. (2011).** MMP16 a 16-bit Didactic Micro-Programmed Micro-Processor. *International Conference on Computer Research and Development*, Vol. 1, pp. 61–65. DOI: 10.1109/ICCRD.2011.5763974
28. **Oztekin, H., Temurtas, F., & Gulbag, A. (2011).** BZK.SAU.FPGA10.0: Microprocessor architecture design on reconfigurable hardware as an educational tool. *IEEE Symposium on Computers & Informatics*, pp. 385–389. DOI: 10.1109/ISCI.2011.5958946
29. **Pereira, M.C., Viera, P.V., Raabe, A.L., & Zeferino, C.A. (2012).** A basic processor for teaching digital circuits and systems design with FPGA. *Southern Conference on Programmable Logic*, pp. 1–6. DOI: 10.1109/SPL.2012.6211804
30. **Hernández Zavala, A., Avante Reyes, J., Duarte Reynoso, Q., & Valencia Pesqueira, J.D. (2011).** RISC-Based Architecture for Computer Hardware Introduction. *International Conference on Computer Research and Development*, pp. 17–21.
31. **Pattersson, D.A. & Hennessy, J.L. (1998).** *Computer Organization and Design, the hardware/software interface* (2nd ed.). Morgan Kaufmann San Francisco CA.
32. **Tocci, R.J., Widmer, N.S., & Moss, G.L. (2007).** *Sistemas Digitales, Principios y aplicaciones* (10th ed.), Pearson, México.
33. **Stallings, W. (1994).** *Organización y Arquitectura de Computadoras* (7th ed.). Pearson, Mexico.
34. **Mano, M. (1994).** *Arquitectura de computadoras* (3th ed.). Pearson, Mexico.
35. **Jaramillo Gómez, J.A., Guzman Dominguez, I., & Molina Lozano, H. (2011).** *VHDL. Guia de Estilo y Prácticas de Laboratorio de Circuitos Lógicos.* Instituto Politécnico Nacional, México.

Antonio Hernández Zavala received the B.Sc. in Computer Systems from the Ecatepec Higher Technological Studies, Mexico, in 2001. He received his M. Sc. in Computer Engineering with specialty in Digital Systems in 2004 and the Ph. D. in Computer Architecture in 2009, both from the Computing Research Centre of Instituto Politécnico Nacional, Mexico. From 2008 to 2011 he was a full time professor at the Interdisciplinary Engineering and Advanced Technologies Professional Unit (UPIITA) of Instituto Politécnico Nacional, where he taught computer architecture to about 200 students using a reduced version of the IPN-8 processor. He is a member of the National System of Researchers (SNI) and a full time professor at the Mechatronics Department of Applied Science and Advanced Technology Research Centre (CICATA) of Instituto Politécnico Nacional, Querétaro (since the end of 2011).

Oscar Camacho Nieto received the B. Sc. in Communications and Electronics Engineering from the Mechanical and Electrical Engineering Faculty (ESIME) of Instituto Politécnico Nacional. He received the M. Sc. degree in Computer Engineering with specialty in Digital Systems from CIDETEC, and the Ph. D. in Computer Science from the Computing Research Centre (CIC), both of Instituto Politécnico Nacional. Besides, he realized doctoral studies in Computer Architecture at Cataluña Polythecnical University in Spain, where he obtained the Researcher Sufficiency degree. Between 2003 and 2010 he was at the Computing Research Centre (CIC), where he realized diverse administrative activities. Currently he is a member of the National System of Researchers (SNI) of CONACYT and the director of CIDETEC, where since 2012 he has been a full time research professor, PhD and master programs' coordinator, chief of the computer center, among other charges.

Jorge Adalberto Huerta Ruelas is a physicist and received his Ph. D. from Universidad Autónoma de San Luis Potosí, specializing in Optoelectronic Devices and Optical Characterization Techniques. Afterwards, he pursued a postdoctoral internship at the Department of Food Science and Technology of Oregon State University. He is a member of the National System of Researchers, Mexico. Since 2010 he has been the director of the Research Center on Applied Science and Advanced Technology of Instituto Politécnico Nacional (National Polytechnic Institute).

Arodí Rafael Carvallo-Domínguez received the B. Eng. degree in Control and Automation Engineering from Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME), Mexico, in 1994. He received his M.Sc. degree and did Doctoral studies in Automatic Control at Automatic Control Department (DCA) of Centro de Investigación y de Estudios Avanzados (CINVESTAV), Mexico, in 1999 and 2001, respectively. Currently he is a Titular Professor at the Department of Advanced Technologies and the director of Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA). He has published more than 20 journal and conference papers. His research interests include vision-based control of robotic systems, intelligent control, industrial automation, and mechatronics systems. He is a member of the IEEE Robotics and Automation Society.

Article received on 27/11/2014; accepted on 15/01/2015.
Corresponding author is Antonio Hernandez Zavala.