A Basic Processor for Teaching Digital Circuits and Systems Design with FPGA

Maicon Carlos Pereira, Paulo Viniccius Viera, André Luis Alice Raabe and Cesar Albenes Zeferino
Master Program on Applied Computer Science
University of Vale do Itajaí, UNIVALI
Itajaí, BRAZIL
{maicon, paulovinicciu s, raabe, zeferino}@univali.br

***Abstract*—Design of digital circuits and systems are topics covered in undergraduate courses on Computer Science, Computer Engineering, and Electrical Engineering. Simple processor architectures are used as example of digital systems to apply and integrate the concepts studied in these courses. In this paper, we discuss the use of a simple processor named BIP (Basic Instruction-set Processor) in courses on digital circuits and systems design. BIP is distinguished from similar processors because it was developed by applying a multidisciplinary approach in order to allow its use in introductory courses on computer programming and in several other courses in the Computer Science area.**

Keywords-FPGA in education; digital circuits; digital systems; computer programming

I. INTRODUCTION

The design of digital circuits and systems is typically studied in undergraduate courses on Computer Science, Computer Engineering and Electrical Engineering.

Courses about digital circuits design cover concepts on data representation, number systems, Boolean algebra, combinational and sequential circuits, memories and technologies for Integrated Circuits (ICs). In courses on digital systems design, these concepts are applied on the design of datapaths, control units, general-purpose and special-purpose processors, by applying design methodologies and technologies, like Register-Transfer Level (RTL) design, Hardware Description Languages (HDLs) and Field Programmable Gate Arrays (FPGAs).

Many books on digital circuits and systems design (as well on computer architecture and organization) use simple general-purpose processors as example of digital systems to apply and integrate the concepts covered along the text. Some of them include SAP (*Simple-As-Possible*) computer [1] and MARIE (*Machine Architecture that is Really Intuitive and Easy*) [2]. These simple processors are useful because they make easier the learning by the students, and are easily implemented by using current technologies and design methodologies and tools.

In this paper, we describe another simple processor architecture that, at the first view, has no major difference with other processors described in the literature. However, it is distinguished from the other processors because it was designed by using an interdisciplinary approach. This processor, named BIP (Basic Instruction-set Processor), was specified by a team of professors that work with courses on "Introductory Programming", "Digital Circuits", "Digital Systems", "Computer Architecture and Organization", and "Compilers Design". All these disciplines of Computer Science were taken into account at the moment of specifying a family of processors with an incremental architecture that could be used in the first weeks of an undergraduate course on Computer Science. Also, we considered its use on advanced courses as a first example of processor architecture and organization, or as an alternative for assembly code generation.

In the digital circuits and systems domain, BIP has shown to be a didactic example of processor. After learning the major concepts on digital circuits, technologies, and design methodologies and tools, any student can easily learn the BIP architecture, design its organization, describe it with any HDL (or with a schematic entry tool) and synthesize it in an FPGA.

In the next sections, we firstly present an overview about the specification process of BIP (Section II). After that, in Section III, we describe the architecture and the organization of the first two BIP models (BIP I and BIP II). Following, we discuss issues regarding the use of BIP in courses on digital circuit and systems design (Section IV). Finally, we present our conclusions in Section V.

II. THE INTERDISCIPLINARY APPROACH TO SPECIFY BIP

A. Motivation

The major motivation for the developing of BIP, was the observation that many undergraduate students in introductory programming had difficult to understand some important concepts. This problem is also discussed in the literature. Khalife [3], for instance, consider that students in their initial studies on computer programming have difficult to develop a proper mental model to capture the internal structure of computers, what makes harder the Teaching and Learning process. Other authors also consider that such students still have not the necessary logic and formal reasoning to understand the high level of abstraction involved in the programming of computers [4-5]. Furthermore, as we had observed, hypothetical computer models used to illustrate some concepts are too abstract, and do not help the students in understanding some underlying concepts of computer programming.

978-1-4673-0186-2/12/$31.00

Considering these issues, we verified the need of adopting a real and simple processor in order to reduce the abstraction level of the computer operation in introductory programming. Although there exists a number of simple processors described in the literature (like SAP [1], MARIE [2], and others), we also observed that it was necessary to specify a new model to take in consideration the needs of introductory courses on introductory programming. At the same time, the processor design should take into account that it should be used in other courses, including the ones on digital circuits and systems design, computer architecture and organization, and compilers design.

B. The Guidelines for BIP Specification

Given the issues discussed above, the following major guidelines were defined before we start the specification of BIP architecture and organization, focusing on the needs of freshmen on Computer Science:

1) The processor architecture and its assembly programming should be as simple as possible in order to make easier its learning by a freshman student on Computer Science;

2) The processor should be able to illustrate the basic concepts on computer programming (by reducing the abstraction level) and help a freshman student on Computer Science to understand these concepts; and

3) The processor organization should be as regular and simple as possible in order to make easier its implementation using the current technologies, design methodologies and tools.

Based on these guidelines, a set of requirements was identified and guided the design of BIP. The major ones are:

1) Adoption of a highly regular architecture: in order to make easier and faster the learning about the processor architecture and how any command written using a high-level programming language is mapped to the BIP assembly, the following choices were taken (a highly regular architecture makes easier the processor implementation [6]):

- All the instructions in the Instruction Set Architecture (ISA) are based on a single format;
- Arithmetic and logic instructions can access the data memory (*i.e.* it is not a *load/store* architecture);
- There are branch instructions for all the comparisons used in high-level programming (*i.e. greater*, *greater than*, *less*, *less than*, *equal*, and *not equal*); and
- The ISA is extensible and the successive generations are fully compatible with the previous ones. Each new generation add functionalities (and complexity) to the ISA.

2) Adoption of a simple processor organization: in order to make easier and faster the implementation of BIP by a student of a course on digital circuits design, the processor organization must use building blocks studied in such courses. A minimal course on digital circuits design cover number systems, Boolean Algebra, combinational logic (logic gates), combinational circuits (mux/demux, coder/decoder and adder/subtractor), and sequencial circuits (latches, flip-flops, registers and counters). Some topics, like FSMs (Finite State Machines), ALU (Arithmetic and Logic Units), Register Files, RTL design, are usually studied in courses on digital systems design. Taking this into account, the following choices were done for the basic organization of BIP:

- Processor should be based on a monocycle/Harvard organization in order to simplify the instruction fetch and execution, and limit the control unit to a combinational circuit, what makes it easier to be designed and implemented;
- Registers should be limited to the ones required by the processor architecture, with no additional registers (like occurs in multi-cycle and pipelined organizations [6]);
- Processor should include a basic arithmetic unit (not an ALU) to support *add* and *subtract* operations; and
- Processor should not have a Register File.

III. BIP Architecture and Organization

After identifying the guidelines and the requirements for BIP, two basic models were designed: BIP I and BIP II. BIP I includes only the support for data memory access and arithmetic operations, while BIP II adds support for conditional and unconditional branches.

A. Architecture

The major architectural attributes of BIP I and BIP II includes:

- Accumulator-oriented architecture;
- 16-bit data width;
- 16-bit instruction width;
- Only one data type (integer);
- Only one instruction format;
- Only two addressing modes (Immediate and Direct);
- Reduced instruction set; and
- Memory-mapped I/O (Input/Output).

Although BIP presents several features of RISC (Reduced Instruction Set Computer) machines, it cannot be considered a RISC processor because it does not use a load-store architecture (and does not have a large number of general purpose registers). The adoption of a full RISC architecture would add complexity to the processor, more than we considered necessary for our purpose.

The instruction format of BIP is shown in Fig. 1. It has only two fields:

- *Opcode*: a 5-bit field to identify the operation to be done by the instruction (up to 32 instructions can be represented); and
- *Operand*: an 11-bit field that identify an operand for the instruction. It can represent an immediate data (a constant), an address in the data memory (a variable), or an address in the program memory (for branches).

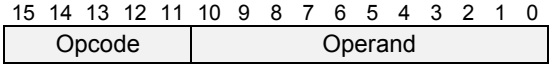

Figure 1. Instruction format

The CPU (Central Processing Unit) has only three registers:

- *PC (Program Counter)*: stores the address of the current instruction (due to the monocyclic approach);
- *ACC (Accumulator)*: works as the implicit operand in many instructions; and
- *STATUS*: stores two flags (Zero and Negative) used by conditional branch instructions in BIP II.

The addressing space is organized in a 2-Kword program space and a 2-Kword data space. The data space is divided into two 1-Kword regions: one for data memory and another for memory-mapped I/O (see Fig. 2). The I/O addressing space can be used to illustrate the access to basic devices, like LEDs and switches, or even to more complex peripherals.

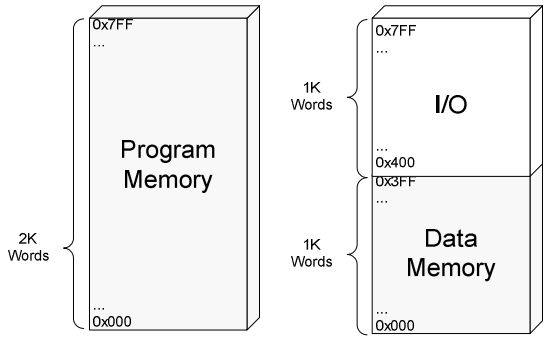

Figure 2. Addressing spaces

The instruction set of BIP is described in Table I (note that BIP I uses only the first 8 instructions).

TABLE I. INSTRUCTION SET

Operation	Opcode	Instruction	Data Memory (DM) and Accumulator (ACC) Updating	Program Counter (PC) updating	Affected Flags	BIP Model
Halt	00000	**HLT**		PC ← PC		I, II
Store Variable	00001	**STO** operand	DM[operand] ← ACC	PC ← PC + 1		I, II
Load Variable	00010	**LD** operand	ACC ← DM[operand]	PC ← PC + 1		I, II
Load Immediate	00011	**LDI** operand	ACC ← operand	PC ← PC + 1		I, II
Add Variable	00100	**ADD** operand	ACC ← ACC + DM[operand]	PC ← PC + 1	Z,N	I, II
Add Immediate	00101	**ADDI** operand	ACC ← ACC + DM	PC ← PC + 1	Z,N	I, II
Subtract Variable	00110	**SUB** operand	ACC ← ACC - DM[operand]	PC ← PC + 1	Z,N	I, II
Subtract Immediate	00111	**SUBI** operand	ACC ← ACC - operand	PC ← PC + 1	Z,N	I, II

Branch on Equal	01000	**BEQ** operand		if (STATUS.Z=1)
				PC ← operand
				else
				PC ← PC + 1
						II

Branch on Not Equal	01001	**BNE** operand		if (STATUS.Z=0)
				PC ← operand
				else
				PC ← PC + 1
						II

Branch on Greater Than	01010	**BGT** operand		if (STATUS.Z=0) and
				(STATUS.N=0)
				PC ← operand
				else
				PC ← PC + 1
						II

Branch on Greater or Equal	01011	**BGE** operand		if (STATUS.N=0)
				PC ← operand
				else
				PC ← PC + 1
						II

Branch on Less Than	01100	**BLT** operand		if (STATUS.N=1)
				PC ← operand
				else
				PC ← PC + 1
						II

Branch on Less or Equal	01101	**BLE** operand		if (STATUS.Z=1) or
				(STATUS.N=1)
				PC ← operand
				else
				PC ← PC + 1
						II

Jump	01110	**JMP** operand		PC ← operand		II

B. Programming

Table II illustrates how commands written in C language can be translated to the assembly of BIP. As one can see, before a conditional branch, it is necessary to subtract the values to be compared by the branch instruction, which analyzes the flags of the STATUS register (Z and N). This is a consequence of the instruction format, which does not support more than one explicit operand. In branch instructions, this operand is already used to define the branch address.

C. Organization

BIP I organization is shown in Fig. 3. It uses a monocycle/Harvard architecture with separated memories for program and data, similar to the organization described for MIPS processor in [6]. A multicycle/Von Neumann architecture could be used, but it would add more complexity to the processor than we considered necessary for our purpose.

BIP's CPU has a port to read instructions from the program memory, and another port to read/write data from/to the data memory. It can also have I/O ports for communication with peripherals, but these ports are connected to the same bus of the data memory port.

Internally, BIP's CPU is structured into two blocks:

- *Control*: it fetches instructions from the program memory, decodes them and commands the operations in the Datapath. Control is composed by a PC register, an 11-bit adder and a combinational instruction decoder;
- *Datapath*: it processes data under the command of Control block. It includes the ACC register, an Arithmetic Unit (a 16-bit adder/subtractor), a block to extend the signal of the 11-bit operand to a 16-bit word, and two multiplexers.

TABLE II. PROGRAMMING IN BIP ASSEMBLY

Code in C language	Code in BIP assembly
`A = 10;`	`LDI 10 ;ACC ← 10` `STO A ;A ← ACC`
`A = B;`	`LD B ;ACC ← B` `STO A ;A ← ACC`

`A = A + 1;`	`LD A ;ACC ← A`
	`ADDI 1 ;ACC ← ACC + 1`
	`STO A ;A ← ACC`

`A = A + B - 3;`	`LD A ;ACC ← A`
	`ADD B ;ACC ← ACC + B`
	`SUBI 3 ;ACC ← ACC - 3`
	`STO A ;A ← ACC`

`if (A==B) {`
`  // Block 1`
`}`
`// Block 2`
	`LD A ;ACC ← A`
	`SUB B ;ACC ← ACC - B`
	`BNE L1`
	`;Block 1`
	`L1:`
	`;Block 2`

`if (A==B) {`
`  // Block 1`
`} else {`
`  // Block 2`
`}`
`// Block 3`
	`LD A ;ACC ← A`
	`SUB B ;ACC ← ACC - B`
	`BNE L1`
	`;Block 1`
	`JMP L2`
	`L1:`
	`;Block 2`
	`L2:`
	`;Block 3`

`i = 0;`
`while (i<10) {`
`  // Block 1`
`  i++;`
`}`
`// Block 2`
	`LDI 0 ;ACC ← 0`
	`STO I ;I ← ACC`
	`L1:`
	`SUBI 10 ;ACC ← ACC - 10`
	`BGE L2`
	`;Block 1`
	`LD I ;ACC ← I`
	`ADDI 1 ;ACC ← ACC + 1`
	`STO I ;I ← ACC`
	`JMP L1`
	`L2:`
	`;Block 2`

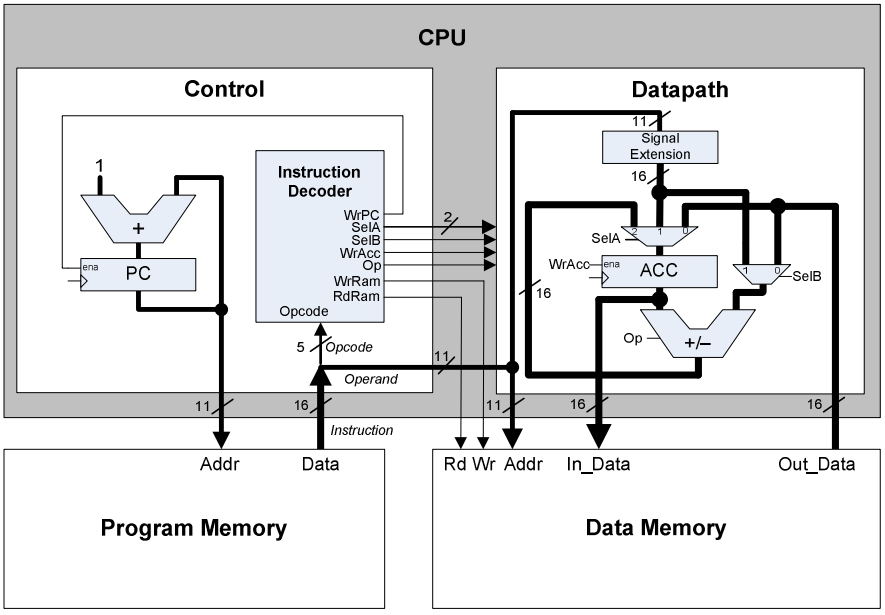

Figure 3. BIP I organization

BIP II organization is shown in Fig. 4. The major differences with BIP I are:

- *Control*: the instruction decoder was modified in order to evaluate the values of Z and N flags of STATUS register when executing a conditional branch instruction, which must be preceded by a subtraction operation (`SUB` or `SUBI` instructions). Furthermore, it was added a multiplexer for loading the operand field when the branch is taken or when a jump instruction is executed;
- *Datapath*: the only change was the adding of the 2-bit STATUS register.

It must be highlighted that all the circuits used in both BIP I and BIP II organizations are covered in digital circuits courses (and books). Although the instruction decoder is an *ad-hoc* combinational circuit, it is simple and easy to be designed by students. The Signal Extension block has no logic and only extends the 11-bit instruction's operand to a 16-bit word by copying the signal bit (bit 10) to bits 11-15.

IV. Using BIP in Courses on Digital Circuits and Systems Design

BIP was already applied on courses on digital circuits and digital systems design with laboratory activities based on design tools and development kits for FPGAs.

A. Use of BIP I in a Digital Circuits Design Course

In the undergraduate course on Computer Science of our university, first year students have a 60-hour course on Digital Circuits. In this course, practical activities are based on the design and simulation of digital circuits using FPGA tools from Altera (MAX+plus II and Quartus II). Since, none HDL is taught, the design entry method is based on the schematic editors available with the tools.

At the end of this course, a 3-hour class is performed in the laboratory. Firstly, BIP I is presented and students learn how it works and is programmed (this consumes about 1 hour). After that, students receive a top-level schematic diagram with all the building blocks interconnected (*i.e.* the diagram block of Fig. 3), but these building blocks are empty. Students are asked to implement them and validate their implementations by simulating the execution of programs that use all the instruction-set of BIP I. Students are then evaluated according to how much they reach a processor that successfully execute these programs.

In a class with 30 students, about 50% of them successfully completed their implementation in two hours. The other ones spent more time to fix some design errors (what was done later as homework).

This approach was shown to be effective for two reasons:

- It allowed evaluation of how much the student learnt about digital circuits design; and
- It showed to students how the digital circuits can be used to implement a programmable processor.

It is important to highlight that the use of a simple processor like BIP I is useful to save the time necessary to learn how a processor works and is programmed. Also, its low complexity allows the fast implementation of its building blocks.

B. Use of BIP II in a Digital Systems Design Course

In the Master Program on Applied Computer Science of our university, we have a course on Digital Systems Design. After a revision on basic concepts on digital circuits and systems, the following topics are covered: design methodologies and trade-offs, VHDL modeling, FPGA architecture, and the design of special-purpose and general-purpose processors.

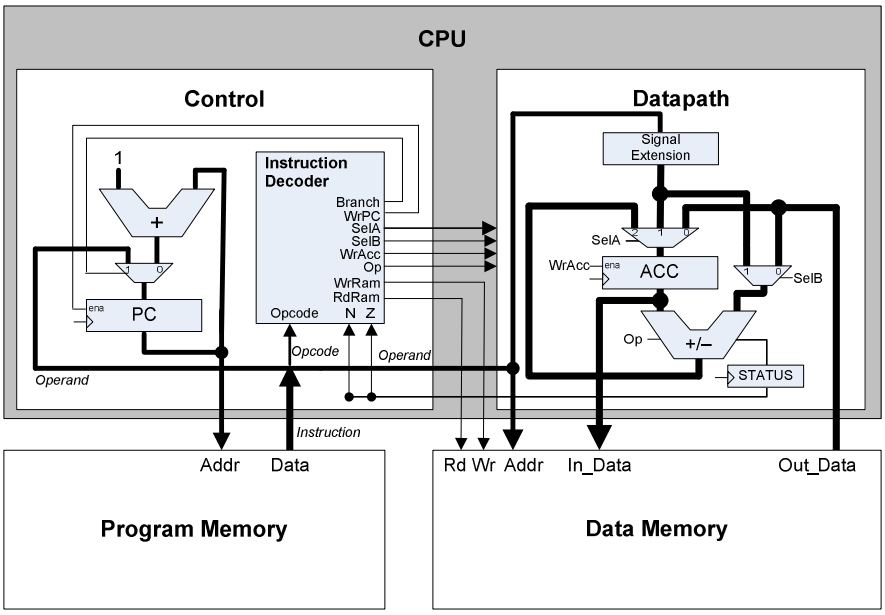

Figure 4. BIP II organization

As example of a general-purpose processor, we use BIP II in an 8-hour practical activity. Firstly, only the architecture and programming of BIP II is presented to the students, which are then asked to do the RTL design based on a monocycle/Harvard architecture (*i.e.* the diagram block of Figure 4). The only information they have is the processor architecture, including its instruction set (Table I). After designing the processor organization, the students are asked to describe the processor with VHDL, simulate the execution of test applications, and synthesize it in FPGA with prototyping in a development kit. In this course, we use Altera Quartus II design tools and Terasic development kits (DE0, DE1 and DE2).

In a class with 5 students (average size of classes in our graduate course), one of them obtained a full operating processor running in the development board in 7 hours. The remaining ones needed more than the 8 hours foreseen to this activity due to errors in their VHDL modeling.

BIP II showed to be effective to demonstrate the use of a design flow for implementing general-purpose processors. The highly regular architecture of BIP II allowed to students to easily design its organization after learning its architecture, what saved time for the other phases in the processor implementation.

V. DISCUSSION

The architecture of BIP family was specified with the goal of making easier the learning of basic concepts on computer architecture and organization by freshmen students. In order to meet this goal, the architecture was defined to be highly regular and simple. This resulted in the choice of using an accumulator-oriented architecture.

Such approach is different of the one adopted in most part of the 32-bit commercial processors used by industry, which are generally based on a register-file-oriented architecture. However, some 8-bit microcontrollers, like Microchip PIC16F628A [7], are based on an accumulator-oriented architecture.

Indeed, the specification of BIP architecture was based on some solutions adopted on PIC microcontrollers. However, while PIC16F628A uses four different formats to represent its instructions, BIP architecture uses a single format to represent all the instructions in the instruction set. Such feature results in a much more regular architecture, what make easier the design and the implementation of the processor organization (in special the instruction decoder).

Because of its simplicity, BIP presents some limitations, like the small addresses spaces and the need of a sequence of subtract and branch instructions to perform a compare-and-branch operation. However, they do not reduce the effectiveness of BIP in allowing students to easily learn how to design and implement a processor applying the concepts covered by courses about the design of digital circuits and systems.

VI. CONCLUSIONS

In this paper, we described the architecture and the organization of a basic processor that was specified under an interdisciplinary approach in order to improve the Teaching and Learning process in different disciplines in the Computer Science area, especially for freshman students.

This processor was applied in courses about the design of digital circuits and systems, in undergraduate and graduate courses. It allowed students to implement the processor in an easy and fast way, and showed to be an useful tool to integrate concepts and demonstrate how digital circuits are used in the implementation of processors.

BIP has also been used in courses on introductory programming, computer architecture and organization, and compilers design.

Beyond the architectures presented in this paper, we had also specified two other extensions: BIP III (with support for logic operations) and BIP IV (with support for procedures). This family of processors has also been used as reference for other works. A student designed and implemented a microcontroller based on BIP architecture, while other students implemented a visual Integrated Development Environment for BIP processors, with compiler, assembler and a full simulator for its architecture and organization.

Future works include a set of experiments to best evaluate the contribution of the use of BIP processors in improving the Teaching and Learning process in the undergraduate courses.

ACKNOWLEDMENTS

The authors thank to CNPq and CAPES, Brazilian funding agencies, which partially funded this research.

REFERENCES

[1] A. P. Malvino and J. A. Brown, Digital Computer Electronics, Career Education, 1992.

[2] L. Null and J. Lobur, The Essentials of Computer Organization and Architecture, 3rd ed., Burlington: Jones & Bartlett Learning, 2012.

[3] J. T. Khalife, “Threshold for the Introduction of Programming: Providing Learners with a Simple Computer Model”. In: Proceedings of the 28th International Conference on Information Technology Interfaces, 2006. pp. 71-76.

[4] V. G. Renumol; D. Janakiram and S. Jayaprakash, “Identification of Cognitive Processes of Effective and Ineffective Students during Computer Programming”, ACM Transactions on Computing Education. New York, v. 10, n. 3, August 2010.

[5] B. Haberman and O. Muller, “Teaching Abstraction to Novice Pattern-based and ADT-based Problems-Solving Processes”. In: Proceedings of Frontiers in Education Conference, 2008. pp. F1C-7-F1C-12.

[6] D. A. Patterson and J. L. Hennesy, Computer Organization and Design: the Hardware/Software Interface, 4th ed., Morgan Kaufmann, 2008.

[7] Microchip. PIC16F627A/628A/648A Data Sheet: Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology, Microchip Technology Inc., 2005.