Students' Experimental Processor: A Processor Integrated with Different Types of Architectures for Educational Purposes

L S K Udugama, PhD
Department of Electrical & Computer Engineering,
Faculty of Engineering Technology,
The Open University of Sri Lanka.
udugama@ou.ac.lk

Janath C Geeganage
Department of Electrical & Computer Engineering,
Faculty of Engineering Technology,
The Open University of Sri Lanka.
janathg@gmail.com

Abstract

Students who are beginners face difficulties in understanding basics of Computer Architecture. Their problems have been identified, and a project has been initiated to address these problems, with defined objectives.

The project is to be implemented in stages: design a processor; build a simulator; develop a compiler and develop an intergraded system. The paper presents the results of the first stage – design a processor. The features and characteristics of the processor are defined and the design process is described in detail.

The SEP (Students' Experimental Processor) integrates different types of architectures: Memory-Memory, Accumulator, Extended Accumulator, Stack, Register Memory, and Load Store. It can be switched to any one of them. It was modeled using VHDL and is ready to be implemented on FPGAs.

The SEP will support the introductory level students to understand the characteristics of computer architectures and their operations. Future developments of Computer Architecture education using the processor are also discussed.

1. Introduction

The Department of Electrical and Computer Engineering of The Open University of Sri Lanka offers specialization in Computer Engineering in the engineering degree program. In the year 2002 a new curriculum was introduced where Computer Architecture is one of the major components in the computer-engineering stream.

Computer Architecture education in our department is basically covered in four different courses taught at different levels. The first course, Communication and Information Technology is taken by freshers, where they are introduced to the components of a processor, how they are interconnected and how they function. Even though theory is explained by using a hypothetical machine, students use a simulator for the 8051 microcontroller to do their exercises. The second course at the next level covers microprocessor-based systems. Here students use the 8051-microcontroller development boards to do their laboratory work. At the next level, students study the five classic components of a computer [6] and other related areas such as performance and peripherals. At the final level students will learn to design a processor and they implement their processors on FPGA development boards in the laboratory.

Students, especially at lower levels, face difficulties in understanding basics of Computer Architecture. Some of these difficulties have been identified, and remedies suggested [1, 5, 7] According to our experience, we can highlight some of the problems faced by our students:

- Understanding the functions of the basic components of a processor.
- Visualizing how a processor actually executes an instruction and how data flows through its components.
- Assembly language programming of complex commercially available microprocessors/ microcontrollers.

- Comprehending the basic types and concepts of Computer Architecture.

We have initiated a project to find solutions to these problems. Here, we have mainly focused on the lowest level course, targeting students who hardly have any knowledge of computers/ processors. However we do not want to limit it to one particular course, but would like to see the possibilities of catering for higher level courses as well. Thus, the following objectives have been drawn up:

- To design a simple processor/s to teach different basic types of architectures, to demonstrate functions of components in a processor and how a processor functions.
- To use the same processor/s to do exercises in the subject.
- To build a simulator for the processor/s and use the same to demonstrate its function, to show the flow of data takes place within the processor and to write assembler programs and execute them.
- To keep provision to use the same processor for microprocessor development systems and to teach concepts in computer architecture.
- To use the processor in the course on, Processor Design, at the final level, as a case study.

We have searched the literature to try to locate a suitable system that meets all our objectives, but found that while some of the systems [3, 4, 8] and simulators [1, 2, 5] did cover our objectives severally no system could be found to cover all in our objectives.

We then decided to commence this project called **C**omputer **A**rchitecture **L**earning **S**ystem of The **O**pen **U**niversity of Sri Lanka (CALS-OU) to achieve our target. The project will be implemented in stages: design a simple processor**;** build a simulator to write, debug and execute assembly programs; develop a compiler to write programs in a high-level language; and finally develop a hardware/ software intergraded system facilitating students to learn concepts in Computer Architecture and to use the same for doing students' experiments. The first stage of the project has been completed, and this paper presents its results – a simple processor designed and named Students' Experimental Processor (SEP), which will accomplish our objectives.

2. How We Made the SEP a Reality

At the very outset of the design we defined the features and characteristics of the processor to be. The first is that the processor should comprise of four basic types of architectures: Accumulator, Memory-memory, Stack and Load Store. We then decided to include other two types, register memory and extended accumulator architectures [6]. We thought of including these as popular processors have these architectural styles. The next feature is that the processor should be able to change to one of the six architectures using hard or soft switches. To keep the processor simple and to make it easy for the programmer, we decided to limit the number of instructions to less than 32 and leave the I/O instructions out. However we did not want to limit the number of addressing modes as it is useful for students to learn to handle them in different situations.

The design process was comprised of the following key steps:

- Design of ISAs and applicable addressing modes, dimensions of data and address buses, size and the number of GPRs and depth of the stack.
- Design of Individual Data paths for the six architectures.
- Design of control sequences of the different instruction classes.
- Design of the top-level architecture. This is the architecture, which integrates all six architectures within one processor.
- VHDL modeling.
- Translator development.
- Testing.
- Documentation.

In the rest of the paper the design process, problems faced and the solutions arrived at are described.

3. Instruction Set Architecture

Basic instructions are included in the instruction set to provide an overall understanding of ISAs. All instructions can be grouped as Arithmetic & Logical, Control, Data Movement and miscellaneous instructions (Table 1).

Same set of instructions in each of the groups Arithmetic & Logical and Control Transfer is applicable to all architectures. Only the Data Movement instructions differ from machine to machine.

Control Transfer instructions refer the Flag register. The Flag register consists of 5 bits corresponding to the Parity flag, Sign flag, Overflow flag, Carry flag, and Zero flag. Instruction count is less than 32 for all machines.

Seven addressing modes are implemented in this processor. They are: immediate, direct, indirect, register direct, register indirect, index and displacement. Except displacement addressing, others are available for the user for accessing operands. Displacement addressing is used by the hardware in Control Transfer instructions. Operands can be in the memory, register bank, stack, or accumulator, depending on the active architecture. Addressing modes are declared in a way that it includes general modes and some popular modes. For clarity and consistency, the same instruction format is applied in all architectures.

Table 1. Instructions available in each architecture

Architecture	Arithmetic and Logic	Control Transfer	*Data Movement*	Misc
Memory to Memory	ADD, SUB, INC, MUL, DIV AND, OR, NOT, XOR, SHL, SHR, ROL, ROR	JC, JS, JP, JZ, JOF Call Ret Looz	LOAD	NOP
Accumulator			LOADacc STOREacc	
Stack			PUSH, POP, DUP, SWAP, ROTATE3	
Extended Accumulator			LOADacc, STOREacc LOAD, STORE	
Register Memory			LOAD, STORE	
Load Store			LOAD, STORE	

Instructions are in two-address format. Targeted implementation device has about 512K of 16-bit memory. A 19-bit address bus is required to address 512K locations. In an instruction there are two addresses (where applicable) and it needs 38-bits.

Instruction format (Figure 1)

- Opcode → 5-bits
- Addressing mode selection → 3-bits
- Addresses of two operands → 38-bits

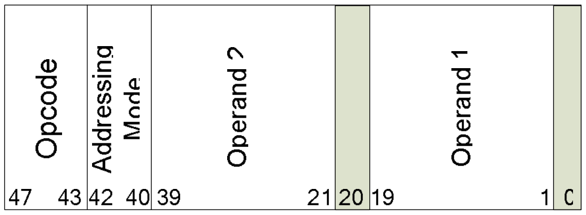


Figure 1. Allocation of bits in an instruction

Altogether 46-bits are required for an instruction. It occupies 3 memory locations. The length of the instruction is 48-bits.

4. Number Representation

Arithmetic and logic unit supports signed fixed-point numbers. Negative numbers are represented in two's complement form. The sign flag is replicated over bits 16 to 19, because the ALU is 19-bit and the data path is 16-bit. It is achieved by coupling the MSB to the three higher order bits.

5. Data Path Design

The structure of these architectures was designed in a simple manner. It has only the basic entities, which clearly shows the operation. For example, memory-to-memory architecture consists of the following entities,

IP; Instruction Pointer
MAR; Memory Address Register
IR; Instruction Register
OP1; a 16-bit register
SR; Status Register/ Flag Register
MDR; Memory Data Register
3S_buff; a three state buffer
Six multiplexers

5.1. Entities

Each of the architectures has a different data path, which facilitates the characteristic operation of its style. Almost all entities are effective in all architectures. There are two types of entities, namely, synchronized and asynchronized. Synchronized entities will function in an edge of the clock signal. Others change the output whenever a respective input is changed. Multiplexers and the ALU are non-synchronised entities. All registers are synchronised to the rising edge of the clock. Types of entities used in the processor are master-slave registers, shift register, 3-state buffer, register bank-stack-accumulator combined unit, Asynchronous SRAM and multiplexers. As the Register bank-stack-accumulator is a special unit, the block diagram is shown in the figure 2.

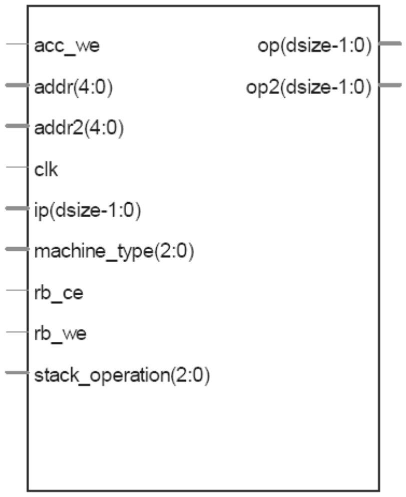


Figure 2. Interface of the Register bank, stack and accumulator combined unit

5.2. Handling Different Sized Busses

In indirect addressing, the address of the location retrieved from memory is 16-bit. It has to go to the MAR. MAR is 19-bit wide. In this case only the lower 16-bits are replaced. The data path facilitates this by coupling 3-MSBs of output of MDR to the input. This technique is used in several places to overcome the above situation. It results some limitations to the design.

5.3. Control Signals

Figure 3 shows the data path of the top-level processor, which facilitates six architectures mentioned above. Control signals to each entity can be found in the same figure.

6. Control Unit Design

A timing diagram was drawn for the execution of each instruction class to determine the sequence of control signals. The Control unit was designed using these timing diagrams.

"Functioning of the entities by the control signals" was important in designing the control unit, as the right signal has to be sent at the right time. A document containing all the entities and their control signals was prepared at the commencement of the design of the control unit.

7. VHDL Modeling

We used the Xilinx ISE Webpack 6.3i for VHDL modeling and for the simulation ModelSim 6.0a.

7.1. Modeling Strategy

Initially, the lower level entities were developed and tested for the expected results. When all entities are working successfully as individual entities, they were connected together to make the data, address and control paths according to the design done in the data path design process.

Behavior models were used in developing individual entities. Structural modeling was used in creating the top-level entity.

Then the control unit was programmed. Control states declared in the earlier stage were programmed using VHDL. The Control unit was tested as a stand-alone unit, and as an integrated entity, to verify its behavior.

Integrated testing was carried out after construction of the control unit.

7.2. Register Bank, Stack and Accumulator - The 3 in 1 Unit

Register bank, stack and the accumulator in the combined architecture was developed in a different way. As far as these three units are concerned, it is clear that two of them won't be available in a single architecture. Accumulator architecture doesn't have a register bank or a stack. Stack architecture doesn't have a register bank or an accumulator. Because of this, these three units were combined and developed as a single unit, which switches the operation according to the selected architecture. It optimizes resource usage.

8. The Assembly Code Translator

This system had to go through a testing process to confirm the proper execution of all instruction classes in the six architectures. In the simulation, machine level instructions had been fed to the memory module of the VHDL model. It needs converting instructions to operational code and addressing modes (if any) and operands to binary. The binary stream needs to be split into 16-bit segments to fit into the memory locations,

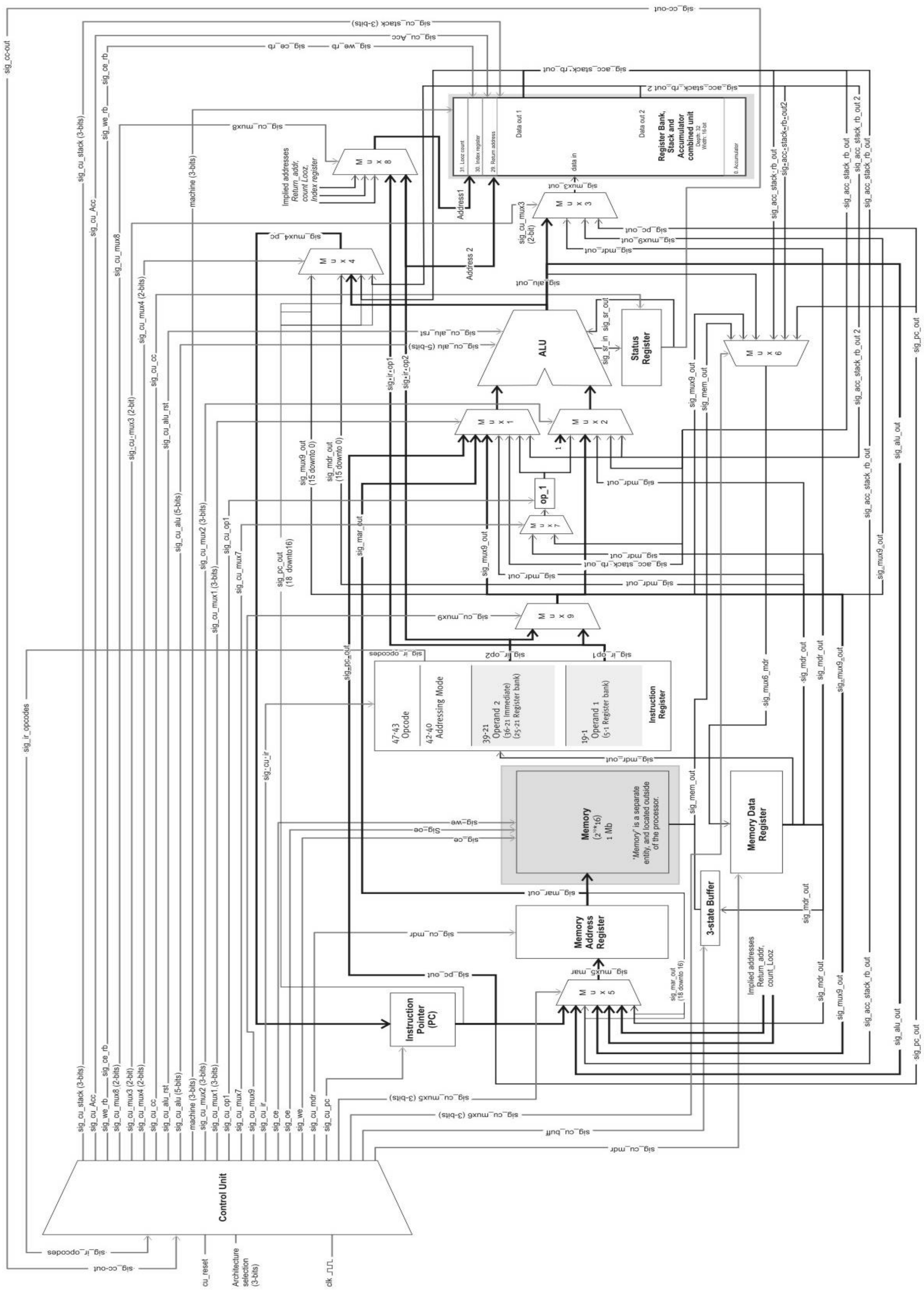

Control Unit
Instruction Pointer (PC)
Memory Address Register
Memory (2**16) 1 Mb
'Memory' is a separate entity, and located outside of the processor.
Memory Data Register
3-state Buffer
Instruction Register
47-43 Opcode
42-40 Addressing Mode
39-21 Operand 2 (36-21 Immediate) (25-21 Register bank)
19-1 Operand 1 (5-1 Register bank)
ALU
Status Register
Register Bank, Stack and Accumulator combined unit
Data out 1
Data out 2
data in
Address1
Address 2
op_1
Implied addresses Return_addr, count_Looz
cu_reset
Architecture selection (3-bits)
clk

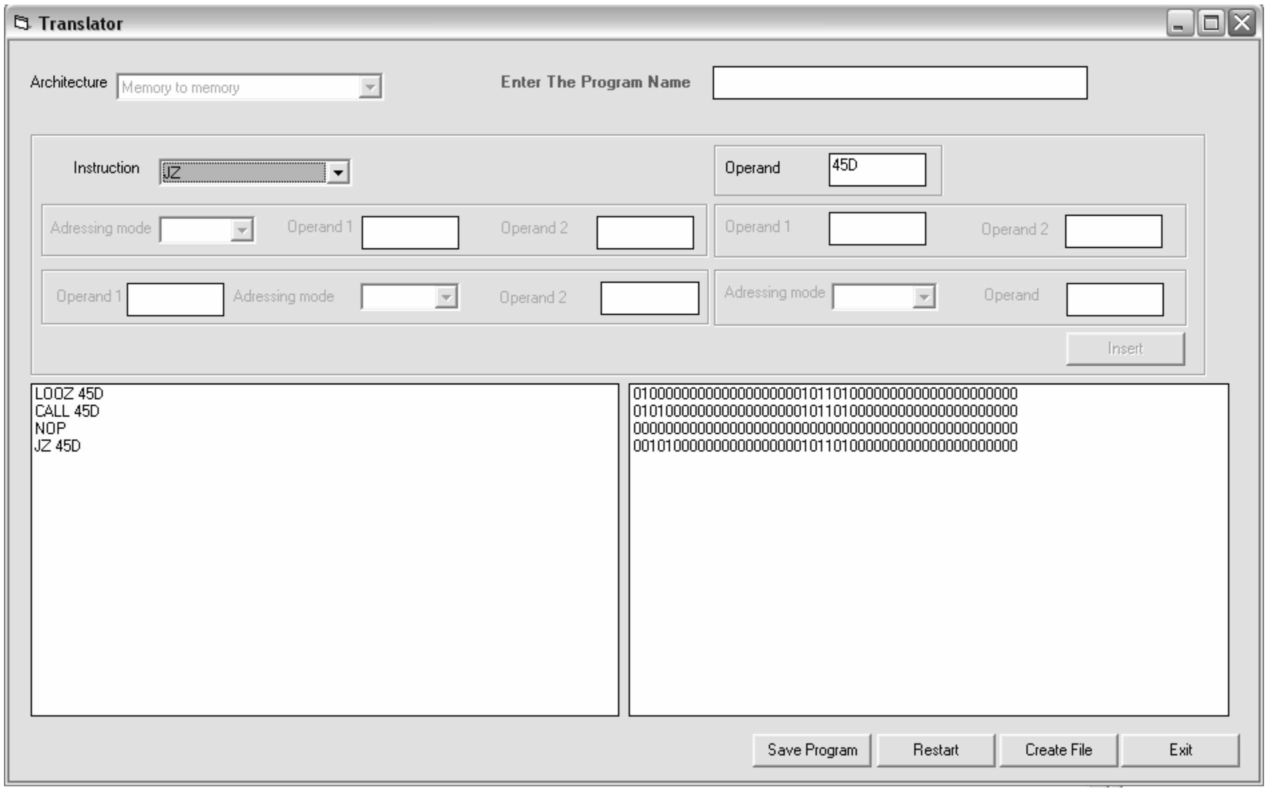
Figure 3. Data path of the top level architecture

starting from the initial memory address. This process was time consuming. One mistake in conversion would produce an entirely different result.

The translator was developed to overcome this situation. The major functions performed by the translator are:

- Validation of instructions, addressing modes and operands
- Automation of machine code generation
- Automation of VHDL memory initialization file creation
- Ensuring the reusability of programs

The Translator was developed using Microsoft Visual Basic 6. It uses a Microsoft Access database to store the written programmes for future reference. Figure 4 shows the interface of the translator.



Architecture: Memory to memory

Instruction: JZ

Operand: 45D

```
LOOZ 45D
CALL 45D
NOP
JZ 45D
```

```
010000000000000000000101101000000000000000000000
010100000000000000000101101000000000000000000000
000000000000000000000000000000000000000000000000
001010000000000000000101101000000000000000000000
```

Figure 4. Interface of the translator

The output of the translator is a text file, which also contains memory initialization data of the program written by the user. However, the user has to copy the contents of that file and paste it in the memory initialization section within the VHDL memory module. Using the translator makes the programming processes interesting.

Even though this was developed originally for testing, it provides a lot of benefits to the users, and helps to make their life easy. Moreover the students can use this translator to write their programmes and generate the binary code mapped to the memory. It makes life easy and saves a lot of time.

9. Testing

Testing was carried out to verify the proper operation of

- Each entity
- Each instruction class
- Each addressing mode under each instruction class
- ALU for status flag settings
- Synchronous operation
- Special cases such as register bank, stack and accumulator combined unit

9.1. Testing Strategy

A bottom-up testing strategy was followed. Bottom level entities were tested and verified first Followed by integrated testing. A testing check-list was prepared to measure progress, and to make the process an organised one.

9.2. Test Results

Successfully tested programmes were stored under the respective architecture. The system was checked with those after a modification is done, to verify the results.

10. Implementation

The target implementation device of this processor is the Spartan-3 FPGA board developed by the Xilinx Corporation, with the following features

- Affordable price
- 400,000-gate Platform FPGA – XC3S400
- Xilinx 2 Mbit Platform Flash configuration PROM – XCF02S
- 1 Mbyte of Fast Asynchronous SRAM
- Daughter card expansion ports

11. Documentation

As the target audience of this project is students, a considerable effort was made in producing proper documentation. It was decided to compile the documentation in separate documents. They are

- Programmer’s Manual (for the six processors) consists of detailed descriptions of six ISAs
- Description of the Processor Architecture, which includes the Functioning of entities over the control signals, Processor Architecture diagrams, Processor architecture, Abstract views of six architectures, Detailed control sequences of the Memory to Memory and Load Store architectures.
- Source code listing

All documentations and related programs are available on web http://www.ou.ac.lk/fac_etec/elec/udugama/.

12. Conclusions and Future Work

All six architectures are working perfectly according to their characteristics.

There are some limitations of this design. Call Return and indirect addressing can only be used in the same segment. In case where a Call instruction is called within a Call, the processor expects the compiler to do the necessary bookkeeping. This is also the case in the Looz instruction. For simplicity of the system, some of them were kept as it is.

Register bank, stack and the accumulator were combined into one unit, to minimize the resource usage on the FPGA. It automatically switches to the relevant storage according to the active architecture.

The next stage of this project is to develop a simulator for the processor. At the same time, we hope to develop a compiler for a high- level language as well. Once they are ready, it will be a good tool for beginners to learn basics of Computer Architecture.

However we hope to use this processor as a case study during this academic year for the course in Processor Design. It is possible to examine the effect of each control signal in each clock cycle using a VHDL simulator. Therefore it will be very helpful to students who are familiar with ModelSim to appreciate the internal operations of a processor.

There is a possibility to develop a microprocessor development board using the SEP. It is possible to implement the design on the FPGA as we targeted for Xilinx Spartan 3. In addition to that the processor is designed to interface the onboard 1 Mbyte asynchronous RAM as well. Therefore Spartan 3 board can be used as a microprocessor development board. However the interfacing units, keypad and a display have to be designed and connected to the board.

This processor can be improved further by pipelining instruction execution.

13. References

[1] Ola Agren, “Virtual machines as an aid in teaching computer concepts”, *Computer Architecture Newsletter*, IEEE Computer Society, September 2000, pp. 72-76.

[2] Jose R. Arias and Daniel F. Garcia, “Introducing computer architecture education in the first course of computer science career”, *Computer Architecture Newsletter*, IEEE Computer Society, February 1999, pp. 37-39.

[3] Milos Becvar, Alois Pluhacek and Jiri Danecek, "DOP — A CPU core for teaching basics of computer architecture," In *Proceedings of the Workshop on Computer Architecture Education*, 2003, pp 14-21.

[4] Jan Gray, "Hands-on Computer Architecture Teaching Processor and Integrated Systems Design with FPGAs", *Computer Architecture Newsletter*, IEEE Computer Society, September 2000, pp. 90-97.

[5] Enric Pastor, Fermin Sanchez, and Anna M. del Corral, "A Rudimentary Machine. Experiences in the Design of a Pedagogic Computer", *Computer Architecture Newsletter*, IEEE Computer Society, February 1999, pp. 51-53.

[6] David A. Patterson and John L. Hennessy, *Computer Organization & Design: The Hardware/ Software Interface*, 2nd edition, Morgan Kaufmann Publishers, San Francisco, 1998.

[7] Timothy D. Stanley and Mu Wang, "An emulated computer with assembler for teaching undergraduate computer architecture", In *Proceedings of the Workshop on Computer Architecture Education*, 2005, pp. 38-45.

[8] Yutaka Sugawara and Kei Hiraki, "A computer architecture education curriculum through the design and implementation of original processors using FPGAs" In *Proceedings of the Workshop on Computer Architecture Education*, 2004, pp. 3-7.