A Coherent Sequence of Computer Architecture Laboratory Assignments

Dorin Patru[1], Konboye Oyake[2], Eric Peskin[3]

Abstract - **The Computer Architecture course at the Rochester Institute of Technology (RIT) is taken by undergraduate students in their fourth year of study, after they have had an Introduction to Digital Systems, to Programming in C, and to Microprocessor Programming. The course gives students the computer hardware designer's perspective, with an emphasis on complete logic design. The objective of the laboratory is the design, simulation and implementation of a processor in a reconfigurable hardware device. Each weekly laboratory assignment builds upon the previous one. The bottom-top design process starts with the design of a combinational logic Arithmetic and Logic Unit, of a Register File and Memory Blocks. The design of the Central Processing Unit is divided into the design of the Data Path and Control Unit. The Instruction Set Architecture is enforced, i.e. the students do not have to come up with their own instruction set. All students must follow general and individual design specifications. The latter are selected using a binary code assigned to each student. The value of each bit chooses between design alternatives such as: Von-Neumann versus Harvard, I/O Mapped versus Memory Mapped Peripherals, 3-bus versus 2-bus architecture, tri-state versus multiplexer data transfer, hardwired versus micro-programmed control unit etc. Each final processor implementation is different from any other, but can run the same machine code. The paper presents the organization of the laboratory sequence, describes each weekly assignment and the lessons learned after offering it to six generations of students.**

Index Terms - Computer Architecture, Laboratory, Introductory Course, Hardware Design, Logic Design

INTRODUCTION

Introductory Computer Architecture or Computer Architecture and Organization courses are taught to junior and senior undergraduate students in Computer Engineering, Computer Science and Electrical Engineering. Depending on the major and of the particular curriculum organization, a course emphasizes the hardware designer's view, or the programmer's view, or strike a balance between the two. This variety in emphasis is also visible in the content of most current textbooks. References [1], [2] and [3] present a system level hardware design perspective. No block is described at gate level, but quantitative and qualitative system level tradeoffs are discussed. In comparison, references [4], [5], [6], [7] and [8] provide a hardware design perspective down to gate level. Finally, reference [9] explicitly states that it describes the computer architecture and organization from a programmer's view, i.e. without logic design implementation details. None of these different approaches to teaching computer architecture is better than the other, because each can serve a different set of instructional objectives [10].

In particular, the Computer Architecture course at Rochester Institute of Technology is preceded by an Introduction to Microprocessors course. In the latter, students learn about a microprocessor from a programmer's perspective, with a sequence of labs in which they learn assembly language programming. Consequently, in the Computer Architecture course, a bottom-up computer architecture design is taught, with an emphasis on complete logic design. The sequence of labs described in this paper is meant to complement such a teaching approach.

The laboratory or project assignments in a Computer Architecture course allow students to apply the computer architecture design methodologies presented in the lecture. While the value of this exercise is generally recognized, few textbooks propose laboratory or project assignments associated with the material covered. One such textbook is [8] and a few others use their accompanying websites. The assignments address individual topics, apparently independent and disconnected of each other. However, following a bottom-up design approach, students could design and implement in successive assignments individual computer hardware blocks. These would then be used to build a complete, low-complexity computer. The advantage of such a coherent and interdependent sequence of assignments is that it exposes students to a complete design and implementation cycle of a computer.

This paper first describes the sequence of laboratory assignments and its organization. Second, it analyzes how these labs help students with different learning styles. Third, the lab policy and grading are discussed. Fourth, student feedback and lessons learned are presented and analyzed. Finally, future enhancements to the sequence are considered.

[1] Dorin Patru, Assistant Professor, Rochester Institute of Technology, dxpeee@rit.edu
[2] Konboye Oyake, formerly with RIT, now with Vanteon Corp., konboye@hotmail.com
[3] Eric Peskin, Assistant Professor, Rochester Institute of Technology, erpeee@rit.edu

LABORATORY ASSIGNMENTS AND SEQUENCE ORGANIZATION

There are currently nine laboratory assignments, each spanning a week. The sequence is tailored to a 10 week quarter system, but as will be shown later it can be expanded to 14 weeks, or 14 assignments. As this is an introductory course, insufficient material will have been taught during the first week and therefore there is no lab in the first week.

The laboratory sequence starts in the second week. Students are first introduced to the instructional objectives of the lab [10]. By the end of this laboratory sequence, the student will be able to:

- Comprehend and explain the function of each hardware unit used to build a computer.
- Apply the computer design methodology to the design of a digital computer system.
- Classify computer architectures by different criteria.
- Analyze and evaluate the tradeoffs of different implementations.
- Design individual hardware units, and use them to synthesize a complete, low-complexity computer.

Then the laboratory sequence schedule is presented, as outlined in Table I.

TABLE I
LABORATORY SEQUENCE SCHEDULE

Lab/Week Number	Lab Title	Date Performed
2	Introduction to the CAD Tool	Week 2
3	Arithmetic and Logic Unit	Week 3
4	Registers, ROM and RAM Memories	Week 4
5	Processor Data Path - (Design)	Week 5
6	Processor Data Path – (Simulation & Implementation)	Week 6
7	Control Unit – (Design)	Week 7
8	Control Unit – (Simulation & Implementation)	Week 8
9	Complete Processor – (Simulation & Implementation)	Week 9
10	Complete Processor – (Emulation)	Week 10

As can be inferred from the table, at the end of the first seven labs, each student will have the necessary hardware units to synthesize a complete, low-complexity computer.

After the reporting requirements and grading policy are presented, the lab commences with an "Introduction to the Computer Aided Design (CAD) Tool". This can be any tool which provides a graphic, i.e. schematic, or text based, i.e. Hardware Description Language – HDL, design entries, is able to logically simulate a circuit, and synthesize it for a reconfigurable device. During this introduction to the CAD tool, students design, capture, simulate and synthesize a decoder and a counter. These are then downloaded to the target reconfigurable device, for example a Field Programmable Gate Array, and debugged. At the end of this lab, the students will have been introduced to the complete logic design flow for a reconfigurable device.

In the third week, the students design, capture and simulate a combinational Arithmetic and Logic Unit (ALU). The operations it implements are shown in Table II. To reduce the amount of hardware necessary to implement the final processor, the operands are only four bits wide. The block diagram of the ALU, which is shown in Figure 1, is given to the students. However, students need to create the gate level design of each block. Although quantitatively small, the design remains qualitatively valid for operands with larger widths.

TABLE II
ARITHMETIC AND LOGIC UNIT FUNCTIONS

Operation Select - FS[3..0]					
FS3	**FS2**	**FS1**	**CIN**	**Operation**	**Function**
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A + 1	Increment A
0	0	1	0	F = A + B	Addition
0	0	1	1	F = A + B + 1	
0	1	0	0	F = A + (not B)	
0	1	0	1	F = A + (not B) + 1	Subtraction
0	1	1	0	F = A – 1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = not A	NOT
1	0	1	X	F = A AND B	AND
1	1	0	X	F = A OR B	OR
1	1	1	X	F = A XOR B	XOR

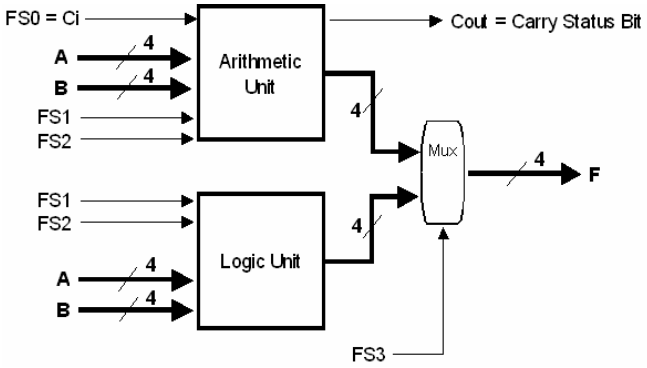

FIGURE 1
BLOCK DIAGRAM OF THE ARITHMETIC AND LOGIC UNIT.

In the fourth week, the students design, capture, initialize the content and simulate storage blocks, i.e. Registers, Read Only and Random Access Memory Blocks.

The computer design starts effectively in week five. Students are given a pre-designed Instruction Set. It contains 16 instructions, covering all fundamental operations, as can be seen in Table III. For the data manipulation instructions the source operands are registers A and/or B, and the destination is always register A. Thus the architecture is load-store or register-register. The special use of register A makes it also an accumulator architecture. The source operands and destination being implicit, the Instruction Word of the manipulation instructions contains only the 4-bit Operation Code.

The data transfer instructions LOAD and STORE have an 8-bit Direct Address value appended to the 4-bit Operation Code, i.e. they require two additional readings from the Program Memory or Code Segment. Similarly, the Flow

TABLE III
Instruction Set

#	4-Bit Instruction Code *	Instruction Mnemonic	RTN Description	Comments
0	0000	ADD	C#A ← (A+B)	Four-bit result gets stored in A. Carry out gets stored in C.
1	0001	SUB	C#A ← (A-B)	
2	0010	INC	C#A ← (A+1)	
3	0011	DEC	C#A ← (A-1)	
4	0100	NOT	A ← (A')	
5	0101	AND	A ← (A•B)	
6	0110	OR	A ← (A+B)	
7	0111	XOR	A ← (A⊕B)	
8	1000	JMPU	See JMPC below for C='0'	Jump Unconditional
9	1001	JMPC	MAH ← M[PC] : PC←PC+1 ; MAL ← M[PC] : PC← PC+1 ; C=0→ PC ← MA	Jump Conditional **← This is an 8-bit transfer!!!**
10	1010	SWAP	A ← B, B ← A	
11	1011	COPY	B ← A	REG-A remains Unchanged
12	1100	STORE	MAH ← M[PC] : PC←PC+1 ; MAL ← M[PC] : PC← PC+1 ; M[MA] ← A	Write contents of REG-A to memory location
13	1101	LOAD	MAH ← M[PC] : PC←PC+1 ; MAL ← M[PC] : PC← PC+1 ; A ← M[MA]	Read contents of memory Location into REG-A
14	1110	IN	MAH ← M[PC] : PC←PC+1 ; MA<7>=1→A←SWH: MA<7>=0→A←SWL	Read Input peripheral into REG-A
		PUSH **	STK[0] ← A: STK[1] ← STK[0]: STK[2] ← STK[1]: STK[3] ← STK[2]	Push A onto the stack
15	1111	OUT	MAH ← M[PC] : PC←PC+1 ; MA<7>=1→DISPH←A MA<7>=0→DISPL← A	Write contents of REG-A to Output Peripheral
		POP **	A ← STK[0]: STK[0] ← STK[1]: STK[1] ← STK[2]: STK[2] ← STK[3]: STK[3] ← 0;	Pop A off the stack

Legend:

A<3..0>: = A Register (4-Bit)	MA<7..0>: = MAH<3..0>#MAL<3..0> Memory Address register (8-Bit, divided into two four-bit halves, MAH and MAL)	
B<3..0>: = B Register (4-Bit)	C: = Carry Bit (a single-bit register that holds the carry from the most recent arithmetic operation)	
PC<7..0>: = Program Counter (8-Bit)	SWH<3..0>: Value on the left-hand (high-order) set of four switches on the board. SWL<3..0>: Value on the right-hand (low-order) set of four switches on the board. DISPH<3..0>: Four-bit register whose value is continuously displayed on the left-hand (high-order) seven-segment display on the board. DISPL<3..0>: Four-bit register whose value is continuously displayed on the right-hand (low-order) seven-segment display on the board.	
IR<3..0> := Instruction Register (4-bit) Holds the OpCode of the currently executing instruction.	STK[0..3]<3..0> Four-bit, four-location stack. **	M[0..255]<3..0> := Memory (256 words of four bits each).

Control Instructions JMPU and JMPC use an 8-bit target address, which is read from the next two locations after the Operation Code. The IN and OUT instruction only read the next 4-bits, as they only use the Most Significant Bit.

Although during weeks five and six students are only concerned with the design and simulation of the Data Path, the General and Individual Design Specifications of the computer are presented here.

The General Design Specifications are:

- The Register File contains two 4-bit registers.
- The Arithmetic and Logic Unit is the one designed in week three.
- The Memories contain 256 4-bit locations, i.e. use 8-bit addressing and 4-bit Input/Output Data Buses.
- The Program Counter is eight bits wide.
- The Memory Address Register is also eight bits wide.
- There are two 4-bit Input Ports, and two 4-bit Output Ports.

Each student is assigned a unique 5-bit binary code number. This is used to infer the Individual Design Specifications using Table IV. Using these individual design specifications accomplishes two purposes. First, it encourages peer tutoring while simultaneously discouraging copying. Second, it allows students to compare different architectural features by learning about their peers' implementations.

The four architectural combinations that arise from combining B0 and B1 are shown in Figure 2.

TABLE IV
INDIVIDUAL DESIGN SPECIFICATIONS

Bit	Value = '0'	Value = '1'	Comments
B0	Von Neumann	Harvard	Memory System Architecture
B1	I/O Mapped	Memory Mapped	I/O System Architecture
B2	Hardwired	Micro-Programmed	CU Implementation
B3	3-Bus	2-Bus	Internal Bus Architecture
B4	Tri-State	Multiplexer	2nd & 3rd BUS implementation

Students whose B1=0 will implement the instructions IN and OUT. Students whose B1=1, will use the LOAD and STORE instructions to access the Input/Output Ports. These latter students will implement an additional hardware based Stack, and add the PUSH and POP instructions to their Instruction Set. Through the use of these individual design specifications, each student designs and implements her/his own unique computer.

In week five, students do a paper and pencil design of the data path. They will have seen a similar, but not identical, Data Path design in the lectures preceding this lab. Thus, the students do not get a design to copy and paste, but rather they use the template from the lecture and create their own solutions for their individual computer. Such a design is not reproduced here, as it can be found in textbooks. The paper and pencil design of the Data Path encompasses the creation of the following documents:

- A schematic with gate level detail.
- An Algorithmic State Machine Chart or State Diagram, which uses:
  - State Boxes, in which operations that happen in one machine cycle are described using Register Transfer Notation[4] Language.
  - Conditional Boxes
  - And (optional) Conditional Output Boxes.
- A Control Signal Table, which contains in the header the list of all control signals and in the body their values for each machine cycle. The information in this table is used as a design input for the Control Unit.

In week six, students capture and simulate their Data Path design. The design can be captured either using a

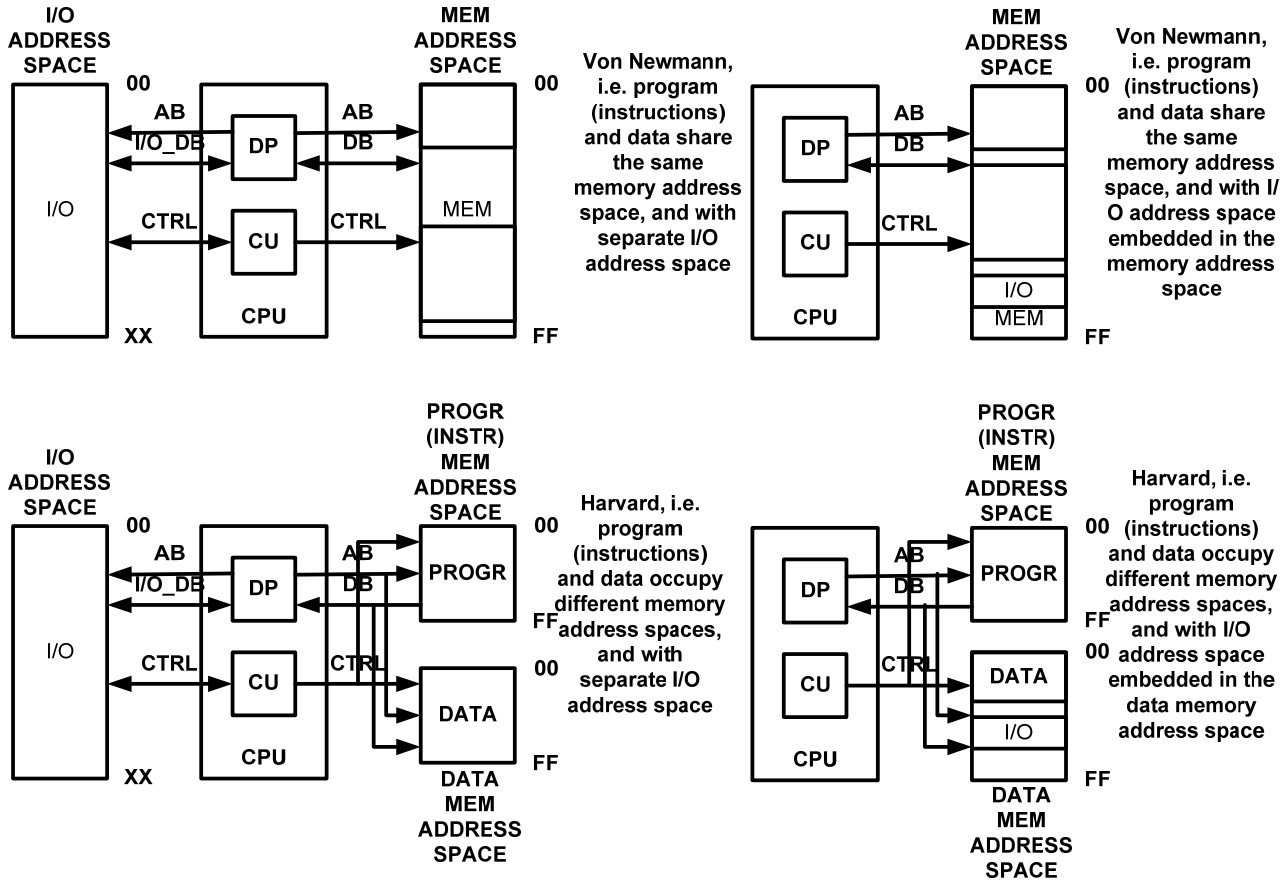

FIGURE 2
THE FOUR ARCHITECTURES THAT ARISE FROM COMBINING B0 AND B1.

schematic, or one of the established Hardware Description Languages: VHDL or Verilog.

The paper and pencil design of the Control Unit is performed in week seven. Based on the value of B2 in Table IV, students design either a Hardwired or Micro-Programmed Control Unit. As with the Data Path, they will have seen in the lecture an example / template design of each kind. The Control Unit is captured and individually simulated in week eight.

In week nine, students connect the Data Path and Control Unit together. Using short instruction sequences, they first verify the entire instruction set. Then they continue with full-length programs. Once these work correctly, they download the computer to the reconfigurable device, i.e. to the Field Programmable Gate Array.

Week ten represents the culmination of their efforts: running and testing programs on their own computer. On the prototype board that carries the Field Programmable Gate Array, the Input Ports pins are connected to eight switches and the Output Ports pins are connected to two seven-segment displays. With these input and output devices, the computer can run programs that would implement functions similar to a handheld calculator. One of the test programs they are required to run successfully performs multiplication through repeated additions.

Although low in complexity, and therefore limited in capabilities, the design of every one of the 32 different computers force the students to get involved in every aspect of the design process of a computer. In a 14 week sequence, one could further:

- Extend the instruction set.
- Introduce several Addressing Modes. Add B5 to differentiate between different Addressing Modes.
- Increase the Main Memory and introduce a small Cache. Add B6 to differentiate between say Direct Mapped and Set Associative Caches.
- Introduce the option of a Pipelined Data Path.

Although these architectural features are covered in the lecture, they cannot be practiced in the 9 week sequence presented above.

ARE ALL STUDENTS BEING SERVED?

The purpose of these laboratory assignments is to complement the lectures, offering a hands-on approach to learning. As is the case with every pedagogical tool, one needs to ask the question: does it serve, i.e. help all students in the learning process? In this section we attempt to answer this question by considering different student learning styles [11].

From a perception point of view, *learners* are classified into *sensing* or *intuitive*. *Sensing learners* are practical, like concrete thinking, hands-on work and are methodical. The laboratory assignments are practical, because they target a final physical implementation, they are hands-on and students have to follow a design methodology. *Intuitive learners* are imaginative, like abstract, model-based thinking, and like variety. Students have to create their own solutions to design problems, which are never the same from one week to another.

From an information input mode point of view, *learners* are classified into *visual* or *verbal*. *Visual learners* like graphic input, such as the schematics and charts the students use in these lab designs. *Verbal learners* like text-based input. These will find the information contained in the tables rather than in the charts, and could capture their designs using one of the Hardware Description Languages.

From an information-processing point of view, *learners* are classified into *active* or *reflective*. *Active learners* like to try out and work in groups. During simulation and verification, students perfect their designs through a lot of trial and error. Although the assignments are individual, peer tutoring is allowed because the specific design requirements make it impossible to copy/paste without an understanding of how to use the block. *Reflective learners* like to think it thoroughly and work solo. The assignments obviously do not preclude such an approach.

Finally, from an understanding point of view, *learners* are classified into *sequential* and *global*. *Sequential learners* can function/work with partial information. A student does not have to understand the whole picture in the first weeks to make steady progress and finish the lab sequence successfully. *Global learners* need the big picture. This is being referred to throughout the lectures, but these Learners will benefit most during the last weeks when the whole design comes together.

The activities associated with these laboratory assignments do not favor any particular learning style, but give each *learner* the opportunity to benefit from completing them.

GRADING, ATTENDANCE AND REPORTING

While each instructor can have her or his own grading policy, for these labs, we have used the following breakdown: Lab of Week2 = 1%, LW3 = 2%, LW4 = 2%, LW5 = 5%, LW6 =5%, LW7 = 5%, LW8 = 5%, LW9 = 5% and LW10 = 10%. Thus, the total lab grade represents 40% of the total grade for the course. The first three labs are lower weighted because these are introductory. The last lab grade is highly weighted to motivate students to complete the entire computer. To receive any points for the final lab, a submission must successfully execute at least six of the sixteen instructions. Each additional instruction is worth one point for a total of ten points possible for the final demonstration.

Attendance is mandatory. If a student misses a lab session, she or he needs to attend another lab session. At Rochester Institute of Technology, lab sessions are conducted by a Teaching Assistant supervising a maximum of 16 students. However, as instructors, we found it helpful to attend lab sessions in weeks 5, 7 and 10. The lab sessions are currently two hours long, and extending them to three hours could give more time for assistance. However, in either case, the lab session is only the starting point of work that has to be completed over the next week. Thus, students must have access to the CAD tool outside the lab sessions, and the instructor and teaching assistant(s) must have a sufficient number of office hours.

While we value the exercise of writing a report for each laboratory assignment, because of the iterative nature of the design process, and the implicit high number of changes and fixes to the designs, we do not require weekly written reports. However, we do require a final laboratory report at the end of the tenth week session. This has to contain:

- A one page text description of the general and individual design specifications.
- A two page text description of design choices, problems encountered, unresolved requirements.
- All schematic diagrams.
- All text based entry files.
- Annotated waveforms showing one complete instruction cycle for each instruction.

STUDENTS FEEDBACK – LESSONS LEARNED

Because the course is offered in two quarters of each academic year, this sequence of computer-architecture labs has been taken by six generations of students, more than 300 students in all, over the last three years.

As for any other lab, at the end students complete a five-question, qualitative lab evaluation. Note that this is in addition to the overall, separate course evaluation. Three of the five questions pertain to the laboratory assignments, the other two ask about the Teaching Assistant's performance.

Question 1: Which aspects of the laboratory assignments or procedures were of most value? Answers: *all, the fact that everything built on everything else, designing my own computer, all parts helped in understanding of how a computer works, the practical implementation of what was taught in the course, being introduced to the entire design process, hands-on.*

Question 2: Which aspects of the laboratory assignments or procedures need improvement? Answers: *none, need three hour lab sessions, more than one Teaching Assistant, more detail in the lab handout, more time*. Despite these issues, in each generation, more than 70% of students completed the design and implementation of the entire computer. Another 25% had the data manipulation instructions working. The remaining 5% did not finish, because they did not complete the early labs on time. The most difficult instructions to complete seem to be the flow control, i.e. JMPU and JMPC, and the data transfer with the memory, LOAD and STORE.

Question 3: Did the laboratory work complement the course lecture and other work? Answers: *yes, very well, everything was relevant to the course material and greatly helped in understanding it, very helpful, yes it would be difficult to understand without applying it.*

In addition to this direct feedback, we have received verbal or email reports from students, who have successfully used the complete computer design in interviews to demonstrate their skills.

We interpret this feedback as very good. As we cannot allocate more than one Teaching Assistant to one lab session, and cannot expand time to mitigate the issues raised in the answers to question 2, we now specifically encourage students to plan and manage their time very carefully. In addition, we closely watch each student's progress, and intervene with help when we notice she or he is at risk of falling behind. The successful use of this sequence of laboratory assignments, definitively requires substantial instructor involvement.

FUTURE ENHANCEMENTS

As capabilities of reconfigurable hardware devices increase, we plan to provide to the student some pre-designed keyboard/mouse and display interface blocks. The student computers would read/write these through the four peripheral ports. This improved user interface will make their computer look and feel more like the personal computers with which they are familiar, stimulating them to write and run more complex programs. Towards this end, the amount of memory could be increased to allow for a small high level language interpreter. Finally, as we have formerly indicated, in a 14 week sequence the computer architecture itself can be very much enhanced.

ACKNOWLEDGMENT

The Authors wish to acknowledge the many students who have provided constructive feedback and helped to refine this sequence of laboratory assignments. We also want to thank the more than 10 Teaching Assistants for their hard work and dedication in the successful use and running of these labs over the past three years.

REFERENCES

[1] Hennessey and Patterson, *Computer Architecture – A Quantitative Approach,* 3rd Edition, Morgan Kaufmann 2003.

[2] Flynn, Michael, J., *Computer Architecture – Pipelined and Parallel Processor Design,* Jones and Bartlett, 1995.

[3] Shen, John, P., and Lipasti, Mikho, H., *Modern Processor Design,* McGraw Hill, 2005.

[4] Heuring, Vincent, P., and Jordan, Harry, F., *Computer Systems Design and Architecture,* 2nd Edition, Pearson Prentice Hall, 2004.

[5] Clements, Alan, *Principles of Computer Hardware,* 4th Edition, Oxford, 2006.

[6] Hayes, John, P., *Computer Architecture and Organization,* 3rd Edition, McGraw Hill, 1998.

[7] Mano, Morris, M., and Kime, Charles, R., *Logic and Computer Design Fundamentals,* 3rd Edition, Pearson Prentice Hall, 2004.

[8] Comer, Douglas, E., *Essentials of Computer Architecture,* Pearson Prentice Hall, 2005.

[9] Null, Linda, and Lobur, Julia, *Computer Organization and Architecture,* Jones and Bartlett, 2003.

[10] Bloom, B.S., and Krathwohl, D.R., *Taxonomy of Educational Objectives – Handbook I – Cognitive Domain*, Addison-Wesley, 1984.

[11] R.M. Felder and L.K. Silverman, "Learning and Teaching Styles in Engineering Education," *Engr. Education, 78*(7), 674-681 (1988)