DESIGN PRACTICES IN COMPUTER ARCHITECTURE TEACHING

Guoping Wang
Indiana University Purdue University
Fort Wayne, Indiana
Email:wang@engr.ipfw.edu

Introduction

Computer Architecture is a common upper-level engineering course that is offered at universities to Computer Science and Computer Engineering students throughout the world. Most of the available educational approaches vary in how they handle computer system and run the microprocessor simulations on the computers.[1,2,3,4,5,8,9] For example, the popular MIPS simulator, SPIM[5] is a MIPS microprocessor simulator developed at the University of Wisconsin for undergraduate teaching in computer architecture. It reads and executes MIPS assembly language programs written for this processor and it also provides a simple debugger and minimal set of operating system services, but it does not execute binary (compiled) programs. SPIM simulator implements both a simple, terminal-style interface and a window interface. VMPIS[3] simulator simulates a MIPS R3000 RISC CPU core and it is an open-source project started and supported by Brian R. Gaeke[3] and distributed under the GNU General Public. VMIPS allows developers to write and test code for a MIPS CPU. VMIPS hopes to be a teaching environment for computer architecture courses where students can write software for the MIPS RISC instruction set. It also hopes to be a development environment for people writing extensions to the simulator, like adding more devices or changing CPU behavior. These MIPS simulator together with other commercial and non-commercial software are adopted in the teaching activities of Computer Architecture.

However, in these approaches, students are only exposed to the instruction set of the computer architecture and the students don't have the chance to build an original processor from scratch. Through the simulation, students understand the computer architecture intricacies through the use of assemblers, architecture simulators, compilers, while it may be enough to Computer Science students, for Computer Engineering students, they require more details about computer architecture than a black box approach can provide. On the other hand, a major problem in teaching computer architecture course is how to assist students make the leap that connects their theory of principles and theories with practical experience. The best way to learn about computer architecture is to design and build one. Unfortunately, computers have become so sophisticated the designing a full-scale CPU and building them in hardware in one semester is not feasible.

In the teaching of Computer Architecture at Indiana University Purdue University Fort Wayne (IPFW), a simplified computer system, which is a modified version[7], is introduced that lets the students design, build and test it on FPGA. Due to the complexity of the system, schematic approach is not practical. Thus, VHDL hardware description language is used through the course. This computer system has a ALU, some registers, and an output interface to the XSA-100 FPGA board. Students will use Xilinx ISE Design and Implementation tools, together with Mentor Graphics Modelsim logic simulator and XESS XSA-100 FPGA board to design, simulate and verify the functionality of this CPU unit. Three projects are assigned to the students. Project 1 requires the students to design and simulate two components of the CPU, the register stacks and the ALU unit. In project 2, students will design, simulate and verify the simplified CPU unit with a datapath, instruction encoder, instruction ROM, register stacks and ALU on the FPGA board. A calculator which uses the simplified CPU

structure is also implemented in this project. Project 3 involves with the further applications of the CPU unit which includes the arithmetic operation of signed binary numbers, a complier application from C code to Assembly language and its implementation on the CPU unit. Through this design process, the compiling theory, the machine language, the internal CPU architecture and the data path will be introduced to help the students better understand the computer architecture.

The rest of this paper is organized as follows: A brief course overview of the current Computer Engineering curriculum in computer hardware area. A description of the teaching background, equipment and facilities of Computer Architecture teaching activities in IPFW. The teaching activities in details are then presented. The conclusion of the paper.

Course Interview

In the area of computer hardware courses for Computer Engineering at IPFW, a four-course sequence is followed which consists of ECE 270 – Introduction to Digital Systems Design, ECE 357 – Introduction to VHDL, ECE 362— Microprocessor Systems and Interfacing and ECE 495A – Computer Design and Prototype. ECE 270 covers the basics of combinational logic circuit analysis and design, sequential logic circuit analysis and design, finite state machine, Karnaugh map., etc. It also has a very important laboratory session which exposes students the design and practice of the digital systems. ECE 357 introduces digital system design using VHDL. Topics covered include VHDL concurrent and sequential statements, signals and variables, state machine design, VHDL synthesis, simulation. Hand-on projects on FPGA board are assigned to the students. ECE 362 covers the topic of 80x86 series microprocessor. The assembly language, computer interfacing are introduced. The MIPS architecture, datapath, pipeline, memory hierarchy and I/O basics are covered in ECE 495A. This four-course sequence gives the students a systematic knowledge of the computer system, from the simple Boolean algebra, logic gates to the computer architecture.

Background

In ECE 495A at IPFW, the most popular textbook for computer architecture by Patterson and Hennessy[6] was adopted in the teaching. MIPS processor architecture is introduced. Three technological advances, 1) powerful EDA tools, 2) fast hardware prototyping facilities (FPGA) and 3) hardware description language, enable teaching this course through the effective implementation of a simplified RISC processor. Due to the complexity of the processor, schematic approach is not practical. VHDL hardware description language is used to design and capture the processor architecture. Xilinx ISE and Mentor Graphics Modelsim logic simulator are used to synthesize, simulate and verify the design. Then the XSA-100 FPGA board from XESS company is used to download the design and test it. The XSA-100 FPGA board has a 100,000-gate Spartan2 FPGA chip on board with interface peripherals such as LEDs, push buttons, DIP switches, etc. It also combines a 16M Bytes DRAM and 256KByte Flash to give you the resources for building a complete, soft-core RISC processor system. Figure 1 shows a picture of the XSA-100 FPGA board.

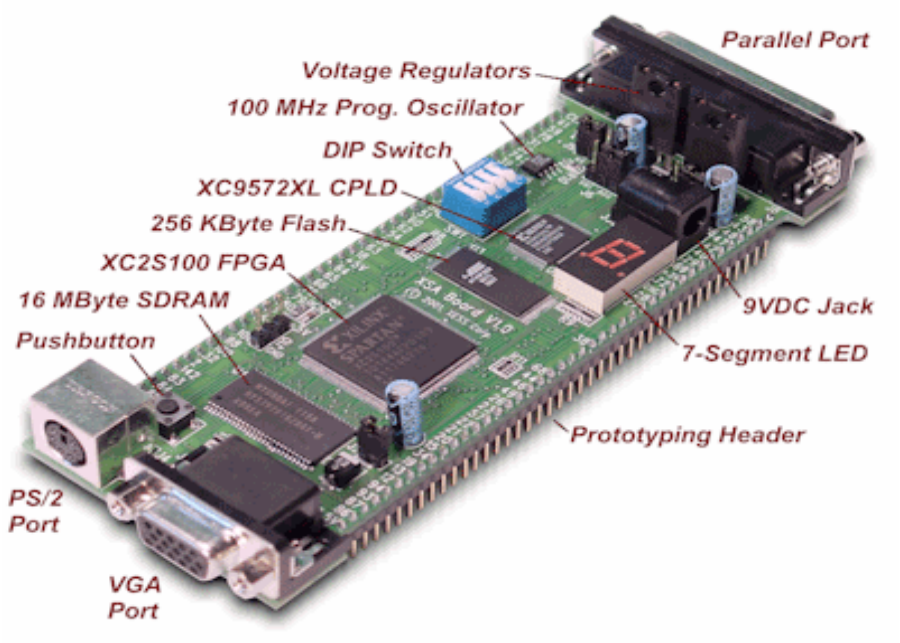

Figure 1. XESS XSA-100 FPGA Board.

Education Activities

In this class students learn and understand the computer architecture through the design a single-cycle processor with a RISC instruction set, datapath, ALU unit and an interface to the FPGA board. Three projects are assigned to the students and each one takes about four weeks.

Project 1: The design of Register Group and ALU unit.

In this project, students will implement and simulate some components of a processor. All the components to be designed in this project are building blocks rather than complete circuits. Therefore, it will not be necessary to test them on the FPGA board. Only the simulation results are required.

- **Project 1 Part 1: Register Group**

This RISC processor will have eight general-purpose, 8-bit wide registers which will be contained in a register file as shown in Figure 2.

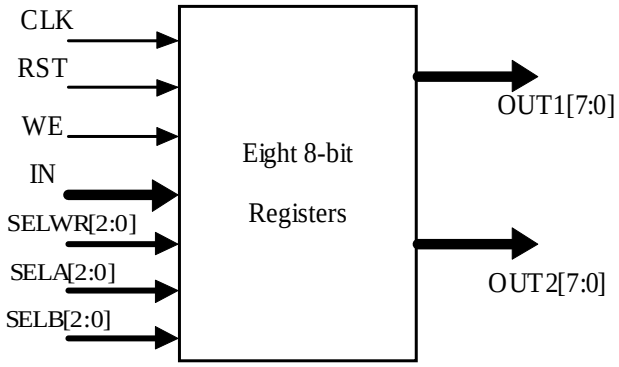

Figure 2. Register File for the Processor.

The register file has one data input and two data outputs. Control signals are used to select which register (if any) the data on the input lines will be written into (i.e. destination register) and which registers will be displayed at the outputs (i.e. operands).

Students will design the register file using VHDL according to the following specification, simulate their design using Modelsim simulator.

- Registers are 8 bit wide, and the register file contains eight registers called R0 through R7.

- The register file has two 8 bit data outputs labeled OUT1 and OUT2 and one 8 bit data input, IN.

- Data is written into the register on the rising edge of the clock.

- The control input `SELWR[2..0]` selects which register will be written; e.g. if `SELWR[2..0]=5` then the data at the input is written into the register R5 on the next rising edge while the content of all other registers remains unchanged (Hint: use the decoder to select the register).

- The control input `WE` is a global write enable signal. If this signal is low, the content of all registers remain unchanged regardless of the value at the `SELWE[2..0]` input. Therefore, `WE` must be high to write into the register (Hint: consider using the 'enable' input on the decoder macro).

- The control signals `SELA[2..0]` and `SELB[2..0]` select which registers will pass their content to outputs OUT1 and OUT2 respectively.

- The register file is clocked by a common clock input signal and can be *asynchronously* cleared with a common reset signal.

A sample of the simulation waveform for this register file is shown as Figure 3.

➢ Project 1 Part 2: ALU Design

In this part, students are asked to design an 8 bit ALU that has two 8 bit inputs IN1 and IN2 and an 8 bit output ALU_OUT as shown in Figure 4. The circuit should also have a 3 bit input SEL which is used to select the operation and a single output ZERO which is high whenever the output of the ALU is all zeros. Based on the settings of the SEL input, the ALU should perform one of the following operations:

SEL	Outputs
000	IN1 and IN2
001	IN1 or IN2
010	IN1 xor IN2
011	not IN1
100	IN1+IN2
101	IN1-IN2
110	IN1<<1
111	IN1>>1

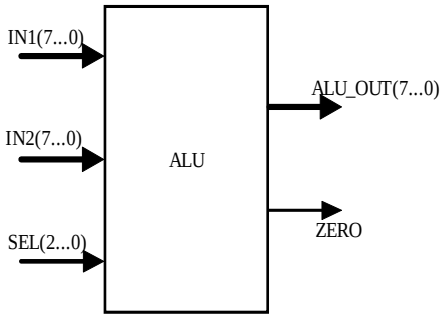

Figure 4. ALU Diagram.

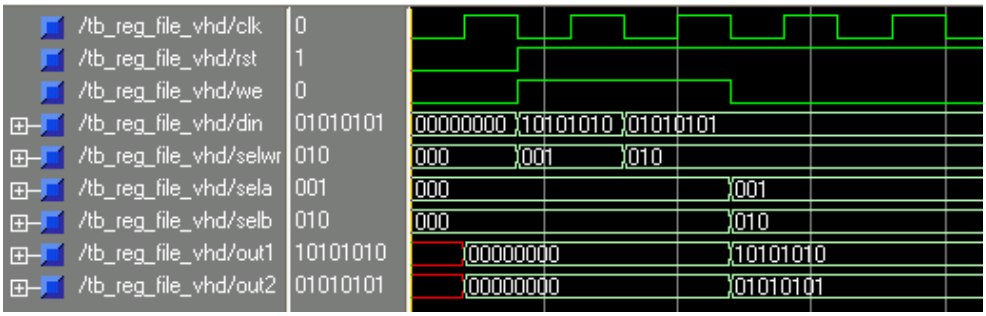

Figure 3. Register File Simulation Waveform.

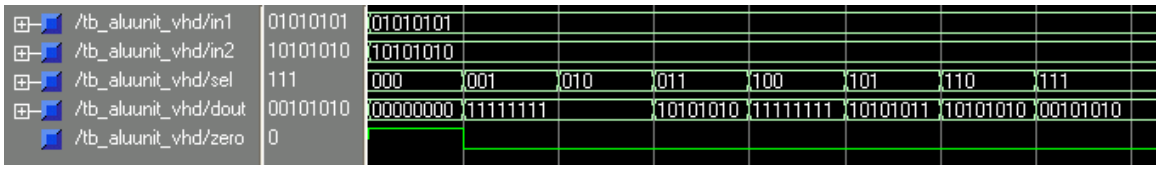

Figure 5. ALU Simulation Waveform.

Students are also required to simulate the ALU design using Modelsim simulator. A good test vector should be composed of all eight ALU functions and some examples to test the ZERO signal output. Figure 5 shows such a sample of the ALU simulation waveform.

Project 2: The design of A Simple RISC Processor

In this project, students will design a simple RISC processor which is similar to MIPS but a simplified version.

➢ The Instruction Set

This microprocessor executes 16 different instructions. Each instruction is 21 bits long and consists of the following five fields: a 4-bit opcode, two 3 bit fields ***rs*** and ***rt*** that denote operand registers, one 3-bit field ***rd*** that denotes the destination register and an 8 bit immediate field. Note that although all of the instructions share the same layout, most instructions do not use all of the available fields. Figure 6 shows such an instruction set layout.

The instructions that the processor will execute are given below:

1. **add rs rt rd**: (opcode = 0) Add the contents of the two registers denoted by the rs and rt fields and store the result in the register indicated in the rd field.

2. **sub rs rt rd**: (opcode = 1) Subtract the contents of the register denoted by the rt field from the register denoted by the rs field and store the result in the register indicated in the rd field.

3. **and rs rt rd**: (opcode = 2) And the contents of the two registers denoted by the rs and rt fields and store the result in the register indicated in the rd field.

4. **or rs rt rd**: (opcode = 3) Or the contents of the two registers denoted by the rs and rt fields and store the result in the register indicated in the rd field.

5. **xor rs rt rd**: (opcode = 4) Xor the contents of the two registers denoted by the rs and rt fields and store the result in the register indicated in the rd field.

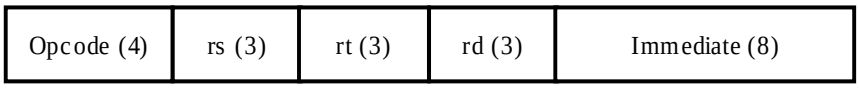

Figure 6. Instruction Set Layout.

6. **not rs rd**: (opcode = 5) Invert the contents of the register denoted by the rs field and store the result in the register indicated in the rd field.

7. **shl rs rd**: (opcode=6) Shift the contents of the register denoted by the rs field left one position (filling the vacated slot on the right with a zero) and store the result in the register indicated in the rd field.

8. **shr rs rd**: (opcode=7) Shift the contents of the register denoted by the rs field right one position (filling the vacated slot on the left with a zero) and store the result in the register indicated in the rd field.

9. **ld rd i**: (opcode=8) Load the register denoted by the rd field with the contents of the immediate field of the instruction.

10. **spc rd**: (opcode=9) Store the current contents of the 8 bit program counter in the register indicated in the rd field.

11. **beq rs rt i**: (opcode=10) If the contents of the two registers denoted by the rs and rt fields are equal then increment the program counter with the value in the immediate field.

12. **ji i**: (opcode=11) Reset the program counter to the value found in the immediate field.

13. **jr rs**: (opcode=12) Reset the program counter to the value stored in the register denoted by the rs field.

14. **ldsw rd**: (opcode = 13) Load the register denoted by the rd field with the current value on the 8 slide switches.

15. **ldbut rd**: (opcode = 14) Load the lower 4 bits of the register denoted by the rd field with the current values on the 4 push button switches.

16. **wleds rs**: (opcode = 15) Write the contents of the register denoted by the rs field into another 8-bit register connected to the 8 LEDs.

The instruction set 16 **wleds rs** interfaces the microprocessor with the FPGA board and it can write out the register value to the LED bar.

Figure 7 shows the logic diagram for this simplified microprocessor.

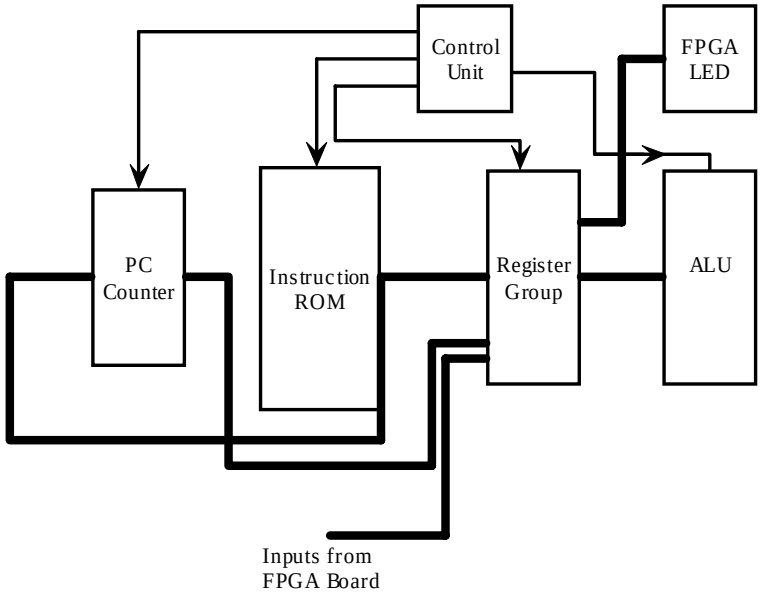

Figure 7. The Simplified Microprocessor Diagram.

A control unit generates all the control signals to PC register, ALU unit, register group, LED output, etc depending on the instruction from the ROM. In addition to the control signals which are internally generated by the control unit, two external signals must be provided: clock and reset. The clock signal must be fed to all registers as in every sequential circuit. The reset signal is used to ensure that on power up, the processor starts the execution of the program from some predefined address, which will be zero on this processor.

Students are required to implement, synthesize, simulate and verify this microprocessor on the FPGA board.

Figure 8 shows a screenshot of the simulation waveform of a sample instruction code for this microprocessor.

After this microprocessor VHDL module has been verified and tested on FPGA boards, students are required to implement a calculator using the assembly program for this processor. The program should work as follows. Whenever Push Button #1 on the FPGA board is pressed the circuit should load the value from the slide switches into register 1. Whenever Push button #2 is pressed the circuit should load the value from the slide switches into register 2 then add this value to the contents of register #1 and display the 8 bit result on the LEDs.

Project 3: The Modified RISC Processor

In this project, the design of the single cycle microprocessor in Project 2 is modified to add another instruction set and students are also needed to manually convert a simple C code into the assembly language of this processor.

➢ Project 3 Part 1: Microprocessor Instruction Modified

It is often useful to be able to increment a register value with an immediate value. The microprocessor from Project 2 is modified to add the following instruction replacing the sub instruction. **addi rs rd i**: (opcode = 1). Add the contents of the register denoted by the ***rs*** field to the contents of the immediate field and store the result in the register indicated in the ***rd*** field. Please explain how a 2's complement subtraction using the available instructions in the modified processor can be carried out. Illustrate the answer with one or more assembly code fragments.

➢ Project 3 Part 2: From C Code to Assembly Language

In this part, given the following piece of C-code act as a compiler and produce the corresponding assembly code for the processor. Use the new addi instruction at least once.

```
int i;
unsigned char l0;
while (1) {
l0 = 0;
for (i=0; i <= 3; i++) {
l0 = l0 >> 2;
/* Display l0 on the LEDs here */
}
for (i=0; i <=3; i++) {
l0 = l0  <<  2;
/* Display l0 on the LEDs here */
}
l0 = 0;
for (i=0; i <= 3; i++) {
l0 = l0 >> 1;
/* Display l0 on the LEDs here */
```

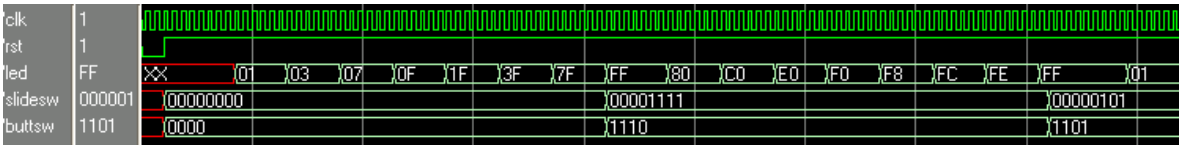

Figure 8. Simulation Waveform of the Microprocessor.

```
}
for (i=0; i <=3; i++) {
l0 = l0  <<  1;
/* Display l0 on the LEDs here */
}
}
```

Students should turn in their assembly code. The compiler theory can be further understood through this procedure.

Summary

The teaching experience of Computer Architecture through hands-on design practices at IPFW is presented in this paper. Due to the complexity of modern RISC microprocessor, the design of a full-scale unit is not feasible in the education activities, thus a simplified model which is similar to the one mentioned in the textbook is introduced. Students need to design, verify and test their design and write some simple application program using it through the learning procedure, thus it will help them to further understand the theories and principles of the computer organization. In the future teaching activities, a processor simulator combined the proposed design practices will be introduced to further facilitate the education process.

References

1. Barua, S. (2001). "An interactive multimedia system on computer architecture, organization, and design", *IEEE Transactions on Education*, 44**,** (1), 41-46.

2. Djordjevic, J., Nikolic, B., and Milenkovic, A.(2005). "Flexible web-based educational system for teaching computer architecture and organization," *IEEE Transactions on Education*, 48, (2), 264-273.

3. Gaeke, B. R. (2005). "VMIPS Project", http://www.dgate.org/vmips/

4. Ibbett, R., and Mallet, F. (2003). "Computer architecture simulation applets for use in teaching," *FIE2003, 33rd Annual Frontiers in Education*, 2**,** F1C-20-5.

5. Larus, J. (2005). "SPIM A MIPS32 Simulator," http://www.cs.wisc.edu/~larus/spim.html

6. Patterson, D. A., and Hennessy, J. L. (2005). "Computer Organization and Design: The Hardware/Software Interface," 3rd Edition, Morgan Kaufmann, San Francisco, CA.

7. Taylor, C. J., (2003), "CSE 371 Digital Systems Organization and Design," http://www.sea.upenn.edu/~cse371

8. Vollmar, K., and Sanderson, D.P. (2005). "A MIPS Assembly Language Simulator Designed in Education," *Consortium for Computing Sciences in Colleges: Midwest Conference.*

9. Yehezkel, C., Eliahu, M., and Ronen, M.(2003). "Learning computer organization and assembly language with the EasyCPU visual environment," *The 3rd IEEE International Conference on Advanced Learning Technologies*, 491-491.

Biographical Information

Dr. Guoping Wang is currently Assistant Professor in the Department of Engineering, Indiana University Purdue University Fort Wayne. He obtained his B.S., M.S., and Ph.D from Tsinghua University, Nanjing University and the University of Oklahoma respectively in 1988, 1991 and 2003. He teaches courses in digital system design, VLSI Design Lab, and computer architecture.