ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/228942202

A Simple Project Paradigm for Teaching Computer Architecture

Article · April 2006

CITATION
1

READS
8,336

1 author:

Yul Chu
University of Texas Rio Grande Valley
40 PUBLICATIONS **68** CITATIONS

SEE PROFILE

All content following this page was uploaded by Yul Chu on 08 January 2015.

The user has requested enhancement of the downloaded file.

A Simple Project Paradigm for Teaching Computer Architecture

Yul Chu
Electrical and Computer Engineering Department
Mississippi State University
chu@ece.msstate.edu

Abstract

This paper presents a teaching method for a possible computer architect by using a simple DCT project for an undergraduate-level computer architecture course. The goal of the project is to let students (two or three students per team) understand the concept of computer hardware and how to design a simple RISC-type 32-bit Instruction Set Architecture (ISA). The project consists of three different tasks: 1) D (Design) - Designing a processor at the abstract level; 2) C (Code) - Writing a simulation program for the ISA; and 3) T (Test) - Running a test program to verify each function of computer hardware. For the first task, students are required to design their own instruction sets, datapath, and control unit. For the second task, they write a simulation program by using a high-level language such as C/C++ or VHDL/Verilog based on the directions provided, and then they run a test program with the simulator to produce the results.

The project has worked well for students since they responded favorably to the project and indicated that they learned the concepts of computer hardware and how to design computer architecture as a professional engineer.

1. Introduction

The main job of a computer architect includes the logical design of computer hardware based on current technology and applications [1]. The logical design, in general, deals with designing the datapath, control unit, memory, and input/output at the abstract level instead of the circuit level [2][3]. Therefore, it is necessary to simulate the design with test programs (or benchmark programs) before chip fabrication to verify whether or not the designed architecture works properly. This design procedure is called DCT (Design, Code, and Test) in this paper. In addition, a computer architect should consider the performance and cost as major factors in determining the specifications for computer hardware [1-3].

Traditionally, simulation tools have been used for computer engineering courses such as computer architecture to let students understand basic operations easily [4][5]. However, some detailed simulators used to discourage students with many options for selection and lengthy lines of code [6]. For a computer architecture class, even if you have a simple simulator for easy understanding, you can just implement the fixed/limited operations repetitively without any trials to design new function logic. Thus, students might be discouraged from designing computer hardware because of this limitation in traditional simulators.

To implement a special function for any purpose, you need to define an instruction, design the datapath & control unit, write a simulation code for the instructions, and test it to check whether or not it works properly. We believe this works well for students to understand more easily and interactively. So *don't just try to use a simulator, but try to write a simple simulator for your clear understanding!*

This paper presents the DCT procedures (as in handling a short-term project instead of laboratory exercises [7]) in detail, as a computer architect would use in designing computer hardware such as a processor. This paper is set out as follows: Section 2 introduces the DCT procedures of a simple project for an undergraduate-level computer architecture class; section 3 discusses how to grade the project and provide for students' evaluation; and section 4 gives the conclusions.

2. DCT Procedures

```
v0 = 0;
v1 = 0;
v2 = 0;
a1 = 10;
While (a1 > 0) do
        {
        a1 = a1 –1;
        t0 = Mem[a0]
        a0 = a0 + 2
        if (t0 > 0) then {
                v0 = v0 * t0;
                Mem[a0-2] = v0; }
        Else {
                v1 = v1 + t0;
                v2 = v1 || t0; }
        }
Return
```

a) Pseudo code (test)

Mem[a0]	1
Mem[a0 + 2]	-1
Mem[a0 + 4]	2
Mem[a0 + 6]	-2
Mem[a0 + 8]	3
Mem[a0 + 10]	-3
Mem[a0 + 12]	4
Mem[a0 + 14]	-4
Mem[a0 + 16]	5
Mem[a0 + 18]	-5

b) Initial Memory data (a0)

Figure 1. Sample pseudo code for the test program and initial memory data

As we discussed in section 1, DCT stands for Design, Code, and Test. This section shows each DCT procedure in detail by using a simple computer architecture project as an

example. The project is to design a 32-bit RISC Instruction Set Architecture (ISA) through MultiCycle Implementation (MCI). MCI means that it takes multi cycles, which are different from instruction to instruction, to execute an instruction [1]. Figure 1 shows a sample pseudo code for the test program, which has Arithmetic and Logic, Data Transfer, and Control functions.

To run the test program, students should complete D (Design) and C (Code) procedures and convert the test program to their own instructions for T (Test) procedure.

2.1 D (Design) Procedure

There are three steps to design an ISA, which is an interface between high-level (system software such as operating system or compiler) and low-level (gate or circuit-level). Those are: 1) Design instruction sets; 2) Design datapath components with clock methodology and datapath; and 3) Design control signals and control unit. Since there are many factors to determine in designing ISA, it would be recommended for students as a team (2 to 3 students per team) to discuss the three steps in detail. Through the discussion, we expect students could build strong and clear concepts for ISA operations.

2.1.1 Design instruction sets. For the first step, each team needs to design instructions to execute a program in an efficient way. For example, MIPS architecture (32-bit) has 3 different types of instruction formats: 1) R (Register) Format for most arithmetic and logical operations; and 2) I (Immediate) Format for immediate addressing modes and memory access operations (data transfer such as load and store); and 3) J (Jump) Format for jump instruction. Figure 2 shows the three instruction formats for MIPS architecture [1].

R (Register) Format:

Opcode (6)	Rs (5)	Rt (5)	Rd (5)	Shamt (5)	Funct (6)

Most arithmetic and logic instructions (except 'immediate')

I (Immediate) Format:

Opcode (6)	Rs (5)	Rt (5)	16-bit Immediate value (16)

Data Transfer, Immediate, and Cond. Branch instructions

J (Jump) Format:

Opcode (6)	26-bit word address (26)

Unconditional Jump instructions

Figure 2. Instruction formats for MIPS architecture [1]

In Figure 2, for the 32-bit instruction format, the high-order 6 bits are used for defining operations, which is called 'opcode'. The opcode will be transferred to the control unit after fetching an instruction from memory. Three types of register are defined in Figure 2 such as 'Rs, Rt, and Rd'. For R-Format, Rs and Rt are used for source registers to compute and Rd is for destination register to store in the Register File (one for datapath component, refer to section 2.2.2). For I-Format, Rs and Immediate field (low-order 16 bits) are used for two inputs of ALU (Arithmetic Logic Unit, refer to section 2.2.2) and Rt would be used for the destination register to save the output (data) from ALU or memory unit. For J-Format, a 26-bit word address is used to compute an unconditional target address for jump instruction. After reviewing the MIPS architecture, each team could design any kind of instructions to execute the test program for its own purpose.

2.1.2 Design Datapath. After designing instructions, students need to design the datapath component to implement its instructions.

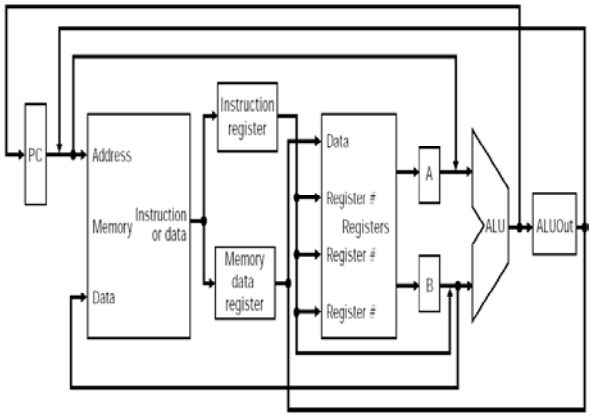


Figure 3. Assemble the datapath [1].

For example, an 'add' instruction (R Format in Figure 2) of MIPS architecture is required to have several datapath components to execute: 1) *PC* (Program Counter) to access Memory to fetch an instruction; and 2) *Memory* to fetch an 'add' instruction; 3) *Register File* to read data from source registers according to the fetched instruction and to store the data into the destination register; and 4) *ALU* (Arithmetic and Logic Unit) to add two register contents (Rs and Rt in Figure 2). In addition, datapath components could include MUX (multiplexer), Adder (kind of ALU), and Sign-Extension Unit for immediate value, etc. to implement instructions for any purpose. In this way, students could design all the datapath components for various instructions such as data transfer (load and store) instructions, control (branch) instructions, etc.

The next step is to assemble the datapath components for various instructions. Figure 3 shows an example of how to assemble the datapath for 32-bit MIPS architecture [1]. In MIPS, there are a maximum of 5 stages to execute an instruction: Fetch, Decode, Execute, Memory Access, and Write Back. The datapath can be assembled according to

those stages and instruction types. There are 4 types of datapath: 1) Instruction Fetch – common for all instructions; 2) Arithmetic and Logical Computation – add, subtract, etc.; 3) Memory Reference – load and store; 4) Control – branch and jump.

The execution procedures are: 1) Fetch instruction from Memory (datapath components: PC, Memory, Adder, and Mux); 2) Decode instructions and read operands (Register File, Sign-extension unit, and Mux); 3) Execute arithmetic and logical computation for output data, condition check, or memory address (ALU, Adder, and Mux); 4) Memory access to read/store data from/into memory (Memory and Mux); and 5) Write back data into Register File to update (Register File, Mux). In this way, students could assemble the datapath for their instructions to work properly.

2.1.3 Design Control Unit. After completing the datapath for all instructions, it is necessary to define control signals to execute each instruction independently since most datapath components are shared for all instructions. In Figure 3, for load instructions, the ALU can compute the memory address with two inputs according to the *ALUOp* control signal, which defines the 'add' operation. After computing the memory address, the data in Memory is read according to the *MemRead* control signal to fetch data from the memory. The fetched data from Memory should be written to Register File according to the *RegWrite* control signal. Therefore, there should be at least three control signals to implement load instructions. In this way, students could define control signals for each instruction.

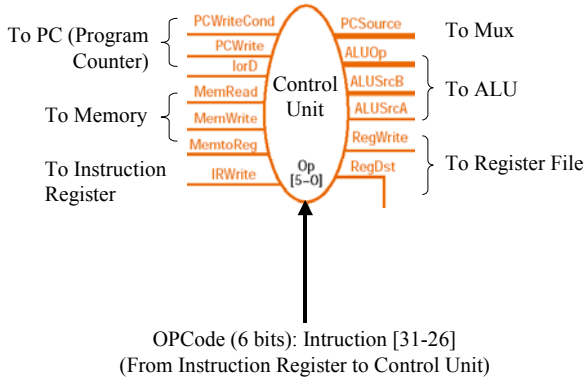


Figure 4. A sample MCI Control Unit

Control signals are issued from the Control Unit during the 'Decode stage' according to the opcode transferred from a fetched instruction. After defining control signals for all instructions, students need to assemble control signals by designing a Control Unit with input (opcode) and outputs (control signals). Figure 4 shows the MIPS MCI Control Unit, which has 13 control signals as an example in [1]. Once this is done by building the data path with control signals, the next step is to build the Finite State Machine (FSM) to implement instructions through MultiCycle Implementation. The sample FSM is shown in Figure 5.

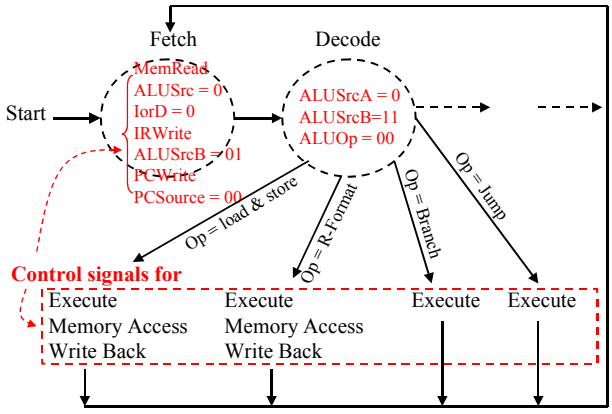


Figure 5. A sample FSM for MCI Implementation

2.2 C (Code) Procedure

```
entity cpu_datapath is
port(clk,reset,zflag,stop : in std_logic;
     drdata,daluout: in std_logic_vector(15 downto 0);
     dr1addr,dr2addr,dwaddr: out std_logic_vector(2 downto 0);
     dmem_address : out std_logic_vector(7 downto 0);
     dwdata,alua,alub,pc_in: out std_logic_vector(15 downto 0);
     dr1data,dr2data,pc_out: in std_logic_vector(15 downto 0);
     fun : out std_logic_vector(2 downto 0);
        pcwritecond,pcwrite,iord    : in std_logic;
        memtoreg: in std_logic;
        irwrite,alusrca,regdst : in std_logic;
        pcsource,alusrcb           : in std_logic_vector(1 downto 0);
         addr : in std_logic_vector(7 downto 0);
        tdata : in std_logic_vector(15 downto 0);
        dtdata : out std_logic_vector(15 downto 0);
        pc_load : out std_logic);
end cpu_datapath;
architecture ab of cpu_datapath is
-- pc
signal       PCLoad : std_logic;
signal       pcout : std_logic_vector (15 downto 0);
signal       pcin  : std_logic_vector (15 downto 0);
-- imem
signal      mem_address:     std_logic_vector(7 downto 0);
signal      instruction_reg: std_logic_vector(15 downto 0);
signal      addr1:     std_logic_vector(7 downto 0);
. . . . . .
```

PC

Instruction Memory

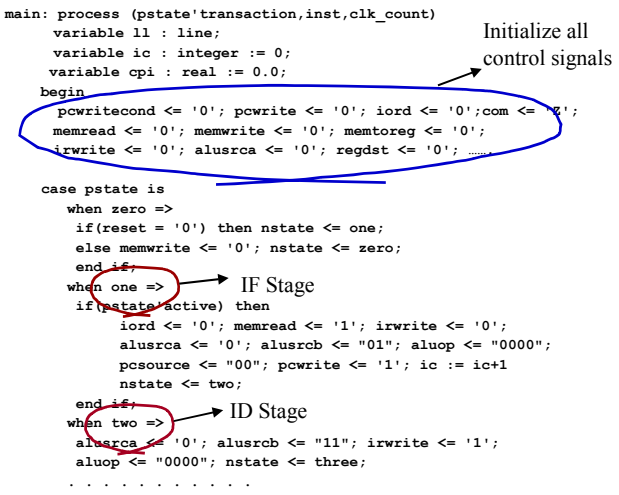
Figure 6. A sample VHDL coding for datapath, control unit.

Since the computer architecture class is an intermediate undergraduate course in most computer engineering schools, programming languages such as C/C++ or VHDL/Verilog would be prerequisites for computer architecture in general [4].

In section 2.1, students could design instructions, datapath, and the control unit for their own purposes. The next step is to write the code for the datapath components and control unit. For example, students would write a code for Register File (consisting of 32 registers) and control signals to update it. Figure 6 shows a sample VHDL datapath coding for cpu_datapath, PC, and instruction memory (imem).

After coding the datapath and control unit, students could write a code to implement each instruction as a Finite

State Machine (FSM). Figure 7 shows a code for control signals, Instruction Fetch (IF) stage, and Instruction Decode (ID) stage for the FSM in Figure 5.




```
main: process (pstate'transaction,inst,clk_count)
      variable ll : line;
      variable ic : integer := 0;
      variable cpi : real := 0.0;
    begin
      pcwritecond <= '0'; pcwrite <= '0'; iord <= '0';com <= 'Z';
      memread <= '0'; memwrite <= '0'; memtoreg <= '0';
      irwrite <= '0'; alusrca <= '0'; regdst <= '0'; ……

    case pstate is
       when zero =>
        if(reset = '0') then nstate <= one;
        else memwrite <= '0'; nstate <= zero;
        end if;
       when one =>
        if(pstate'active) then
             iord <= '0'; memread <= '1'; irwrite <= '0';
             alusrca <= '0'; alusrcb <= "01"; aluop <= "0000";
             pcsource <= "00"; pcwrite <= '1'; ic := ic+1
             nstate <= two;
        end if;
       when two =>
        alusrca <= '0'; alusrcb <= "11"; irwrite <= '1';
        aluop <= "0000"; nstate <= three;
       . . . . . .
```

Figure 7. A sample VHDL coding for FSM in Figure 5.

2.3 T (Test) Procedure

After students complete the Design and Code procedure, they need to test their architecture with the test program. The input for the simulator would be a sequence of designed instructions converted from the test program. After the instructions (machine code) are placed into Memory, all instructions would be fetched from Memory according to PC and be executed in the simulator. The output of the simulator would be placed in the Register file or Memory for the test program. So, students need to print the contents of the Register file and Memory to verify whether the simulator works properly or not. The steps for T procedure are: 1) Input operations – Clear contents of Memory and Register File, and place the instructions (machine code) into Memory and initialize PC; 2) Execution of instructions – Print initial contents of Memory and Register File, and Execute the instruction by feeding it into the FSM; and 3) Output operation – Print final contents of Memory & Register File after executing the instructions.

After completing the DST procedures, students are required to prepare and submit a final project report. The final report would include the following:

- An explanation of the architecture (datapath and control unit);
- A discussion of how to test the architecture;
- A discussion of errors in the architecture;
- A discussion of how to optimize the errors;
- Simulation results.

3. Grading Projects and students' Evaluation

The grading for the DCT project is mainly based on the work for the three procedures (DCT). For D (Design) procedure, we need to check the efficiency of the designed ISA and the usage of clocking methodology (rising edge trigger or falling edge trigger). For C (Code) procedure, the major point is to check whether each instruction works properly or not. For T (Test) procedure, the whole test procedure would be checked with the results. In addition to the DCT grading, we need to check the discussion among team members since the goal of the project is to share ideas and get a clear concept through discussion.

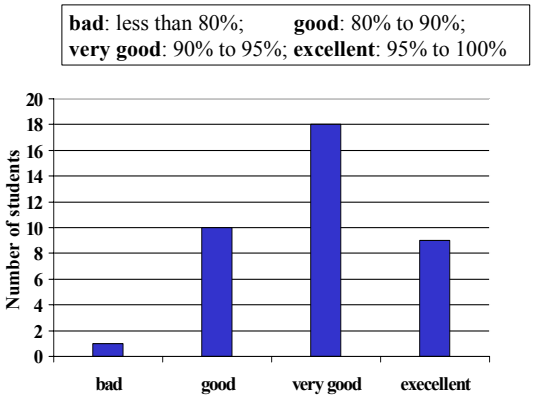


Figure 8. Grading for DCT project (Fall semester, 2004)

As a case study, Figure 8 shows the project grading for a DCT project (Fall semester, 2004 at Mississippi State University). There were 14 teams (2 to 3 students per team). Their final reports for the DCT project were graded based on efficiency (20%), clocking methodology (10%), correctness (30%), testing (15%), results (15%), and discussion (10%).

Figure 8 shows that most students (71.0%) got grade A (more than 90%) and other students (26.3%) had grade B (between 80% and 90%) except 1 student (2.6%). Therefore, we could say the DCT project was successful to let students understand the concepts and design process for the 32-bit ISA as a computer architect.

Figure 9 shows the evaluation from computer architecture (ECE4713) students in Fall 2004 at Mississippi State University. In Figure 9, most of the students (84.2%) responded that they learned a great deal about fundamental-level datapath design and concepts since the DCT project is easy to follow and a good experience for getting a grasp on a professional career. Especially, they mentioned that the project was closely related to the class work. However, there were 5 students (13.1%) who did not agree like the project since they wanted more detailed guidelines regarding how to write a simulator, and some students wanted to use a FPGA hardware design instead of a software simulation program. Therefore, we believe that it is feasible to expand the DCT project from hardware to a

software simulator if students take the FPGA class as a prerequisite for the computer architecture class.

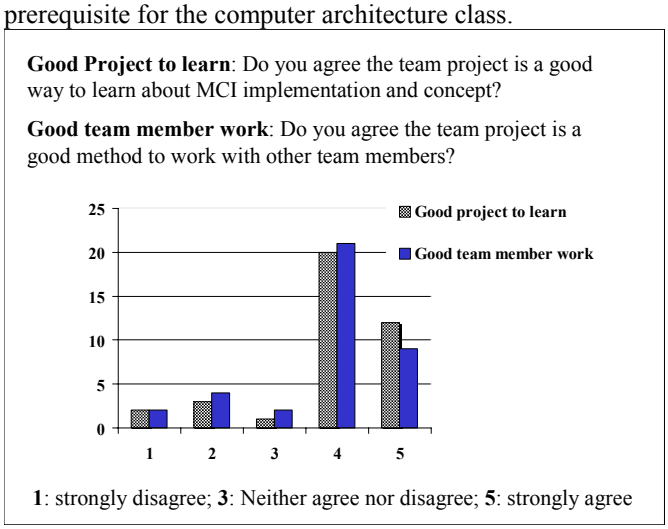


Figure 9. Student Evaluation for the project (38 students and Fall semester, 2004).

Figure 9 also shows that 30 students (78.9%) responded that the team project (2 to 3 students per team) would be a good method to work with other members since they could share ideas and re-establish the concept clearly by discussing the DCT procedures step by step. However, 6 students (15.7%) responded that they did not agree since some team members did not attend team meetings at all during the project, and there was some difficulty in finding good team members in a short period of time. Therefore, it would be necessary for an instructor to help students who cannot find a team by using communication tools such as class email or bulletin board. In addition, it is required to let students write a contribution report for their work to differentiate some students who do not attend the project actively.

The project term in Figure 9 was from Nov. 11, 2004 to Dec. 3, 2004 and there was a Thanksgiving break (from Nov. 23 to Nov. 28, 2004) for one week. Because of the break, most students were short of time to finish the project on time. Therefore, it would be better to start the project one week earlier than Nov. 11, 2004. Another valuable comment from students was that they wanted to have feedback regarding their project results. So it would be a good idea to open their project grading with comments before the final exam as well.

4. Conclusion

There have been so many software tools developed to teach computer engineering courses such as computer architecture. Traditionally, those tools have many options to choose from for proper operations and consist of a lengthy line of code to figure out. Therefore, it is possible for students to figure out the options first and then to learn the operations through the tools. In addition, since the tools used to have limited functions to operate, it is difficult to design a different type of instruction with the tools. Therefore, those tools can be used to let students understand the limited operations instead of creative design since they lack experience of the designing process.

This paper introduces DCT procedures to accommodate students to design an ISA with their own ideas by: 1) Designing instructions, datapath, and control unit; 2) Coding the simulator for the architecture from step 1); and 3) Testing the architecture through the simulator with a test program.

According to the grading project and student evaluation from Mississippi State University, we found that the DCT procedures worked successfully for the undergraduate level computer architecture class since most students (97.3%) who participated in the DCT project had As and Bs for their grades and 78.9% of the students evaluated the project as favorable (agree and strongly agree) since they could learn fundamental concepts and the design process clearly and gain confidence in the area of computer architecture.

5. References

[1] David A. Patterson & John L. Hennessy, Computer organization and design: the hardware/software interface, *second edition, Morgan-Kaufmann,* San Francisco, California, 1998.

[2] Vincent P. Heuring and Harry F. Jordan, Computer Systems Design and Architecture, *second edition*, Prentice Hall, Upper saddle river, New Jersey, 2004.

[3] John L. Hennessy & David A. Patterson, Computer Architecture: A Quantitative Approach, *third edition*, *Morgan-Kaufmann,* San Francisco, California, 2003.

[4] Lillian Cassel et al, Distributed Expertise for Teaching Computer Organization & Architecture, *Working Group Reports in the 5th Annual Conference on Innovation and Technology in Computer Science Education*, Helsinki, Finland, July 2000.

[5] D. Ellard, D. Holland, N. Murphy, and M. Seltzer, On the Design of a New CPU Architecture for Pedagogical Purposes, *in Proc. WCAE 02 – workshop on Computer Architecture Education, on 29th International Symposium on Computer Architecture*, Anchorage, AK (USA), 2002, pp.28-34.

[6] Christopher T. Weaver, Eric Larson, and Todd Austin, Effective Support of Simulation in Computer Architecture Instruction, *Workshop on Computer Architecture Education (WCAE02) held in conjunction with the 29th International Symposium on Computer Architecture*, Anchorage, AK, May 2002.

[7] Daniel C. Hyde, Teaching Design In a Computer Architecture Course, *IEEE Micro, Volume 20, Number 3*, May/June 2000, pp23-28.

View publication stats