

COMPUTER ORGANIZATION: HARDWARE DESCRIPTION LANGUAGE APPROACH

AWAD, Abdelkarim

*Birzeit University, Faculty of Information Technology
Department of Computer Systems Engineering, Palestine
akarim@birzeit.edu*

GHANEM, Wasel

*Birzeit University, Faculty of Information Technology
Electrical and Computer Systems Engineering, Palestine
ghanem@birzeit.edu*

Abstract

Teaching of computer organization and architecture concepts is not an easy task for instructors. Usually, Students face difficulties in grasping many (even simple) ideas. In this paper we will present a stereotype of using Computer Aided Design CAD tools, namely, Altera's Quartus II simulator, in teaching Computer Organization concepts. We will build a simple computer using verilog HDL language. We will elaborate on the Instruction Set Architecture (ISA) concepts. In our approach we will provide students with a running simple computer, which will make the simulation possible and easy. This way, they can trace the execution of the simple computer that has few instructions stored in the memory. After that students can route the program on a FPGA which is important because it will teach them how to build their own processor. At the end, students can expand the computer in several dimensions (e.g. more instructions, more addressing modes, cache, speculative execution, little or big Endian,...). We present at the end of this paper a typical exercise that can help students to improve their knowledge about the computer organization. As a future work we introduced Moodle as a good candidate to be used in the e-learning process.

Keywords: *Simple computer, Instruction Set Architecture (ISA), Verilog HDL*
ACM classification: K.3.1, C.1.1

1. Introduction

Learning an Idea is the core to understand it, nevertheless. It is known that practicing something has high impact on the level of understanding this thing. Instructors in computer organization course put huge effort in order to enable students to understand concepts like fetch-decode-execute cycle. If we make it possible for students to build their own simple computer in an easy way then it will be easy for them to understand the concepts of computer organization. Moreover they can start extending their computer and discover many important issues in designing computers (i.e backward compatibility).

The main topics that students should learn in a computer architecture and organization job include the logical design of computer hardware based on current technology and applications(Stallings, 2009).The logical design, in general, deals with designing the data path, control unit, memory, and input/output at the abstract

level instead of the circuit level (Heuring, 2004; Patterson and Hennessy, 2005). Therefore, it is necessary that students simulate the design with test programs (or benchmark programs) before chip fabrication to verify whether or not the designed architecture works properly. In addition, students should consider the performance and cost as major factors in determining the specifications for computer hardware (Hennessy and Patterson, 2003; Heuring, 2004; Patterson, 2005; Stallings 2009)

Using simulation is common in education and industrial sectors. Using very simple simulator can be very abstracted and hides important details. On the other side using a complicated simulator (especially at the beginning) can be very daunting for the students. So it is important to have a flexible simulation environment. Students at the beginning need to see a running example that has enough details. In our approach we give students a running example with few instructions then they start to expand it and test it after each step.

There are several works have been done in this field (Cassel, 2000; Ellard, 2002, Chu 2005) form building a very simple computer using breadboard and chips (Pilgrim, 1993), to use a complicated simulator (Weaver et al, 2002). The work in this paper was influenced by (Daniel C. Hyde, 1998). Nevertheless our work is different in many ways, for example we build our simple computer using not only using simulation but also on real FPGA. (Daniel C. Hyde, 1998) have used delay instead of clock which makes it difficult to translate it to a real implementation. In our approach we build the states of the instruction execution. The main problem in using chips and breadboard to build a simple computer is the limitation on expanding the project. On the other hand using a sophisticated tool doesn't suit the undergraduate students. Therefore we will try to build a simple computer that can start very simple and ends as a complicated one. For this goal we will use a CAD tool from Altera called Quartus II. This tool can be used as a simulator as well as it can be used to route the computer on a Field Programmable Gate Array FPGA or Complex Programmable Logic Device CPLD.

We will use verilog HDL language in the implementation and we assume that students had learned it in a prerequisite course (i.e. digital design). If students did not already learnt, the instructor can ask them to learn its basics which should not be a difficult one.

The rest of the paper is organized as follows. The second section shows the implementation of simple computer. Afterwards the simulation results are shown in section 3. Finally Section 4 concludes the paper.

2. General remarks

In this section we will show how to implement a running simple computer. We will use von Neumann model, shown in Figure 1, which is illustrated in (Stallings, 2009), we will call this computer **simcomp1**. Register Transfer Language (RTL) will be used to illustrate the implementation.

Simcomp1 is a single accumulator machine; it has a two byte-addressable memory with size of 128byte. The memory is synchronous to the CPU, and the CPU can read or write a word in single clock period. The memory can only be accessed through the memory address register (MAR) and the memory buffer register (MBR): `MBR <= Memory [MAR];` (to read), and `Memory [MA] <= MBR;` (to write).

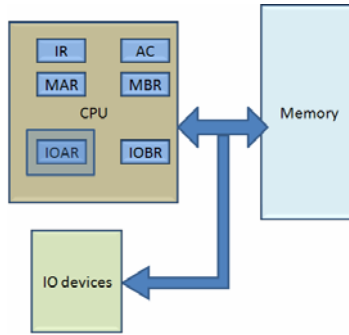


Figure 1. *Simple Computer Components*

Simcomp1 has also one input (INPORT) and one output (OUTPORT) devices we can access these devices using only Input-Output Buffer Register IOBR. (Notice that here we do not need Input-Output Address Register IOAR because we have only one input and one output). The size of INPORT, OUTPORT and IOBR is one byte. The CPU has also an accumulator (AC), a program counter (PC) and an instruction register (IR). The size of these registers is 16 bit.

Table 1: *Instruction set of simcomp1*

| opcode | Instruction | Meaning |
|--------|-------------------|---|
| 0011 | LOAD M(X) | loads the contents of memory location <i>X</i> into the AC. |
| 1011 | STORE M(X) | stores the contents of the AC in memory location <i>X</i> . |
| 0111 | ADD M(X) | adds the contents of memory location <i>X</i> to AC. |
| 1100 | IN | reads the content of <i>INPORT</i> into the AC. |
| 1101 | OUT | writes the content of AC in <i>OUTPORT</i> . |

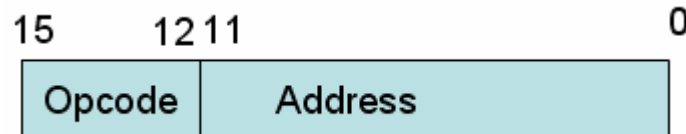


Figure 2. *Instruction Format*

2.1. *Instruction Set*

Simcomp1 has only five instructions-- Load, Store, Add, IN and OUT. The size of all instructions is 16 bits; the instruction format is shown in the Figure 2. The operation code (opcode) is 4 bits and the remaining 12 bits are used for the operand. All the instructions are single address instructions and access a word in memory. The opcodes are listed in Table 1.

A simple task for students is to extend the instructions, for example adding a branch instruction (e.g. Jump) or logical instructions (e.g. AND). Other important things that can be discussed with students include:

- Instruction format
- Number of opcodes,
- Number and type of operands,
- Addressing modes.

2.2. Verilog Code

In this section, the verilog code will be presented, at the beginning any verilog file should has a name the same as the module as shown in Figure 3-A line (1) then we define the inputs and outputs as shown in lines 2-12. We used clock to execute each phase of the instruction cycle. The states represent the phases of instruction cycle.

```
1 module simcomp1(clock, PC, IR, MBR, AC, MAR, INPORT, OUTPORT);
2     input clock;
3     output PC, IR, MBR, AC, MAR;
4     reg [15:0] IR, MBR, AC;
5     reg [11:0] PC, MAR;
6     reg [15:0] Memory [0:63];
7     reg [2:0] state;
8     input [7:0] INPORT;
9     output [7:0] OUTPORT;
10    reg [7:0] OUTPORT;
11    reg [7:0] IOBR;
12    parameter load = 4'b0011, store = 4'b1011, add=4'b0111, in=4'b1100, out=4'b1101;
```

Figure 3-A. Part1: parameters of the simple computer program

```
13 initial begin
14     // program stored in the memory
15     Memory [10] = 16'h3020; //LOAD M(20H)
16     Memory [11] = 16'h7021; //ADD M(21H)
17     Memory [12] = 16'hB030; //STORE M(14H)
18     // data stored in the memory
19     Memory [32] = 16'd6; //data stored at location 20H
20     Memory [33] = 16'd5; //data stored at location 21H
21     //set the program counter to the start of the program
22     PC = 10; state = 0;
23     end
```

Figure 3-B. Part2: parameters of the simple computer program

The initialization part (lines 13-23) contains the program that will be executed. Moreover it contains the data initially stored in the memory which will be used by the program. To execute the program, we must point the Program Counter (PC) to first instruction of the program. The first instruction is stored in location 10. The machine code for this instruction is 3020H, we commented the corresponding assembly code in the same line (LOAD M(20H)). Notice that some numbers are written in decimal and others are written in hexadecimal (20H=32).

The code of the instruction cycle is shown the Figure 3-C which illustrates the fetch-decode-execute phases. As it can be seen in the code, the memory can be accessed through MBR and MAR registers only. Furthermore the input out devices can be accessed only through IOBR. In this implementation we did not use IOAR, needless to say that students can be asked to add more input and output devices, this way they have to use IOAR.

3. Simulation

In Quartus II, there are two types of simulation: Timing and Functional. If the Timing simulation is selected, the simulation will be based on timing analysis of the selected chips, on the other side functional simulation will display only the functionality of the system without taking care of the timing. Figure 4 shows a

timing simulation of the program listed in Figure 3. The only input that is needed here is the clock which triggers the execution of the program. The Timing simulation mode allows students to see the propagation delay along various paths.

```

24 always @(posedge clock) begin
25 case (state)
26 0: begin
27     MAR <= PC;
28     state<=1;
29 end
30 1: begin // fetch the instruction from memory
31     MBR <= Memory[MAR];
32     PC <= PC + 1;
33     state<=2; //next state
34 end
35 2: begin //transfer the instruction from MBR to IR
36     IR <= MBR;
37     state<= 3;
38 end
39 3: begin //Instruction decode
40     if ((IR[15:12]==in)|| (IR[15:12]==out))
41         state<=4;
42     else
43     begin
44         MAR <= IR[11:0];
45         state<= 4;
46     end
47 end
48 4: begin // Operand fetch
49     state <=5;
50     case (IR[15:12])
51         load : MBR <= Memory[MAR];
52         add  : MBR <= Memory[MAR];
53         store: MBR<=AC;
54         in  : IOBR<=INPORT;
55         out : IOBR<=AC[7:0];
56     endcase
57 end
58 5: begin //execute
59     if (IR[15:12]==add) begin
60         AC<= AC+MBR;
61         state<=0; // next state
62     end
63     else if (IR[15:12] == load) begin
64         AC <= MBR;
65         state <=0;
66     end
67     else if (IR[15:12] == store) begin
68         Memory[MAR] <= MBR;
69         state <= 0;
70     end
71     else if (IR[15:12] == in) begin
72         AC[7:0] <= IOBR;
73         state <= 0;
74     end
75     else if (IR[15:12] == out) begin
76         OUTPORT <= IOBR;
77         state <= 0;
78     end
79 end
80 endcase
81 end
82 endmodule
83

```

Figure 3-C. Part3: Code of instruction cycle of the simple computer

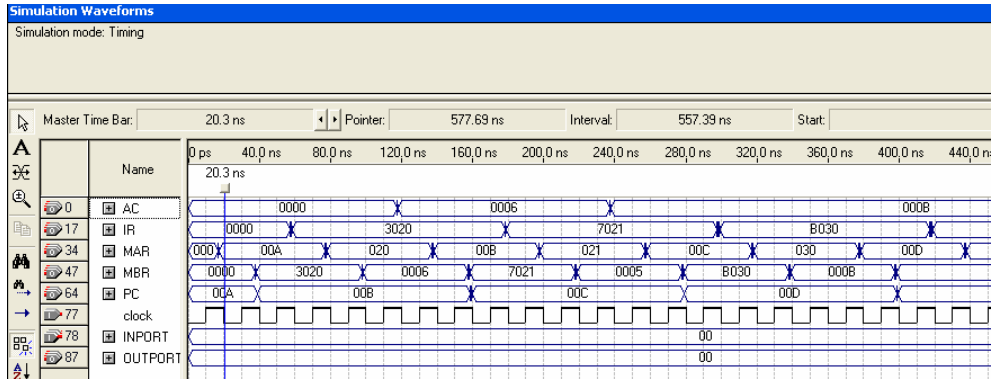


Figure 4. Simulation Results of the program stored in the memory in the Figure 3 (Timing simulation)

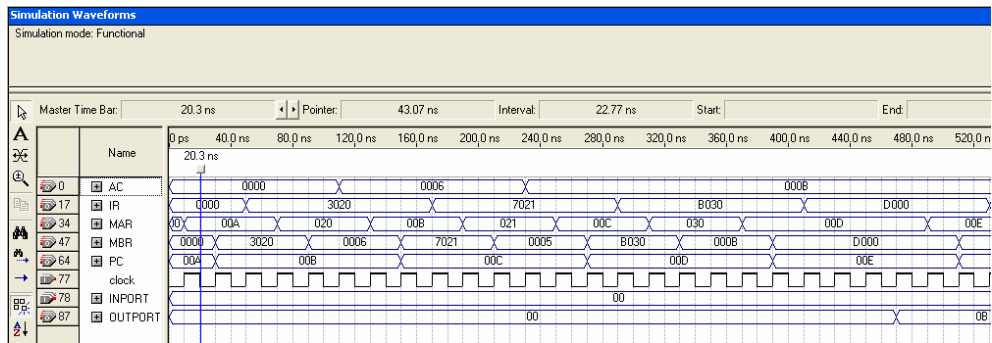


Figure 5. Functional simulation results of Program stored in the memory in the Figure 3 plus addition instruction to write in Output device

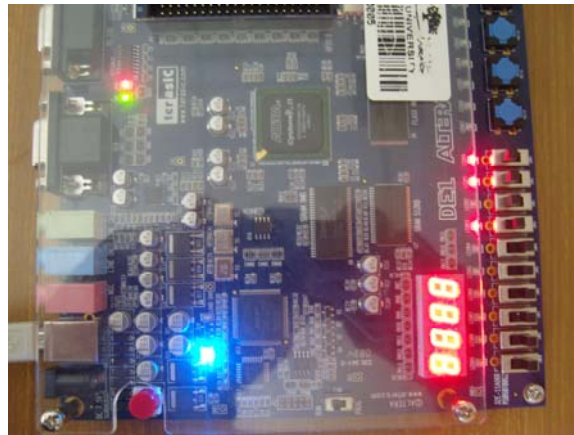


Figure 6: Altera's DE1 Kit, simcom1 is routed on the FPGA. The Leds points to the output (0BH).

In figure 5 we show Functional simulation of the same program with extra Instruction in which the results will be stored in the output device. The extra instruction stored in address 13 is:

```
Memory [13] = 16'hD000; //OUT
```

In the last portion of the execution, the result (0BH) is stored in the output device (OUTPORT). The students can see the difference between the timing and functional simulation modes.

In order to give the student the feeling that he was productive, we can route the program on an FPGA. For this purpose we used Altera's DE1 (Figure 6) kit for the demonstration. The student has to determine the FPGA chip installed in the board which is EP2C20F484C7N then assign the pins of OUTPORT to LEDs. Figure 6 illustrate the execution of the program which sends a 0Bhex to the port (LEDS).

4. Simple computer 2

SIMCOMP1 is a single accumulator machine. Based on SIMCOMP1 we asked the students to implement **SIMCOMP2** such that, it has two 16 bit general purpose registers R0 and R1 in addition to PC MAR MBR IR, IOAR, IOBR. This computer should execute the following instructions (shown in Table 2), the instruction format is shown Figure 7.

Table 2. *SIMCOMP2 Instruction Set*

| Opcode | Instruction | Description |
|--------|----------------|--|
| 0110 | ADD Reg, M(x) | Reg=Reg+M(x) |
| 0100 | AND Reg, M(X) | Reg=Reg&M(X)(bitwise) |
| 0001 | LOAD Reg, M(X) | Reg=M(X) |
| 0010 | STORE Reg,M(X) | M(X)=Reg |
| 0101 | IN | Reads the content of INPORT into the R0 (lower byte). |
| 0011 | OUT | writes the content of the lower byte of R0 in OUTPORT. |
| 1011 | JMP X | Jump to location x |
| 1101 | Loop X | Decrement R1 and jump to location x if the value of x not zero |
| 1110 | ADD Reg, X | Reg=Reg+X |
| 1100 | AND Reg, X | Reg=Reg and X (bitwise) |
| 1001 | LOAD Reg, X | Reg=X |
| 1010 | ADD Reg1,Reg2 | Reg1=Reg1+Reg2 |

| | | |
|---------------|------------|--------------------------|
| Opcode (4bit) | Reg(1 bit) | Address (11 bit)/IMM/REG |
|---------------|------------|--------------------------|

Figure 7: *Instruction format for simple computer 2*

Students have to modify the verilog code of SIMCOMP1 and simulate the new code and show the results of simulation and justify the results based on the assembly code used in the simulation. We suggest also that each student to submit a report that includes (or Students can work in groups with maximum 2 students per group).

1. Verilog code
2. Provide Simulation in Quartus II.
3. A text that describes the code and the simulation results.

In this way it is easy to get feedback if students understand what they are doing or do not.

To test SIMCOMP2 design, students have to execute the following program where PC starts at 8.

```
1. JMP 10
2. loadR1,20
3. Load R0,6
4. Load R1,3
5. StoreR0,M(100)
6. AddR0,5
7. loop 12
8. out
```

5. Future work-Integrating with Moodle

Moodle is a powerful and flexible open source content management system for managing, presenting and distributing course materials; moreover, its modules help support independent learning by allowing students to access course materials “on demand,” thus encouraging reflective and recursive thinking to occur. In addition, many of Moodle’s modules help support the social mediation of learning. Moodle promotes social constructionist pedagogy to address collaborative and social dimensions of learning. To this end it supports many interactivities such as assignments, chats, choices, forums, glossaries, lessons, quizzes, resources, individual learning journals, surveys, wikis, and workshops. Moodle also supports the inclusion of SCORM® learning objects which offer not only interoperability of content with other systems but also the opportunity to extend the range of learning experiences while remaining within the Moodle tracking and monitoring framework.

The system also includes many possibilities of assessment and testing – like various versions of quizzes and assignments. With respect to quizzes, the platform enables to prepare questions from several categories and choose either specific questions for the test or choose questions randomly, including both questions and answers shuffling. The types of questions include such elementary choices as multiple-choice, true/false and numerical, but also calculated type, description, essay and short answer, matching type and random short-answer matching. The assessment and grading is also very flexible.

Similarly there exist nice options in the Moodle software for working with surveys and questionnaires, which is an important part of the education process at universities. Usually the teacher is interested in the perception of his lecture and opinions concerning the content of both classroom and laboratory exercises. Good questionnaire enables the teacher to get much information concerning the effectiveness of teaching and possibly improve some parts of the course. On the other hand the faculty and university authorities are frequently interested in students’ opinions with respect to specific courses and other teaching process components and dedicated questionnaire system can be very helpful for this.

6. Conclusions

In this paper we have introduced a simple computer design that aid students to streamline the computer Organization and Architecture course by using HDL and simulation to implement a working computer. Giving the students a working model makes it simple to lunch simulation and then perform different

enhancements. The simple computer gives students a clearer understanding of the basics of Instruction Set architecture by offering them a point design from which to compare and contrast other approaches. The Instructor can ask students to extend the simple computer in several ways. For example students can add more instructions, use different addressing modes, implement more registers. Furthermore students can add more hardware like cache or improve the performance by using pipelining techniques or speculative execution.

References

1. Cassel L., 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, 2000.
2. Chu, Y., *A Simple Project for Teaching Instruction Set Architecture*. “ICALT 2005”, 69-71, 2005.
3. Ellard, D., Holland, D., Murphy, N., Seltzer, M., *On the Design of a New CPU Architecture for Pedagogical Purposes*, “Proc. WCAE 02 – workshop on Computer Architecture Education, on 29th International Symposium on Computer Architecture”, Anchorage, AK (USA), 28-34, 2002.
4. Hennessy, J.L., David A. Patterson, D.A., *Computer Architecture: A Quantitative Approach*, 3rd ed., “Morgan-Kaufmann”, San Francisco, California, 2003.
5. Heuring V.P., Harry F. Jordan, H.F., *Computer Systems Design and Architecture*, 2nd ed., “Prentice Hall”, Upper saddle river, New Jersey, 2004.
6. Hyde D. C., *Using Verilog HDL to Teach Computer Architecture Concepts*, “Proceedings of the 1998 workshop on Computer architecture education”, 10, 1998.
7. Patterson, D.A., John L. Hennessy, J.L., *Computer Organization and Design: the Hardware/Software Interface*, 3rd ed., “Morgan-Kaufmann”, San Francisco, California, 2005.
8. Pilgrim R.A., *Design and Construction of the Very Simple Computer (VSC): a Laboratory Project for an Undergraduate Computer Architecture Course*, “ACM SIGCSE Bulletin”, 25, 1, 151-154, 1993.
9. Stallings, W., *Computer Organization and Architecture: Designing for Performance*, 8th ed., “Prentice Hall PTR”, Upper Saddle River, NJ, USA, 2009.
10. Weaver, C.T., Larson, E., Austin, T., *Effective Support of Simulation in Computer Architecture Instruction*, “Proc. Workshop Computer Architecture Education”, 48 – 55, 2002.