Step-by-Step Design and Simulation of a Simple CPU Architecture

Derek C. Schuurman
Redeemer University College
777 Garner Road East
Ancaster, Ontario, CANADA
dschuurman@cs.redeemer.ca

ABSTRACT

This paper describes a sequence of assignments, each building upon the next, leading students to a working simulation of a simple 8-bit CPU (Central Processing Unit). The design features a classic Von Neumann architecture comprising a simple data path with a few registers, a simple ALU (Arithmetic Logic Unit), and a microprogram to direct all the control signals. The first step involves the design of the ALU which is capable of eight basic operations. The second step guides students to construct a datapath complete with several 8-bit registers. The third step involves the design and implementation of a control unit which uses a microprogram to implement machine code instructions. The microprogram implements nine basic machine language instructions which are sufficient for writing many simple programs. The final step involves adding program memory and an input and output device to form a simple working simulation of a computer. At this point, students may hand-assemble code for their CPU and simulate its execution.

All simulations are performed using a free and open source simulator called Logisim which performs digital logic simulations with the ability to build larger circuits from smaller subcircuits. Students can set an adjustable clock rate and observe the internal CPU state and registers as it retrieves instructions and steps through the microcode. The basic CPU architecture provides many opportunities for more advanced exercises, such as adding an instruction fetch unit, adding pipelining, or adding more machine language instructions. The assignments were introduced in a second year course on computer organization, providing an effective hands-on approach to understanding how a CPU actually operates.

Categories and Subject Descriptors

C.0 [**General**]: Modeling of computer architecture

SIGCSE'13, March 6–9, 2013, Denver, Colorado, USA.

General Terms

Design, Theory

Keywords

computer organization; CPU simulation; education

1. INTRODUCTION

Computer organization and architecture form important parts of the computer science "body of knowledge" as identified by the ACM and IEEE computing curriculum task force. The 2008 computer science curriculum guidelines continue to identify computer organization and architecture as important topics in the computer science curriculum. According to this report "A professional in any field of computing should not regard the computer as just a black box that executes programs by magic ... Students need to understand computer architecture in order to make best use of the software tools and computer languages they use to create programs" [2].

One way to learn about this topic is to study existing CPU architectures. Many different CPU architectures abound, and different textbooks have selected different processor architectures as examples. Several commercial processors have been used as exemplars to describe computer organization and architecture concepts, including the 8051 [7], the MIPS processor [6], and the popular x86 and ARM architectures [9].

Other authors have developed their own architectures designed with pedagogical principles in mind to help students understand the fundamentals of computer organization. Some simple processors used for pedagogical purposes include the picocontroller described in [5] and the Mic-1 described in [11]. Other simple computer architectures can be found in popular textbooks such as [3] and [8].

Another strategy is to guide students in a step-by-step manner to construct their own basic processor. This approach seeks to help students go beyond simply understanding computer architecture and guide them to apply the concepts to the design of a simple computer architecture.

There are many different approaches to having students build a CPU. Using real digital hardware is fraught with many engineering pitfalls: chip fan-outs, ground bounce, clock ringing, stray inductance and capacitance, open and short circuits, and propagation delays to name just a few. These low-level issues are important for electrical and computer engineers to master, but can be a source of frustration and distraction when the objectives are to teach basic computer architecture principles. Using programmable logic de-

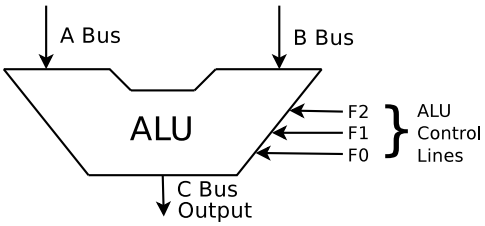

Figure 1: Block diagram of a simple ALU.

vices such as FPGAs (Field Programmable Gate Arrays) can reduce the issues associated with assembling digital circuits, but the development environments associated with FPGAs are complex and can take considerable time to master.

A better approach is to use a basic logic simulator, which provides a simpler environment free of FPGA architecture constraints and low-level electrical issues. These environments provide digital logic elements at a level of abstraction that allows students to experiment with basic digital design issues. In particular, Logisim provides a friendly educational tool for performing basic digital simulations [4]. Logisim enables designing and simulating digital circuits ranging from small logic circuits up to entire CPUs. A recent comparison of logic simulators for computer design courses found Logisim to be a good option for educational use [10]. Furthermore, Logisim is an open source tool that can be freely downloaded by students from the web [1].

2. THE SIMPLE CPU DESIGN

This paper will describe a series of assignments using Logisim to provide a step-by-step design and simulation of a simple CPU. The design of the processor is inspired and derived from the Mic-1 architecture as described by Andrew Tanenbaum and Todd Austin in their text *Structured Computer Organization* [11]. The steps are broken down into several assignments, each building upon the next. These steps are described in the following subsections.

2.1 Step 1: A Simple ALU

The students are first provided with a hands-on lab tutorial to teach them the fundamentals of using Logisim. The first assignment includes some initial logic design exercises such as designing a 2-bit demultiplexer and testing it using Logisim.

After the students gain a working knowledge of Logisim, they are given the first assignment. This assignment requires the students to design and simulate an 8-bit ALU using logic gates. The ALU must have two 8-bit bus inputs (labelled A and B) and one 8-bit output as illustrated in Figure 1. The function of the ALU is selected using three ALU control lines ($F0$, $F1$ and $F2$) which select one of eight operations in the ALU as indicated in Table 1. The ALU can be constructed in a straightforward manner by building the digital circuit for each operation and using a 1-of-8 multiplexer to select which appears at the output. Once the ALU is constructed and tested, the ALU can be turned into a subcircuit and used as a building block in subsequent assignments.

2.2 Step 2: The CPU Datapath

The next step in the process is to take the ALU subcircuit built in the previous step and place it inside a datapath using

Table 1: Summary of simple ALU operations.

F2	F1	F0	Output
0	0	0	`A`
0	0	1	`B`
0	1	0	`A+1`
0	1	1	`B+1`
1	0	0	`A+B`
1	0	1	`A-B`
1	1	0	`A AND B`
1	1	1	`A OR B`

Table 2: List of CPU Registers.

Register	Description
A	General purpose A register
B	General purpose B register
OUT	Output register
MBR	Memory Byte Register
PC	Program Counter
MIR	Microprogram Instruction Register
MPC	Microprogram Program Counter

Logisim. The datapath is a simplified version of the Mic-1 datapath and represents a basic Von Neumann architecture. Using 8-bit buses and registers, the students are guided to create a datapath that forms the basis of a simple CPU as illustrated in Figure 2. (Note that the 7-segment display and input switch are not part of the CPU, but are included as helpful diagnostic devices). The CPU will eventually include two general-purpose registers, one output register, and four special-purpose registers as summarized in Table 2. The destination register for the output of the ALU (C bus) is determined by selecting the write control line on the appropriate register. The A Bus input to the ALU is tied directly to the A register, hence the A register figures prominently in nearly all ALU operations. The B Bus input to the ALU uses a multiplexer to select which register output will appear on the bus. The control of the multiplexer is summarized in Table 3.

Operation of the CPU datapath can be simulated by preloading the A register with a preset value (a helpful feature in Logisim allows pre-loading registers with a click of the mouse). Next, students must discern how to toggle the appropriate ALU control lines and write lines to move the value in the A register through the ALU and into the OUT register. If the operation is done correctly, the current value in the A register will appear in the OUT register. At this point, the students have just simulated a "Move" operation in a simple CPU datapath!

After completing this step, students can either demonstrate their working CPU datapath, or submit a screenshot of Logisim showing the 7-segment display output along with a description of the sequence of control lines that were asserted to perform the move operation. At this point, the students are manually performing operations that will later be automated by the control unit.

2.3 Step 3: The Control Unit

The manual control of the datapath simulated in the previous step highlights the need to manage the control signals in the CPU. In previous assignments, the students built an

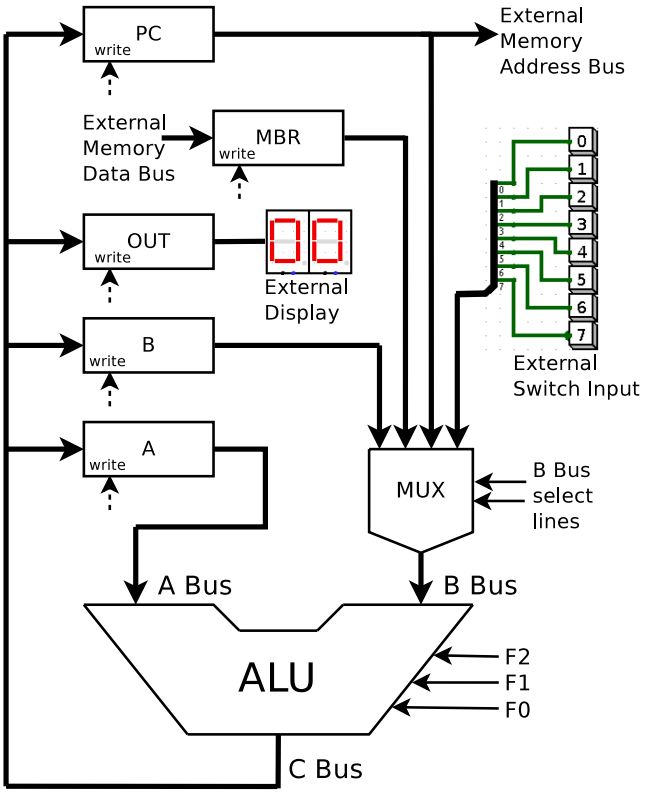

Figure 2: The block diagram of the CPU datapath.

Table 3: B Bus Multiplexer Select Lines.

Select Lines	ALU B Bus Input
00	B Register
01	MBR (Memory Byte Register)
10	PC (Program Counter)
11	External input

ALU and added registers to simulate a simple CPU datapath using Logisim. In this next step, students complete the final piece in the CPU puzzle by adding the control unit.

The CPU requires a sequencer to step through the operations necessary to execute each machine language instruction. A microprogram can be used to orchestrate and coordinate the various registers and control lines within the CPU. To accomplish this, a ROM (Read Only Memory) is used as the control store for the microprogram. The microprogram and control unit used in the assignment are derived from the Mic-1 architecture as described in sections 4.1 and 4.3 of *Structured Computer Organization* [11]. For this assignment, students are required to use a 32x16-bit ROM for the control store as arranged in Figure 3. The 5-bit address bus for the control store is connected to a special purpose register called the Microprogram Program Counter (MPC). The 32-bit data bus from the control store is latched into another special purpose register called the Microprogram Instruction Register (MIR).

Students are given the following specifications for the control store and surrounding logic:

- the control store ROM address is provided by a 5-bit MPC register. It is synchronized by the rising edge of the clock
- the next address loaded into the MPC register should be selected by a multiplexer controlled by a JMPC line as defined in Table 4
- a 16-bit data bus from the control store ROM connects to the MIR register and is synchronized by the falling edge of the clock
- The MIR bits are arranged as follows (from most-significant to least-significant bits):
  - Next address (5-bits)
  - JMPC (Jump control) bit
  - ALU control lines (3-bits)
  - Register write control lines (5 bits)
  - ALU B Bus input select lines (2 bits to select one of four registers)

Students are then coached to connect the output bits from the MIR to the various control lines within the CPU. Logisim provides a convenient "tunnel" connector to connect

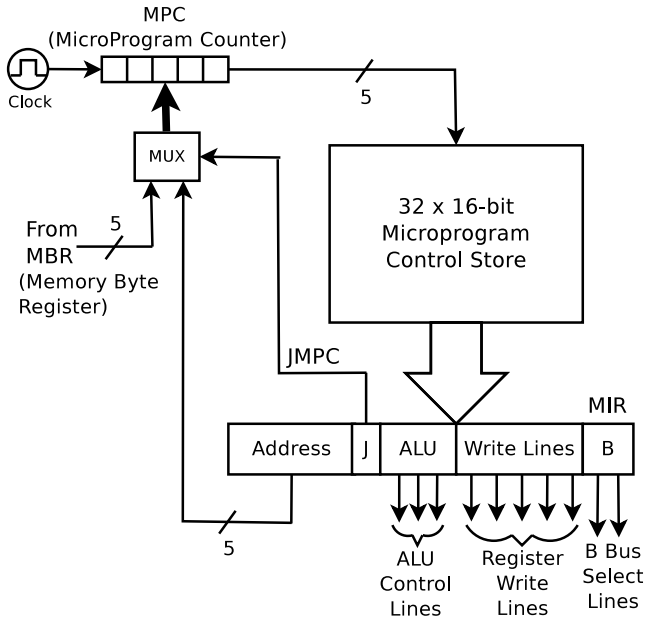

Figure 3: Block diagram of the control unit.

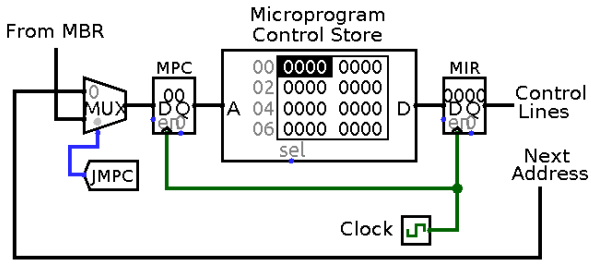

Figure 4: Logisim blocks for the control unit.

Table 4: JMPC Operation.

JMPC	Description
0	Next address loaded from the top 5-bits of MIR
1	Next address is loaded from MBR (bottom 5-bits of the Memory Byte Register)

Table 5: List of simple CPU instructions.

Instruction	Description
`NOP`	No operation
`INPUT`	Load Input into A register
`OUTPUT`	Move A register to Output register
`JMP <addr>`	JMP to address <addr> in program memory
`LOAD A,#<value>`	Load <value> into the A register
`INC A`	Increment the value in register A by 1
`MOV B,A`	Move register A into register B
`ADD A,B`	Add register A to register B and store result in register A
`HALT`	Halt execution

lines using labels rather than drawing individual wires. This avoids having a diagram containing a "rat's nest" of wires and connections which are hard to follow and debug. A Logisim diagram of the control unit is shown in Figure 4.

Once the control unit wiring is complete, students will need to enter a microprogram into the control store ROM. The microprogram directs the flow of the datapath for each of the various machine language instructions. For simplicity, the instruction set is limited to the nine simple instructions summarized in Table 5.

The microcode begins at location 00h in the control store with the main loop which continually fetches and executes instructions from program memory. The main loop fetches the machine language instructions (opcodes) from program memory at the address stored in the Program Counter (PC) register and loads them into the MBR (Memory Byte Register). After the instruction is loaded, the PC register is incremented to the next location. The main loop then branches to execute the microcode for the current instruction (for convenience in this design, the opcode corresponds to the actual address in the control store). Once the microcode for the current instruction is completed (which may require several microinstructions and clock cycles) control is transferred back to the main loop (at location 00h in the control store) and the process repeats.

The microcode for each of the assembly language instructions can be given to the students, or added as a further assignment. The use of a spreadsheet can be helpful in coding the microprogram by assigning a column to each control signal and a row to each memory location in the control store (a sample of a few spreadsheet rows is shown in Figure 5). The bit patterns in the resulting table can be converted into 16-bit hexadecimal words which are then placed in the corresponding locations in the control store. Logisim provides an intuitive interface for entering data into a ROM device.

Once the microprogram has been entered into the control store ROM, the CPU is complete. The CPU, including the registers, ALU and control unit can all be placed inside a single Logisim subcircuit and represented using a single block. The CPU block can now be connected to an external program memory and input/output components to simulate a simple computer. The address and data bus of the program memory should connect to the CPU block. The address bus should internally connect to the output of the PC register, and the data bus should internally connect to the input of the MBR register. The CPU block can also be connected to external input switches and the 7-segment output display as illustrated in Figure 6.

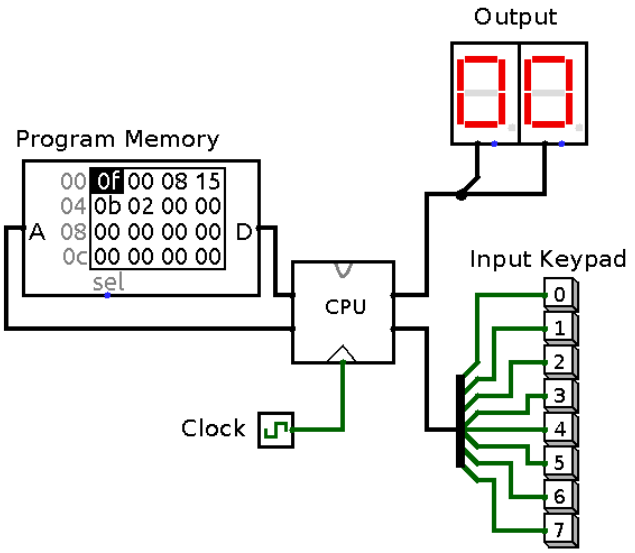

Figure 6: Final schematic diagram of computer with CPU represented as a Logisim subcircuit and connected to input and output devices and program memory.

2.4 Step 4: Putting It All Together

Students are given the following simple assembly language program to test their design. The students must first hand-assemble the program into machine language and then enter it into program memory starting at address 00h. The opcodes for each assembly language instruction can be deduced from their starting addresses in the microprogram ROM. Students can then run this program on their CPUs and observe what it does.

```
00h  LOAD A,#0
02h  OUTPUT
03h  INC A
04h  JMP 2
```

If everything is working properly, students should observe the CPU counting in the 7-segment display output. If something is not working correctly, students can debug their design in Logisim by single-stepping the clock and tracking the state of the program or even the microprogram. Logisim also supports adjustment of the clock speed, so students can observe how the CPU clock rate effects the counting speed on the 7-segment display.

Even with the limited set of assembly language instructions, a variety of additional programs can be assigned. For example, students can code a program that continuously

Opcode	Hex Address	Decimal Address	Next Addr	JMPC	ALU2	ALU1	ALU0	R1	R0	OUT$_{WE}$	PC $_{WE}$	MBR$_{WE}$	B$_{WE}$	A$_{WE}$		Comment
(main)	00	0	1	0	0	0	0	0	0	0	0	1	0	0		Fetch MBR
	01	1	2	0	0	1	1	1	0	0	0	0	0	0		PC = PC + 1
	02	2	3	0	0	1	1	1	0	0	1	0	0	0		Store PC
	03	3	0	1	0	0	0	0	0	0	0	0	0	0		Goto MBR
NOP	04	4	0	0	0	0	0	0	0	0	0	0	0	0		goto main
	•															
	•															
	•															

Figure 5: Sample of a spreadsheet showing a portion of the microprogram listing.

reads the input switches and displays their values (in hexadecimal) on the 7-segment display output. For homework submissions, students can be asked to hand in their assembly code and machine language translation along with a print out of the completed CPU with the input/output devices and program memory.

3. FUTURE WORK

This simple CPU design provides a foundation for many possible future enhancements. One obvious enhancement is to add support for additional instructions in the microprogram. Some instructions that are conspicuously absent in the instruction set include compare operations and conditional jump instructions. The ALU could be further enhanced by adding a more elaborate status register to support more useful compare operations. Furthermore, the ALU could be modified to provide an 8-bit integer multiply or divide instruction. Moreover, students could be asked to enhance the architecture by adding an instruction fetch unit to independently update the PC register and fetch the next instructions. Additional enhancements can be explored such as adding various stages of pipelining.

To give students a glimpse of all the levels of computer organization from the digital logic layer up to the assembly language layer, the students can be guided to make a simple two-pass assembler for the CPU they have designed. Logisim has support for a binary text file format that can be loaded into a ROM block rather than typing in the hex codes manually. The assembler could be programmed to output an assembled program directly to a binary text file that can be read directly into Logisim. The students will then be able to write assembly language programs in a text editor, assemble them them with their assembler, load the resulting binary code into the program memory block, and then run it on their CPU.

4. CONCLUSIONS

Logisim is a friendly and simple tool for performing digital logic simulations, yet it is powerful enough to simulate a basic CPU. Using a simulator avoids many of the pitfalls that accompany working with real hardware and allows students to focus on understanding basic computer organization concepts. Having students build their own simple CPU has pedagogical value which augments the description of existing industrial CPU architectures. This paper describes a sequence of step-by-step assignments that gradually guide students to create their own simple CPU simulation. As each assignment builds on the next, students gain an insight into the layers of computer organization from the digital logic level up to the assembly language level. Designing a simple CPU from the ground up provides an opportunity to grasp basic computer architecture concepts while students experience the satisfaction of making their own CPU. The simulation provides many opportunities for further exercises such as adding additional instructions, features and performance enhancements to the basic CPU architecture.

5. ACKNOWLEDGEMENTS

The author would like to thank all the contributors to the Logisim project for making it available to educators and students.

6. REFERENCES

[1] `http://ozark.hendrix.edu/~burch/logisim/`.

[2] Computer science curriculum 2008. `http://www.acm.org/education/curricula-recommendations`, 2008. Report from the ACM/IEEE Interim Review Task Force.

[3] S. Brown and Z. Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw Hill, second edition, 2005.

[4] C. Burch. Logisim: a graphical system for logic circuit design and simulation. *Journal on Educational Resources in Computing*, 2(1):5–16, Mar. 2002.

[5] F. Cady. *Microcontrollers and Microcomputers: Principles of Software and Hardware Engineering*. Oxford University Press, second edition, 2010.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, third edition, 2003.

[7] I. S. MacKenzie. *The 8051 Microcontroller*. Prentice Hall, 1999.

[8] M. M. Mano and C. R. Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, second edition, 2004.

[9] W. Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall, eighth edition, 2010.

[10] T. Stanley, V. Chetty, M. Styles, S.-Y. Jung, F. Duarte, T.-W. J. Lee, M. Gunter, and L. Fife. Teaching computer architecture through simulation: (a brief evaluation of cpu simulators). *Journal of Computing Sciences in Colleges*, 27(4):37–44, Apr. 2012.

[11] A. S. Tanenbaum and T. Austin. *Structured Computer Organization*. Prentice Hall, sixth edition, 2013.