Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs

Timothy D. Stanley, Thanh Quach Xuan, Leslie Fife, Don Colton

School of Computing
Brigham Young University Hawaii
Laie, HI 96762 USA

StanleyT@byuh.edu, xq003@byuh.edu, fifel@byuh.edu, ColtonD@byuh.edu

Abstract

Students learn better when they both hear and do. In computer architecture courses "doing" can be difficult in small schools without hardware labs hosted by computer engineering, electrical engineering, or similar departments. Software solutions exist. Our success with George Mills' Multimedia Logic (MML) is the focus of this paper. We have found that students learn and understand more, and experience less frustration, without the additional complexity of hardware details. MML provides a graphical computer architecture solution with convenient I/O support and the ability to build and emulate a variety of computer designs. It has proven highly motivational to upper-division computer science students designing and constructing emulated computers. Student projects resulted in excellent student understanding of the detailed inner workings of computers. Students also developed better teamwork skills and produced useful training aids for the lower-division computer organization class. Designs implemented include 8-bit and 16-bit, von Neumann and Harvard architectures, from single-cycle to twelve-cycle instructions. Issues resolved during the learning process include timing, initialization, instruction set architecture, I/O, and assembler design. We provide two demonstration computers used to illustrate to students a design approach and an expected outcome in their individual design activities. One example is an eight-bit Harvard architecture with eight instructions that execute in a single clock cycle. The second is an eight-bit von Neumann architecture that has four instructions and executes each instruction in three clock cycles. This paper describes these two example computers.

Keywords: Computer Organization & Architecture, Emulation Software, Pedagogy.

1 Introduction

Effective learning comes from doing, not just hearing. Computer Architecture is an area where doing is a natural extension to the course material. There are many approaches to teaching a lab component for Computer Architecture. These include no lab at all, emulated hardware, actual hardware, and a mix of emulated and actual. Software tools range from register- and memory-level computer simulators to high-level chip design languages. Between these extremes is George Mills' Multimedia Logic (MML). We discuss the use of computer simulators and chip definition languages. We then introduce Multimedia Logic and our approach.

However taught, Computer Architecture is an essential part of any computer science curriculum. In the periodic curriculum report issued by a joint task force headed by the ACM and IEEE-CS, Architecture and Organization is one of 14 core bodies of knowledge (CC 2001). The 36 minimum core hours in Architecture and Organization represents more than 12% of all core topic hours (CC 2001). While not binding, this curriculum recommendation is a good indication of the central place Computer Architecture occupies in the curriculum. The core architecture topics include machine level representation of data, memory system organization and architecture, and alternative architectures (CC 2001). The teaching of each of these can be enhanced through the MML software tool. For example, alternative architectures can be clearly shown. In our examples, both Harvard and von Neumann designs are shown.

1.1 Computer/Logic Simulators

Many computer architecture classes use emulators. The popular textbook "Computer Organization and Design" (Paterson, and Hennessy 1994) features SPIM by James Larus (Larus 2006). It shows memory contents and registers and includes an assembler but does not simulate the datapath.

Moving closer to our goal, many computer architecture classes use the text by Null and Lobur that introduces MARIE (2003a). They may also use MarieSim (Null, and Lobur 2003b). MARIE uses a von Neumann architecture with a simple instruction set. Using MarieSim, students can write their own programs and watch them execute, seeing the effect these programs have on system state. MARIE has been implemented with a "Data Path Simulator" that highlights registers and data paths visually. This shows the steps the processor goes through when running a program. However, students cannot build their own computers and simulate them.

At the other end of the complexity spectrum is the implementation of a simulator of an entire real architecture. Clark, Czezowski, and Strazdins implemented a simulator for the Sparc V9 (2001). Even if it had a GUI, this is of limited usefulness for a typical undergraduate course in computer architecture. The level of complexity this embodies is not a good starting point for learning computer architecture. In addition, pedagogically, learning only one machine's architecture is limiting. The Alfa-1 simulator (Wainer, Daicz, Simoni, and Wasserman 2001) also works specifically with the Sparc processor. This tool allows the user to experiment with the entire architecture, including extending it in some ways. However, the limited user interface and the restriction to a single architecture are limitations. The Alfa-1 was designed to replace tools that simulated obsolete architectures. This approach ensures that an eventual replacement for Alfa-1 will be needed.

Some simulation tools address only a single problem. The KScalar simulator (Moure, Rexachs, and Luque 2002) provides a tool for learning about microprocessors. This is, of course, only part of what must be covered in a typical computer architecture course. However, this might be a good choice for an advanced course on microprocessors or for a few labs within a larger course. A similar approach is used by Holland, Harris, and Hauck (2003). They provide an incomplete processor and have students design the missing pieces. They can simulate any non-floating-point instruction in their 8-instruction MIPS processor. This creates an opportunity for students to learn processor details. However, the student is still limited to a single processor type without floating point. Other parts of the machine architecture still must be learned in some fashion. Another single-purpose tool is SIMT (Tao, Schulz, and Karl 2003). This simulator allows the evaluation of shared-memory systems. While these tools may be very useful in specialized computer architecture courses, having to learn several unconnected single-purpose tools takes extra effort in a general computer architecture course.

1.2 Chip Design Languages

Several chip design languages are available.

Logisim (Burch 2002) is another design and simulation tool. A basic package, Logisim allows the user to design circuits from basic logic components and some basic devices. The primary drawbacks are the lack of a clock for timing and the somewhat primitive I/O capabilities.

Logic Works 5 from Capilano Computing is a wonderful package, but the emphasis is on detailed timing and simulation of circuits to be exported to silicon (2006). It seems more suited to advanced students that already grasp computer architecture and are looking for the next step. It does not have the input and output devices available in Multimedia logic.

DLSim by Matthew Leslie was developed while an undergraduate student with additional development funded by Cambridge (2006). It is an open source Java program available from http://www.sourceforge.net/. DLSim has the capability to use macros and can display logic states propagated through the circuits. But the devices are simply blocks and the rich set of IO devices available in Multimedia logic is not available.

A number of schools have used VHDL as a design medium for computer architecture. While VHSIC is used commercially for chip design, we believe it is too abstract to visualize and is too much like a software design to be physically satisfying.

1.3 Multimedia Logic (MML)

The package we chose is Multimedia Logic (MML), open source free software by George Mills (2006). MML strikes a good balance. We can design and implement logic at the gate level, but still use high-level I/O operations. (This is parallel to the C programming language. C provides near-assembler access to the machine, but still includes a standard I/O library in the basic distribution.)

We have seen how a variety of tools exist to simulate imaginary and real computers based on simple and complex instruction sets. Generally these tools allow the simulation of only a single architecture and they often hide many of the underlying details that we might like to reveal. You can write programs and see the results in various registers, but the computer itself remains a black box. Alternately, some tools allow the student to access the logic gate level of design but have limited I/O capabilities.

MML allows the student to define an Instruction Set and build the enabling architecture. This means the student is not limited to von Neumann computers or the instruction sets designed by others. With an excellent graphical user interface and ASCII I/O, the tool is easy to learn and use. Because MML is open source software, functions can be added and the tool recompiled. Despite being easy to use it is capable of sophisticated designs. For example, see the work of James Larson (2006).

Multimedia Logic is a very decent environment for virtual computer construction. We were able to implement all of the necessary components in order to run programs written for MARIE. The benefits of MML for our project were extensive. Ease of use compared to physical hardware is obvious. In addition, MML uses abstraction to simplify the hardware components it offers and reduces the need to build all of one's own components, such as an arithmetic logic unit.

2 Architecture Course Design

The philosophy of our computer architecture course is that students will only truly understand computer architecture when they design and build a computer. Students are shown how to build a computer through a couple of designs that are the focus of this paper. We demonstrate starting from an Instruction Set Architecture and continuing through the building of emulated hardware to implement the instruction set. After working through these example designs, students were assigned to invent an original design, including an instruction set, and the registers to support the instructions. Our students are then required to implement their design in MML (Mills 2006). We have used this approach for three years now and students have produce some wonderful designs. Students feel empowered and are highly motivated to participate in the labs and survive the debugging process. They report having a great sense of accomplishment and achieving a profound understanding of how computers work at the logic level.

3 Example Computer One, the von Neumann Design

We start our discussion of MML with a "Hello World" example shown in Figure 1. This seems to be the first program shown in every programming language text, and it is fun to do it in an architecture course as well.

In a recent semester, to set the stage for the design assignment, the instructor provided as an example an 8-bit, accumulator-based, von Neumann design that uses three clock cycles to execute each instruction. The von Neumann architecture uses a single memory for data and instructions. To make this computer as simple as possible, but still able to demonstrate operation with useful programs, it was designed with four instructions. The instructions are: Load the accumulator from memory, Save the accumulator to memory, Add from a memory location to the accumulator, and Jump if the last add produced a zero result. These four instructions were supplemented by two memory mapped commands, Output on Save to memory location x3F, and Halt on Save to memory location x3E. Figure 1 shows the main page of this computer running a "Hello World" program. A second page of multiplexers is shown in Figure 2.

One of the most difficult challenges in a design like this is to develop an instruction decoder state machine in "Read Only" memory. This decoder takes as inputs a step count, the op-code and the zero flag, and provides as outputs control signals to set paths through multiplexers and enable writing to memory and registers

One very nice feature of designs done in Multimedia Logic is the ease of attaching diagnostic displays to the computer. Displays include hexadecimal, binary, and

ASCII display terminal. Also available are text displays that provide one of sixteen text strings depending on a provided four-bit binary value. These text displays are used to show the current state of the computer in the execute cycle, and the instruction currently being executed.

Even though this computer has only four instructions they cover the basics needed to illustrate an accumulator-based von Neumann architecture.

One significant learning to come from this design is the elegance of self-modifying code to implement a program like the "Hello World" program shown. An effective indirect load can be designed in the von Neumann architecture by just recursively incrementing the load instruction. This can lead to a discussion on self-modifying code.

In the Harvard architecture discussed next, with twice as many instructions, the "Hello World" program is a series of output commands for characters stored in the data memory because the program memory can not be modified by the running program.

4 Example Computer Two, the Harvard Design

An alternate example, developed three years ago, just redesigned to improve readability, is a single cycle, 8-bit, Harvard architecture describe in the proceedings of the ACM Workshop on Computer Architecture Education (Stanley 2005). This design, shown in figures 3 and 4, uses two memories, one, read only, for program storage and a second, read-write, for data storage. It also has a read-only memory used to decode operation codes into control line states. This design also uses two ALU devices, one to increment the program counter and a second to execute mathematical instructions. Having two ALUs enables instructions to execute in one cycle, with the program counter incrementing while the other processes are also occurring. Also, this architecture can simultaneously access data and program memory eliminating the "von Neumann Bottle neck". But since mathematical operations are on data memory, creating self modifying code is not possible in this design. This makes some program tasks awkward. For example, the hello world program implemented in the Harvard design consists of a series of output commands.

One of the nice features of designs in MultiMedia Logic is the freedom to lavishly include display devices to show hexadecimal values, binary values, ASCII text, and value dependent comments, like the display "OutM" which is the mnemonic for the current instruction.

In this design, as with the von Neumann design, one of the most difficult challenges is designing the operation decoder, although for the Harvard design, this task is simpler since each instruction takes just one cycle. One nice capability with both designs is the ability to single step through instructions, by pressing the "Clock Pulse" button or have the computer run by lifting the "Enable Clk" switch.

5 Conclusions

These two designs, while not very advanced, have proven to be very useful for teaching principles of computer architecture. The have also served as a spring board for some very elaborate designs from our students. Some of the student designs include sixteen bit computers, hardware multipliers, a 128 bit key hardware encryption unit, and a fully multiplexed sixteen register array of sixteen bit registers. Also, since hardware is not used in these designs, students can take them with them at the end of the semester.

Some comments from students taking this class include the following:

"This project was very motivating for our team. We spent many hours to insure that a quality project was built. We learned a lot through the design and assembly as well as in debugging. Our fellow students also learned from our presentation of this project."

Another student added:

"We appreciate for this new method of teaching the computer architecture. We think that we learn much more than if we physically build computers. I personally understand more how to build and design a computer from the ground up. Although our computer is in a logical form, it does represent our knowledge and our work. Instead of spending more time on connecting wire or circuit, our team actually spends more time on the primary elements in designing, planning, debugging, and understanding how our computer works."

Patterson says "The processor comprises two components: data path and control..." (1994). These students know what that means because they have built both.

6 Acknowledgements

The authors gratefully acknowledge George Mills, the author of Multimedia Logic, for making his logic simulation package available without cost and including the source code on his web site, http://www.softronix.com/.

The authors are also deeply grateful to the students they have taught who have been the motivation for this development. Special thanks to Elliot Manning who was the team leader of the first student team and set the bar for the following classes.

7 References

Burch, C. (2002): Logisim: A Graphical System for Logic Circuit Design and Simulation. *ACM Journal of Educational Resources in Computing* **2**(1): 5–16.

Capilano Computing. http://www.capilano.com/. Accessed 1 June 2006.

Clark, B., Czezowski, A. and Strazdins, P. (2001): Implementation Aspects of a SPARC V9. *Proc. 25th Australasian Computer Science Conference*, Melbourne, Australia, 23–32.

Computing Curricula 2001 Computer Science, Final Report, ACM/IEEE-CS Task Force, 15 December 2001. http://acm.org/education/curric_vols/cc2001.pdf. Accessed 9 August 2006.

Holland, M., Harris, J. and Hauck, S. (2003): Harnessing FPGAs for Computer Architecture Education. *Proc. 2003 IEEE International Conference on Microelectronic Systems Education*, 12.

Larson, J. http://www.dst-corp.com/james/MMLogic.html. Accessed 1 June 2006.

Larus, J. SPIM, A MIPS32 Simulator, http://www.cs.wisc.edu/~larus/spim.html. Accessed 1 June 2006.

Leslie, M. http://urchin.earth.li/~mleslie/project.html, Accessed 1 June 2006.

Mills, G. Multimedia Logic, available for free download at http://www.softronix.com. Accessed 1 June 2006.

Moure, J., Rexachs, D. and Luque, E. (2002): The KScalar Simulator. *ACM Journal of Educational Resources in Computing* **2**(1): 73–116.

Null, L. and Lobur, J. (2003): The Essentials of Computer Organization and Architecture. Sudbury, MA, Jones and Bartlett Computer Science.

Null, L. and Lobur, J. (2003): MarieSim: The MARIE Computer Simulator. *ACM Journal of Educational Resources in Computing* **3**(2): 1–29.

Paterson, D. and Hennessy, J. (1994): Computer Organization and Design, The Hardware/Software Interface, Morgan Kaufmann.

Stanley, T. (2005): An emulated computer with assembler for teaching undergraduate computer architecture. *Proc. of the Workshop on Computer Architecture Education*.

Tao, J., Schulz, M. and Karl, W. (2003): A Simulation Tool for Evaluating Shared Memory Systems. *Proc. 36th Annual Simulation Symposium (ANNS'03).*

Wainer, G., Daicz, S., Simoni, L. and Wassermann, D. (2001): Using the Alfa-2 Simulated Processor for Educational Purposes. *ACM Journal of Educational Resources in Computing* **1**(2): 111–151.

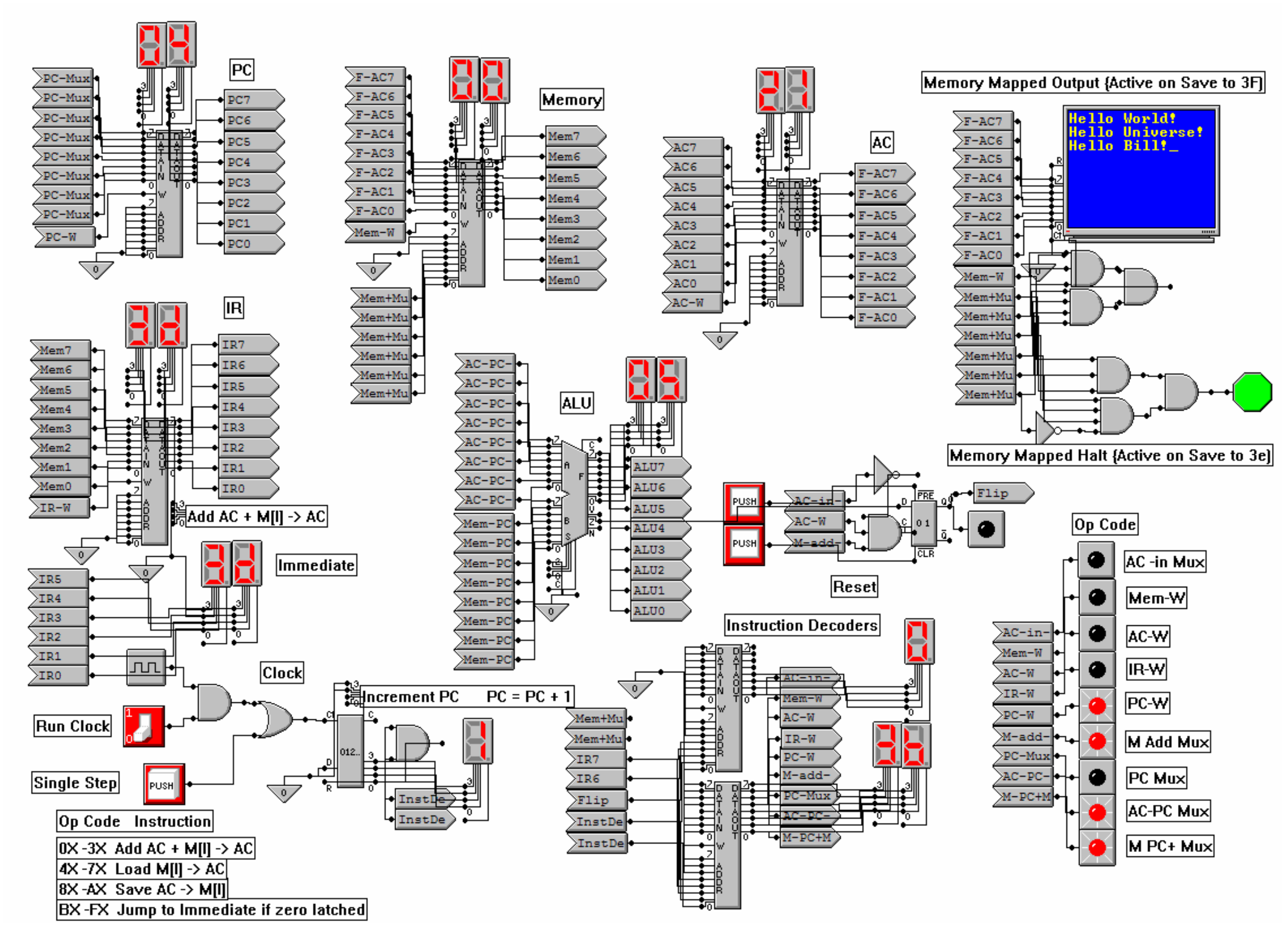

Figure 1 An example computer using an 8-bit von Neumann design

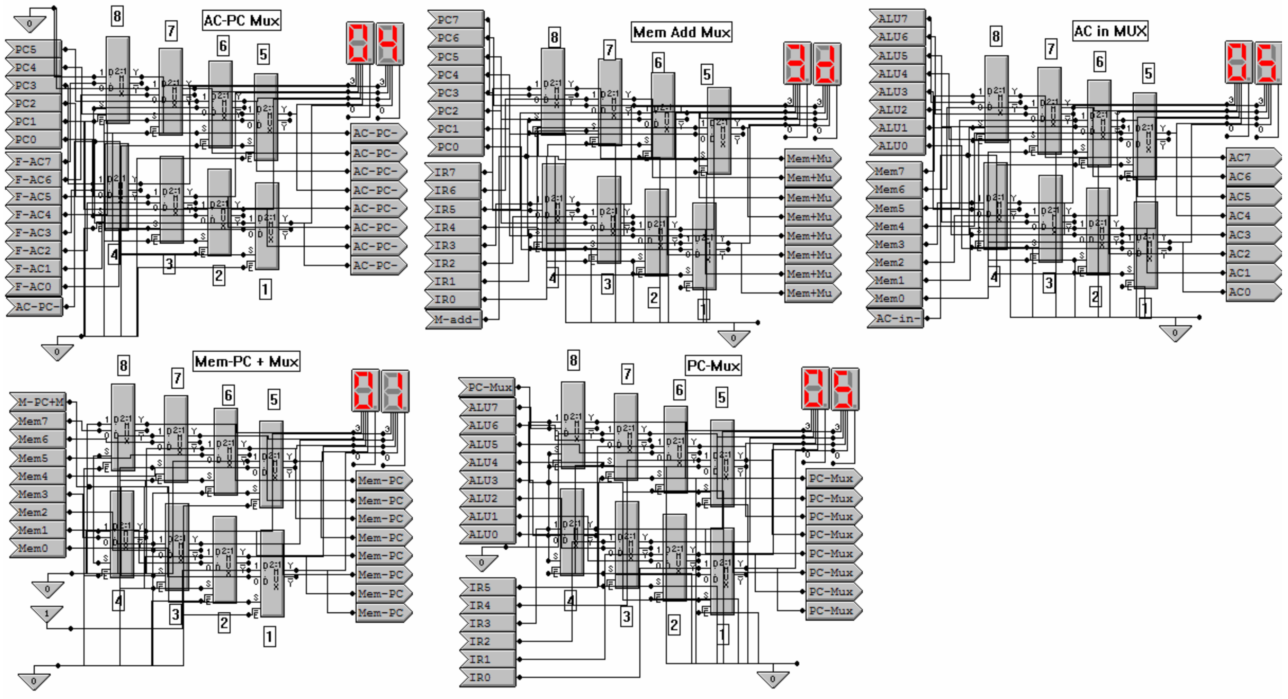

Figure 2 Multiplexer array for the 8-bit von Neumann design

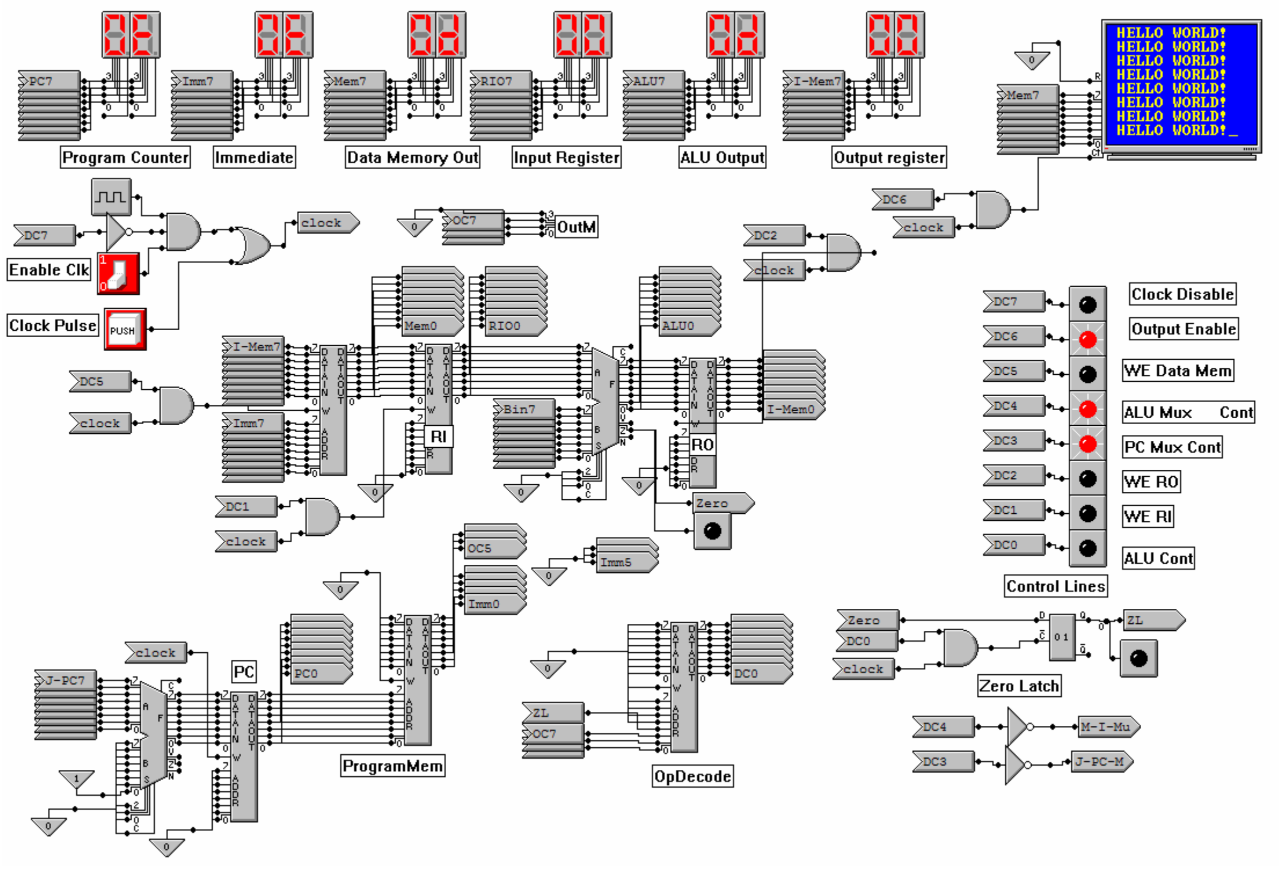

Figure 3 Example computer two, the eight bit Harvard design

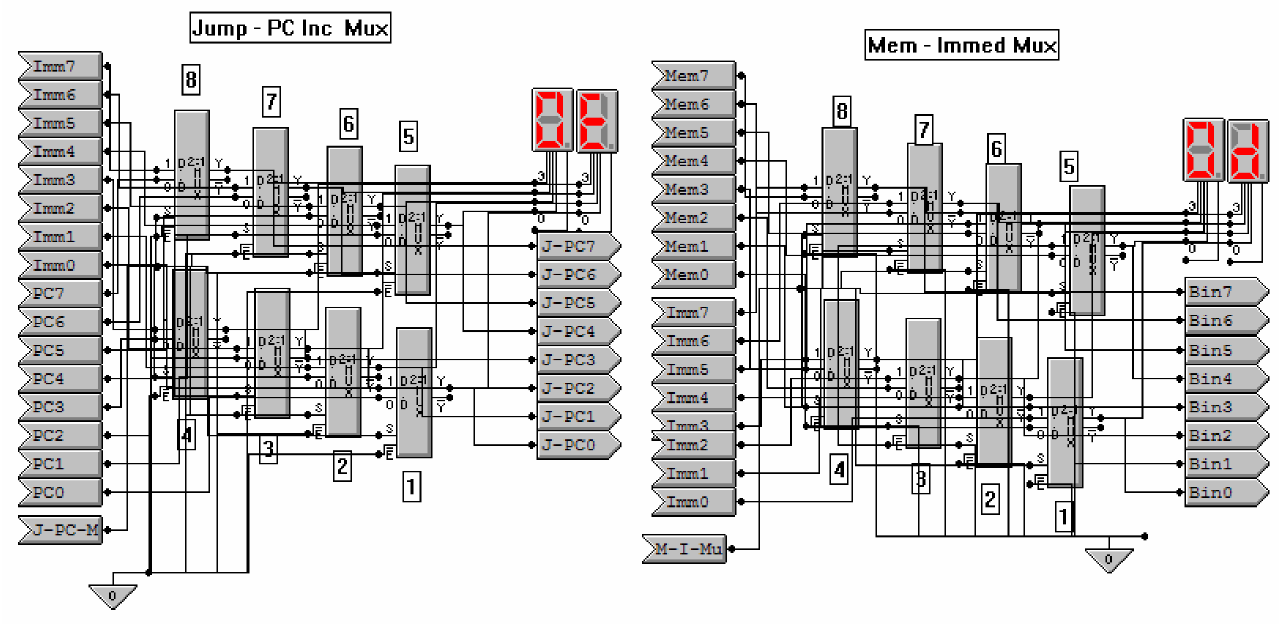

Figure 4 Multiplexer array for Harvard design