A Demonstration of CPU Organization Using a Simple Apparatus and Sixteen People

Alexander A. Sherstov, Jr., John J. Krupczak, Jr.
Department of Computer Science/Department of Physics and Engineering
Hope College
Holland, MI 49423

Abstract
We have developed a laboratory activity to demonstrate the basic central processing unit (CPU), input, output, and memory of a computer. The activity is intended to help beginning engineering students or non-engineering students to understand basic computer architecture. The activity is based on sixteen individuals who are assigned to conduct specific tasks in a manner analogous to specific computer hardware elements. Tasks assigned to humans include: I/O, memory, control unit, ALU, and registers. The central processing unit consists of five internal registers, an arithmetic/logic unit, and a control unit. These components are interconnected via an internal bus. Bus arbitration is implemented using tri-state buffers. The CPU communicates with memory and the I/O devices through an 8-bit data bus, a 4-bit address bus, and a control bus. The instruction set includes a total of 9 instructions designed to perform arithmetic/logic operations, control flow, and I/O operations involving memory and external devices. Prior to simulation, computer programs are converted to binary code and loaded into memory. Program code comprises a data section and an instruction section. The data section features integers in two's complement notation. The instruction section is composed of a sequence of fixed-length instructions, each consisting of an identifying 4-bit opcode and an optional 4-bit operand. Operands represent absolute memory addresses. Execution starts at address 0000, which corresponds to the beginning of the instruction section. The fetching and execution of an instruction takes up 4 to 5 phases, each requiring a single CPU cycle. Each phase is characterized by a collection of control signals output by the control unit to the rest of the system. There are a total of 18 distinct phases. The control unit can uniquely determine the next phase from the current phase. Although this system supports only the most basic CPU functionality and lacks many features found in modern CPUs (such as multiple addressing modes, variable-length instructions, and exception handling), it can be effectively used to illustrate a variety of fundamental computing concepts. Among these are the fetch-decode-execute cycle, sequential execution, conditional and unconditional branching, and iteration.

1 Introduction

The quality of education in science and technology for all undergraduates is becoming an area of increasing concern [1]. In the United States, the National Science Foundation is requesting that Science, Math, Technology and Engineering (SME&T) programs concentrate more effort on the 80% of college students who are not SME&T majors. In response, science and engineering faculty are developing courses intended to specifically address the needs of the non-SME&T students. A review of some historical background information and relevant new developments

has been compiled by Byars [2]. In these initiatives for non-SME&T students, a need exists for laboratory exercises to accompany the lecture component of the course. The work reported here describes a laboratory activity developed to accompany a technological literacy course for non-science majors taught at Hope College [3,4].

The goal of the activity is to help non-SME&T students to learn the rudiments of how a computer processor works. This particular activity addresses basic computer architecture, the concept of an instruction set, algorithmic operations, and complex functions performed through permutations of simple operations.

Lecture-based approaches exist to demonstrate the basic central processing unit (CPU), input, output, and memory of a computer. In addition, basic electronic laboratory microprocessor demonstrations are available. Our experience with non-SME&T students has shown that to capture and retain student interest, novel and amusing activities are essential. A lecture-based presentation of this topic was considered a problematic approach with non-SME&T students. It is difficult to keep this audience engaged in following the evolution of what may be deemed arcane algorithmic procedures. Use of a microprocessor electronics laboratory may appear overly complex and inaccessible to the non-SME&T student.

2 Activity Requirements

A primary requirement for the activity was to involve the students in an active and immediate manner in the operation of a simple computer. Another essential requirement was to develop an organization for the simulated computer that is simplified but not simplistic. A level of sophistication that is challenging but still understandable by the target audience was sought.

Other requirements influenced the design of the activity. These were derived from the nature of laboratory projects for non-SME&T students. The activity must be able to be set up and taken down in less than approximately 15 minutes. No permanent installations are possible. Materials needed should be inexpensive and readily obtained. Total cost for any materials needed should not exceed US$ 200. The space needed should not exceed that of a gymnasium floor or small parking lot. Materials should be durable and be able to withstand some rough handling without the need for extensive repairs or adjustments.

3 Organization of the Simulated Computer

The organization of the simulated computer is shown in Fig. 1. The four major components are the central processing unit (CPU), memory, input device, and output device. These units are interconnected by a system of three buses: the address bus, data bus, and control bus. Memory and the input and output devices are atomic in that they are not subdivided into smaller components; each of them is simulated by a single human participant. The CPU has a complex internal organization and is simulated by a team of 13 participants.

3.1 CPU

The CPU performs all arithmetic and logic computations. In addition, it is responsible for coordinating the operation of the system's components so that the overall task can be performed. The CPU is connected to each of the other components by one or two control lines, which collectively make up the control bus. These lines carry instructions from the CPU to the rest of the system. For example, an asserted *W* ("write") line of the control bus instructs memory to

perform a memory write using the values on the data and address buses. Likewise, the CPU-driven control bus governs the operation of the input and output devices.

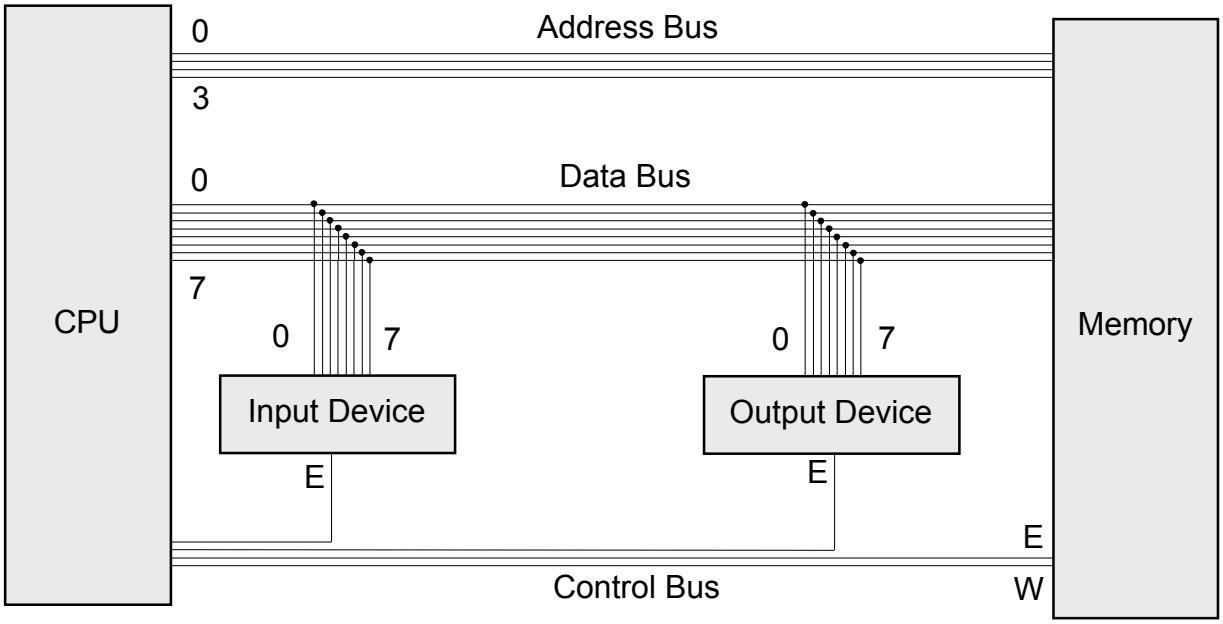

Fig. 1. System Organization.

3.2 Memory

Memory stores loaded programs (i.e., sequences of fixed-length instructions expressed in binary code) and temporary results of the CPU's calculations (expressed in 2's complement binary numbers). Memory comprises 16 individually addressable 8-bit storage cells, numbered 0000 through 1111 in binary. Memory is connected to the data bus (which supplies the data to be written to memory or receives data being retrieved from memory) and to the address bus (which supplies the address of the memory location involved in a write or read operation). In addition, memory is connected to the CPU by means of two control lines, *E* and *W*. The *E* ("enable") signal is used to activate memory prior to a read or write operation. When the *E* signal is asserted, the *W* ("write") line indicates whether a write or a read operation is to be performed.

3.3 Input/Output Devices

The input and output devices make it possible for the computer system to communicate with the outside world. Each is connected to the data bus, which receives data from the input device and supplies data to the output device. The input and output devices are each activated by a control line from the CPU. By asserting a device's control line, the CPU requests that it perform an input or output operation, resulting in data being placed on or read from the data bus.

4 CPU Organization

The architecture of the CPU (Fig. 2) features 13 components interconnected by an internal bus. Essential aspects of this design were borrowed from a basic CPU architecture described in Carpinelli [5]. The specific components of the CPU are the control unit, the arithmetic-logic unit (ALU), 5 registers, 5 tri-state buffers, and a NOR gate.

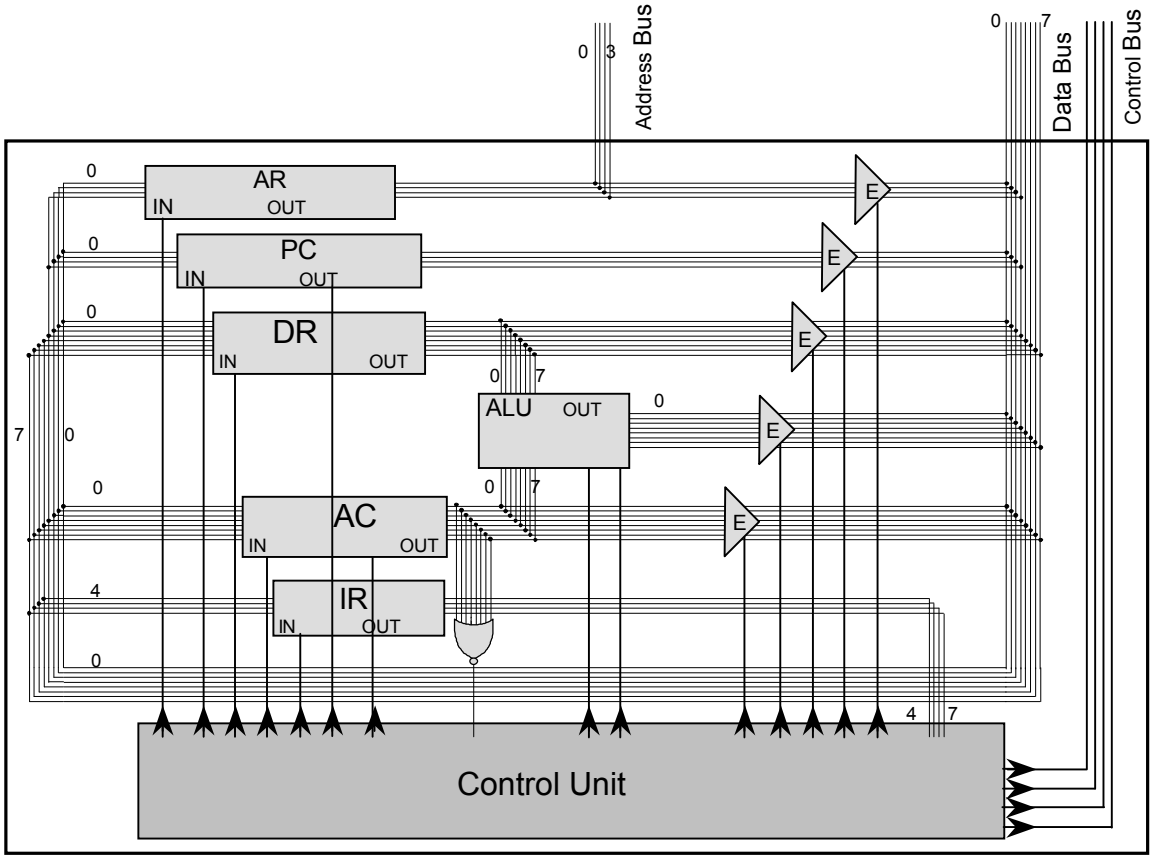

Fig. 2: CPU Organization

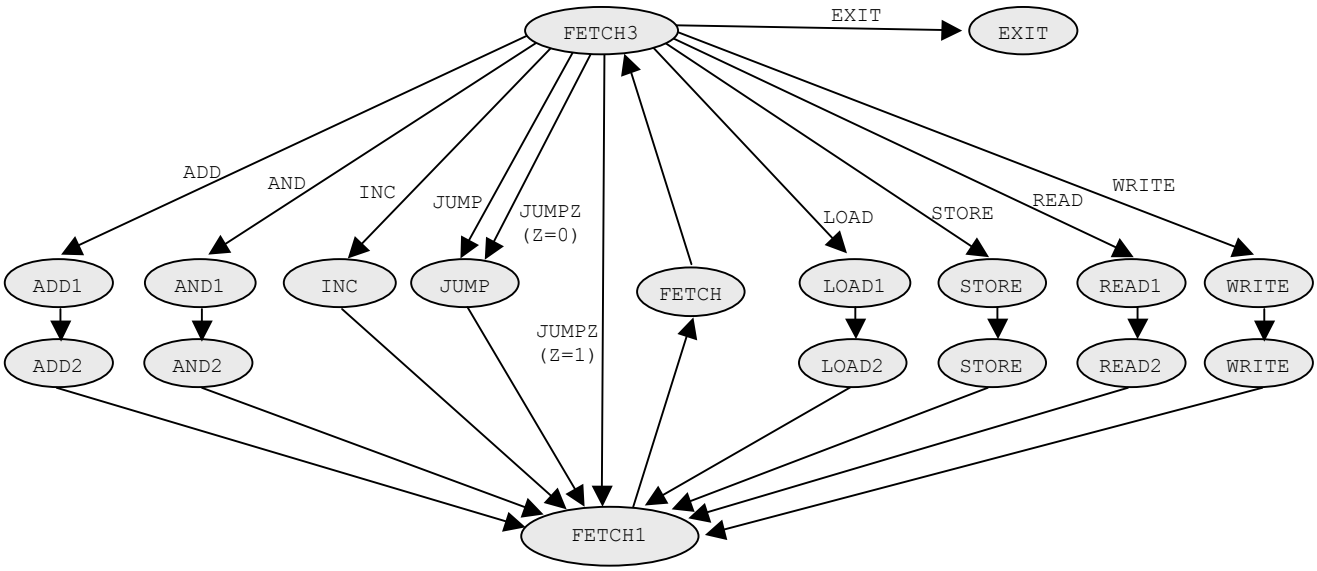

Fig. 3. Control Unit Phase Transition Diagram.
The initial phase is `FETCH1`. Labels on the arrows refer to the name of the instruction being processed.

Phases	Operations	Control Signals																		
		AR_LOAD	AR_BUS	PC_LOAD	PC_BUS	PC_INC	DR_LOAD	DR_BUS	AC_LOAD	AC_BUS	AC_INC	IR_LOAD	ALU_BUS	ALU_OP	ALU_ENABLE	MEM_ENABLE	MEM_WRITE	INP_ENABLE	OUT_ENABLE
FETCH1	PC → AR	x			x														
FETCH2	M → DR, PC++					x	x									x			
FETCH3	DR[7-4]→IR, DR[3-0]→AR	x						x				x							
ADD1	M → DR						x									x			
ADD2	AC + DR → AC								x				x		x				
AND1	M → DR						x									x			
AND2	AC & DR → AC								x				x	x	x				
INC	AC++										x								
JUMP	AR → PC		x	x															
LOAD1	M → DR						x									x			
LOAD2	DR → AC							x	x										
STORE1	AC → DR						x			x									
STORE2	DR → M							x								x	x		
READ1	IN → DR						x											x	
READ2	DR → AC							x	x										
WRITE1	AC → DR						x			x									
WRITE2	DR → OUT																		x

Fig. 4. Control Signals, by Phase. Asserted signals are marked with an **'x'.**

4.1 Control Unit

The purpose of the control unit is to fetch and execute program instructions stored in memory. The control unit drives the control bus of the system. In addition, it uses internal control signals to operate the other 12 components of the CPU. Among these signals are the "enable" and "operation select" signals of the ALU; the "enable" signals of the tri-state buffers; and the "load" and "increment" signals of the registers. A major responsibility of the control unit is to assert the needed signals at the right time to enable the transfer of data within the CPU or between the CPU and memory or input/output devices.

The control unit executes an instruction in several phases, each taking up a single clock cycle. The first 3 phases serve to fetch the instruction from memory to the CPU. The following one or two phases execute the newly fetched instruction, at which point control branches back to the instruction-fetch phases. Fig. 3 gives a diagram of control flow between phases.

At the beginning of a phase, the control unit asserts a specified subset of the control lines. The lines remain asserted throughout the phase. There are a total of 18 distinct phases defined for the system. Fig. 4 features, for each phase, the subset of the control signals to be asserted during that phase. As the phase transition diagram (Fig. 3) shows, the next phase of the control unit is fully determined by its current phase.

4.2 ALU

The purpose of the ALU is to compute the sum or bitwise conjunction of its two 8-pin inputs and place the result on the 8 output pins. The *OP* ("operation-select") line driven by the control unit selects the specific operation (ADD or AND) to be performed. The ALU starts processing the data on the input pins as soon as the control unit asserts its *E* ("enable") signal.

5 Registers

The CPU contains 5 registers. The registers store intermediate calculation results and CPU state information. The address register, *AR*, stores a memory address and is involved in all data transfer operations between the CPU and memory. The data register, *DR,* receives data from or places data on the data bus and is likewise used for transferring data between the CPU and memory. The program counter, *PC,* holds the address of the next instruction to be fetched from memory; its value is incremented every time an instruction is fetched. The accumulator, *AC,* supplies one of the arguments to, and receives the result of, every arithmetic/logic operation. *AC* is unique in that no other register is directly accessible by a program instruction. The instruction register, *IR,* holds the opcode (i.e., the identifying 4-bit binary pattern) of a newly fetched instruction. When an instruction is being executed, the control unit determines what signals to output based on the contents of the instruction register.

Each register has 4 or 8 input pins and the same number of output pins. The input pins supply data to be stored in the register; the output pins receive the register's current data. Each register is also equipped with an *LD* ("load") signal controlled by the control unit. When its *LD* signal is asserted, the register reads in and stores the value currently on the input pins and places it immediately on the output pins. In addition, some registers have an *INC* ("increment") signal used to increment the register's value by 1.

5.1 Tri-state Buffers

The tri-state buffers enable the control unit to govern the CPU components' access to the shared internal bus. Bus arbitration is necessary to prevent data loss or corruption caused by several components outputting data to the bus at the same time. The outputs of the ALU or a register pass through a tri-state buffer before reaching the bus. A buffer allows this passage only when the control unit asserts its *E* ("enable") input; at all other times, the buffer outputs infinite impedance.

6 Instruction Set

The instruction constitutes the basic unit of computation by the CPU. The computer's instruction set comprises 9 instructions, each identified uniquely by a 4-bit opcode. Each instruction occupies 8 bits. The 4 high-order bits represent the opcode. The low-order 4 bits represent a memory address and serve as an argument to the instruction. If a given instruction does not require an argument, the address bits are ignored. A list of the 9 instructions and their respective formats is given in Fig. 5.

Name	Format	Function
EXIT	0000@@@@	Terminates operation of the computer.
ADD	0001@@@@	Adds the value stored at memory location @@@@ to the accumulator.
AND	0010@@@@	Computes the bitwise AND of the value at memory location @@@@ and the accumulator.
INC	0100xxxx	Increments the accumulator by 1.
JUMP	1000@@@@	Transfers control to memory location @@@@.
JUMPZ	1001@@@@	Transfers control to memory location @@@@ if the accumulator equals zero.
LOAD	1100@@@@	Loads the value stored at memory location @@@@ into the accumulator.
STORE	1101@@@@	Stores the value of the accumulator at memory location @@@@.
READ	1110xxxx	Reads a value from the external input device and places it in the accumulator.
WRITE	1111xxxx	Writes the value of the accumulator to the external output device.

Fig. 5. Instruction Set.

Two sample programs in the form of assembly-language code and the corresponding binary code are presented in Fig. 6 and Fig. 7. Program code consists of a sequence of instructions. If the program uses variables, their values are stored in memory locations following the program code. Programs are loaded into memory starting at address 0000, the memory location at which execution starts.

```
Program Section:    Memory Contents:
0000    LOAD        0000    11001000
1000                0001    00010011
0001    ADD 1001    0010    00011010
0010    ADD 1010    0011    11110000
0011    WRITE       0100    00000000
0100    EXIT        0101    00000000
                    0110    00000000
Data Section:       0111    00000000
1000    3           1000    00000011
1001    7           1001    00000111
1010    15          1010    00001111
                    1011    00000000
                    1100    00000000
                    1101    00000000
                    1110    00000000
                    1111    00000000
```

Fig. 6. Sample Program #1.
The program computes the sum of 3, 7, and 15 and sends the result to the output device.

```
Program Section:    Memory Contents:
0000    READ        0000    11100000
0001    JUMPZ       0001    10010101
0101                0010    00011000
0010    ADD 1000    0011    11011000
0011    STORE       0100    10000000
1000                0101    11110000
0100    JUMP 0000   0110    00000000
0101    WRITE       0111    00000000
0110    EXIT        1000    00000000
                    1001    00000000
Data Section:       1010    00000000
1000    0           1011    00000000
                    1100    00000000
                    1101    00000000
                    1110    00000000
                    1111    00000000
```

Fig. 7. Sample Program #2.
The program computes the sum of a sequence of numbers read from the input device (0 marks the end of the sequence) and sends the result to the output device.

7 Hardware Implementation

The hardware implementation was designed to be as simple as possible so that the workings would be apparent to the participants. Besides the human participants, it was desired to create the

computer from essentially only switches, wire, and lights. The hardware implementation uses LED's turned OFF or ON to indicate a 0 or 1 condition. Manually actuated 4 or 8 pole single throw switches are used to load data onto a bus. Other signals are activated using single pole switches. Construction is accomplished using solderless breadboards. Board-to-board connections are made using 14 conductor ribbon cables equipped with DIP connectors. This style of connector was chosen for its low cost and because it can be attached directly into the solderless breadboards. Since at most, only 8 conductors are required for the bus, some of the remaining 6 conductors are used to provide 5 VDC and ground to each unit. The physical layout closely follows that used to describe the CPU and computer organization given in Fig. 1 and Fig. 2.

8 Operation with Human Participants

The simulation involves 16 individuals. Memory and the input and output devices are each operated by a single human participant. Simulation of the CPU involves 13 individuals: 1 to operate the control unit, 1 to operate the ALU, 5 to operate the 5 registers, and 5 to operate the 5 tri-state buffers. An additional participant is needed to simulate the system clock (a crystal oscillator that synchronizes the actions of the system components and sets the pace for the overall operation). A simple sound-producing device, such as a gong, can be conveniently used for such a simulation. Two distinct sound patterns are needed to mark the "rising edge" of the clock (i.e., the beginning of the first half of the clock cycle) and the "falling edge" (i.e., the beginning of the second half of the clock cycle).

Each participant is responsible for a specific task and has limited knowledge of how the rest of the system works. The overall complex behavior of the system implicitly emerges from many individuals' working on simple tasks in a coordinated fashion. While fifteen of the students are engaged in the simulation, the rest of the class can observe the operation of individual components or of the overall system. After a few minutes, the participants and observers switch roles. Specifications for the actions of each participant follow.

Memory. If the *E* ("enable") signal is asserted and the *W* ("write") signal is not asserted, locate the memory cell at the address specified by the address bus and output its value onto the data bus. If the *E* and *W* signals are both asserted, locate the memory cell at the address specified by the address bus and write to it the value that is currently on the data bus (on the rising edge of the clock).

Input device. If the *E* ("enable") signal is asserted, place the next data item on the data bus.

Output device. If the *E* ("enable") signal is asserted, read in the value from the data bus on the rising edge of the clock.

Control unit. Follow the control unit phase diagram (Fig. 3), changing phases at the end of every clock cycle. Immediately after entering a new phase, assert the corresponding set of signals, as specified by the control signal table (Fig. 4).

ALU. When the *E* ("enable") signal is asserted and the *OP* ("operation select") signal is not asserted, perform the addition of the values on the data pins and output the result to the output

pins. When the *E* and *OP* signals are both asserted, compute the bitwise AND of the values on the data pins and output the result to the output pins.

Registers. When the *LD* ("load") signal is asserted, read in the value on the input pins on the rising edge of the clock. When the *INC* ("increment") signal is asserted, increment the value by 1 on the rising edge of the clock.

Tri-state buffers. When the *E* ("enable") signal is asserted, relay the incoming signals unchanged to the output pins. When the *E* ("enable") signal is not asserted, output infinite impedance.

Clock. Strike the gong at strictly periodic intervals. A single clock cycle encompasses two gong beats, one to mark the rising edge of the clock and another one to mark the falling edge.

9 Results and Conclusions

The simple computer described in this paper meets the stated goals of developing a demonstration of computer architecture in which non-SM&T majors, or other beginning students, can participate. The activity is complex enough to convey the essential features of a processor, but simple enough that little prior knowledge is needed by the students. This activity provides a novel approach to familiarizing students with CPU architecture.

10 Acknowledgments

The authors would like to thank the faculty and staff of the Hope College Department of Physics and Engineering, and the Hope College Science Division for their contributions to this work. This research was supported in part by the National Science Foundation under grant DUE-9752693.

References

[1] National Science Foundation, "Shaping The Future: New Expectations for Undergraduate Education in Science, Mathematics, Engineering, and Technology, *NSF 96-139*, October 1996.
[2] N. A. Byars, "Technological Literacy Classes: The State of the Art," *Journal of Engineering Education*, January 1998. 53-61.
[3] J. J. Krupczak, Jr., "Science and Technology of Everyday Life: A course in technology for liberal arts students," *Proceedings of the 1996 American Society for Engineering Education Annual Conference*, Washington, D.C., June 23-26, 1996, session 2261.
[4] J. J. Krupczak, Jr. "Demystifying Technology," *ASEE PRISM*, October 1997. 30-34.
[5] J. D. Carpinelli, *Computer Systems Organization & Architecture*, Addison-Wesley, October 2000.

Biographical Information

ALEXANDER A. SHERSTOV JR.
Alexander Sherstov is currently a senior majoring in computer science at Hope College in Holland, Michigan.

JOHN KRUPCZAK, JR.
John Krupczak, Jr., is an Associate of Engineering at Hope College in Holland, Michigan. He has a BS degree in physics from Williams College and MS and Ph.D. degrees in mechanical engineering from the University of Massachusetts at Amherst. His email address is: krupczak@hope.edu.