An Interactive Web-Based Simulation of a General Computer Architecture

William Yurcik*, Joaquin Vila, and Larry Brumbaugh

***Abstract* — This paper describes a web-based simulation based on a conceptual paradigm used to introduce computer architecture and assembly language programming to undergraduate students. An application has been developed using Java and made available on the web (http://138.87.169.30/LMC/). We show how the web-based application is used to teach basic programming concepts. In addition, experience has been that interactive use of these simulations enable student-centered learning of more complex issues such as addressing modes and operating system concepts. By visualizing all parts of the computer architecture simultaneously during the execution of the program, this application facilitates the comprehension of the von Neumann architecture and its relationship to assembly language. Examples of programs, system development trade-offs, and a description of the user interface are presented. Future extension of this type of simulation for other computer architectures is discussed.**

***Index Terms* — web-based simulation, computer architecture, computer organization, assembly language, von Neumann architecture.**

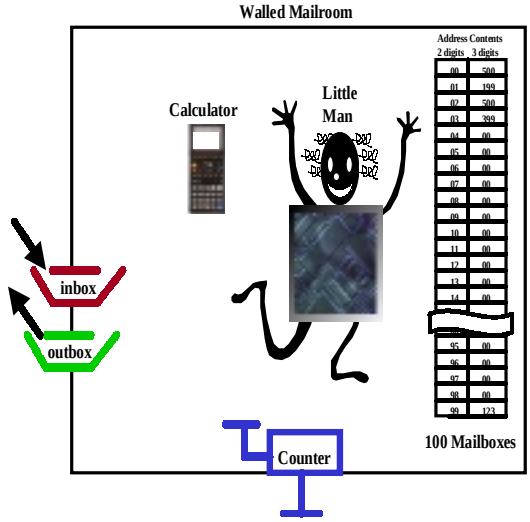
I. Introduction

Students taking introductory courses in computer organization and assembly language find it difficult to understand the various concepts related to computers without a proper model/tool that represents the architecture of a working computer.[3-9] We have developed a web-based simulation tool for this purpose based on the Little Man Computer (LMC) model developed by Stuart Madnick of MIT.[1] This model has a set of instructions that can be executed on the LMC and operates in a manner very similar to actual computers, helping students to understand the basics of computer architecture.

II. The LMC Paradigm

The LMC paradigm consists of a walled mailroom, 100 mailboxes numbered 00 through 99, a calculator, a two-digit location counter, an input basket, and an output basket. Each mailbox is designed to hold a single slip of paper upon which is written a three-digit decimal base number. Note that each mailbox has a unique address and the content of each mailbox is not the same as the address. The calculator can be used to enter and temporarily store numbers and to add and subtract. The two-digit location counter is used to increment the count each time the Little Man executes an instruction. The reset for the location counter is located outside of the mailroom. Finally there is the "Little Man" himself, depicted as a cartoon character, who performs tasks within the walled mailroom. Other than the reset switch for the location counter, the only communication a user has with the Little Man is via slips of paper with three-digit numbers put into the input basket or retrieved from the output basket.

FIGURE I
THE LITTLE MAN COMPUTER PARADIGM



We have implemented the LMC paradigm as a web-based application implemented in Java embedded within an applet. The only user requirement is a Java-enabled browser such as Internet Explorer 4.0 or Netscape Navigator/Communicator 3.0 and LMC can be accessed anywhere via the Internet. User documentation is available via separate web help menus and within the application itself.

The LMC simulation visually shows a one-pass assembly process (mnemonic source code to machine code) and load process (moving machine code into mailboxes). In the edit mode users can write source code (which is automatically checked for syntax errors) or load source code from an existing file. For convenience of the programmer, three different execution modes are provided: (1) Burst Mode, where all instructions in the program are executed until a HALT instruction is encountered; (2) Step Into, which

Manuscript received on March 8, 2000. This work was supported in part by grants from John Deere Corporation and State Farm Insurance Co.

All authors are affiliated with the Department of Applied Computer Science, Illinois State University USA.

* Corresponding author; additional contact information: Email wjyurci@ilstu.edu, voice/fax 309-438-8016/5113, hard copy: Campus Box 5150, 202 Old Union, Normal IL 61790 USA.

executes one instruction at a time; and (3) Step Over, which is similar to Step Into except for SKIP instructions (for SKIP instructions, if the condition is met the program executes the next instruction and then waits for user interaction). Break points are available for all three execution modes for debugging. A flag register indicates error conditions and is set/reset for corresponding overflow/underflow conditions and zero/negative/positive values within the calculator.

A. Computer Architecture

Although there have been experimental computer architectures, the von Neumann architecture continues to be the general architecture for computers. It is significant that in a field where technological changes is so rapid that the general computer architecture is virtually unchanged since 1951.[1] Today's CISC and RISC architectures are consistent with the broad characteristics that define a von Neumann computer. The major characteristics that define a von Neumann architecture include:

- stored program concept: memory holds both programs and data
- linear memory addressing: there is a unique sequential numeric address for every memory location
- memory is addressed by location without consideration of memory contents.
- sequential execution of instructions (unless an instruction or outside event causes a branch to occur)
- functional organization: control unit (executes instructions), Arithmetic Logic Unit (ALU- arithmetic and logic operations), Program Counter (PC), Input/Output interfaces (I/O), and memory

It should be observed that the LMC is a direct example of a von Neumann architecture. The calculator corresponds to the ALU, mailboxes correspond to memory, location counter corresponds to the PC, I/O corresponds to input/output baskets, and the Little Man himself corresponds to the control unit. Both data and instructions are stored in the mailboxes. There is no distinction between data and instructions except when a specific operation is taking place. The exact steps performed by the Little Man are important for students to visualize because they reflect the steps performed in a real CPU when executing an instruction.

It should also be noted that the analogy is not perfect. In a real computer, memory (mailboxes) is actually separated both physically and functionally from the central processing unit (CPU). In a most computers, general-purpose registers (accumulators) are available to temporarily hold data being processed. In LMC, the calculator display panel loosely serves the purpose of an accumulator. The path of Little Man performing tasks is loosely equivalent to computer bus connections but does not allow for data to be on different busses simultaneously (Little Man in two places at once). Clock timing and interrupts are not part of the LMC paradigm. Lastly, the LMC instruction set is based on the decimal system and a real CPU operates in binary. We make this numbering system trade-off to highlight novice understanding of computer architecture while rigorously covering binary/octal/ hexadecimal representations elsewhere in a course.

B. Instruction Set

The Little Man performs tasks by following simple instructions which are described by three-digit numbers. The LMC instruction set is fundamentally similar to the instruction sets of many different computers. In fact, the LMC instructions – data movement, arithmetic, and branching – are central to the instruction set of every computer. In a LMC instruction, the first digit describes the operation (opcode) and the last two digits specify the mailbox address to be acted upon (operand). The instructions provide a way to move data between the inbox, outbox, calculator, and mailboxes. There are also instructions that cause Little Man to stop (HALT) or branch conditionally/unconditionally (SKIP).

Table I defines the LMC instruction set. These nine instructions are sufficient to perform the steps of any computer program.

TABLE I
LMC INSTRUCTON SET

Opcode	Description	Mnemonic
1	LOAD contents of mailbox address into calculator	LDA XX
2	STORE contents of calculator into mailbox address	STA XX
3	ADD contents of mailbox address to calculator	ADD XX
4	SUBtract mailbox address contents from calculator	SUB XX
500	INPUT value from inbox into calculator	IN
600	OUTPUT value from calculator into outbox	OUT
700	HALT - LMC stops (coffee break)	HLT
	SKIP	
801	SKN - skip next line if calculator value is negative	SKN
802	SKZ - skip next line if calculator value is zero	SKZ
803	SKP - skip next line if calculator is positive	SKP
9	JUMP – goto address	JMP XX

NOTE: XX represents a two-digit mailbox address

When working with students, we emphasize two things about the LMC instruction set:

1. although any program can theoretically be implemented in LMC assembly source code, the actual implementation may be painful!
2. expanded instruction sets on modern computers do not change the fundamental operations of the computer!

Like most assembly language instruction sets, it is difficult to write useful functions in a small number of source code lines. We use this to emphasis the relationship of high level languages to assembly language in terms of programmability, user friendliness, and efficiency. Computer instruction sets on real computers are more sophisticated and flexible, providing additional instructions that make programming easier, however, these additional instructions do not change the fundamental operations of the computer. Students learn the important idea that a computer is nothing more than a

machine capable of performing simple instructions at high speed. Later in the course we discuss instruction set variations as the major difference between types of computers and show specific examples of different code which performs the same task.

C. Sample Programs

Table II shows an example program in LMC assembly language source code. We save for later discussion how these instructions are stored in the mailboxes starting at mailbox address 00. For this example program the user must place two numbers in the Input Box. The sum of the two numbers will appear in the Output Box.

TABLE II
PROGRAM TO ADD TWO NUMBERS

mailbox	LMC assembly source code	machine code
00	IN ; input 1st number from inbox to calculator	500
01	STA 99; store number from calculator in 99	299
02	IN ; input 2nd number from inbox to calculator	500
03	ADD 99; add contents of 99 to calculator	399
04	OUT ; output result from calculator to outbox	600
05	HLT ; stop	700
99	DAT 00; reserve and clear memory location	000

First we assume the location counter has been reset to 00. The Little Man looks at the location counter (00) to find the memory address containing the next instruction to be executed. The Little Man goes to memory address 00, gets the instruction from the contents of the memory address 00, puts it on his forehead (to keep his hands free – the forehead is often referred to as the instruction register in class discussion), and then executes the instruction. When Little Man completes executing an instruction, he goes to the location counter and increments it (by 1 in his case). Then Little Man looks at the location counter to find the memory address of the next instruction and so forth until the HALT instruction is executed. Thus the Little Man will execute the instructions in the mailboxes sequentially starting at mailbox 00. Since the location counter is reset outside of the mailroom, the user can restart the program simply by resetting the counter.

D. Fetch-Execute Cycle

We refer to the part of the repetitive cycle in which Little Man finds an instruction to execute as the "fetch" portion and the part of the cycle in which Little Man actually performs the task specified in the instruction as the "execute" portion. The fetch portion of the cycle is identical for each instruction and occurs before the execution portion. The execute portion is different for each instruction.

This analogy is close to the fetch-execute cycle of a real CPU. In a real CPU a detailed fetch-execute cycle would entail: (1) transferring the contents of the PC to the Memory Address Register (MAR) resulting in a the contents of the memory address specified by the PC (the instruction) to be moved to the Memory Data Register (MDR); (2) transferring the contents of the MDR (the instruction) to the Instruction Register (IR); and (3) the remaining steps are the execution portion and thus instruction dependent. We discuss this detail with students up to the point of optimizing microcode for each instruction.

III. LMC TECHNICAL DESCRIPTION

A. Hardware and Software Requirements

The hardware and software requirements for LMC are generally available PCs running Windows 95/98/NT. LMC should work on any Java-enabled web browser although only tested with Microsoft Explorer and Netscape Navigator.

B. Java Implementation

LMC was developed using Java JDK1.2 and is embedded in an applet so as to provide ubiquitous access to users over the Internet without the need for JDK1.2 to be installed locally. Another advantage of using this approach is that the applet is loaded in a separate window allowing the user to run the application as well as look at the HTML documentation in separate windows. To enable control of the LMC simulation, multiple threads were implemented: one thread for executing the program and a second thread listening for a user interrupt to halt the program.

C. Security

Applets have restricted access to computer file systems on most browsers so to enable saving and loading of source code the user has to install a signed certificate. Signed applets have full functionality while unsigned applets may still execute in a restrictive "sandbox" mode preventing hard disk access and web site connections. Signing certificates for Microsoft Internet Explorer and Netscape Navigator are handled in different ways. Complete documentation is provided online.

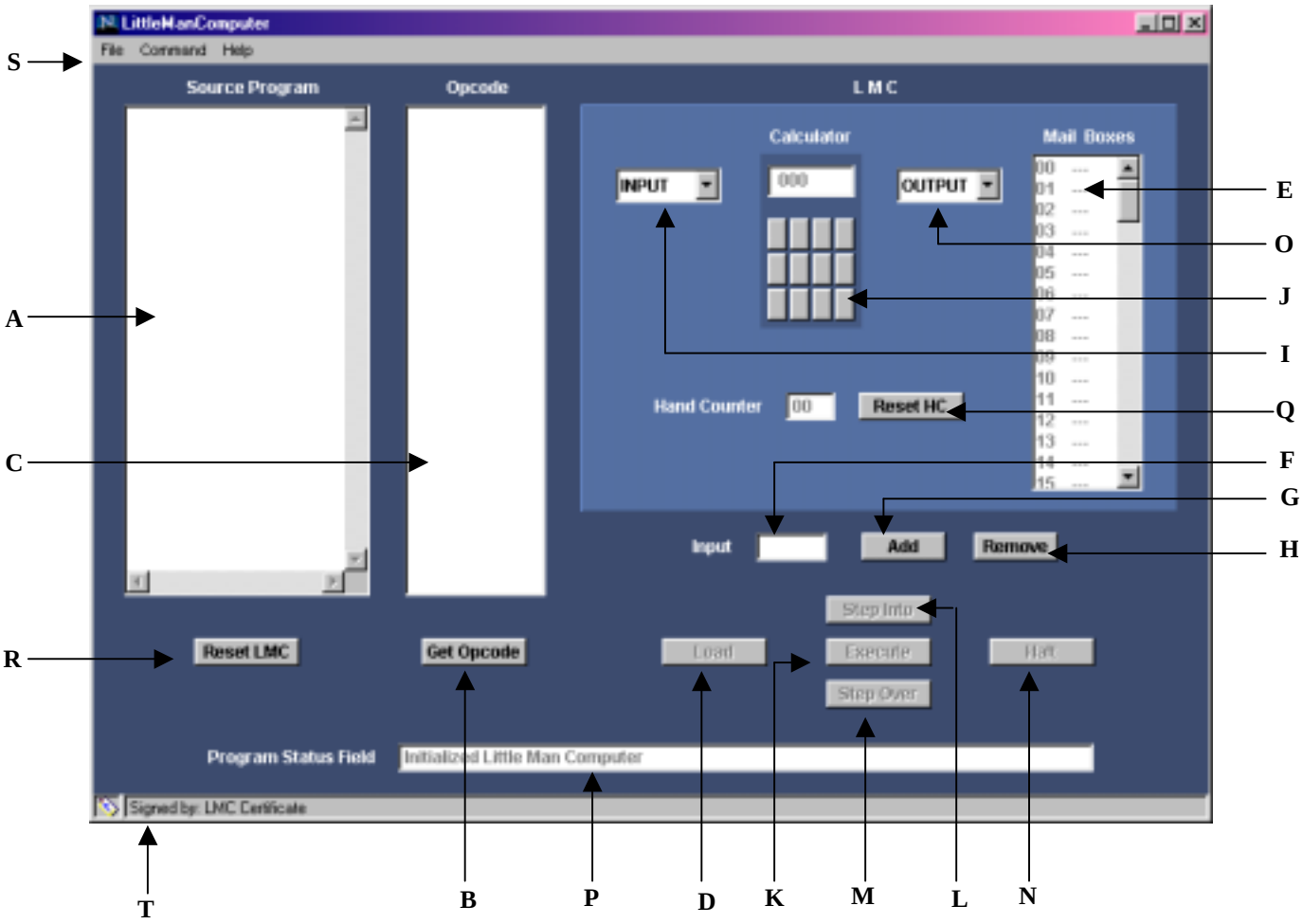
E. User Interface

The following are the general principles used to design the user interface to LMC:

- an intuitive Graphical User Interface (GUI)
- on-line help should be available at all times
- standardized error messages with explanations provided
- white user modifiable fields, gray unmodifiable fields
- equally-sized buttons that are enabled only when required
- no more than five colors on the screen
- screen resolution is 800 X 600 (commonly available)

Our intent is to provide students a visualization of all components within a computer architecture simultaneously, an ability to observe the fetch-execute cycle during program execution, and a highly interactive problem-solving environment with flexible input/output demands.

Below is a screen shot of LMC found at **http://138.87.169.30/LMC/** . The various fields are explained as follows:



A: Editor to write source program or load existing file
B: Assembler, converts mnemonics to machine code
C: if syntax is correct the assembled machine code is here
D: Loader, loads machine code into mailboxes (memory)
E: Mailboxes, 100 from 00 to 99
F: input data here by typing here
G: click here to load input data to Input Box
H: click here to remove highlighted data from Input Box
I: Input Box, contains data entered from F and G
J: Calculator, user cannot interact with calculator buttons
K: Execution: Burst Mode
L: Execution: Step-Into Mode
M: Execution: Step-Over Mode
N: Halt, safety to stop Burst Mode Execution
O: Output Box
P: Program Status Field, contains flags and error messages
Q: Reset Location Counter (or Hand Counter)
R: Clear, clears all fields except source program in A
S: Menu Bar for various operations not shown here including: loading and saving of files, setting breakpoints, help etc.
T: Security, indicates certificate and author of certificate

III. Educational Uses of LMC

A. *von Neumann Architecture*

This LMC simulation is an instance of the von Neumann architecture. We have found that LMC visualization provides an opportunity to examine the concepts underlying this von Neumann architecture.

After introducing digital logic, students understand that a small set of logic components can be combined to implement almost any function (Boolean logic, truth tables, karnaugh maps). The question is then posed to students: if a particular computation is to be performed why not "hardwire" a configuration of logic components. From this students eventually converge to the general purpose configuration of logic functions which can perform various operation depending on control signals applied to hardware.

A next question is posed to students: how shall control signals be applied? The answer is simple but subtle.[5] With direction, students converge to providing a unique code for each control signal and then creating a hardware device which accepts this code and generates control signals.

At this point it can be revealed to students that they have defined software programming as code (instructions)

executed by hardware (interpreting each instruction and generating control signals) without the need for rewiring.

One more component is necessary. While instructions and data can be input, a program is not necessarily executed in the same order as the code is input, there must be a place to temporarily store both instructions and data. Without much help students converge to the concept of memory which can lead to discussion of the memory hierarchy and virtual memory.

B. Addressing Modes

Addressing modes add flexibility and convenience to the programmer without significantly altering the fundamental simplicity of LMC. The Madnick LMC paradigm is limited to direct, absolute addressing but we have extended our LMC simulation to include immediate and indirect addressing (indexed addressing to be added in the near future). Students find these addressing modes allow them to make connections with data structures such as arrays and linked lists using pointers learned in previous algorithm courses.

There are situations where it is both acceptable and convenient to store date within an instruction itself. For these situations we have implemented immediate addressing. In immediate addressing (denoted by # symbol) the operand field of the instruction is the actual value to be acted upon. For example, the instruction LDA #77 loads the value 77 into the calculator. Immediate addressing is not compatible with all instructions. For example, the store instruction cannot use immediate addressing – STA #77 is both ambiguous and incomplete – does it mean store the value 77 somewhere or store some value in mailbox address 77.

Indirect addressing provides a second level of indirection: the operand field of an instruction contains the address of the address of the value to be acted upon. Students found it intuitive to understand this concept as degrees of separation between the instruction and the data to be acted (i.e., six degrees of separation – Kevin Bacon Game http://www.cs.virginia.edu/oracle/bacon_info.html). Usually addressing is one of the more challenging concepts to communicate but the visualization of addressing modes (including immediate and indirect addressing) has made these concepts intuitive such that focus can be placed on more advanced concepts using these underlying techniques.

C. Operating Systems Concepts

LMC also introduces the fundamental concepts of Operating Systems (OS) in several different ways. First, students understand that the external reset for the LMC location counter is similar to the bootstrap process of a real computer accessing a predetermined address in ROM to load the OS kernel. Second, students realize the LMC is a one-trick pony in that it only executes a single program and stops. A real computer would have an operating system allocating resources between different processes and would not stop and require rebooting after the execution of each program. Lastly, the LMC machine code output of the assembler generally assumes a fixed program starting point, memory location 00. Most OS do not assign programs to a particular location, rather the OS assumes that the program is relocatable and assigns it a location that is convenient under the dynamic computer conditions at load time. With accelerated course progress, an assignment may be given to implement this OS operation as an LMC loader program.

IV. Conclusion

We have presented a web-based simulation of a general computer architecture based on the LMC paradigm of Stuart Madnick and showed how LMC can be used as the central model/tool for Computer Architecture and Assembly Language Education (CAALE). We feel that the use of interactive simulation is a powerful tool to enable CAALE active learning and report our teaching techniques here with the hope that others in the CAALE community will use LMC and provide us feedback.

With the success of this LMC simulation of a general computer architecture, we have embarked on simulating more complex computer architectures modeled on real CPUs. Specifically we want students to visualize different ways of implementing the von Neumann architecture based on cost/speed tradeoffs. We have a prototype 8085 CPU simulator in beta test and a VAX11 CPU simulator under development. See the following URL for future contributions: http://www.acs.ilstu.edu/faculty/wjyurci/caale/

V. Acknowledgments

The authors wish to acknowledge the following students who were instrumental in the development of this LMC simulation: Anita Knap who programmed the first LMC prototype, Rahul Gedupudi who programmed this LMC second version of LMC simulation as a Masters Project[2] and continues to maintain and provide upgrades, and lastly the ACS 254 classes at Illinois State University during the Fall 1999 and Spring 2000 semesters who learned along with us the value of active learning using interactive simulation.

References

[1] I. Englander, *The Architecture of Hardware and Systems Software: An Information Technology Approach*, John Wiley & Sons, 1996.

[2] R. Gedupudi, *Simulation of Little Man Computer*, Masters Project Documentation, Department of Applied Computer Science, Illinois State University, February 29, 2000.

[3] L. Beck, *System Software: An Introduction to Systems Programming*, Addison-Wesley, 1985.

[4] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface 2nd* edition, Morgan Kaufmann, 1998.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach 2nd* edition, Morgan Kaufmann, 1996.

[6] W. Stallings, *Computer Organization & Architecture 5th* edition, Prentice Hall, 2000.

[7] M.M. Mano, *Computer System Architecture 3rd edition*, Prentice Hall, 1993.

[8] A. S. Tanenbaum, *Structured Computer Organization 4th edition*, Prentice Hall, 1999.

[9] J. E. Brink and R.J. Spillman, *Computer Architecture and VAX Assembly Language Programming*, Benjamin Cummings, 1987.