Extending the Spreadsheet to Illustrate Basic CPU Operations in a Computer Literacy Course

by
Richard E. Smith, Ph.D.
Department of Quantitative Methods and Computer Science
University of St. Thomas
402 O'Shaughnessy Science Hall
2115 Summit Avenue
St. Paul, Minnesota 55015-1079
Email: resmith@stthomas.edu

Abstract

The computer literacy course at the University of St. Thomas seeks to provide students with a broad background in computing so that they can understand concepts they will encounter in real-world computers, like the meaning of a cycle within a central processing unit (CPU). The behavior of a cycle establishes a CPU's cycle speed, which is often an important discriminator when selecting a computer. Teaching the concept of a CPU cycle has become more difficult as the course has moved away from simple programming languages, like BASIC, towards problem-solving applications like spreadsheets and desktop databases. SimpleCPU is a macro package that uses a spreadsheet to provide an inside view of CPU operation, and it has been used to introduce CPU fundamentals in the computer literacy course. The instructor uses the package interactively to illustrate CPU operation to the class on a large screen. Students use the package individually to write and trace the execution of their own "machine language" programs. SimpleCPU has two advantages over "black box" CPU simulations, like the "Little Man Computer." First, SimpleCPU is integrated into the spreadsheet that is already part of the course curriculum, which simplifies matters for novice computer users as well as the instructor. Second, students can examine any part of the simulation directly by looking in spreadsheet cells and reading the single page of macro definitions. Nothing is hidden in object code or an unfamiliar language. The package can also be used with a computer architecture course, giving students a low-cost way to build and test their own simple processor designs.

Introduction

Introductory computer literacy has always posed a challenge to instructors. Although students generally enter college with a lot of personal computer experience, those students taking introductory literacy courses can still pose a challenge when trying to convey a sense of how computers operate internally. As with many topics, a hands-on approach seems most effective.

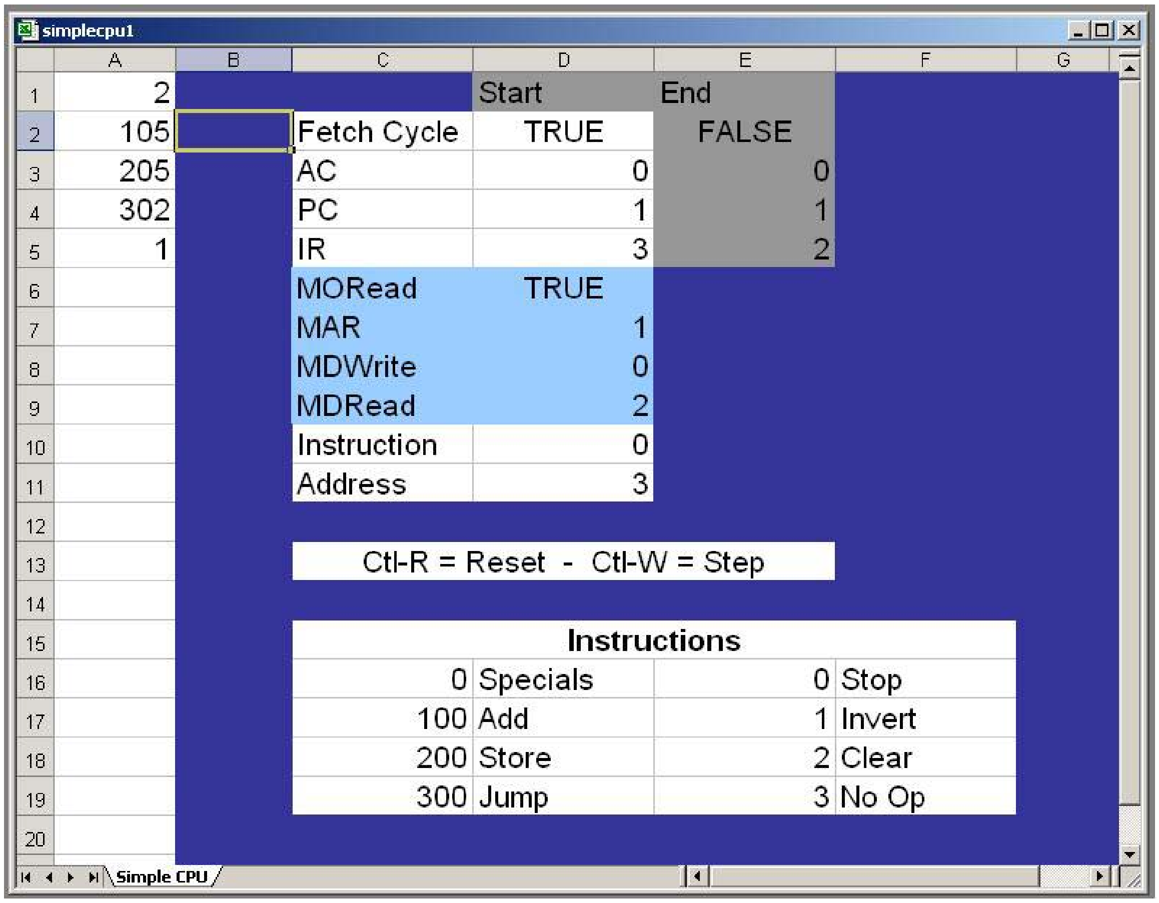
For teaching internal computer operation, many rely on simulations of a computer's central processing unit (CPU). A well-known example is the Little Man Computer (LMC) which is used in at least one computer design textbook [1]. The LMC was originally developed in 1965 by Stuart Madnick at MIT and is generally used as a stand-alone program. This can be a shortcoming in an introductory computer literacy course. A significant number of students in these courses are already intimidated by computer programs and the LMC represents another program to learn. The instructor dilutes the message when the students must both learn how to use the LMC program and also learn what it means.

At the University of St. Thomas we have introduced a CPU simulation called the "SimpleCPU." The simulation operates within a Microsoft Excel spreadsheet. Since Excel is already part of the introductory literacy course, the CPU simulation simply adds on to a program the students have already learned. The SimpleCPU lets students see how the insides of a simple CPU works. There are a small number of "registers" whose contents control what happens. There is the notion of a "processor cycle" by which the CPU takes the steps necessary to execute a computer program. The cycle corresponds directly to the concept behind widely-advertised computer speeds ("the new Pentium operates at a rate of 3.0 GHz"). The SimpleCPU gives the students a way of relating those statistics to a tangible phenomenon. The SimpleCPU also allows students to experiment with programming in a realistic "machine language" represented numerically instead of textually. This combination of capabilities gives the students hands-on experience with essential concepts of computer internals.

The SimpleCPU Architecture

Figure 1 shows the SimpleCPU spreadsheet as it appears to the students. The leftmost column on the spreadsheet, column A, provides the RAM used by the SimpleCPU. Each cell from A1 to A99 represents a single location in RAM. Instructions or data may reside in RAM cells. Each cell has a unique address that is the same as its row number. Since there is no row 0 in a spreadsheet, there is no location 0, either.

The SimpleCPU uses an instruction set that is as simple as possible. At present, there are only seven instructions. As with classic computer instruction sets, the CPU begins executing a program at a selected location in RAM and executes the instructions stored there in sequential order, one after another. A summary of the SimpleCPU instruction set appears at the bottom of the window in Figure 1.



	A	B	C	D	E	F	G
1	2			Start	End		
2	105		Fetch Cycle	TRUE	FALSE		
3	205		AC	0	0		
4	302		PC	1	1		
5	1		IR	3	2		
6			MORead	TRUE			
7			MAR	1			
8			MDWrite	0			
9			MDRead	2			
10			Instruction	0			
11			Address	3			
12							
13			Ctl-R = Reset - Ctl-W = Step				
14							
15			Instructions				
16			0	Specials	0	Stop	
17			100	Add	1	Invert	
18			200	Store	2	Clear	
19			300	Jump	3	No Op	
20							

Figure 1: SimpleCPU spreadsheet with the sample program

Both the *Add* and *Store* instructions contain a reference to a location in memory. The Add instruction retrieves the data stored in the referenced location and adds that data to the contents of the internal accumulator register, called the AC. The Store instruction simply takes the contents of the AC and stores it in the referenced memory location. The *Jump* instruction also refers to a memory location. When the CPU executes a Jump, it changes the location from where it retrieves its instructions, and starts retrieving instructions from the location given in the Jump instruction.

In addition to the Add, Store, and Jump instructions, there are four additional instructions that do not include a memory address. The *Invert* instruction negates the contents of the AC. The *Clear* instruction sets the contents of the AC to zero. The *Stop* instruction causes the CPU to stop executing instructions until it is manually restarted. Since “No-Op” is an abbreviation for “no operation,” the CPU does nothing when it encounters a *No-Op* instruction.

Although this instruction set may still seem too trivial to perform any sort of serious calculation, it is in fact based on a real computer. The computer was called the TX-0, it was finished in 1957, and it was one of the earliest computers to used transistor circuits [2,3]. The SimpleCPU is missing certain features of the TX-0 CPU, but features will be added as required to improve the teaching of introductory CPU concepts. Potential improvements are discussed later in this paper.

The CPU uses two memory cycles to execute each instruction. The first cycle, called the Fetch cycle, retrieves from RAM the instruction to be performed. The second cycle, the Execute cycle, executes that instruction, fetching or storing data in RAM if required. At the end of the Execute cycle, the CPU locates the next instruction so that the next Fetch cycle can retrieve it.

In the SimpleCPU, the operator runs a program by directing the spreadsheet to execute one CPU cycle at a time. Each cycle is performed by executing a macro tied to the Control-W keystroke. The operator can reset and initialize the CPU by running a different macro tied to the Control-R keystroke.

Executing the Sample Program

Below is a list of SimpleCPU instructions that form a very simple program. For as long as that program runs, it doubles the number in storage location 5. The RAM in Figure 1 contains the same program, encoded in the numeric form that the SimpleCPU understands.

1. Clear
2. Add from location 5
3. Store to location 5
4. Jump to location 2
5. 1

Each time the CPU executes instruction 2, it doubles the contents of the accumulator. Each time it executes instruction 3, it saves the doubled number. Each time it executes instruction 4, it jumps back to instruction 2, which repeats the process. Note that the program will never try to treat location 5 as an instruction, since it always jumps back to 2 before reaching location 5.

The SimpleCPU Implementation

Fundamentally, a computer consists of a lot of storage registers (RAM) combined with some calculating logic and some registers to manage the execution of the computer's program. The spreadsheet's cells provide the storage registers needed and, unlike hardware registers, the contents are immediately visible to the operator. The calculating functions of the spreadsheet provide the calculating logic needed both to implement arithmetic instructions like Add and control functions like Jump.

The SimpleCPU simulation implements CPU state and memory modification through the "cycle" macro that is tied to the Control-W keystroke. This macro implements a general-purpose CPU cycle that may include writing data to a location in RAM. To set up the cycle, the spreadsheet looks at its current state as given by the Fetch cycle flag, AC, PC, and IR, located in spreadsheet cells D2 through D4. The spreadsheet calculates new values for each of these, using calculations stored in locations E2 through E4. When executed, the cycle macro performs two steps. One step copies the new values (E2 through E4) into the current values (D2 through D4). The other step writes a value to a location in RAM if required by the current instruction. Both of these steps must be performed by a macro since there is no other way to directly modify spreadsheet cells in this manner.

Experience with Students: Informal Observations

Introductory computer literacy students performed much more successfully on exams that covered CPU concepts after being taught with the SimpleCPU simulation. The students had an opportunity to write machine language programs and watch them execute in a controlled environment. The two-cycle simulation gave them a chance to see what it meant to execute computer operations across multiple steps. Unlike other approaches to introductory sequential programming, the students could see exactly what the computer was doing with their short programs at every step in the process.

Moreover, students curious about the details of CPU operation could directly examine every portion of the simulation. “Next state” calculations were immediately available by examining the appropriate cells. The cycle macro was also directly available for inspection in Visual Basic. Students of computer organization and architecture could even extend the definitions to support additional cycles and more sophisticated instruction sets. While this capability is also available in variants of the SMC [4], the spreadsheet-based simulation usually requires less support overhead since the spreadsheet software is already available to most students.

Potential Improvements

The following improvements would expand the range of programs the students could experiment with:

1. Conditional control flow – there need to be one or more instructions that use the results of calculations to alter the program’s control flow. The TX-0’s Jump instruction was actually a conditional instruction that only jumped if the accumulator contained a negative number. That is one way to do it.
2. Input/output – students find it hard to relate what they see at the CPU level to what they see in large-scale computer behavior, and that involves input/output devices.
3. Address arithmetic – there needs to be a way for programs to compute the addresses of storage locations they use, instead of always storing and fetching from static locations. The TX-0 used self-modifying code, but that’s probably not a good thing to teach in an introductory course.

References

1. Englander, Irv, The Architecture of Computer Hardware and Software Systems: An Information Technology Approach, 2003: John Wiley & Sons, New York.
2. Gilmore, J. T., Jr., and H. P. Peterson, “A Functional Description of the TX-0 Computer,” Memorandum 6M-4789-1, MIT Lincoln Laboratory, Lincoln, MA, October 1958. On-line (retrieved 8 July 2004) at http://bitsavers.org/pdf/mit/tx-0/6M-4789-1_TX0_funcDescr.pdf
3. Bell, C. Gordon, J. Craig Mudge, and John E. McNamara, Computer Engineering: A DEC View of Hardware Design, Digital Press, Bedford MA, 1978.
4. Osborne, Hugh, "The Postroom Computer," Journal on Educational Resources in Computing, Volume 1, Issue 4 (December 2001), pp. 81 - 110.