A spreadsheet-based simulation of CPU instruction execution

Richard E. Smith

University of St. Thomas

*appearing in the **Proceedings of the 2007 ASEE Annual Conference**, Honolulu, HI, June 2007.*

Abstract

The Spreadsheet CPU simulates a central processing unit for teaching purposes. The simulator provides interactive instruction execution like the "Little Man Computer," the LC-3, and other simulators, but it is not a stand-alone program. Instead, it is implemented atop an off-the-shelf copy of the Microsoft Excel spreadsheet. The spreadsheet cells make it easy for students to observe the simulator's internal operation and to modify its operation if necessary. The Spreadsheet CPU was originally used in introductory computer literacy classes to present the concept of the instruction cycle. Since then it has been used to study instruction set design in an undergraduate computer design class.

The Spreadsheet CPU can present CPU design to a broader range of undergraduate students than conventional simulators. Unlike other approaches, the Spreadsheet CPU does not require a background in digital logic or associated design languages in order to modify the CPU's operation or to add or modify instructions. Students can observe all important intermediate results during the cycling of the Spreadsheet CPU, since these results appear in spreadsheet cells. This allows students to study CPU design and perform design experiments without prerequisites in digital design or even in binary arithmetic.

The Spreadsheet CPU simulation is distributed with a Creative Commons license and is freely available for use and modification by interested instructors.

Introduction

The design and operation of a central processing unit (CPU) is a truly fundamental computing concept. Unfortunately, few students study it in detail outside the confines of digital design, electrical engineering, and computer architecture courses. This is primarily because experiments with CPU design require extensive coursework in binary arithmetic, combinatorial logic, digital state machines, and digital design tools such as register transfer languages. These prerequisites prevent a broader range of students from developing the deeper understanding of computing systems that arises from a study of CPU design.

The Spreadsheet CPU makes it possible to introduce elements of basic CPU design and operation to a broader range of students without digital design prerequisites. It was originally developed to present the fundamentals of CPU operation and machine language programming to computing non-majors. Next, it was used to illustrate multi-cycle instruction execution and instruction set design in an undergraduate computer organization course. At present it is being adapted to support a broader range of instruction set design exercises and to illustrate CPU pipelining concepts.

As implied by the name, the Spreadsheet CPU is built atop a standard commercial spreadsheet: Microsoft Excel. This makes the CPU simulation and experimentation much more accessible to undergraduates and other introductory level students. By using a standard application program that is familiar to most students, the Spreadsheet CPU presents students with fewer uncertainties regarding semantics and behavior. Since the spreadsheet's basic number system is decimal, the Spreadsheet CPU is a decimal machine and does not require training in the binary number system. Since spreadsheets use a simple and well-known algebraic notion for their formulas, students can easily study the formulas that implement the CPU's instructions. Students do not need to master Boolean algebra or logic design to examine, analyze, or modify the Spreadsheet CPU. Students can also directly observe intermediate results of instruction execution by examining the cells containing the formulas. Students do not need to learn an esoteric programming interface or register transfer language in order to modify or extend the Spreadsheet CPU: they simply modify the spreadsheet.

CPU Simulations and Spreadsheets

Instructors have relied on CPU simulators for decades to teach machine level instruction operation. In 1965, Stuart Madnick at MIT introduced the Little Man Computer (LMC) as a teaching tool to address this problem. The LMC is a very simple computer designed expressly for teaching the basics of computer operation. Typically, it is implemented as a software simulation instead of in hardware, and the students experiment with the simulation. At least one textbook uses the LMC to introduce computer design (Englander, 2003). Typically the LMC is provided as a stand-alone program that the students can run as a simulation. Other authors have also implemented simulated computers to teach computer operation and processor design: for example, Patt and Patel introduced the LC-2 (now the LC-3) processor as part of their text (Patt and Patel, 2004), and simulators are available for it on line. Patterson and Hennessey (2005) uses a subset of a real world instruction set architecture, the MIPS, and distributes simulators that can execute examples in the text.

These simulators pose shortcomings when being used either to teach computer basics in a computer literacy class or to teach more advanced topics such as instruction set architecture. In many cases, students in computer literacy classes start out with limited computer skills: those with more computer knowledge are more likely to place out of the class or take a more advanced class. When introducing novices to a new computer program, the instructor must teach them the mechanics of using the program and interpreting its output before they can make any use of it. This dilutes the message of the simulator for introductory students.

In computer organization and architecture classes, CPU simulations are intended to illustrate CPU operation in detail. Ideally, the simulations should show both instruction level results in terms of changes to register contents, and intermediate results produced by individual clock cycles. In fact, many simulators, like the LMC and the LC-3, only provide instruction level results.

Moreover, students should be able to modify elements of the CPU's design within the simulator in order to better understand instruction set design and issues of CPU architecture. In practice, many simulators have been implemented using design languages such as VHDL, Verilog, or specialized register transfer languages. At some level of the curriculum, computer architecture students should be working with models of CPUs implemented in standard design languages, and experiment with design changes using simulators; this approach becomes even more practical with the development of open source tools, simulators, and processor designs, like those being promoted by the "opencores.com" project (Goering, 2005).

To simulate a CPU, the environment must be able to implement a sequential machine, which generally requires three components:

- state variables or registers to store the previous state of the system,
- combinatorial logic to calculate the next state from a combination of the previous state and any input signals,
- a periodic signal, like a clock, to tell the machine to update its state based on the outputs of the combinatorial logic.

Digital logic, or a capable software simulation of digital logic, will provide all of these components. The same is true of a typical machine language programming environment, and it's true of its offspring, the procedural programming language.

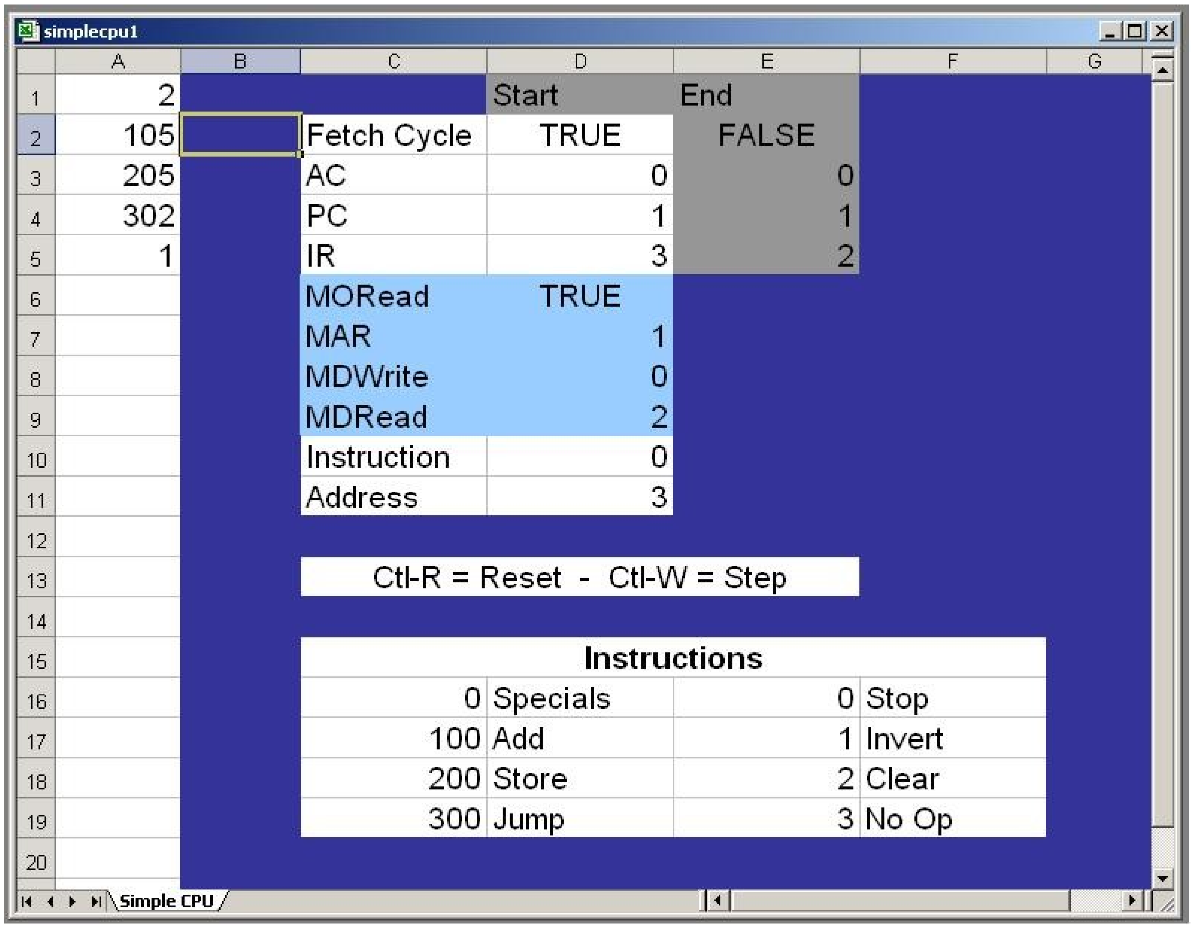
On the other hand, a spreadsheet presents a very different environment. Alan Kay used the term *value rule* to summarize a spreadsheet's operation: a cell's value relies solely on the formula the user has typed into the cell (Kay, 1984). The formula may rely on the value of other cells, but those cells are likewise restricted to user-entered data or formulas. There are no "side effects" to calculating a formula: the only output is to display the calculated result inside its occupying cell. There is no natural mechanism for permanently modifying the contents of a cell unless the user manually modifies the cell's contents. In the context of programming languages, this yields a limited form of first-order functional programming (Walpole Djang and Burnett, 1998; Burnett et al, 2001; Hannah, 2002).

Superficially this might seem to provide the raw materials to simulate a processor. Spreadsheet formulas can implement the combinatorial logic needed to derive the CPU's next state. Spreadsheet cells can serve as registers, or simulate an entire random access memory (RAM). However, Kay's rule gives us no way of permanently modifying a cell, so we have no way of "latching" the combinatorial outputs that define the CPU's next state. It might be possible to implement CPU cycles using circular cell references (i.e. using a formula that directly or indirectly refers back to the cell it resides in), just like feedback from two gates are used to implement a flip flop. Unfortunately, circular references are not implemented in a way that yields a reliable mechanism. The Spreadsheet CPU solves this problem by falling back on procedural programming. The Spreadsheet CPU uses Microsoft Excel's built-in Visual Basic macro facility to implement storage registers, RAM operations, and CPU cycle execution.

The Spreadsheet CPU was originally developed as an example for a computer literacy course at the University of St. Thomas, QMCS 110, as described in an earlier paper (Smith, 2005). Later it was extended to illustrate CPU operation and instruction set design for a computer organization course, QMCS 300. Several variants exist, each with a different set of instructions, internal registers, and processor cycles or cycles.

The "Literacy Version" of the Spreadsheet CPU

Figure 1 shows the version of the Spreadsheet CPU used by computer literacy students. The leftmost column on the spreadsheet, column A, provides the RAM used by the Spreadsheet CPU. 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.



	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: Spreadsheet CPU, literacy version, with a sample program

The basic Spreadsheet CPU uses an instruction set with 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. A summary of the Spreadsheet CPU instruction set appears at the bottom of the window in Figure 1.

CPU instructions in the literacy version consist of three decimal digits. The first digit indicates the type of operation. The lower two digits indicate a storage address for memory reference instructions. In Table 1, the two letters “xx” represent two-digit RAM addresses used in memory reference instructions.

Although this instruction set may 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 (Mitchell and Olsen, 1956; Gilmore and Petersen, 1958).

In the literacy CPU, cycles are organized around memory cycles, and each instruction has two cycles. The first cycle is a conventional Fetch cycle that 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 updates the program counter to point to the next instruction so that the next Fetch cycle can retrieve it.

Instruction	Code	Example Text	Example Code
Add xx	1xx	Add 23	123
Store xx	2xx	Store 23	223
Jump xx	3xx	Jump 5	305
Stop	0	Stop	0
Invert	1	Invert	1
Clear	2	Clear	2
Noop	3	Noop	3

Table 1: Spreadsheet CPU Instruction Set, literacy version

The actual implementation consisted of ten spreadsheet formulas stored in cells E2:E5 and D6:11. Cells D6 through D9 controlled RAM access as follows:

- MAR – calculates the RAM address to reference during this cycle. During Fetch, it uses the PC, during Execute it uses the address embedded in the fetched instruction.
- MORead – a flag that, if true, indicates a RAM read cycle
- MDRead – formula that retrieves the data stored in the address indicated by the MAR
- MDWrite – formula that calculates the data to be stored during a write operation.

The cells E2:E5 implemented most of the instruction fetch and execution. Their formulas worked as follows:

- Fetch flag: cycled between True and False to go between Fetch and Execute cycles
- AC: updates its value during the Execute cycle of the Add, Invert, and Clear instructions.
- PC: increments itself at the end of each Fetch cycle. During the Execute cycle, the PC decrements itself for a Halt and loads a new address from a Jump instruction.
- IR: retrieves the instruction from RAM (via MDRead) during the Fetch cycle

A Visual Basic macro initialized the processor to begin execution by setting the AC and IR to zero, the starting PC to 1, and the cycle to Execute. The macro to "run" the simulation would step the processor forward a single cycle, based on the contents of the previous cycle registers. If no RAM store operation need occur, then cycle changing is simply a matter of copying the "End" register values (E2:E5) into the "Start register values (D2:D5). If the MORead flag is False, then the macro must also perform a RAM data store operation. The macro uses the contents of the MAR to index into column A of the spreadsheet, and writes the contents of MDWrite into the selected RAM location.

The resulting Spreadsheet CPU is very simple yet it fully captures the operation of a CPU. Execution relies on formulas stored in ten spreadsheet cells combined with less than 20 statements in Visual Basic. Students can experiment with machine language programming using a decimal notation and a simple set of instructions. They do not need to observe or modify the Visual Basic software. To practice with machine language, they do not need to modify any of the spreadsheet except for the leftmost column that represents the RAM contents.

Classroom Use

The Spreadsheet CPU has been used in two different courses. These experiences are summarized below.

Computer Literacy Class

One of the requirements of the department's computer literacy course was to ensure that students thoroughly understood the concept of CPU cycles and instruction execution cycles. Students are constantly bombarded by references to cycle times in computer advertisements ("Our new Pentium-based system operates at 2.66 gigahertz"). Initially, the recommendation was to drill the students in the details of processor instruction cycles so that they would understand in detail what the concept meant. The Spreadsheet CPU was originally developed to give students a hands-on approach to working with processor cycles.

Introductory computer literacy students performed much more successfully on exams that covered CPU concepts after being taught with the Spreadsheet CPU 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.

Computer Organization Class

Experience with the Spreadsheet CPU in computer literacy suggested that the technique could be adapted for teaching computer organization and instruction set design. In the process of adaptation, the Spreadsheet CPU morphed into several versions:

- Two cycles with additional arithmetic and branch instructions
- Two cycles with an index register (IX)
- Three cycles with indirection

These implementations made better use of the spreadsheet environment to implement separate elements of the instruction set. Instructions were assigned to separate columns, and processor cycles were assigned separate rows. Thus, there was an individual cell for each formula to calculate the contents of a particular register executing a particular instruction at the end of each cycle. This arrangement made it simple to add instructions by adding another column to the right of the spreadsheet.

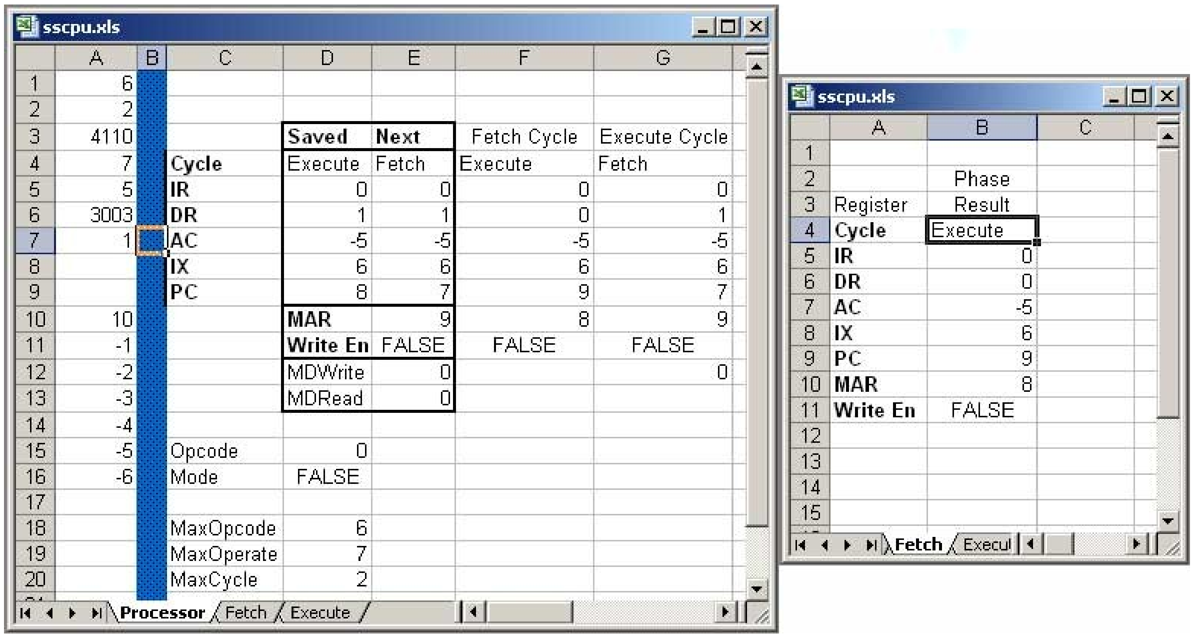
In the class, students wrote sample machine language programs and reported on intermediate results of instruction operations they observed within the spreadsheet. The index register made it practical to use a broader range of programming examples: the restrictions of absolute addressing were too challenging for novice machine language programmers. The differences required between the two-cycle and three-cycle CPUs helped clarify the concept of indirect addressing. Students also explored the process of adding new instructions to the CPU.

Unfortunately, it was difficult to modify instruction cycles in these spreadsheets. An important design objective was to eliminate any need for the students to have to modify the Visual Basic macros to change the machine. Since both cycles and registers were represented with rows, changes to cycles required changes to the macros. Adding a cycle required moving all registers apart by an additional row. Since register values had to be explicitly addressed in the Visual Basic macros, the cycle change required a macro change.

Improved Spreadsheet CPU

Figures 2 and 3 show a recent version that Spreadsheet CPU that incorporates an index register and fourteen instructions. This version has also eliminated the need to modify Visual Basic macros when adding or changing instruction cycles. Instead of implementing cycles in rows parallel to register contents, this CPU implements separate instruction cycles in separate “worksheets” within the same spreadsheet workbook.

Figure 2 shows two of the three worksheets appearing in the CPU workbook file. The first worksheet, titled “Processor,” contains the processor registers and the addressable RAM, similar to Figure 1. As before, the leftmost column A contains the addressable RAM. The next column is left blank. The right hand columns provide the CPU simulation. The rightmost columns contain the formulas needed to compute new register values for the different processor cycles.



sscpu.xls

	A	B	C	D	E	F	G
1	6						
2	2						
3	4110			**Saved**	**Next**	Fetch Cycle	Execute Cycle
4	7		**Cycle**	Execute	Fetch	Execute	Fetch
5	5		**IR**	0	0	0	0
6	3003		**DR**	1	1	0	1
7	1		**AC**	-5	-5	-5	-5
8			**IX**	6	6	6	6
9			**PC**	8	7	9	7
10	10			**MAR**	9	8	9
11	-1			**Write En**	FALSE	FALSE	FALSE
12	-2			MDWrite	0		0
13	-3			MDRead	0		
14	-4						
15	-5		Opcode	0			
16	-6		Mode	FALSE			
17							
18			MaxOpcode	6			
19			MaxOperate	7			
20			MaxCycle	2			

Processor / Fetch / Execute

sscpu.xls

	A	B	C
1			
2		Phase	
3	Register	Result	
4	**Cycle**	Execute	
5	**IR**	0	
6	**DR**	0	
7	**AC**	-5	
8	**IX**	6	
9	**PC**	9	
10	**MAR**	8	
11	**Write En**	FALSE	
12			
13			
14			
15			

Fetch / Execu

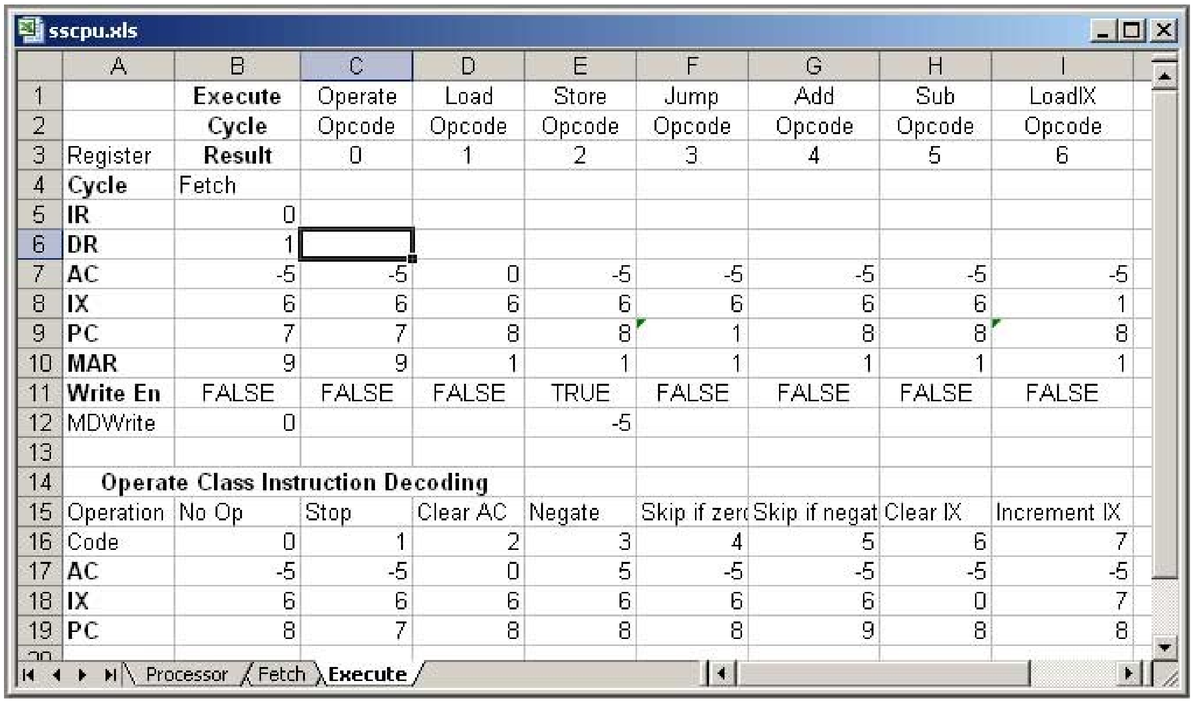
Figure 2: Spreadsheet CPU: Processor and Fetch cycle worksheets

This CPU incorporates two additional registers beyond the four in the literacy version:

- DR – data register that stores partial results while calculating the actual storage address being referenced in an instruction
- IX – index register for indexed addressing of RAM (optional)

As with the literacy version, this CPU contains two copies of each of these registers, called Saved and Next in this version. The Saved copy is used by formulas in the spreadsheet to calculate the results of the current instruction cycle. The calculated result appears in the Next copy of the same register. The CPU moves from one cycle to the next by copying the contents of the Next cell into the Saved cell for each of the six registers. As before, a Visual Basic macro performs the copy operation and also implements RAM “store” operations when required.

The right hand columns titled “Fetch” and “Execute” calculate register values for those two cycles. The “Next” value for each register is set to the value in the column corresponding to the next cycle. The column values are in turn copied from each cycle's worksheet. The worksheet for the Fetch cycle appears to the right of the Processor sheet. In the current version, the Fetch cycle is fairly simple, but it was placed in its own worksheet for consistency. Moreover, it may be necessary to add columns to the Fetch cycle to implement address modes like indirection.



	A	B	C	D	E	F	G	H	I
1		**Execute**	Operate	Load	Store	Jump	Add	Sub	LoadIX
2		**Cycle**	Opcode	Opcode	Opcode	Opcode	Opcode	Opcode	Opcode
3	Register	**Result**	0	1	2	3	4	5	6
4	**Cycle**	Fetch							
5	**IR**	0							
6	**DR**	1							
7	**AC**	-5	-5	0	-5	-5	-5	-5	-5
8	**IX**	6	6	6	6	6	6	6	1
9	**PC**	7	7	8	8	1	8	8	8
10	**MAR**	9	9	1	1	1	1	1	1
11	**Write En**	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
12	MDWrite	0			-5				
13									
14	**Operate Class Instruction Decoding**								
15	Operation	No Op	Stop	Clear AC	Negate	Skip if zerc	Skip if negat	Clear IX	Increment IX
16	Code	0	1	2	3	4	5	6	7
17	**AC**	-5	-5	0	5	-5	-5	-5	-5
18	**IX**	6	6	6	6	6	6	0	7
19	**PC**	8	7	8	8	8	9	8	8

Figure 3: The Instruction Execute Cycle

Calculations of worksheet cells are straightforward. During the Fetch cycle, the next value for the IR will be the contents of the RAM location indicated by the PC; otherwise the IR retains its previous contents. PC contents are generally preserved, except to be incremented for the next instruction or replaced during a Jump instruction.

Figure 3 shows the worksheet for the Execute cycle, indicating the tentative results for executing all of the CPU's instructions. The upper rows 1 through 12 decode and interpret the opcode digit of the instruction. If the opcode is a zero, that indicates an “operate class” instruction whose behavior is defined in the lower table (rows 14 through 19). An operate class instruction will only affect three registers.

Conclusions

By using a spreadsheet as the platform for implementing a CPU simulation, the following benefits were achieved:

- Novice computing students gained access to a CPU simulation and performed simple exercises in a simulated machine language. This improved their comprehension of basic computer architecture concepts.
- Computer organization students gained access to a CPU simulation that allowed them to write machine language programs in several similar but slightly different instruction sets. They were able to observe the internal operation of instruction calculations and make

minor modifications to instructions.

The following issues appeared while using the Spreadsheet CPU:

- The TX-0 relied exclusively on absolute addressing, and this is too restrictive for implementing more sophisticated machine language programming exercises. This problem was fixed by introducing an index register.
- It was difficult for computer organization students to make significant modifications to the CPU because there was too strong of an interdependence between the arrangement of cycles and the implementation of the Visual Basic macros that drove the simulation. The implementation was modified to eliminate that interdependence.

Work continues on the Spreadsheet CPU as described in the next section.

Status and Future Plans

To provide a good basis in introductory computer organization and instruction set design, a CPU simulator should provide a minimum of the following:

- An instruction set sophisticated enough to support realistic machine language programming exercises. This was a fundamental shortcoming of the literacy version. While the current instruction set remains unsophisticated, it can be easily extended to support a broad range of programming exercises.
- An environment to simulate instruction execution and display the results. The Spreadsheet CPU simulates execution and provides a detailed view of the CPU's inner workings during execution.
- A simple mechanism to modify the processor architecture for student exercises. The Spreadsheet CPU provides this capability.
- Modern architectural features like pipelining and caches. Work is underway to implement such features.

To support pipelining, the Spreadsheet CPU must go through certain changes. First, the instruction set should be overhauled to better fit the needs of a pipelined processor (Patterson and Hennessy, 2005). This will probably involve adding general registers to store intermediate arithmetic results as well as changes to instruction types and layouts. Second, the spreadsheet needs a way of handling pipeline registers. A simple approach might be to insert the registers into each cycle's worksheet. This will allow sheets to retain intermediate instruction results from one cycle to the next. However, this makes modifications more difficult, since changes to registers generally involve changes to the Visual Basic macros. Alternatively, the CPU might centralize all registers on a single worksheet, and the Visual Basic macros may be modified to dynamically detect the number of registers to update.

Acknowledgments

This work was funded in part by a new faculty grant from the University of St. Thomas in St. Paul, Minnesota. The author would also like to thank the students in the QMCS 110, QMCS 200, and QMCS 300 courses for their suggestions and feedback regarding the Spreadsheet CPU.

Bibliography

Burnett, M., Atwood, J., Walpole Djang, R., Reichwein, J., Gottfried, H., and Yang, S. 2001. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *J. Funct. Program.* 11, 2 (Mar. 2001), 155-206.

Englander, Irv, *The Architecture of Computer Hardware and Software Systems: An Information Technology Approach*, 2003: John Wiley & Sons, New York.

Everett and Swain, “Whirlwind I Computer Block Diagrams,” Report R-231, MIT Servomechanisms Laboratory, 1946.

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

Goering, Richard, “Doors 'open' to hardware,” *EE Times*, 6 June 2005.

Hanna, K. 2002. Interactive visual functional programming. In *Proceedings of the Seventh ACM SIGPLAN international Conference on Functional Programming* (Pittsburgh, PA, USA, October 04 - 06, 2002). ICFP '02. ACM Press, New York, NY, 145-156.

Kay, A., “Computer Software,” *Scientific American* 251(3), September 1984, 52-59.

Mitchell, J. L., and K. H. Olsen, “TX-0: A Transistor Computer,” *AFIPS Conf Proc. EJCC 10*, 1956, pp. 93-101.

Patt, Yale, and Sanjay Patel, *Introduction to Computing Systems: from bits to C and beyond*, second edition, Boston: McGraw Hill, 2004.

Patterson, David, and John Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, third edition, San Francisco: Morgan Kaufmann, 2005.

Smith, R.E., “Extending the Spreadsheet to Illustrate Basic CPU Operations in a Computer Literacy Course,” 2005 ASEE North Midwest Regional Conference, Milwaukee, WI, October, 2005.

Walpole Djang, Rebecca, and Margaret M. Burnett, "Similarity Inheritance: A New Model of Inheritance for Spreadsheet VPLs," p. 134, IEEE Symposium on Visual Languages, 1998.