The Knob & Switch Computer: A Computer Architecture Simulator for Introductory Computer Science

GRANT BRAUGHT
Dickinson College
and
DAVID REED
Creighton University

The Knob & Switch Computer is a computer architecture simulator designed to teach beginning students the basics of computer organization. It differs from existing simulators in two significant ways: (1) it incorporates "cognitive hooks" in the form of knobs and switches that encourage exploration and discovery on the part of the student, and (2) it can be presented one component at a time, starting with a simple interactive data path and building incrementally to a full-featured stored program machine. Both of these features make it possible to engage beginning students and effectively convey an understanding of how computers work. The Knob & Switch Computer Simulator can also motivate the study of other computing topics such as data representation, assembly language programming, and RISC vs. CISC architectures. In addition to describing the Knob & Switch Computer, experiences using the simulator in breadth-based introductory courses both at Dickinson College and Creighton University are discussed.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization – General**]: Modeling of Computer Architecture; I.6.5 [**Simulation and Modeling**]: Model Development; K.3.1 [**Computer Uses in Education**]: Computer Uses in Education
General Terms: Design
Additional Key Words and Phrases: computer architecture simulator, education, knob & switch

---

1. INTRODUCTION

Educators in computer science have long debated the relative advantages of breadth vs. programming depth in introductory computer science courses. The adoption of introductory courses that provide broad perspectives on computer science was strongly advocated by the IEEE/ACM Computing Curricula 1991 [Tucker 1992] and similar reports [Foley & Standish 1988] in the late 1980's and early 1990's. In recent years, the breadth-first approach has received renewed attention, partially due to the emergence of the World Wide Web as a unifying theme [Gurwitz 1998, Reed 2001a, Reed 2001b] and also due to the perceived need for all citizens to be fluent with computer technology and its capabilities [NRC Committee on IT Literacy 1999]. As further evidence, the IEEE/ACM Computing Curricula 2001 includes a breadth-first option in its proposed curricular models, describing both a stand-alone non-majors course (commonly referred to as CS0) and a breadth-based course that can be integrated into a three course introductory sequence [CC2001 Task Force 2001].

Authors' addresses: Grant Braught, Department of Mathematics and Computer Science, Dickinson College, Carlisle, PA 17013. braught@dickinson.edu; David Reed, Department of Mathematics and Computer Science, Creighton University, Omaha, NE 68178. davereed@creighton.edu

A potential drawback of the breadth-first approach is that students may have limited experience with computers, and thus have little context in which to place computing concepts. This is especially true with respect to hardware issues and the underlying organization of computers, as these areas are least likely to be familiar to beginning students. Clearly, an understanding of the basic components of computers (e.g., CPU, datapath, memory), as well as their capabilities and limitations, is essential to understanding the field of computer science and also serves students as consumers of computing technology. However, the amount of technical detail involved is often overwhelming for students. Even if they are able to memorize the components of a computer and their roles, a deeper understanding of why modern computers are organized as they are is often elusive.

2. ARCHITECTURE SIMULATORS

A promising approach to the presentation of computer organization at the introductory level involves the use of computer architecture simulators. Unlike their expensive and complex real-world counterparts, simulators are able to present cost-effective, simplified models of computers. In addition, simulators encourage active learning by allowing students to interact with the simulated components and observe their behavior. This paper describes one such simulator, the Knob & Switch Computer, developed in 1998 by Grant Braught for a breadth-based CS0 course at Dickinson College [Braught 2001]. Like many other simulators targeted at introductory students (e.g. [Biermann 1997], [Decker & Hirshfield 1998] [Sample and Arnold 1997], [Arias and Garcia 1999], [Pastor et al. 1999], and [Yurcik and Brumbaugh 2001], see [Yurcik et al. 2001] for a survey.) the Knob & Switch (K&S) Computer presents a graphical representation of the computer's components and illustrates the flow of data between them using animation. However, despite these similarities the K&S Computer supports a significantly different approach to introducing the concepts of computer architecture than is supported by other simulators.

The approach taken by the majority of introductory architecture simulators is to use graphics and animations to illustrate how the computer hardware processes machine or assembly language instructions. The PIPPIN machine simulator [Decker & Hirshfield 1998] and the Little Man Computer (LMC) simulator [Yurcik and Brumbaugh 2001] are representative of this approach. With PIPPIN, a machine language program and data are loaded (or entered manually) into PIPPIN's main memory. PIPPIN's datapath is then animated to illustrate the flow of control and information as each machine language instruction is fetched and executed. With the Little Man Computer (LMC), an assembly language program is first assembled into machine language and is then loaded into the LMC's main memory. Each machine language instruction is then fetched by an imaginary Little Man who executes it by using the computer's components. While this approach effectively demonstrates the execution of machine or assembly language instructions, it is most effective when these types of instructions as well as the computer's components have a real and significant meaning to the students. While this is likely to be the case in a sophomore level computer organization course [Scragg 1991], it typically is not true in an breadth-based introductory course.

The K&S Computer was designed specifically to address the above concern, and it does so through the use of two novel features: (1) the K&S is configured via knobs and switches which provide "cognitive hooks" that encourage exploration and discovery on the part of the student, and (2) the K&S can be presented one component at a time, starting with a simple interactive data path and building incrementally to a full-featured stored program machine. Through the combination of these two features a student's introduction to the K&S Simulator begins with a simple datapath that is controlled through the intuitive actions of turning knobs and opening and closing switches. The "cognitive hook" provided by the knobs and switches makes it possible to engage beginning students, and the incremental construction of a

complete computer effectively conveys both an understanding of how computers work and why they work the way that they do.

In addition to its knobs and switches and its incremental development, the use of the K&S Computer has several other benefits as compared to other simulators. The majority of the simulators referenced above use a single accumulator design typical of older CISC style architectures. The K&S datapath however, contains a register bank allowing it to implement the register-to-register operations typical in modern RISC style load-and-store architectures. Also, the K&S simulators are written entirely in HTML and JavaScript, which has several advantages for an introductory course. The Web interface utilized by the simulator is already familiar to most students, and thus less intimidating to beginners. Knobs and switches are represented as images, changeable via a simple click of the mouse. Individual components of the K&S Computer (e.g., datapath, main memory, and control unit) are presented in separate frames, allowing for the addition of new components while leaving existing components unchanged in their own frame. The simulator will run using any JavaScript-enabled Web browser, with no proprietary tools or environments required. Thus, students may be encouraged to access and experiment with the simulator regardless of their physical location or computer platform. If desired, the instructor may even download the source code for modification[1] and local execution. The simulator as presented in the following sections has been successfully adopted in courses at Dickinson College, Creighton University, and at other institutions for use in breadth-based introductory courses and also in several computer organization courses

3. THE KNOB & SWITCH COMPUTER SIMULATOR

The defining feature of the Knob & Switch (K&S) Computer Simulator is its innovative use of knobs and switches as "cognitive hooks" that encourage student interaction. Students are already familiar with the use of knobs (which can be set to numerous positions) and switches (which can be toggled) to control real-world devices such as televisions and radios. The K&S Computer builds upon this familiar metaphor to enable students to directly manipulate the components of the simulator. For example, by clicking the mouse on knobs associated with the register bank, students select the registers from which operands are taken and to which results are stored. Likewise, by clicking on switches, students open or close connections that direct the flow of data between the datapath and main memory. In all of these interactions, each student action has an immediate and visual effect on the configuration of the hardware settings within the computer. Thus, students are encouraged to explore the behavior of the different hardware components by experimenting with different settings and observing the results. The familiar knob and switch interface provides a cognitive hook that increases student confidence, encourages exploration, and provides a framework for actively building an understanding of the workings of a computer. (See [Scragg 1991] for additional discussions on cognitive hooks in computer organization.)

The other significant innovation provided by the K&S Computer Simulator is its modular, incremental design. In its final form, the K&S Computer models a complete stored-program computer with its own microprogramming and machine languages. However, the complexity of this complete model prevents it from being immediately accessible to beginning students. To provide a gentler introduction to the internal workings of a computer, the K&S Computer was designed to allow for an incremental presentation. The first component, a simple datapath, may be introduced in isolation. Through direct manipulation of knob and switch settings, students can configure the hardware within the datapath, observe its behavior, and understand the role of registers and the ALU without the added complexity of other hardware components. The limited number of registers in the datapath leads naturally to the next

[1] The Knob & Switch source code is released under the GNU General Public License. (See http://www.gnu.org/copyleft/gpl.html)

increment of the K&S Computer, which adds a separate main memory component. Building upon their understanding of the datapath, students can focus on the role of memory and its integration with the existing datapath. The idea of encoding knob and switch settings using the 0's and 1's that computers "understand" motivates the third increment of the K&S Computer, which adds a simple control unit for microinstructions. The limitations of microprogramming and a two-memory layout motivate a single-memory, stored-program architecture with machine language programming. Finally, programmer convenience and efficiency motivate the development of an assembly language. While the end result is an understanding of a full-featured, stored-program computer, the incremental development of the computer allows students to focus on new features as they are introduced and build their understanding gradually. And since each increment addresses a specific issue or shortcoming of the previous increment, students not only understand how the individual components work but also gain insight as to why the computer works in the way that it does.

The following sections describe each increment of the K&S Computer, starting with the interactive datapath and building to the fully programmable computer. For each increment, the important concepts introduced by that increment are discussed, as well as factors that motivate the next increment. Experiences using the simulator in breadth-based introductory computer science courses are described, including sample exercises that have been assigned to encourage student exploration.

3.1 K&S Datapath Simulator

The first increment of the K&S Computer is an interactive datapath simulator. The datapath consists of a bank of four registers, a simple arithmetic logic unit (ALU), and buses that connect the two. Students are able to enter decimal numbers directly into the registers and control the flow of data from the registers, to the ALU, and back into a register using clickable knobs. Clicking the mouse on the image of a knob turns it clockwise, allowing students to select the desired register. For example, suppose a student wanted to simulate the addition of two numbers. They must enter the numbers directly into the text boxes corresponding to registers, say register 0 (R0) and register 1 (R1), and select those registers by clicking on the knobs so that the arrow labeled A Bus Address points to R0 and arrow labeled B Bus Address points to R1. Similarly, they must select the addition operation in the ALU by clicking on the knob until it points to A+B, and also select a destination for the result by clicking on the knob labeled C Bus Address. Figure 1 shows the datapath configured to add 43 and -296, stored in registers R0 and R1, and place the result in R2.

Clicking the execute button within the datapath simulator causes one complete machine cycle to be performed. As the datapath performs the operation, it is animated to show how data moves through the computer (the animation speed can be controlled for more careful study). The knob metaphor is a natural one to students, and provides an immediate "cognitive hook" for interacting with the simulation. Students are encouraged to experiment with the machine and are provided with exercises such as the following:

- Describe the settings that would result in the contents of R2 being doubled.
- Describe the settings that would result in a 0 being placed in R3.
- How many cycles would be required to add the contents of R0, R1, and R2 and place the sum in R3? Describe the settings for each cycle.
- How many cycles would be required to negate the contents of R1? Describe the settings for each cycle.

One immediate consequence of experimentation with the datapath is that students develop a solid understanding of clock speed. While many texts and articles describe clock speed in terms of the number of "operations" per second that a computer can perform, the actual

definition of an "operation" is usually left vague. Students often have no idea exactly what is meant by the term, or else mistakenly assume that it includes any single computation. After experimenting with the datapath, a definition of clock speed as number of datapath cycles per second is clear to students. The fact that a 1 GHz processor can perform 1 billion such datapath cycles in a second is understandable although still astounding to most students. Plus, by recognizing that different CPUs have different numbers of registers and different ALU operations, and thus support different capabilities in a single datapath cycle, students reach the conclusion that clock speed alone is not a valid means of comparing the performance of different CPUs.

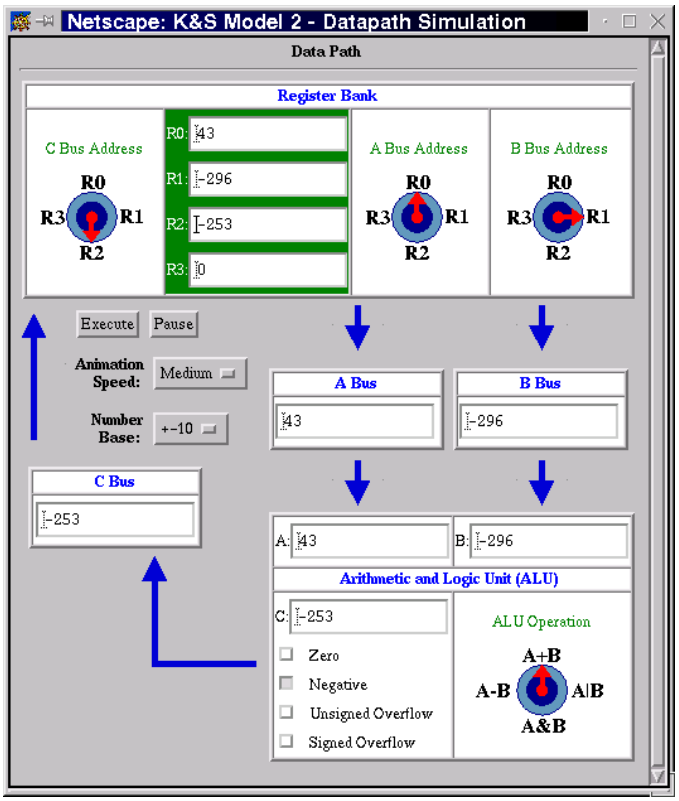

Figure 1. K&S Datapath Simulator

Perhaps more noteworthy than what they learn from the datapath simulation is what students observe about its limitations. After experimenting with the simulator for a period of time and answering questions such as the ones above, students typically have two complaints: “It can only store 4 numbers!” and “It’s not programmable!” The addition of components for external data storage (Section 2.2) and microprogramming (Section 2.3) are thus directly motivated by student experience.

3.2 K&S Datapath Simulator with Main Memory

The next increment of the K&S Computer adds a Main Memory component that provides a separate storage area for more data. This main memory consists of 32 memory locations, numbered from 0 to 31. Similar to the registers in the datapath, students can enter numbers directly into the memory locations. A read/write radio button is provided to allow students to select a particular memory location. Likewise, switches (in the form of clickable images) are added to the datapath between the C Bus and main memory, between the ALU and the C Bus and between the C Bus and the Register Bank to control the flow of data. By clicking on the

switches, students can set the buses so that data is loaded from memory into a register, or so that the output of the ALU is stored directly in memory. For example, suppose a student wanted to perform an operation on two numbers stored in registers, as in the previous example, but store the result in main memory instead of a register. As before, the numbers must be entered into the registers and the knobs controlling the A Bus, B Bus, and ALU Operation must be set appropriately. In addition, the student must click on the C Bus switches so that the switches connecting the ALU output to the Main Memory are closed, while the other two switches are open. Finally, the read/write radio button next to the desired memory location must be selected. Figure 2 shows the datapath and main memory configured to subtract 72 from 892, stored in R0 and R1, and store the result in memory location 1.

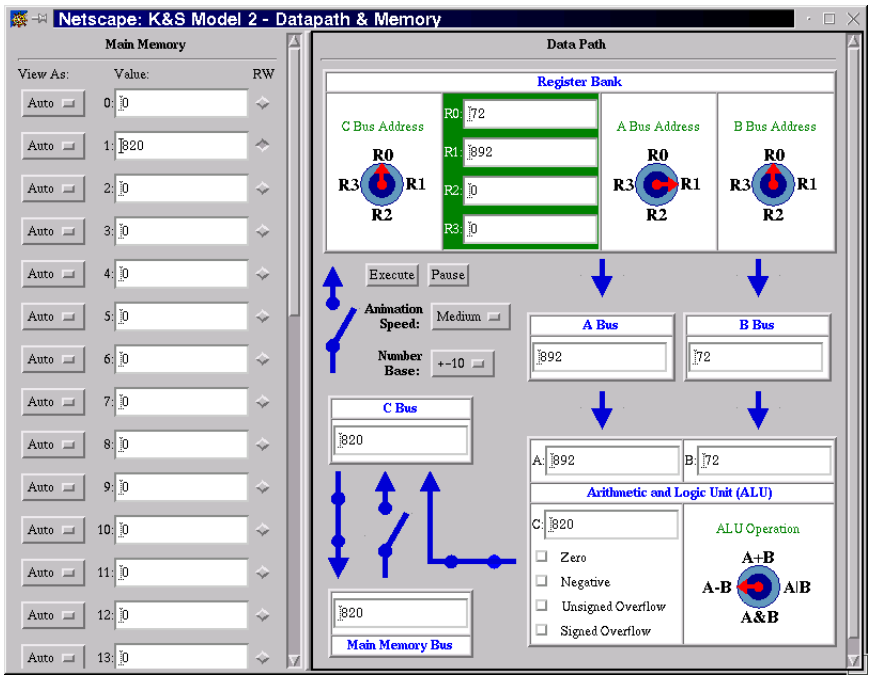

Figure 2: K&S Datapath Simulator with Main Memory.

Once again, interaction with the simulator is intuitive and visually clear to the students. Clicking on a switch causes it to open and close, and students are encouraged to try various settings to see how memory can be integrated with the datapath. In particular, exercises that involve the transfer of data between memory locations and the datapath itself, such as the ones listed below, are provided to encourage exploration.

- Describe settings that would result in the contents of memory location 4 being copied into register R0.
- Describe settings that would result in the contents of register R0 being copied into memory location 4.
- How many cycles would be required to copy the contents of memory location 5 into memory location 6? Describe the settings for each cycle.
- How many cycles would be required to add the contents of memory locations 0, 1, and 2 and store the result in memory location 3? Describe the settings for each cycle.

The second question above, how to move a value from a register to a memory location, raises interesting questions for the students. Typically, the settings for the register knobs and

the bus switches are easy enough, but passing a value through the ALU unchanged requires some thought. Most students will come upon the idea of adding zero to the register (or perhaps subtracting zero), and may even suggest generating the zero value by subtracting a register from itself. Those who have explored the behavior of the bitwise AND (&) and OR (|) operations may also propose these operations for passing a value through the ALU unchanged. Having students compare their answers to exercises such as these can reinforce their understanding and motivate the upcoming increments of the simulator. For example, observing a fellow student struggle to interpret an English description of datapath settings may suggest the need for a more precise notation (e.g., binary microinstructions).

Beyond introducing main memory, this second increment of the K&S Computer introduces several other important ideas. First, it has a load-and-store architecture in the tradition of modern machines. Second, many things in the simulation happen in parallel and many operations are performed even though their results are never used. For example, when reading a value from memory the ALU performs an operation even though the result is discarded (due to the open switch between the ALU and the C Bus). Similarly, memory is read on every machine cycle but its contents are only transferred into a register if the switches are set appropriately. The idea that computers do a significant amount of "unnecessary" work is surprising to most students. Finally, memory accesses in the simulator take longer than register accesses, with a noticeable delay in the animation to reflect the relative slowness of RAM vs. registers. This effectively plants the seeds for a discussion of caches and the memory hierarchy.

The introduction of main memory in the simulator also makes this an appropriate time to discuss data representation. Although data in the memory locations is displayed in decimal notation by default, a pull-down menu is provided for each memory location that allows it to be viewed in binary notation. Similarly, an option is available for viewing numbers in the datapath in binary. Thus, students may compare decimal and binary representations by entering data and switching back and forth between the two views (a third option for unsigned decimal numbers is also provided). If desired, algorithms for binary arithmetic may be discussed at this point, and students may experiment with the simulator to observe the behavior of the ALU on binary numbers.

3.3 K&S Computer Simulator with Microprogramming

After repeated exercises in which they must describe the settings of the datapath in English, students are amenable to the idea of defining a more precise and concise notation. Students are asked to "Describe how a sequence of 1's and 0's could be used to instruct the K&S Computer to perform multiple operations in sequence." In other words students are asked to develop a way to write down a program for the machine. The nature of the K&S Computer leads many students to develop ideas analogous to microprogramming. Microprogramming is usually seen as difficult, arcane and complex, however the knobs and switches of the datapath provide a strong intuitive feel for what the 1's and 0's mean. Students propose using 1's to represent closed switches and 0's to represent open switches, or vice versa. Common suggestions for representing knob positions are a positional notation (0001 = 1, 0010 = 2, 0100 = 3 etc.), a string length notation (1 = 1, 11 =2, 111 = 3 etc.), and a pattern notation (00 = 0, 01 = 1, 10 = 2, 11 = 3 or some other ordering). If binary number representation was discussed in conjunction with main memory, students generally focus on the natural mapping of knob settings to binary numbers (00 for register 0, 01 for register 1, etc.).

The third increment of the K&S Computer adds a separate microprogram control unit to the datapath and main memory. The microprogram control unit contains microinstruction registers for up to five microinstructions. The A Addr., B Addr, and C Addr. fields of each microinstruction encode the Register Bank knob positions using the register number represented as a 2 bit unsigned binary integer. The ALU Op. is encoded using a 2 bit unsigned binary integer starting with the + operation as 00 and going clockwise. Switches are encoded

using a 1 for a closed switch and a 0 for an open switch. The Switch pos. field of the microinstruction lists the switches starting with the ALU output in the least significant bit and going clockwise. Finally, the RW Addr. field represents the memory address to be read or written and is encoded using the unsigned binary representation of the memory address.

By entering a series of microinstructions corresponding to datapath settings, students are able to program the machine to carry out a series of datapath cycles. For example, suppose a student wanted to add the contents of two memory locations and store the result in memory. This task requires three separate datapath cycles, one each to load a number from memory and a third cycle to add the numbers and store the result in memory. Loading the contents of memory location 0 into R0 requires a microinstruction in which the switch positions are set at 1010 (connecting main memory and the registers), the C Addr. is 00 (for register 0), and the R/W Addr. is 00000 (for memory location 0). Inserting zeros in the other irrelevant fields yields the microinstruction 0000010100000000. Similarly, loading the contents of memory location 1 into R1 requires a microinstruction with switch settings 1010, C Addr. 01, and R/W Addr. 00001. Again, assuming zeros in the other fields yields 0000010100100001. Finally, adding the contents of R0 and R1 and storing the result in memory location 3 requires a microinstruction in which the A Addr. is 00 (for R0), the B Addr. is 01 (for R1), the ALU Op. is 00 (for A+B), the switch positions are set at 0101 (connecting the ALU with main memory), and the R/W Addr. is 00011 (for memory location 3). Thus, 0001000101000011. Figure 3 shows the simulator with these three microinstructions entered into the Microprogram Memory. Clicking on the Execute button will cause these three microinstructions to be executed in order, resulting in three datapath cycles with the specified configurations.

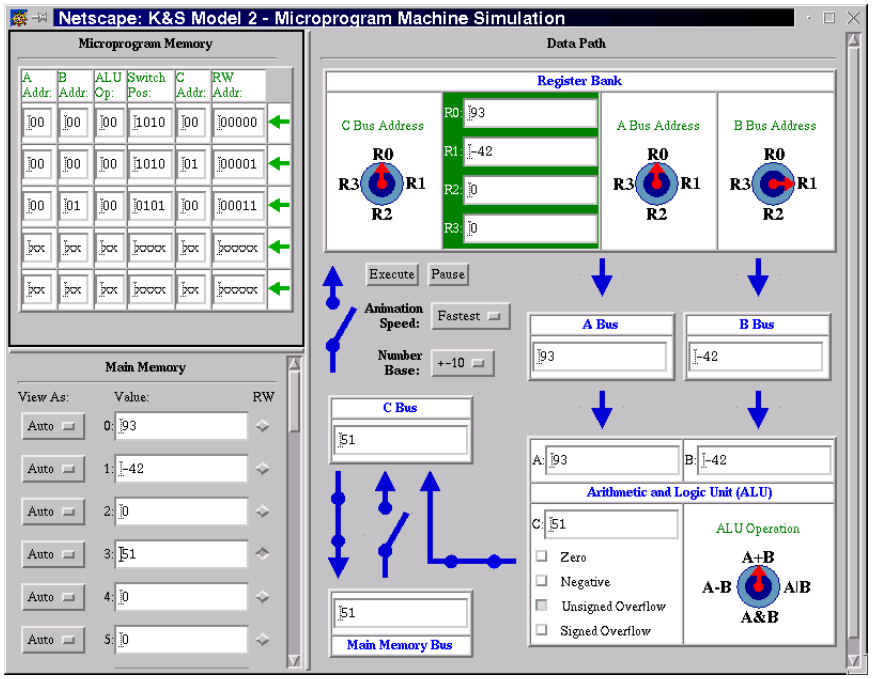

Figure 3: K&S Computer Simulator with Microprogramming.

To visually connect the behavior of the microprogram control unit with the datapath, each microinstruction is animated as it is executed. The microinstruction blinks in its register, and the corresponding settings in the datapath are animated before the cycle begins. As such, students see the direct connection between the microinstructions and the datapath cycles they manually controlled in previous versions of the simulator. Utilizing this visual feedback,

students may experiment by entering bit patterns in the microinstruction registers and observe the resulting settings in the datapath. To encourage experimentation, the simulator also allows for the automatic generation of a microinstruction based on the current knob and switch settings. To generate a microinstruction a student can manually set the knobs and switches of the datapath as in previous versions of the simulator. Then clicking on the arrow to the right of any one of the microinstruction registers causes that register to be automatically loaded with a microinstruction corresponding to the current settings of the knobs and switches.

After having completed exercises where they must describe a series of cycles that complete a given task, the process of programming with microinstructions is already familiar to students. Additional exercises that focus on the transition from manual manipulation to microprogramming are provided, such as the following.

- Give the microinstruction that subtracts R1 from R0 and stores the result in memory location 4.
- Write a microprogram that changes the sign of the number in memory location 4.
- Write a microprogram that stores 4 times the number in memory location 1 into memory location 2.
- Write a microprogram that computes the sum of the numbers in memory locations 0, 1 and 2 and stores the result in memory location 3.

3.4 K&S Computer Simulator with Machine Language Programming

The move from the two-memory Harvard-like architecture of the microprogrammable K&S Computer to a single-memory stored-program version is a difficult step. There are two conceptual changes that take place simultaneously: (1) the program is moved from the microprogram memory to main memory, and (2) the representation of the program is changed from microinstructions to machine language. The advantages of a single memory system are readily understandable to the students (simpler architecture, more efficient use of shared memory, the ability to load multiple programs in memory, etc). The less obvious transition from microinstructions to machine language requires more motivation. First, students must recognize that in real computers, the number of microprogrammable components in the datapath can be quite large, and many of the configurations of those components are not useful in practice (e.g., simultaneously loading data into a register from the ALU and memory). In addition, other potentially useful instructions do not correspond to datapath cycles, such as branch instructions that conditionally or unconditionally alter the flow of control of a program. The necessity of such instructions is easily motivated by exercises like trying to find the absolute value of a number stored in R0.

Table 1: The Machine and Assembly Language Instructions for the K&S Computer.

Machine Language	Example	Meaning	Assembly Language
1 000 0001 0 RR MMMMM	1 000 0001 0 10 01101	R2 = MM[13]	LOAD R2 13
1 000 0010 0 RR MMMMM	1 000 0010 0 11 01000	MM[8] = R3	STORE 8 R3
1 001 0001 0000 RR RR	1 001 0001 0000 10 00	R2 = R0	MOVE R2 R0
1 010 0001 00 RR RR RR	1 010 0001 00 11 10 01	R3 = R2 + R1	ADD R3 R2 R1
1 010 0010 00 RR RR RR	1 010 0010 00 11 01 00	R3 = R1 - R0	SUB R3 R1 R0
1 010 0011 00 RR RR RR	1 010 0011 00 00 11 01	R0 = R3 & R1	AND R0 R3 R1
1 010 0100 00 RR RR RR	1 010 0100 00 10 10 11	R2 = R2 | R3	OR R2 R2 R3
0 000 0001 000 MMMMM	0 000 0001 000 01010	PC = 10	BRANCH 10
0 000 0010 000 MMMMM	0 000 0010 000 00010	if Zero Flag set, PC = 2	BZERO 2
0 000 0011 000 MMMMM	0 000 0011 000 00111	if Neg. Flag set, PC = 7	BNEG 7
0000 0000 0000 0000		no operation	NOP
1111 1111 1111 1111		halt execution	HALT

Table 1 presents the machine language instruction set for the K&S Computer. While this instruction set is limited, it is in fact Turing complete when given the ability to manually pre-load constants into memory locations and/or registers. This ability to pre-load constant values was favored over the inclusion of immediate mode or I/O instructions in order to preserve the simple nature of the K&S Computer.

Since the control unit for the K&S Computer must fetch, interpret, and execute machine language instructions from memory, it is understandably more complex than the microprogram control unit. A Program Counter (PC) must keep track of the next instruction to load, and an Instruction Register (IR) is needed to load and interpret each instruction. The Instruction Interpreter translates the machine language instruction from the IR, displays the corresponding microinstruction, and executes the datapath cycle using the specified settings. The PC is automatically incremented, or otherwise updated in the case of branch instructions. For example, suppose a student wanted to compare the contents of two memory locations and store the larger of the two in a separate memory location. This can be accomplished by entering a sequence of 8 machine language instructions in main memory. First, the contents of the desired memory locations must be loaded into registers (1000000100001010 loads the contents of memory location 10 into R0, and 1000000100101011 loads the contents of memory location 11 into R1). The next instruction (1010001000100001) subtracts R1 from R0 and stores the result in R2. Next, a branch-if-negative instruction (0000001100000110) causes the PC to jump to location 6 in memory if the negative flag in the ALU is set, i.e., if R1 > R0. If this occurs, the instruction at memory location 6 (1000001000101100) will store the value from R1 into memory location 12, followed by a halt instruction (1111111111111111). If a jump does not occur, the instruction in memory location 4 (1000001000001100) will store the value from R1 into memory location 12, followed by a halt instruction (1111111111111111). Figure 4 shows the K&S Computer with this machine language program loaded into main memory. Clicking on the Execute button will cause these instructions to be fetched, interpreted, and executed by the Control Unit, resulting in the desired behavior.

Stepping through a small example similar to the one shown in Figure 4 has proven to be an effective way to illustrate how a program stored in main memory is executed. While stepping through this example, the role of the program counter (PC) and instruction register (IR) are explained. Based on their experience with microprogramming, students have an intuitive understanding of the purpose of the "Instruction Interpretation" part of the control unit. At this point, the students are presented with machine language programming tasks to complete, for example:

- If the number in memory location 13 is less than zero, then add 1 to the number in memory location 14, otherwise subtract 1 from the number in memory location 15.
- If the number in memory location 13 is greater than zero, then add 1 to the number in memory location 14, otherwise subtract 1 from the number in memory location 15.
- Calculate the sum of all of the numbers between 1 and 100 and store the result in memory location 15.
- Multiply the number in memory location 14 by the number in memory location 15 and store the result in memory location 13.

The distinction between RISC and CISC processors can be introduced at this point by noting that each machine language instruction corresponds to at most one microinstruction. While this limits the complexity of the machine language instructions, it ensures that each instruction can be executed in a single datapath cycle, i.e., it is a Reduced Instruction Set Computer. If there were machine language instructions that could not be translated into a

single microinstruction (such as adding the values in 2 memory locations), then a Complex Instruction Set Computer would result.

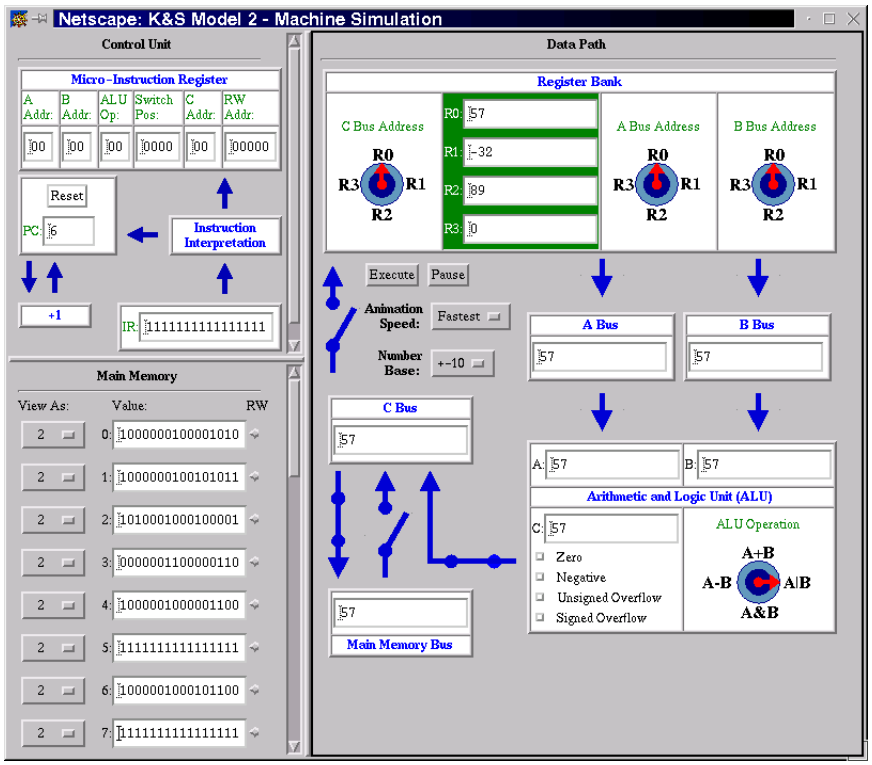

Figure 4: K&S Computer Simulator with Machine Language Programming.

3.5 K&S Computer Simulator with Assembly Language Programming

The final increment of the K&S Computer, as described in the previous section, also supports assembly language programming. The development of assembly languages is motivated by gains in programmer convenience and efficiency achieved by introducing mnemonic names for the machine language instructions. In Table 1, the last column gives the corresponding assembly language instruction for each machine language instruction. For example, Figure 5 shows the same program as in Figure 4, but viewed as assembly instead of binary machine language instructions.

Admittedly, the one-to-one correspondence of assembly language instructions to machine language instructions in the K&S Computer blurs the distinction between the two. However, introducing the notion that a separate assembler could perform the translation from assembly to machine language can reinforce the distinction. Discussions of the assembler can include the creation of pseudo-instructions such as BEQ (Branch Equal), BLE (Branch Less Than) and BGT (Branch Greater Than). The job of the assembler then becomes translating the single pseudo-instruction into an equivalent sequence of machine language instructions. Depending on the students' backgrounds and the goals of the course, a discussion of compilers and high-level language translation may also be appropriate at this point.

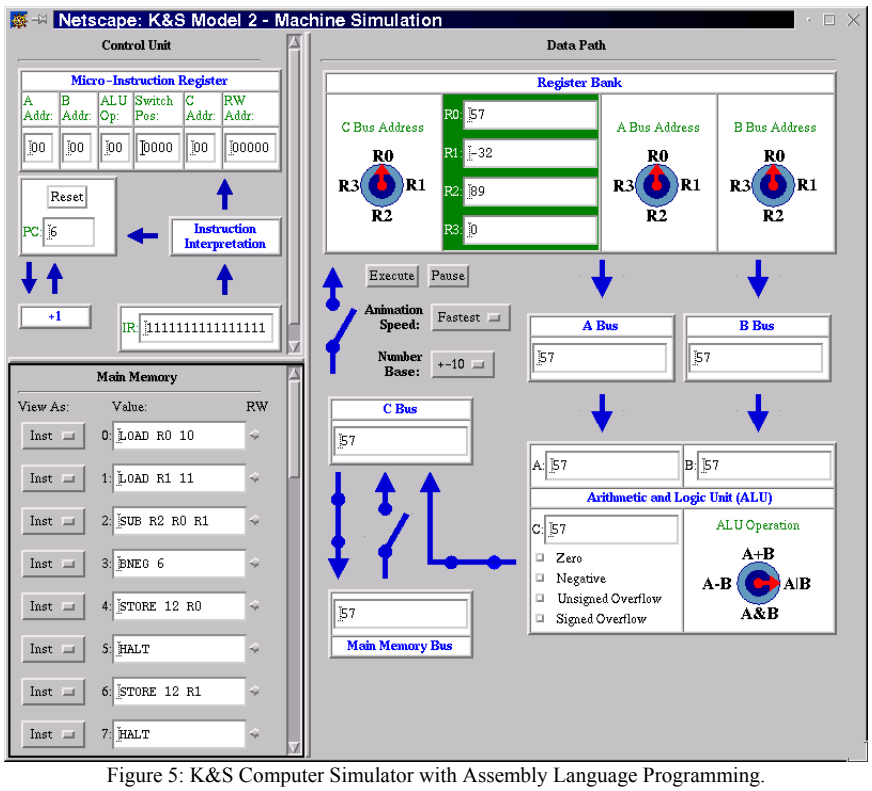

Figure 5: K&S Computer Simulator with Assembly Language Programming.

4. DISCUSSION

The Knob & Switch Computer Simulator has been successfully adopted by the authors in breadth-based introductory computer science courses at Dickinson College and Creighton University, as well as in courses at a variety of other colleges and universities (see the K&S Computer homepage at www.dickinson.edu/~braught/kands/kands.html for a current list of adopters). The two courses at Dickinson and Creighton demonstrate how the K&S Computer Simulator can be adapted to courses with different goals. The introductory course at Dickinson (www.dickinson.edu/~cs131) serves a mixture of non-majors and potential majors, meeting college requirements as a laboratory science course and a quantitative reasoning course. This course also provides an entry point to the computer science major for students with no prior programming experience. The goal is to instill an understanding of computer organization and its relation to high-level programming, culminating in the ability to write and execute simple assembly language programs. This full range of topics is presented using all four increments of the K&S Computer Simulator over a two-week period. At Creighton University, the K&S Computer Simulator is used in a more traditional CS0 course for non-majors (www.creighton.edu/~csc107). The goal there is for students to understand the basic structure of a stored-program computer and its components, with only one week devoted to the topic. Students experiment with the first two increments of the simulator, but are only exposed to microprogramming and machine language programming via instructor-led demonstrations of the final two increments.

Student response to the K&S Computer Simulator at both Dickinson and Creighton has been overwhelmingly positive. The majority of students have found the simulator intuitive and easy to use, and appreciate the incremental development of the simulator as a technique for gradually building their understanding. At least qualitatively, students are coming away from these courses with a more complete picture of how computers work than they have in the past. At Dickinson, where the K&S Computer Simulator has been used since 1998, there

has been a noticeable effect on the sophomore level computer organization course, which is now able to assume a top-to-bottom conceptual understanding of how computer work from incoming students.

ACKNOWLEDGEMENTS

Portions of this paper originally appeared in [Braught 2001]. They are reprinted here with permission from the Consortium for Computing in Small Colleges.

REFERENCES

ARIAS, J. AND GARCIA, D. 1999 Introducing computer architecture education in the first course of computer science career. *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, (July).

BIERMANN, A. 1997 *Great Ideas in Computer Science, 2nd ed*. MIT Press.

BRAUGHT, G. 2001 Computer organization in the breadth-first course. *Journal of Computing in Small Colleges 16*(4): 182-195.

CC2001 TASK FORCE 2001 *Computing Curricula 2001,* (Dec. 2001). WWW: http://www.acm.org/sigcse/cc2001/steelman.

DECKER, R. AND HIRSHFIELD, S. 1998 *The Analytical Engine: An Introduction to Computer Science Using the Internet*. Brooks/Cole Thomson Learning.

FOLEY, J. AND STANDISH, T. (Eds.). 1988 *Undergraduate Computer Science Education, Report of a Workshop Sponsored by The National Science Foundation*, held at The George Washington University, (March 10-11).

GURWITZ, C. 1998 The Internet as a motivating theme in math/computer science core courses for non-majors. *SIGCSE Bulletin 30*(1): 68-72.

NATIONAL RESEARCH COUNCIL COMMITTEE ON INFORMATION TECHNOLOGY LITERACY 1999 *Being Fluent with Information Technology*. National Academy Press.

NEWSOME, M. AND PANCAKE, C. 1992 A graphical computer simulator for systems programming courses. *SIGCSE Bulletin 24*(1): 157-162.

PASTOR, E., SANCHEZ, F. AND DEL CORRAL, A. 1999 A rudimentary machine: Experiences in the design of a pedagogic computer. *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, (July).

REED, D. 2001a Rethinking CS0 with JavaScript. *SIGCSE Bulletin 33*(1): 100-104.

REED, D. 2001b Developing empirical skills in an introductory computer science course. *Proceedings of the 34th Midwest Instruction and Computing Symposium*, Cedar Falls, IA, (April).

REED, D., MILLER, C. AND BRAUGHT, G. 2000 Empirical investigation throughout the CS curriculum. *SIGCSE Bulletin 32*(1): 202-206

SAMPLE, N., AND ARNOLD, M. 1997 JavaScript for simulation education. *NAU/web.97 Conference*, Flagstaff, Arizona, (June).

SCRAGG, G. 1991 Most computer organization courses are built upside down. *SIGCSE Bulletin 23*(1): 341-346.

TUCKER, A. (Ed.). 1992 *Computing Curricula 1991, Report of the ACM/IEEECS Joint Curriculum Task Force*. ACM Press and IEEE Computer Society Press.

YURCIK, W. AND BRUMBAUGH, L. 2001 A Web-based little man computer simulator. *SIGCSE Bulletin* 33(1): 204-208.

YURCIK, W., WOLFFE, G. AND HOLLIDAY, M. 2001 A survey of simulators used in computer organization/architecture courses. *Proceedings of the 2001 Summer Computer Simulation Conference*, Orlando, Florida, (July).

Received: November 2001; Accepted: February 2002