

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

**Grant Braught**

**Dept of Math and Computer Science  
Dickinson College  
Carlisle, PA 17013  
[brought@dickinson.edu](mailto:brought@dickinson.edu)**

**David Reed**

**Dept of Math and Computer Science  
Creighton University  
Omaha, NE 68178  
[davereed@creighton.edu](mailto:davereed@creighton.edu)**

## **Abstract**

This paper describes the Knob & Switch Computer, a computer architecture simulator designed to teach beginning students the basics of computer organization. This Knob & Switch Computer differs from existing simulators in two significant ways: (1) 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, and (2) it incorporates "cognitive hooks" in the form of knobs and switches that encourage exploration and discovery on the part of the student. Both of these features make it possible to engage beginning students and effectively convey an understanding of how computers work. The 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 simulator, experiences using the simulator in breadth-based introductory courses both at Dickinson College and Creighton University will be discussed.

## **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 breadth-first 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 upcoming 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-first course that can be integrated into a three course introductory sequence [CC2001 Task Force 2001].

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.

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. In recent years, numerous breadth-first texts have integrated computer architecture simulators with their presentation of hardware concepts, e.g., [Biermann 1997], [Decker & Hirshfield 1998]. In addition, educators and software developers have developed a variety of simulators that demonstrate the internal workings of computers and their components, e.g., [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.) While these simulators may present the layout of the CPU and other components and allow the student to visualize the flow of information inside a computer, most have features that limit their effectiveness for beginning students. In particular, they can still overwhelm students with complexity and often provide only minimal capabilities for interaction and exploration.

In 1998, Grant Braught developed a series of computer architecture simulators for use in a breadth-first CS0 course at Dickinson College [Braught 2001]. The Knob & Switch Computer Simulator differs from existing simulators in two significant ways: (1) 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, and (2) it incorporates "cognitive hooks" in the form of knobs and switches that encourage exploration and discovery on the part of the student. Both of these features make it possible to engage beginning students and effectively convey an understanding of how computers work. Written in HTML and JavaScript, the Knob & Switch (K&S) Computer Simulator utilizes the intuitive Web interface and is readily accessible using any JavaScript-enabled browser. The simulator has been successfully adopted in breadth-first introductory courses at Dickinson College, Creighton University, and other institutions such as Wheaton College and the University of Northern Colorado.

## **2. The Knob & Switch Computer Simulator**

implementation details (HTML & JavaScript, buttons&boxes&selectmenus&images → portable & machine indep.), uses frames to integrate components

incremental approach – avoids complexity overload, can study one component at a time  
interactive using cognitive hook – encourages exploration

use in CS0 – provide exercises, student questions lead from one increment to the next (not just how it is laid out, but why it is that way)

## 2.1 K&S Datapath Simulator

The first component of the Knob & Switch 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 setting. For example, Figure 1 shows the datapath configured to add the value in register 0 (R0) to the value in register 1 (R1) and place the result into register 2 (R2).

### SCREEN SHOT

Figure 1. K&S Datapath Simulator

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 speed of the animation 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 settings that would result in the contents of R2 being doubled.
- Describe 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, the idea that clock speed is not necessarily comparable with different CPUs can be understood by noting that different CPUs will have different numbers of registers, bus sizes, and ALU operations, and thus provide different capabilities in a single datapath cycle.

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.

## 2.2 K&S Datapath Simulator with Main Memory

The next component that can be added to the Knob & Switch Computer is separate storage for more

data. 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 checkbox 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 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, Figure 2 shows the datapath and main memory configured to subtract R0 from R1 and to store the result in memory location 2.

## SCREEN SHOT

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.
- Is it possible to copy the contents of R0 into memory location 0 in a single cycle? If so, describe the settings. If not, how many cycles would be required (using what settings)?
- 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 knob and switch settings 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.

Beyond introducing main memory, this second increment of the Knob & Switch 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 often 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.

## 2.3 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 Knob & Switch 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 Knob & Switch 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 Knob & Switch Computer adds a separate control unit to the datapath and main memory, with registers for containing 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 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 R/W 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 the appropriate microinstructions corresponding to datapath settings, students are able to program the machine to carry out a series of datapath cycles. For example, Figure 3 shows the simulator with microinstructions for adding the contents of memory locations 0 and 1 and storing the result in memory location 2. The first two microinstructions load the values from memory locations 0 and 1 into registers R0 and R1, respectively. The third instruction adds the value in R0 to the value in R1 and stores the result in R2. The fourth instruction stores the value in R2 into memory location 2.

### SCREEN SHOT

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 even allows for the automatic generation of microinstructions. Students can manually set the knobs and switches of the datapath as in previous versions of the simulator, then click on the arrow next to a microinstruction register to automatically load the corresponding microinstruction.

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.

## **2.4 K&S Computer Simulator with Machine Language Programming**

The move from the two memory Harvard-like architecture of the microprogrammable Knob & Switch 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 alter the program counter. All of these factors suggest that a new set of instructions is needed for stored-program computer.

Table 1 presents a machine language instruction set for the K&S Computer, utilizing opcodes and general instruction formats. 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.

Machine Language Instruction	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: The Machine and Assembly Language Instructions for 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, Figure 4 shows the K&S Computer with machine language program that compares the contents of memory locations 10 and 11, and stores the larger of the two in memory location 12. The first two machine language instructions load the contents of the memory locations 10 and 11 into registers R0 and R1, respectively. The next instruction subtracts R1 from R0 and stores the result in R2. Next, a branch-if-negative instruction 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 will store the value from R1 into memory location 12, followed by a halt instruction. If a jump does not occur, the instruction in memory location 4 will store the value from R0 into memory location 12, followed by a halt instruction.

## SCREEN SHOT

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

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 the 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.

- 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 machine language instructions, it does ensure that each instruction can be executed in a single datapath cycle, i.e., it is a Reduced Instruction Set Computer. If it were possible to translate a single machine language instruction into multiple microinstructions, then a Complex Instruction Set Computer would result.

## SCREEN SHOT

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

The concept of assembly language programming can be motivated 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 code 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. The distinction can be reinforced by introducing the notion of a separate assembler that performs the translation from assembly to machine language. 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 student's background a discussion of the job of the compiler may be appropriate at this point.

## 5. Discussion

Use at Dickinson (since F99): mixture of non-majors & potential majors. Devote 2 weeks, full coverage

Use at Creighton (since F01), non-majors. Devote 1-1.5 weeks, cover datapath + memory + overview of rest.

Adopted at other schools...

positive response of students, anecdotal evidence suggests better understanding of concepts. Able to ask deeper, more interesting questions on tests. At Dickinson, noticeable effect on performance of students in sophomore level computer organization course.



## 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 1999.
- 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).
- CC2001 TASK FORCE 2001 Computing Curricula 2001, Steelman Draft (Aug 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, 1988.
- 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 1999.
- 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 2001.
- 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 1997.
- 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 2001.