5.12 Historical Perspective and Further Reading

Maurice Wilkes learned computer design in a summer workshop from Eckert and Mauchly and then went on to build the first full-scale, operational, stored-program computer—the EDSAC. From that experience he realized the difficulty of control. He thought of a more centralized control using a diode matrix and, after visiting the Whirlwind computer in the United States, wrote [Wilkes 1985]:

I found that it did indeed have a centralized control based on the use of a matrix of diodes. It was, however, only capable of producing a fixed sequence of eight pulses—a different sequence for each instruction, but nevertheless fixed as far as a particular instruction was concerned. It was not, I think, until I got back to Cambridge that I realized that the solution was to turn the control unit into a computer in miniature by adding a second matrix to determine the flow of control at the microlevel and by providing for conditional microinstructions.

Wilkes [1953] was ahead of his time in recognizing that problem. Unfortunately, the solution was also ahead of its time: To provide control, microprogramming relies on fast memory that was not available in the 1950s. Thus Wilkes's ideas remained primarily academic conjecture for a decade, although he did construct the EDSAC 2 using microprogrammed control in 1958 with ROM made from magnetic cores.

IBM brought microprogramming into the spotlight in 1964 with the IBM 360 family. Before this event, IBM saw itself as a cluster of many small businesses selling different machines with their own price and performance levels, but also with their own instruction sets. (Recall that little programming was done in high-level languages, so that programs written for one IBM machine would not run on another.) Gene Amdahl, one of the chief architects of the IBM 360, said that managers of each subsidiary agreed to the 360 family of computers only because they were convinced that microprogramming made it feasible. To be sure of the viability of microprogramming, the IBM vice president of engineering even visited Wilkes surreptitiously and had a "theoretical" discussion of the pros and cons of microcode. IBM believed that the idea was so important to its plans that it pushed the memory technology inside the company to make microprogramming feasible.

Stewart Tucker of IBM was saddled with the responsibility of porting software from the IBM 7090 to the new IBM 360. Thinking about the possibilities of microcode, he suggested expanding the control store to include simulators, or interpreters, for older machines. Tucker [1967] coined the term *emulation* for this, meaning full simulation at the microprogrammed level. Occasionally, emulation on the 360 was actually faster than on the original hardware.

Once the giant of the industry began using microcode, the rest soon followed. (IBM was over half of the computer industry in 1964, measured in revenue.) One difficulty in adopting microcode was that the necessary memory technology was not widely available, but that was soon solved by semiconductor ROM and later RAM. The microprocessor industry followed the same history, with the limited resources of the earliest chips forcing hardwired control. But as the resources increased, the advantages of simpler design, ease of change, and the ability to use a wide variety of underlying implementations persuaded many to use microprogramming.

In the 1960s and 1970s, microprogramming was one of the most important techniques used in implementing machines. Through most of that period, machines were implemented with discrete components or MSI (medium-scale integration—fewer than 1000 gates per chip), and designers had to choose between two types of implementations: *hardwired control* or **microprogrammed control**. Hardwired control was characterized by finite state machines using an explicit next state and implemented primarily with random logic. In this era, microprogrammed control used microcode to specify control that was then implemented with a microprogram sequencer (a counter) and ROMs. Hardwired control received its name because the control was implemented in hardware and could not be easily changed. Microprograms implemented in ROM were also called **firmware** because they could be changed somewhat more easily than hardware, but not nearly as easily as software.

microprogrammed control A method of specifying control that uses microcode rather than a finite state representation.

firmware Microcode implemented in a memory structure, typically ROM or RAM.

The reliance on standard parts of low- to medium-level integration made these two design styles radically different. Microprogrammed approaches were attractive because implementing the control with a large collection of low-density gates was extremely costly. Furthermore, the popularity of relatively complex instruction sets demanded a large control unit, making a ROM-based implementation much more efficient. The hardwired implementations were faster, but too costly for most machines. Furthermore, it was very difficult to get the control correct, and changing ROMs was easier than replacing a random logic control unit. Eventually, microprogrammed control was implemented in RAM, to allow changes late in the design cycle, and even in the field after a machine shipped.

With the increasing popularity of microprogramming came more sophisticated instruction sets. Over the years, most microarchitectures became more and more dedicated to support the intended instruction set, so that reprogramming for a different instruction set failed to offer satisfactory performance. With the passage of time came much larger control stores, and it became possible to consider a machine as elaborate as the VAX with more than 300 different instruction opcodes and more than a dozen memory-addressing modes. The use of RAM to store the microcode also made it possible to debug the microcode and even fix some bugs once machines were in the field. The VAX architecture represented the high-water mark for instruction set architectures based on microcode implementations. Typical implementations of the full VAX instruction set required 400 to 500 Kbits of control store.

The VAX architecture was laid to rest and replaced by the Alpha architecture, another RISC architecture. With the disappearance of the VAX, traditional microprogramming, in which the control is implemented with one major control store, has disappeared from conventional microprocessor designs. The recent Intel IA-32 processors, which we described in this chapter, use a mix of direct hardware translation via the microoperations and a central control store used only for complex instructions. Intel's newest architecture, the IA-64, originally designed as a replacement for IA-32, uses a RISC-style instruction set, which can be implemented without a large central control store.

Of course, control unit design will continue to be a major aspect of all computers, and the best way to specify and implement the control will vary, just as computers will vary, from streamlined RISC architectures with simple control, to special-purpose processors with potentially large amounts of more complex and specialized control.

The history of the development of interrupts in equally interesting. Professor Mark Smotherman has assembled a brief history at *http://www.cs.clemson.edu/~mark/interrupts.html* or in the library under Section 5.12. Among the notable innovations are the concept of arithmetic overflow introduced in the Univac-1 and vectored interrupts introduced in Stretch (see Chapter 6 Section 6.13 Historical Perspectives).

Further Reading

A basic Verilog tutorial is included on the CD. (See Tutorials) There are also many books both on Verilog and on digital design using Verilog.

Kidder, T. [1981]. *Soul of a New Machine*, Little, Brown, and Co., New York.

Describes the design of the Data General Eclipse series that replaced the first DG machines such as the Nova. Kidder records the intimate interactions among architects, hardware designers, microcoders, and project management.

Levy, H. M., and R. H. Eckhouse, Jr. [1989]. *Computer Programming and Architecture: The VAX*, second ed., Digital Press, Bedford, MA.

Good description of the VAX architecture and several different microprogrammed implementations.

Patterson, D. A. [1983]. "Microprogramming," *Scientific American* 248:3 (March) 36–43.

Overview of microprogramming concepts.

Tucker, S. G. [1967]. “Microprogram control for the System/360,” *IBM Systems J.* 6:4, 222–41.

Describes the microprogrammed control for the 360, the first microprogrammed commercial machine.

Wilkes, M. V. [1985]. *Memoirs of a Computer Pioneer*, MIT Press, Cambridge, MA.

Intriguing biography with many stories about industry pioneers and the trials and successes in building early machines.

Wilkes, M. V., and J. B. Stringer [1953]. “Microprogramming and the design of the control circuits in an electronic digital computer,” *Proc. Cambridge Philosophical Society* 49:230–38. Also reprinted in D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*, McGraw-Hill, New York, 158–63, 1982, and in “The genesis of microprogramming,” in *Annals of the History of Computing* 8:116, 1982.

These two classic papers describe Wilkes’s proposal for microcode.