CHAPTER SUMMARY

This chapter presented a simple architecture, MARIE, as a means to understand the basic fetch–decode–execute cycle and how computers actually operate. This simple architecture was combined with an ISA and an assembly language, with emphasis given to the relationship between these two, allowing us to write programs for MARIE.

The CPU is the principal component in any computer. It consists of a datapath (registers and an ALU connected by a bus) and a control unit responsible for sequencing the operations and data movement and creating the timing signals. All components use these timing signals to work in unison. The I/O subsystem accommodates getting data into the computer and back out to the user.

MARIE is a very simple architecture designed specifically to illustrate the concepts in this chapter without getting bogged down in too many technical details. MARIE has 4K 16-bit words of main memory, uses 16-bit instructions, and has seven registers. There is only one general-purpose register, the AC. Instructions for MARIE use 4 bits for the opcode and 12 bits for an address. Register transfer notation was introduced as a symbolic means for examining what each instruction does at the register level.

The fetch–decode–execute cycle consists of the steps a computer follows to run a program. An instruction is fetched and then decoded, any required operands are then fetched, and finally the instruction is executed. Interrupts are processed at the beginning of this cycle, returning to normal fetch–decode–execute status when the interrupt handler is finished.

A machine language is a list of binary numbers representing executable machine instructions, whereas an assembly language program uses symbolic instructions to represent the numerical data from which the machine language program is derived. Assembly language *is* a programming language, but does not offer a large variety of data types or instructions for the programmer. Assembly language programs represent a lower level method of programming.

You would probably agree that programming in MARIE's assembly language is, at the very least, quite tedious. We saw that most branching must be explicitly performed by the programmer, using jump and branch statements. It is also a large step from this assembly language to a high-level language such as C++ or Ada. However, the assembler is one step in the process of converting source code into something the machine can understand. We have not introduced assembly language with the expectation that you will rush out and become an assembly language programmer. Rather, this introduction should serve to give you a better understanding of machine architecture and how instructions and architectures are related. Assembly language should also give you a basic idea of what is going on behind the scenes in high-level C++, Java, or Ada programs. Although assembly language programs are easier to write for x86 and MIPS than for MARIE, all are more difficult to write and debug than high-level language programs.

Intel and MIPS assembly languages and architectures were introduced (but by no means covered in detail) for two reasons. First, it is interesting to compare the

various architectures, starting with a very simple architecture and continuing with much more complex and involved architectures. You should focus on the differences as well as the similarities. Second, although the Intel and MIPS assembly languages looked different from MARIE's assembly language, they are actually quite comparable. Instructions access memory and registers, and there are instructions for moving data, performing arithmetic and logic operations, and branching. MARIE's instruction set is very simple and lacks many of the "programmer friendly" instructions that are present in both Intel and MIPS instruction sets. Intel and MIPS also have more registers than MARIE. Aside from the number of instructions and the number of registers, the languages function almost identically.

FURTHER READING

A MARIE assembly simulator is available on this textbook's home page. This simulator assembles and executes your MARIE programs.

For more detailed information on CPU organization and ISAs, you are referred to the Tanenbaum (1999) and Stallings (2006) books. Mano (1991) contains instructional examples of microprogrammed architectures. Wilkes, Renwick, and Wheeler (1958) is an excellent reference on microprogramming.

For more information regarding Intel assembly language programming, check out the Abel (2001), Dandamudi (1998), and Jones (1997) books. The Jones book takes a straightforward and simple approach to assembly language programming, and all three books are quite thorough. If you are interested in other assembly languages, you might refer to Struble (1975) for IBM assembly, Gill, Corwin, and Logar (1987) for Motorola, and SPARC International (1994) for SPARC. For a gentle introduction to embedded systems, try Williams (2000).

If you are interested in MIPS programming, Patterson and Hennessy (1997) give a very good presentation and their book has a separate appendix with useful information. Donovan (1972) and Goodman and Miller (1993) also have good coverage of the MIPS environment. Kane and Heinrich (1992) wrote the definitive text on the MIPS instruction set and assembly language programming on MIPS machines. The MIPS home page also has a wealth of information.

To read more about Intel architectures, please refer to Alpert and Avnon (1993), Brey (2003), Dulon (1998), and Samaras (2001). Perhaps one of the best books on the subject of the Pentium architecture is Shanley (1998). Motorola, UltraSparc, and Alpha architectures are discussed in Circello and co-workers (1995), Horel and Lauterbach (1999), and McLellan (1995), respectively. For a more general introduction to advanced architectures, see Tabak (1991).

If you wish to learn more about the SPIM simulator for MIPS, see Patterson and Hennessy (1997) or the SPIM home page, which has documentation, manuals, and various other downloads. Waldron (1999) is an excellent introduction to RISC assembly language programming and MIPS as well.