5.9 SUMMARY

The instruction set architecture level is what most people think of as "machine language" although on CISC machines it is generally built on a lower layer of microcode. At this level the machine has a byte- or word-oriented memory consisting of some number of megabytes or gigabytes, and instructions such as MOVE, ADD, and BEQ.

Most modern computers have a memory that is organized as a sequence of bytes, with 4 or 8 bytes grouped together into words. There are normally also between 8 and 32 registers present, each one containing one word. On some machines (e.g., Core i7), references to words in memory do not have to be aligned on natural boundaries in memory, while on others (e.g., OMAP4430 ARM), they must be. But even if words do not have to be aligned, performance is better if they are.

Instructions generally have one, two, or three operands, which are addressed using immediate, direct, register, indexed, or other addressing modes. Some machines have a large number of complex addressing modes. In many cases, compilers are unable to use them in a effective way, so they are unused. Instructions are generally available for moving data, dyadic and monadic operations, including arithmetic and Boolean operations, branches, procedure calls, and loops, and sometimes for I/O. Typical instructions move a word from memory to a register (or vice versa), add, subtract, multiply, or divide two registers or a register and a memory word, or compare two items in registers or memory. It is not unusual for a computer to have well over 200 instructions in its repertoire. CISC machines often have many more.

Control flow at level 2 is achieved using a variety of primitives, including branches, procedure calls, coroutine calls, traps, and interrupts. Branches are used to terminate one instruction sequence and begin a new one at a (possibly distant) location in memory. Procedures are used as an abstraction mechanism, to allow a part of the program to be isolated as a unit and called from multiple places. Abstraction using procedures in one form or another is the basis of all modern programming. Without procedures or the equivalent, it would be impossible to write any modern software. Coroutines allow two threads of control to work simultaneously. Traps are used to signal exceptional situations, such as arithmetic overflow. Interrupts allow I/O to take place in parallel with the main computation, with the CPU getting a signal as soon as the I/O has been completed.

The Towers of Hanoi is a fun little problem with a nice recursive solution that we examined. Iterative solutions to it have been found, but they are far more complicated and less elegant than the recursive one we studied.

Last, the IA-64 architecture uses the EPIC model of computing to make it easy for programs to exploit parallelism. It uses instruction groups, predication, and speculative LOADs to gain speed. All in all, it may represent a significant advance over the Core i7, but it puts much of the burden of parallelization on the compiler. Still, doing work at compile time is always better than doing it at run time.

PROBLEMS

1. A word on a little-endian computer has the numerical value of 9. If it is transmitted to a big-endian computer byte by byte and stored there, with byte 0 in byte 0, and so on, what is its numerical value on the big endian machine?

2. Design an expanding opcode to allow all the following to be encoded in a 36-bit instruction:

   7 instructions with two 15-bit addresses and one 3-bit register number
   500 instructions with one 15-bit address and one 3-bit register number
   40 instructions with no addresses or registers

3. A certain machine has 24-bit instructions and 8-bit addresses. Some instructions have one address and others have two. If there are n two-address instructions, what is the maximum number of one-address instructions?

4. Given the memory values below and a one-address machine with an accumulator, what values do the following instructions load into the accumulator?

   word 20 contains 40
   word 30 contains 50
   word 40 contains 60
   word 50 contains 70

   a. LOAD IMMEDIATE 20
   b. LOAD DIRECT 20
   c. LOAD INDIRECT 20
   d. LOAD IMMEDIATE 30
   e. LOAD DIRECT 30
   f. LOAD INDIRECT 30

5. Given the memory values below and a one-address machine with an accumulator, what values do the following instructions load into the accumulator?

   word 10 contains 25
   word 20 contains 35
   word 25 contains 45
   word 35 contains 60

   a. LOAD IMMEDIATE 10
   b. LOAD DIRECT 10
   c. LOAD INDIRECT 10
   d. LOAD IMMEDIATE 20
   e. LOAD DIRECT 20
   f. LOAD INDIRECT 20

6. Compare 0-, 1-, 2-, and 3-address machines by writing programs to compute

   $$X = (A + B \times C) / (D - E \times F)$$

   for each of the four machines. The instructions available for use are as follows:

0 Address	1 Address	2 Address		3 Address	
PUSH M	LOAD M	MOV	(X = Y)	MOV	(X = Y)
POP M	STORE M	ADD	(X = X+Y)	ADD	(X = Y+Z)
ADD	ADD M	SUB	(X = X–Y)	SUB	(X = Y–Z)
SUB	SUB M	MUL	(X = X*Y)	MUL	(X = Y*Z)
MUL	MUL M	DIV	(X = X/Y)	DIV	(X = Y/Z)
DIV	DIV M				

M is a 16-bit memory address, and *X*, *Y*, and *Z* are either 16-bit addresses or 4-bit registers. The 0-address machine uses a stack, the 1-address machine uses an accumulator, and the other two have 16 registers and instructions operating on all combinations of memory locations and registers. SUB X,Y subtracts *Y* from *X* and SUB X,Y,Z subtracts Z from *Y* and puts the result in *X*. With 8-bit opcodes and instruction lengths that are multiples of 4 bits, how many bits does each machine need to compute *X*?

7. Devise an addressing mechanism that allows an arbitrary set of 128 addresses, not necessarily contiguous, in a large address space to be specifiable in a 7-bit field.

8. Convert the following formulas from infix to reverse Polish notation.

a. A+B+C–D–E
b. (A–B)×(C+D)+E
c. (A×B)+(C×D)+E
d. (A–B)×(((C–D×E)/F)/G)×H

9. Which of the following pairs of reverse Polish notation formulas are mathematically equivalent?

a. RS+T+ and RST++
b. RS–T– and RST––
c. RS×T+ and RST+×

10. Convert the following reverse Polish notation formulas to infix.

a. AB–C+D×
b. AB/CD/+
c. ABCDE+××/
d. ABCDE×F/+G–H/×+

11. Write three reverse Polish notation formulas that cannot be converted to infix.

12. Convert the following infix Boolean formulas to reverse Polish notation.

a. (A OR B) AND C
b. (A OR B) AND (A OR C)
c. (A AND B) OR (C AND D) OR E

13. Convert the following infix formula to reverse Polish notation and generate IJVM code to evaluate it.

(5 × 2 + 7) – (4 / 2 + 1)

14. How many registers does the machine whose instruction formats are given in Fig. 5-24 have?

15. In Fig. 5-24, bit 23 is used to distinguish the use of format 1 from format 2. No bit is provided to distinguish the use of format 3. How does the hardware know to use it?

16. Describe one advantage and one disadvantage of program-counter-relative addressing.

17. The Core i7 has a condition code bit that keeps track of the carry out of bit 3 after an arithmetic operation. What good is it?

18. One of your friends has just come bursting into your room at 3 A.M., out of breath, to tell you about his brilliant new idea: an instruction with two opcodes. Should you send your friend off to the patent office or back to the drawing board?

19. Tests of the form

```
if (k == 0) ...
if (a > b) ...
if (k < 5) ...
```

are common in programming. Devise an instruction to perform these tests efficiently. What fields are present in your instruction?

20. For the 16-bit binary number 1001 0101 1100 0011, show the effect of:

a. A right shift of 4 bits with zero fill.
b. A right shift of 4 bits with sign extension.
c. A left shift of 4 bits.
d. A left rotate of 4 bits.
e. A right rotate of 4 bits.

21. How can you clear a memory word on a machine with no CLR instruction?

22. Compute the Boolean expression (*A* AND *B*) OR *C* for

A = 1010 1110 0001 0000
B = 1010 1111 1010 1001
C = 0000 0000 0010 0000

23. Devise a way to interchange two variables *A* and *B* without using a third variable or register. *Hint*: Think about the EXCLUSIVE OR instruction.

24. On a certain computer it is possible to move a number from one register to another, shift each of them left by different amounts, and add the results in less time than a multiplication takes. Under what condition is this instruction sequence useful for computing "constant × variable"?

25. Different machines have different instruction densities (number of bytes required to perform a certain computation). For the following Java code fragments, translate each one into Core i7 assembly language and IJVM. Then compute how many bytes each expression requires for each machine. Assume that *i* and *j* are local variables in memory, but otherwise make the most optimistic assumptions in all cases

a. i = 3;
b. i = j;
c. i = j – 1;