# Chapter Two

DATA MANIPULATION

**Chapter Summary**

This chapter introduces the role of a computer's CPU. It describes the machine cycle and the various operations (or, and, exclusive or, add, shift, etc.) performed by a typical arithmetic/logic unit. The concept of a machine language is presented in terms of the simple yet representative machine described in Appendix C of the text. The chapter also introduces some alternatives to the von Neumann architecture such as multiprocessor machines.

The optional sections in this chapter present a more thorough discussion of the instructions found in a typical machine language (logical and numerical operations, shifts, jumps, and I/O communication), a short explanation of how a computer communicates with peripheral devices, and alternative machine designs.

The machine language in Appendix C involves only direct and immediate addressing. However, indirect addressing is introduced in the last section (Pointers in Machine Language) of Chapter 7 after the pointer concept has been presented in the context of data structures.

**Comments**

1. Much of Comment 1 regarding the previous chapter is pertinent here also. The development of skills in the subjects of machine architecture and machine language programming is not required later in the book. Instead, what one needs is an image of the CPU/main memory interface, an understanding of the machine cycle and machine languages, an appreciation of the difference in speeds of mechanical motion compared to CPU activities, and an exposure to the limited repertoire of bit manipulations a CPU can perform.

2. To most students at this stage the terms millisecond, microsecond, nanosecond, and picosecond merely refer to extremely short and indistinguishable units of time. In fact, most would probably accept the incorrect statement that activities within a computer are essentially instantaneous. Once a student of mine wrote a recursive routine for evaluating the determinant of a matrix in an interpreted language on a time-sharing system. The student tried to test the program using an 8 by 8 matrix, but kept terminating the program after a minute because "it must be in a loop." This student left with an understanding of microseconds as real units of time that can accumulate into significant periods.

3. A subtle point that can add significantly to the complexity of this material is combining notation conversion with instruction encoding. If, for example, all the material in Chapters 1 and 2 is new to a student, the problem, "Using the language of Appendix C, write an instruction for loading register 14 with the value 124" can be much more difficult than the same problem stated as, "Using the language of Appendix C, write an instruction for loading register D with the (hexadecimal) value 7C." In general, notation conversion is a subject of minor importance and should not be allowed to cloud the more important concerns.

4. If you want your students to develop more than a simple appreciation of machine language programming, you may want to use one of the many simulators that have been developed for the machine in Appendix C. A nice example is included on the Addison-Wesley website at http://www.aw.com/brookshear or you can find other simulators by searching the Web.

5. Here are some short program routines in the machine language presented in Appendix C of the text, followed by their C language equivalents. (These examples are easily converted into Java, C++, and C#.) Each machine language routine starts at address 10. I've found that they make good examples for class presentations or extra homework problems in which I give the students the machine language form and ask them to rewrite it in a high-level language.

Address Contents Address Contents Address Contents

0D 00 14 20 1B 0F

0E 00 15 5A 1C 50

0F 00 16 30 1D 12

10 20 17 0F 1E 30

11 5C 18 11 1F 0D

12 30 19 0E 20 C0

13 0E 1A 12 21 00

C language equivalent:

{int X,Y,Z;

X = 92;

Y = 90;

Z = X + Y;

}

If the contents of the memory cell at address 1C in the preceding table is changed from 50 to 60 the C equivalent becomes:

{float X, Y, Z;

X = 1.5;

Y = 1.25;

Z = X + Y;

}

Here's another example:

Address Contents Address Contents Address Contents

0E 00 19 0F 24 20

0F 00 1A 20 25 01

10 20 1B 04 26 50

11 02 1C B1 27 01

12 30 1D 2C 28 30

13 0E 1E 12 29 0F

14 20 1F 0E 2A B0

15 01 20 50 2B 18

16 30 21 12 2C C0

17 0F 22 30 2D 00

18 11 23 0E

C equivalent:

{int X, Y;

X = 2; Y =1;

while (Y != 4) {X = X + Y; Y = Y + 1;}

}

6. Here are two C program segments that can be conveniently translated into the machine language of Appendix C.

{int X, Limit;

X = 0;

Limit = 5;

do X = X + 1 while (X != Limit);

}

Program segment in machine language:

Address Contents Address Contents Address Contents

0E 00 (X) 18 22 (R2 = 1) 22 10 (R0 = Limit)

0F 00 (Limit) 19 01 23 0F

10 20 (X = 0) 1A 11 (R1 = X) 24 B1 (go to end

11 00 1B 0E 25 28 if X == Limit)

12 30 1C 50 (X = X+1) 26 B0 (return)

13 0E 1D 12 27 1A

14 20(Limit = 5)1E 30 28 C0 (halt)

15 05 1F 0E 29 00

16 30 20 11 (R1 = X)

17 0F 21 0E

{int X, Y, Difference;

X = 33;

Y = 34;

if (X > Y)Difference = X - Y

else Difference := Y – X}

Program segment in machine language:

Address Contents Address Contents Address Contents

0D 00 (X) 1D 01 2D 16 (Diff = X-Y)

0E 00 (Y) 1E 24 (R4=FF) 2E 30

0F 00 (Diff) 1F FF 2F 0F

10 20 (X = 33) 20 96 (R6=not Y) 30 B0 (branch to

11 21 21 24 31 3A halt)

12 30 22 56 (R6= -Y) 32 90 (R0=not X)

13 0D 23 36 33 14

14 20 (Y = 34) 24 50 (R0=X-Y) 34 50 (R0 = -X)

15 22 25 16 35 03

16 30 26 25 (R5=80Hex) 36 50 (R0 = Y-X)

17 0E 27 80 37 02

18 11 (R1=X) 28 80 (mask low) 38 30 (Diff = Y-X)

19 0D 29 50 7 bits) 39 0F

1A 12 (R2=Y) 2A B5 (if R0=R5 3A C0 (halt)

1B 0E 2B 32 then Y>X 3B 00

1C 23 (R3=1) 2C 50

**Answers to Chapter Review Problems**

1. a. General purpose registers and main memory cells are small data storage cells in a computer.

b. General purpose registers are inside the CPU; main memory cells are outside the CPU.

(The purpose of this question is to emphasize the distinction between registers and memory cells—a distinction that seems to elude some students, causing confusion when following machine language programs.)

2. a. 0010001100000100

b. 1011

c. 001010100101

3. Eleven cells with addresses 98, 99, 9A, 9B, 9C, 9D, 9E, 9F, A0, A1, and A2.

4. CD

5. Program Instruction Memory cell

counter register at 02

02 2211 32

04 3202 32

06 C000 11

6. To compute x + y + z, each of the values must be retrieved from memory and placed in a register, the sum of x and y must be computed and saved in another register, z must be added to that sum, and the final answer must be stored in memory.

A similar process is required to compute (2x) + y. The point of this example is that the multiplication by 2 is accomplished by adding x to x.

7. a. OR the contents of register 2 with the contents of register 3 and place the result in register 1.

b. Move the contents of register E to register 1.

c. Rotate the contents of register 3 four bits to the right.

d. Compare the contents of registers 1 and 0. If the patterns are equal, jump to the instruction at address 00. Otherwise, continue with the next sequential instruction.

e. Load register B with the value (hexadecimal) CD.

8. 16 with 4 bits, 64 with 6 bits

9. a. 2677 b. 1677 c. BA24 d. A403 e. 81E2

10. The only change that is needed is that the third instruction should be 6056 rather than 5056.

11. a. Changes the contents of memory cell 3C.

b. Is independent of memory cell 3C.

c. Retrieves from memory cell 3C.

d. Changes the contents of memory cell 3C.

e. Is independent of memory cell 3C.

12. a. Place the value 55 in register 6. b. 55

13. a. 1221 b. 2134

14. a. Load register 2 with the contents of memory cell 02.

Store the contents of register 2 in memory cell 42.

Halt.

b. 32

c. 06

15. a. 06 b. 0A

16. a. 00, 01, 02, 03, 04, 05

b. 06, 07

17. a. 04 b. 04 c. 0E

18. 04. The program is a loop that is terminated when the value in register 0 (initiated at 00) is finally incremented by twos to the value in register 3 (initiated at 04).

19. 11 microseconds.

20. The point to this problem is that a bit pattern stored in memory is subject to interpretation—it may represent part of the operand of one instruction and the op-code field of another.

a. Registers 0, 1, and 2 will contain 32, 24, and 12, respectively.

b. 12

c. 32

21. The machine will alternate between executing the jump instruction at address AF and the jump instruction at address B0.

22. It would never halt. The first 2 instructions alter the third instruction to read B000 before it is ever executed. Thus, by the time the machine reaches this instruction, it has been changed to read "Jump to address 00." Consequently, the machine will be trapped in a loop forever (or until it is turned off).

23. a. b. c.

14D8 14D8 2000

34B3 15B3 1144

C000 358D B10A

34BD 22FF

C000 B00C

2201

3246

C000

24. a. The single instruction B000 stored in locations 00 and 01.

b. Address Contents

00,01 2100 Initialize

02,03 2270 counters.

04,05 3109 Set origin

06,07 320B and destination.

08,09 1000 Now move

0A,0B 3000 one cell.

0C,0D 2001 Increment

0E,0F 5101 addresses.

10,11 5202

12,13 2333 Do it again

14,15 4010 if all cells

16,17 B31A have not

18,19 B004 been moved.

1A,1B 2070 Adjust values

1C,1D 3071 that are

1E,1F 2079 location

20,21 3075 dependent.

22,23 207B

24,25 3077

26,27 208A

28,29 3087

2A,2B 2074

2C,2D 3089

2E,2F 20C0

30,31 30A4

32,33 2000

34,35 20A5

36,37 B070 Make the big jump!

c. Address Contents

00,01 2000 Initialize counter.

02,03 2100 Initialize origin.

04,05 2270 Initialize destination.

06,07 2430 Initialize references

08,09 1530 to table.

0A,0B 310D Get origin

0C,0D 1600 value.

0E,0F B522 Jump if value must be adjusted.

10,11 3213 Place value

12,13 3600 in new location.

14,15 2301 Increment

16,17 5003 R0,

18,19 5113 R1, and

1A,1B 5223 R2.

1C,1D 233C Are we done?

1E,1F B370 If so, jump to relocated program.

20,21 B00A Else, go back.

22,23 2370 Add 70 to

24,25 5663 value being

26,27 2301 transferred and

28,29 5443 update R4 and

2A,2B 342D R5 for next

2C,2D 1500 location.

2E,2F B010 Return (from subroutine).

30,31 0305 Table of

32,33 0709 locations that

34,35 0B0F must be

36,37 111F updated for

38,39 212B new location.

3A,3B 2FFF

25.

20A0

21A1

6001

21A2

6001

21A3

6001

30A4

C000

26. The machine would place a halt instruction (C000) at memory location 04 and 05 and then halt when this instruction is executed. At this point its program counter will contain the value 06.

27. The machine would continue to repeat the instruction at address 08 indefinitely.

28. It copies the data from the memory cells at addresses 00, 01, and 02 into the memory cells at addresses 10, 11, and 12.

29. Let R represent the first hexadecimal digit in the operand field;

Let XY represent the second and third digits in the operand field;

If the pattern in register R is the same as that in register 0,

then change the value of the program counter to XY.

30. Let the hexadecimal digits in the operand field be represented by R, S, and T;

Activate the two's complement addition circuitry with registers S and T

as inputs;

Store the result in register R.

31. Same as Problem 24 except that the floating-point circuitry is activated.

32. a. 02 b. AC c. FA d. 08 e. F2

33. a. b. c. d.

1044 1034 10A5 10A5

30AA 21F0 210F 210F

8001 8001 8001

3034 12A6 4001

21F0 A104

8212 7001

7002 30A5

30A6

34. a. 101001 b. 000000 c. 000100 d. 110011 e. 111001 f. 111110

g. 010101 h. 111111 i. 010000 j. 101101 k. 000101 l. 001010

35. a. OR the byte with 11110000.

b. XOR the byte with.10000000.

c. XOR the byte with 11111111.

d. AND the byte with 11111110.

e. OR the byte with 01111111.

36. XOR the input string with 10000001.

37. First AND the input byte with 10000001, then XOR the result with 10000001.

38. a. 11010 b. 00001111 c. 010 d. 001010 e. 10000

39. a. CF b. 43 c. FF d. DD

40. a. AB05 b. AB06

41. Address Contents

00,01 2008 Initialize registers.

02,03 2101

04,05 2200

06,07 2300

08,09 148C Get the bit pattern;

0A,0B 8541 Extract the least significant bit;

0C,0D 7335 Insert it into the result.

0E,0F 6212

10,11 B218 Are we done?

12,13 A401 If not, rotate registers

14,15 A307

16,17 B00A and go back;

18,19 338C If yes, store the result

1A,1B C000 and halt.

42. The idea is to complement the value at address A1 and then add. Here is one solution:

21FF

12A1

7221

13A2

5423

34A0

43. An uncompressed video stream of the specified format would require a speed of about 1.5 Gbps. Thus, both USB 1.1 and USB 2.0 would be incapable of sending a video stream of this format. A USB 3.0 serial port would be required. It is interesting to note that with compression, a video stream of 1920 X 1080 resolution, 30 fps and 24 bit color space could be sent over a USB 2.0 port.

44. The typist would be typing 40 x 5 = 200 characters per minute, or 1 character every 0.3 seconds (= 300,000 microseconds). During this period the machine could execute 150,000,000 instructions.

45. The typist would be producing characters at the rate of 4 characters per second, which translates to 32 bps (assuming each character consists of 8 bits).

46. Address Contents

00,01 2000

02,03 2101

04,05 12FE Get printer status

06,07 8212 and check the ready flag.

08,09 B004 Wait if not ready.

0A,0B 35FF Send the data.

47. Address Contents

00,01 20C1 Initialize registers.

02,03 2100

04,05 2201

06,07 130B

08,09 B312 If done, go to halt.

0A,0B 31A0 Store 00 at destination.

0C,0D 5332 Change destination

0E,0F 330B address,

10,11 B008 and go back.

12,13 C000

48. 15 Mbps is equivalent to 1.875 MBs / sec (or 6.75 GBs / hour). Therefore, it would take 29.63 hours to fill the 200 GB drive.

49. 1.74 megabits

50. Group the 64 values into 32 pairs. Compute the sum of each pair in parallel. Group these sums into 16 pairs and compute the sums of these pairs in parallel. etc.

51. CISC involves numerous elaborate machine instructions that can be time consuming. RISC involves fewer and simpler instructions, each of which is efficiently implemented.

52. How about pipelining and parallel processing? Increasing clock speed is another answer.

53. In a multiprocessor machine several partial sums can be computed simultaneously.