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, which we call The Vole, 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 of Chapter 7 (Pointers in Machine Language) after the pointer concept has been presented in the context of data structures.

Comments

1. When describing Computer Architecture in Section 2.1, remind students that this architecture applies, in general, to every computer whether it be a supercomputer, desktop, tablet, laptop, or phone.

2. Students will often be confused with the idea and implementation of machine language, so go very slowly when first teaching this. In the "Questions and Exercises" at the end of Section 2.2, problem #7 starts with commands in English and asks students to translate them into Vole. Using this approach first will help students better see what the Vole language is trying to accomplish.

3. The concepts of Program Counter and Instruction Register in Section 2.3 will make more sense to students if the instructor does an interactive example in which these values are changing as the program is hand-simulated. Because a single command requires 4 Hex digits, but each memory cell holds 2 Hex digits, the program counter in the Vole language must increase by 2 after each instruction. This is demonstrated in Figure 2.11. Students may need some help seeing why this is required, and may also need reminders of this fact throughout the chapter.

4. While it could be possible to write an interpreter for the Vole language, students will benefit in the long run by hand-simulating these programs rather than entering them into a simulator.

The ability to understand that a program is executed one command at a time, and that unintended commands still execute, lay the groundwork for debugging programs, no matter the language.

5. It may be helpful to hand out to your students a summary of the Vole language, from Appendix C, on a single sheet of paper.

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 0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F, 0xA0, 0xA1, and 0xA2.

4. 0xCD

5.

Program counter	Instruction register	Memory cell at 0x02
0x02	0x2211	0x32
0x04	0x3202	0x32
0x06	0xC000	0x11

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 0x2 with the contents of register 0x3 and place the result in register 0x1.

b. Move the contents of register 0xE to register 0x1.

c. Rotate the contents of register 0x3 four bits to the right.

d. Compare the contents of registers 0x1 and 0x0. If the patterns are equal, jump to the instruction at address 0x00. Otherwise, continue with the next sequential instruction.

e. Load register 0xB with the value (hexadecimal) 0xCD.

8. 16 with 4 bits, 64 with 6 bits

9. a. 0x2677 b. 0x1677 c. 0xBA24 d. 0xA403 e. 0x81E2

10. The only change that is needed is that the third instruction should be 0x6056 rather than 0x5056.

11. a. Changes the contents of memory cell 0x3C.

b. Is independent of memory cell 0x3C.

c. Retrieves from memory cell 0x3C.

d. Changes the contents of memory cell 0x3C.

e. Is independent of memory cell 0x3C.

12. a. Place the value 0x55 in register 0x6. b. 0x55

13. a. 0x1221 b. 0x2134

14. a. Load register 0x2 with the contents of memory cell 0x02.
Store the contents of register 0x2 in memory cell 0x42.
Halt.

b. 0x32

c. 0x06

15. a. 0x06 b. 0x0A

16. a. 0x00, 0x01, 0x02, 0x03, 0x04, 0x05 b. 0x06, 0x07

17. a. 0x04 b. 0x04 c. 0x0E

18. 0x04. The program is a loop that is terminated when the value in register 0x0 (initiated at 0x00) is finally incremented by twos to the value in register 0x3 (initiated at 0x04).

19. 11 microseconds, because 11 instructions were executed.

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 0x0, 0x1, and 0x2 will contain 0x32, 0x24, and 0x12, respectively.

b. 0x12

c. 0x32

21. The machine will alternate between executing the jump instruction at address 0xAF and the jump instruction at address 0xB0.

22. It would never halt. The first 2 instructions alter the third instruction to read 0xB000 before it is ever executed. Thus, by the time the machine reaches this instruction, it has been changed to read "Jump to address 0x00." Consequently, the machine will be trapped in a loop forever (or until it is turned off).

23. As the question states, assume the program is loaded into memory starting at address 0x00

a.	b.	c.
0x14D8	0x14D8	0x2000
0x34B3	0x15B3	0x1144
0xC000	0x35D8	0xB10A
	0x34B4	0x22FF
	0xC000	0xB00C
		0x2201
		0x3246
		0xC000

24. a. The single instruction 0xB000 stored in locations 0x00 and 0x01.

b.

Address	Contents
0x00	0x2100 Initialize
0x02	0x2270 counters.
0x04	0x3109 Set origin
0x06	0x320B and destination.
0x08	0x1000 Now move
0x0A	0x3000 one cell.
0x0C	0x2001 Increment
0x0E	0x5101 addresses.
0x10	0x5202
0x12	0x2333 Do it again
0x14	0x4010 if all cells
0x16	0xB31A have not
0x18	0xB004 been moved.
0x1A	0x2070 Adjust values
0x1C	0x3071 that are
0x1E	0x2079 location
0x20	0x3075 dependent.
0x22	0x207B
0x24	0x3077
0x26	0x208A
0x28	0x3087
0x2A	0x2074
0x2C	0x3089
0x2E	0x20C0
0x30	0x30A4
0x32	0x2000
0x34	0x20A5
0x36	0xB070 Make the big jump!

c.

Address	Contents
0x00	0x2000 Initialize counter.
0x02	0x2100 Initialize origin.
0x04	0x2270 Initialize destination.
0x06	0x2430 Initialize references
0x08	0x1530 to table.
0x0A	0x310D Get origin
0x0C	0x1600 value.
0x0E	0xB522 Jump if value must be adjusted.
0x10	0x3213 Place value
0x12	0x3600 in new location.
0x14	0x2301 Increment
0x16	0x5003 R0,
0x18	0x5113 R1, and
0x1A	0x5223 R2.
0x1C	0x233C Are we done?
0x1E	0xB370 If so, jump to relocated program.
0x20	0xB00A Else, go back.
0x22	0x2370 Add 70 to
0x24	0x5663 value being
0x26	0x2301 transferred and
0x28	0x5443 update R4 and
0x2A	0x342D R5 for next
0x2C	0x1500 location.
0x2E	0xB010 Return (from subroutine).
0x30	0x0305 Table of
0x32	0x0709 locations that
0x34	0x0B0F must be
0x36	0x111F updated for

0x38	0x212B	new location.
0x3A	0x2FFF	

25.

0x20A0
0x21A1
0x6001
0x21A2
0x6001
0x21A3
0x6001
0x30A4
0xC000

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. 0x02 b. 0xAC c. 0xFA d. 0x08 e. 0xF2

33.

a.	b.	c.	d.
0x1044	0x1034	0x10A5	0x10A5
0x30AA	0x21F0	0x210F	0x210F
	0x8001	0x8001	0x8001
	0x3034	0x12A6	0x4001
	0x21F0	0xA104	
	0x8212	0x7001	
	0x7002	0x30A5	
	0x30A6		

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.

f. AND the 24-bit RGB bitmap pixel with 111111110000000011111111.

g. XOR the 24-bit RGB bitmap pixel with 111111111111111111111111.

h. OR the 24-bit RGB bitmap pixel with 111111111111111111111111.

36. a. print(bin(byteVariable | 0b11110000))

b. print(bin(byteVariable ^ 0b10000000))

c. print(bin(byteVariable ^ 0b11111111))

d. print(bin(byteVariable & 0b11111110))

e. print(bin(byteVariable | 0b01111111))

f. print(bin(pixel & 0b111111110000000011111111))

g. print(bin(pixel ^ 0b 111111111111111111111111))

h. print(bin(pixel | 0b 111111111111111111111111))

37. XOR the input string with 10000001.

38. print(bin(inputString ^ 0b10000001))

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

40. tempString = inputString & 0b10000001

print(bin(inputString ^ 0b10000001))

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

42. a. 0xCF b. 0x43 c. 0xFF d. 0xDD

43. a. 0xAB05 b. 0xAB06 (2 bits to the left is equivalent to 6 bits to the right)

44.

```
Address      Contents
0x00         0x2008 Initialize registers.
0x02         0x2101
0x04         0x2200
0x06         0x2300
0x08         0x148C Get the bit pattern;
0x0A         0x8541 Extract the least significant bit;
0x0C         0x7335 Insert it into the result.
0x0E         0x6212
0x10         0xB218 Are we done?
0x12         0xA401 If not, rotate registers
0x14         0xA307
0x16         0xB00A   and go back;
0x18         0x338C If yes, store the result
0x1A         0xC000   and halt.
```

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

```
0x21FF
0x12A1
0x7221
0x13A2
0x5423
0x34A0
```

46. 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.

47. 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.

48. 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).

49.

```
Address     Contents
0x00        0x2000
0x02        0x2101
0x04        0x12FE Get printer status
0x06        0x8212 and check the ready flag.
0x08        0xB004 Wait if not ready.
0x0A        0x35FF Send the data.
```

50.

```
Address     Contents
0x00        0x20C1 Initialize registers.
0x02        0x2100
0x04        0x2201
0x06        0x130B
0x08        0xB312 If done, go to halt.
0x0A        0x31A0 Store 00 at destination.
0x0C        0x5332 Change destination
0x0E        0x330B address,
0x10        0xB008 and go back.
0x12        0xC000
```

51. 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.

52. 1.74 megabits

53. 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.

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

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

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

57.

```
radius = float(input('Please enter a radius '))
circumference = 2 * 3.14 * radius
radius = 3.14 * radius * radius
print('Circumference ' is ' + str(circumference))
print('Area is ' + str(area))
```

58.

```
message = input('Please enter message ')
ntimes = int(input('Please enter no. times to repeat the message '))
print(message * ntimes)
```

59.

```
import math
side1 = float(input('Please enter first side of a right triangle '))
side2 = float(input('Please enter second side of a right triangle '))
hypotenuse = math.sqrt(side1 * side1 + side2 * side 2)
perimeter = side1 + side2 + hypotenuse
are = side1 * side2 / 2
print('Hypontenuse ' is ' + str(hypotenuse))
print('Perimeter is ' + str(perimeter))
print('Area is ' + str(area))
```