Chapter 2(2.1,2.2,2.5,2.8,2.13,2.9(HW)) Machine Instructions and Programs

2.1 **Represent the decimal values 5, −2, 14, −10, 26, −19, 51, and −43, as signed, 7-bit numbers in the following binary formats:**

(*a*) Sign-and-magnitude
(*b*) 1's-complement
(*c*) 2's-complement

2.1. The three binary representations are given as:

Decimal values	Sign-and-magnitude representation	1's-complement representation	2's-complement representation
5	0000101	0000101	0000101
−2	1000010	1111101	1111110
14	0001110	0001110	0001110
−10	1001010	1110101	1110110
26	0011010	0011010	0011010
−19	1010011	1101100	1101101
51	0110011	0110011	0110011
−43	1101011	1010100	1010101

2.2 (*a*) Convert the following pairs of decimal numbers to 5-bit, signed, 2's-complement, binary numbers and add them. State whether or not overflow occurs in each case.

(*a*) 5 and 10
(*b*) 7 and 13
(*c*) −14 and 11
(*d*) −5 and 7
(*e*) −3 and −8
(*f*) −10 and −13

(*b*) Repeat Part *a* for the subtract operation, where the second number of each pair is to be subtracted from the first number. State whether or not overflow occurs in each case.

2.2. (a)

(a)	00101	(b)	00111	(c)	10010
	+ 01010		+ 01101		+ 01011
	———		———		———
	01111		10100		11101
	no overflow		overflow		no overflow
(d)	11011	(e)	11101	(f)	10110
	+ 00111		+ 11000		+ 10011
	———		———		———
	00010		10101		01001
	no overflow		no overflow		overflow

(b) To subtract the second number, form its 2's-complement and add it to the first number.

(a)	00101	(b)	00111	(c)	10010
	+ 10110		+ 10011		+ 10101
	———		———		———
	11011		11010		00111
	no overflow		no overflow		overflow
(d)	11011	(e)	11101	(f)	10110
	+ 11001		+ 01000		+ 01101
	———		———		———
	10100		00101		00011
	no overflow		no overflow		no overflow

2.3 Given a binary pattern in some memory location, is it possible to tell whether this pattern represents a machine instruction or a number?

2.3. No; any binary pattern can be interpreted as a number or as an instruction.

2.4 A memory byte location contains the pattern 00101100. What does this pattern represent when interpreted as a binary number? What does it represent as an ASCII code?

2.4. The number 44 and the ASCII punctuation character "comma".

2.5 Consider a computer that has a byte-addressable memory organized in 32-bit words according to the big-endian scheme. A program reads ASCII characters entered at a keyboard and stores them in successive byte locations, starting at location 1000. Show the contents of the two memory words at locations 1000 and 1004 after the name "Johnson" has been entered.

2.5. Byte contents in hex, starting at location 1000, will be 4A, 6F, 68, 6E, 73, 6F, 6E. The two words at 1000 and 1004 will be 4A6F686E and 736F6EXX.Byte 1007 (shown as XX) is unchanged. (See Section 2.6.3 for hex notation.)

2.5 Consider a computer that has a byte-addressable memory organized in 32-bit words according to the big-endian scheme. A program reads ASCII characters entered at a keyboard and stores them in successive byte locations, starting at location 1000. Show the contents of the two memory words at locations 1000 and 1004 after the name "Johnson" has been entered.

2.6 Repeat Problem 2.5 for the little-endian scheme.

2.6 Byte contents in hex, starting at location 1000, will be 4A, 6F, 68, 6E, 73, 6F, 6E. The two words at 1000 and 1004 will be 6E686F4A and XX6E6F73.Byte 1007 (shown as XX) is unchanged. (See section 2.6.3 for hex notation.)

2.7 Clear the high-order 4 bits of each byte to 0000.

2.8 Write a program that can evaluate the expression

$$A \times B + C \times D$$

in a single-accumulator processor. Assume that the processor has Load, Store, Multiply, and Add instructions, and that all values fit in the accumulator.

2.8 A program for the expression is:

```
Load      A
Multiply  B
Store     RESULT
Load      C
Multiply  D
Add       RESULT
Store     RESULT
```

2.9 The list of student marks shown in Figure 2.14 is changed to contain *j* test scores for each student. Assume that there are *n* students. Write an assembly language program for computing the sums of the scores on each test and store these sums in the memory word locations at addresses SUM, SUM + 4, SUM + 8, ... The number of tests, *j*, is larger than the number of registers in the processor, so the type of program shown in Figure 2.15 for the 3-test case cannot be used. Use two nested loops, as suggested in Section 2.5.3. The inner loop should accumulate the sum for a particular test, and the outer loop should run over the number of tests, *j*. Assume that *j* is stored in memory location *J*, placed ahead of location *N*.

2.9 Memory word location J contains the number of tests, *j*, and memory word location N contains the number of students, *n*. The list of student marks begins at memory word location LIST in the format shown in Figure 2.14. The parameter Stride = 4(*j* + 1) is the distance in bytes between scoreson a particular test for adjacent students in the list.

The Base with index addressing mode (R1,R2) is used to access the scores on a particular test. Register R1 points to the test score for student 1, and R2 is incremented by Stride in the inner loop to access scores on the same test by successive students in the list.

```
        Move       J,R4          Compute and place Stride = 4(j + 1)
        Increment  R4              into register R4.
        Multiply   #4,R4
        Move       #LIST,R1      Initialize base register R1 to the
        Add        #4,R1           location of the test 1 score for student 1.
        Move       #SUM,R3       Initialize register R3 to the location
                                   of the sum for test 1.
        Move       J,R10         Initialize outer loop counter R10 to j.
OUTER   Move       N,R11         Initialize inner loop counter R11 to n.
        Clear      R2            Clear index register R2 to zero.
        Clear      R0            Clear sum register R0 to zero.
INNER   Add        (R1,R2),R0    Accumulate the sum of test scores in R0.
        Add        R4,R2         Increment index register R2 by Stride value.
        Decrement  R11           Check if all student scores on current
        Branch>0   INNER           test have been accumulated.
        Move       R0,(R3)       Store sum of current test scores and
        Add        #4,R3           increment sum location pointer.
        Add        #4,R1         Increment base register to next test
                                   score for student 1.
        Decrement  R10           Check if the sums for all tests have
        Branch>0   OUTER           been computed.
```

2.10 (*a*) Rewrite the dot product program in Figure 2.33 for an instruction set in which the arithmetic and logic operations can only be applied to operands in processor registers. The two instructions Load and Store are used to transfer operands between registers and the memory.

(*b*) Calculate the values of the constants k_1 and k_2 in the expression $k_1 + k_2 n$, which represents the number of memory accesses required to execute your program for Part *a*, including instruction word fetches. Assume that each instruction occupies a single word.

2.10. (*a*)

```
                                              Memory
                                              accesses
                                              ————
         Move       #AVEC,R1                     1
         Move       #BVEC,R2                     1
         Load       N,R3                         2
         Clear      R0                           1
LOOP     Load       (R1)+,R4                     2
         Load       (R2)+,R5                     2
         Multiply   R4,R5                        1
         Add        R5,R0                        1
         Decrement  R3                           1
         Branch>0   LOOP                         1
         Store      R0,DOTPROD                   2
```

(*b*) $k_1 = 1 + 1 + 2 + 1 + 2 = 7$; and $k_2 = 2 + 2 + 1 + 1 + 1 + 1 = 8$

2.11 Repeat Problem 2.10 for a computer with two-address instructions, which can perform operations such as

$$A \leftarrow [A] + [B]$$

where A and B can be either memory locations or processor registers. Which computer requires fewer memory accesses?

2.11. (*a*) The original program in Figure 2.33 is efficient on this task.

(*b*) $k_1 = 7$; and $k_2 = 7$

This is only better than the program in Problem 2.10(*a*) by a small amount.

2.12 "Having a large number of processor registers makes it possible to reduce the number of memory accesses needed to perform complex tasks." Devise a simple computational task to show the validity of this statement for a processor that has four registers compared to another that has only two registers.

2.12. The dot product program in Figure 2.33 uses five registers. Instead of using R0 to accumulate the sum, the sum can be accumulated directly into DOTPROD. This means that the last Move instruction in the program can be removed, but DOTPROD is read and written on each pass through the loop, significantly increasing memory accesses. The four registers R1, R2, R3, and R4, are still needed to make this program efficient, and they areall used in the loop. Suppose that R1 and R2 are retained as pointers tothe A and B vectors. Counter register R3 and temporary storage register R4 could be replaced by memory locations in a 2-register machine; but the number of memory accesses would increase significantly.

2.13 Registers R1 and R2 of a computer contain the decimal values 1200 and 4600. What is the effective address of the memory operand in each of the following instructions?

(*a*)	Load	20(R1),R5
(*b*)	Move	#3000,R5
(*c*)	Store	R5,30(R1,R2)
(*d*)	Add	−(R2),R5
(*e*)	Subtract	(R1)+,R5

2.13. 1220, part of the instruction, 5830, 4599, 1200.

c-1200+4600+30=5830

d. Effective Address=R2−1=4600−1=4599 (pre decrement)

e.Effective Address=R1=1200 (post increment)

2.14 Assume that the list of student test scores shown in Figure 2.14 is stored in the memory as a linked list as shown in Figure 2.36. Write an assembly language program that accomplishes the same thing as the program in Figure 2.15. The head record is stored at memory location 1000.

2.14. Linked list version of the student test scores program:

```
        Move        #1000,R0
        Clear       R1
        Clear       R2
        Clear       R3
LOOP    Add         8(R0),R1
        Add         12(R0),R2
        Add         16(R0),R3
        Move        4(R0),R0
        Branch>0    LOOP
        Move        R1,SUM1
        Move        R2,SUM2
        Move        R3,SUM3
```

2.15 Consider an array of numbers $A(i,j)$, where $i = 0$ through $n - 1$ is the row index, and $j = 0$ through $m - 1$ is the column index. The array is stored in the memory of a computer one row after another, with elements of each row occupying m successive word locations. Assume that the memory is byte-addressable and that the word length is 32 bits. Write a subroutine for adding column x to column y, element by element, leaving the sum elements in column y. The indices x and y are passed to the subroutine in registers R1 and R2. The parameters n and m are passed to the subroutine in registers R3 and R4, and the address of element A(0,0) is passed in register R0. Any of the addressing modes in Table 2.1 can be used. At most, one operand of an instruction can be in the memory.

2.15. Assume that the subroutine can change the contents of any register used to pass parameters.

Subroutine

```
                Move        R5,−(SP)      Save R5 on stack.
                Multiply    #4,R4         Use R4 to contain distance in
                                            bytes (Stride) between successive
                                            elements in a column.
                Multiply    #4,R1         Byte distances from A(0,0)
                Multiply    #4,R2           to A(0,x) and A(0,y)
                                            placed in R1 and R2.
LOOP            Move        (R0,R1),R5    Add corresponding
                Add         R5,(R0,R2)      column elements.
                Add         R4,R1         Increment column element
                Add         R4,R2           pointers by Stride value.
                Decrement   R3            Repeat until all
                Branch>0    LOOP            elements are added.
                Move        (SP)+,R5      Restore R5.
                Return                    Return to calling program.
```

2.16 Both of the following statements cause the value 300 to be stored in location 1000, but at different times.

```
ORIGIN      1000
DATAWORD    300
```

and

```
Move   #300,1000
```

Explain the difference.

2.16. The assembler directives ORIGIN and DATAWORD cause the object program memory image constructed by the assembler to indicate that 300 is to be placed at memory word location 1000 at the time the program is loaded into memory prior to execution.

The Move instruction places 300 into memory word location 1000 when the instruction is executed as part of a program.

2.17 Register R5 is used in a program to point to the top of a stack. Write a sequence of instructions using the Index, Autoincrement, and Autodecrement addressing modes to perform each of the following tasks:

(*a*) Pop the top two items off the stack, add them, and then push the result onto the stack.

(*b*) Copy the fifth item from the top into register R3.

(*c*) Remove the top ten items from the stack.

2.17. (a)

```
Move  (R5)+,R0
Add   (R5)+,R0
Move  R0,−(R5)
```

(b)

```
Move   16(R5),R3
```

(c) Add   #40,R5