B.Tech II-CO-Unit II- CO Tutorials2

Ref 1. "Computer Organization," by Carl Hamacher, Zvonko Vranesic and Safwat Zaky. Fifth Edition McGraw-Hill, 2002. (Chapter 2-problems -pps. 98-102)

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) I's-complement
(c) 2's-complement

(

CO Tutorials2

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) 1 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.

CO Tutorials2

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

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 ASGI code?

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.

CO Tutorials2

6) Repeat Problem 2.5 for the little-endian scheme.

7) A program reads ASCII characters representing the digits of a decimal numbers they are entered at a key board and stores the characters in successive memory bytes. Examine the ASCII code in Appendix E and indicate what operation is needed to convert each character into an equivalent binary number.

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.

CO Tutorials2

9)The list of student marks shown in Figure 2.14 is changed to contain 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 -f 8.....number of tests, j, is larger than the number of registers in the processor, so the type of program shown in Figure 2.1S 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's j is stored in memory location J, placed ahead of location N.

CO Tutorials2

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 k1 and k2 in the expression k1+k2n, 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.

CO Tutorials2

11)Repeat Problem 2.10for 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? (Chapter 8 on pipelining gives a different perspective on the answer to this question.)

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.

Solutions For the CO Tutorials

13) Registers R1 andR2of 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

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.

Solutions For the CO Tutorials

15)Consider an array of numbers 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 RO. Any of the addressing modes in Table 2.1 can be used. At most, one operand of an instruction can be in the memory.

Solutions For the CO Tutorials

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.

17)Register RS is used in a program to point to the top of a stack. Write a sequence of instructions using the Index, Auto increment, and Auto decrement 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 flie top ten items from the stack.

Solutions For the CO Tutorials

18)A FIFO queue of bytes is to be implemented in the memory, occupying a fixed region of k bytes. You need two pointers, an IN pointer and an OUT pointer. The IN pointer keeps track of the location where die next byte is to be upended to the queue, and the OUT pointer keeps track of the location containing the next byte to be removed from the queue.

(a)Asdataitemsareaddedtothequeue,theyareaddedatsuccessivelyhigheraddresses until the end of the memory region is reached. What happens next, when a new item is to be added to the queue?

(b) Choose a suitable definition for the IN and OUT pointers, indicating what they point to in the data structure. Use a simple diagram to illustrate your answer.

(c)Showthatifthestateofthequeueisdescribedonlybythetwopointers,thesituations when the queue is completely fill and completely empty are indistinguishable.

(d) What condition would you add to solve the problem in part c?

(e) Propose a procedure for manipulating the two pointers IN and OUT to append and remove items from the queue.

Solutions For the CO Tutorials

19)Consider the queue structure described in Problem2.18. Write APPEND and REMOVE routines that transfer data between a processor register and the queue. Be careful to inspect and update the state of the queue and the pointers each time an operation is attempted and performed.

20)Consider the following possibilities for saving the return address of a subroutine:

(a) In a processor register

(b) In a memory location associated with die call, so that a different location is used when the subroutine is called from different places

(c) On a stack Which of these possibilities supports subroutine nesting and which supports subroutine recursion (that is, a subroutine that calls itself)?

Solutions For the CO Tutorials

22)Assume you want to organize subroutine calls on a computer as follows: When routine Main wishes to call subroutine SUB1, it calls an intermediate routine, CALLSUB, and passes to it the address of SUB1 as a parameter in register R1. CALLSUB saves the return address on a stack, making sure that the upper limit of the stack is not exceeded. Then it branches to SUB1. To return to the calling program, subroutine SUB1 calls another intermediate routine, RETRN. This routine checks that the stack is not empty and then uses the top element to return to the original calling program.

Write routines CALLSUB and RETRN, assuming that the subroutine call instruction saves the return address in a link register, RL. Ihe upper and lower limits of the stack are recorded in memory locations UPPERLIMIT and LOWERLIMIT, respectively.

Solutions For the CO Tutorials

21)The subroutine call instruction of a computer saves the return address in a processor register called the link register, RL. What would you do to allow subroutine nesting? Would your scheme allow the subroutine to call itself?

23)The linked-list insertion subroutine in Figure 2.37 does not check if the E) of the new record matches that of a record already in the list. What happens in the execution of the subroutine if this is the case? Modify the subroutine to return the address of the matching record in register ERROR if this occurs or return a zero if the insertion is successful.

Solutions For the CO Tutorials 2

24)The linked-list deletion subroutine in Figure 2.38 assumes that a record with the ID contained in register RIDNUM is in the list. What happens in the execution of the subroutine if there is no record with this ID? Modify the subroutine to return a zero in RIDNUM if deletion is successful or leave RIDNUM unchanged if the record is not in the list.

Solutions For the CO Tutorials 2

Ref. Solution manual of "Computer Organization," by Carl Hamacher, Zvonko Vranesic and Safwat Zaky. Fifth Edition McGraw-Hill, 2002.(unit ii-pps.1=9)

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

Solutions For the CO Tutorials 2

2.2. (a)

$$
\begin{array}{lrlrlr}
(a) & 00101 & (b) & 00111 & (c) & 10010 \\
 & +\ 01010 & & +\ 01101 & & +\ 01011 \\
 & \hline & & \hline & & \hline \\
 & 01111 & & 10100 & & 11101 \\
 & \text{no overflow} & & \text{overflow} & & \text{no overflow} \\
 & & & & & \\
(d) & 11011 & (e) & 11101 & (f) & 10110 \\
 & +\ 00111 & & +\ 11000 & & +\ 10011 \\
 & \hline & & \hline & & \hline \\
 & 00010 & & 10101 & & 01001 \\
 & \text{no overflow} & & \text{no overflow} & & \text{overflow}
\end{array}
$$

Solutions For the CO Tutorials 2

(*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

Solutions For the CO Tutorials 2

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

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

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.)

2.6. 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.

Solutions For the CO Tutorials 2

2.8. A program for the expression is:

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

Solutions For the CO Tutorials 2

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 scores on 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.
```

Solutions For the CO Tutorials 2

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 $k_2 = 2 + 2 + 1 + 1 + 1 + 1 = 8$

Solutions For the CO Tutorials 2

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. 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 are all used in the loop. Suppose that R1 and R2 are retained as pointers to the 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. 1220, part of the instruction, 5830, 4599, 1200.

Solutions For the CO Tutorials 2

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
```

Solutions For the CO Tutorials 2

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) to A(0,x) and A(0,y) placed in R1 and R2.
	Multiply	#4,R2	
LOOP	Move	(R0,R1),R5	Add corresponding column elements.
	Add	R5,(R0,R2)	
	Add	R4,R1	Increment column element pointers by Stride value.
	Add	R4,R2	
	Decrement	R3	Repeat until all elements are added.
	Branch>0	LOOP	
	Move	(SP)+,R5	Restore R5.
	Return		Return to calling program.

Solutions For the CO Tutorials 2

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. (*a*)

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

(*b*)

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

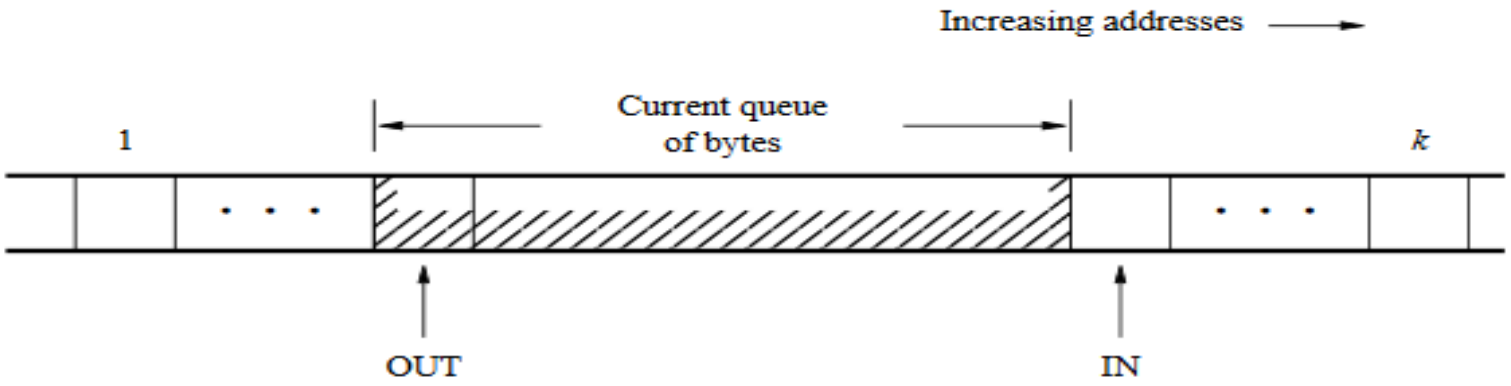
(*c*)

```
Add    #40,R5
```

Solutions For the CO Tutorials 2

2.18. (*a*) Wraparound must be used. That is, the next item must be entered at the beginning of the memory region, assuming that location is empty.

(*b*) A current queue of bytes is shown in the memory region from byte location 1 to byte location *k* in the following diagram.



The IN pointer points to the location where the next byte will be appended to the queue. If the queue is not full with *k* bytes, this location is empty, as shown in the diagram.

The OUT pointer points to the location containing the next byte to be removed from the queue. If the queue is not empty, this location contains a valid byte, as shown in the diagram.

Initially, the queue is empty and both IN and OUT point to location 1.

Solutions For the CO Tutorials 2

(*c*) Initially, as stated in Part *b*, when the queue is empty, both the IN and OUT pointers point to location 1. When the queue has been filled with k bytes and none of them have been removed, the OUT pointer still points to location 1. But the IN pointer must also be pointing to location 1, because (following the wraparound rule) it must point to the location where the next byte will be appended. Thus, in both cases, both pointers point to location 1; but in one case the queue is empty, and in the other case it is full.

(*d*) One way to resolve the problem in Part (*c*) is to maintain at least one empty location at all times. That is, an item cannot be appended to the queue if ([IN] + 1) Modulo k = [OUT]. If this is done, the queue is empty only when [IN] = [OUT].

(*e*) Append operation:

- LOC ← [IN]
- IN ← ([IN] + 1) Modulo k
- If [IN] = [OUT], queue is full. Restore contents of IN to contents of LOC and indicate failed append operation, that is, indicate that the queue was full. Otherwise, store new item at LOC.

Remove operation:

- If [IN] = [OUT], the queue is empty. Indicate failed remove operation, that is, indicate that the queue was empty. Otherwise, read the item pointed to by OUT and perform OUT ← ([OUT] + 1) Modulo k.