National University of Computer and Emerging Sciences



**Laboratory Manual**

*for*

*Computer Organization and Assembly Language*

Course Instructors

Lab Instructor(s)

Section

Semester

**Department of Computer Science**

# COAL Lab 12 Manual

|  |
| --- |
| **Objectives:**     * Revision * Extended Addition & Subtraction using ADC, SBB * Problems & Assignments |

**12.1 Extended Addition Subtraction**

The ADC (add with carry) instruction adds both a source operand and the contents of the Carry flag to a destination operand.

|  |  |
| --- | --- |
| **Example1:** Adding two 8-bit integers (FFh + FFh), producing a 16-bit sum in DL:AL, which is 01FEh.    **mov** **DL,0** **mov AL,0FFh**  **add AL,0FFh** ; AL = FEh **adc** **DL,0** ; DL/AL = 01FEh | **Example2:** Adding two 32-bit integers (FFFFFFFFh + FFFFFFFFh), producing a 64bit sum in EDX:EAX: 00000001FFFFFFFEh:    **mov EDX,0 mov EAX,0FFFFFFFFh add EAX,0FFFFFFFFh adc EDX,0** |

The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand.

64-bit subtraction

**mov EDX,7** ; upper half **mov EAX,1** ; lower half **sub EAX,2** ; subtract 2 **sbb EDX,0** ; subtract upper half

# Problem(s) / Assignment(s)

|  |  |
| --- | --- |
| **Discussion & Practice** | **Estimated completion time: 1 hr, 30 mins** |

|  |  |
| --- | --- |
| **PROBLEM 12.3:** *Finding relative prime numbers* | **Estimated completion** |

|  |  |
| --- | --- |
| **PROBLEM 12.1:** *Extended Addition*  Write a program that have following procedures,  **Extended\_Add,** calculates the sum of two extended integers stored as arrays of bytes.  **Display\_Sum,** displays the sum in its proper order, starting with the high-order byte, and working its way down to the low-order byte **Use the arrays,**  **op1 BYTE 34h,12h,98h,74h,06h,0A4h,0B2h,0A2h op2 BYTE 02h,45h,23h,00h,00h,87h,10h,80h**  **Sum comes out to be,**  **0122C32B0674BB5736**  **Solution:**  include Irvine32.inc  .data  op1 BYTE 34h,12h,98h,74h,06h,0A4h,0B2h,0A2h  op2 BYTE 02h,45h,23h,00h,00h,87h,10h,80h  temp dword 0  .code  main proc  call Extended\_Add  ;  call display\_sum  exit  main endp  Extended\_Add proc  pop temp  mov esi , offset op1  mov edi , offset op2  mov ecx ,8  l1:  mov eax,0  mov edx,0  mov al,[esi]  add al ,[edi]  adc al,0  ;;movzx ebx,al  ;;mov eax,0  ;;mov eax,ebx  push ax;  inc esi  inc edi  loop l1  push temp  ret  Extended\_Add endp  display\_sum proc  pop temp  mov ecx,8  l1:  pop ax  call writehexb  loop l1  push temp  ret  display\_sum endp  end main | **Estimated completion time:30mins** |

|  |  |
| --- | --- |
| **PROBLEM 12.2:** *Encryption using rotate operation*  Write a program that performs simple encryption by rotating each plaintext byte a varying number of positions in different directions.  For example, in the following array that represents the encryption key, a negative value indicates a rotation to the left and a positive value indicates a rotation to the right. The integer in each position indicates the magnitude of the rotation:  **key BYTE** -**2, 4, 1, 0,** -**3, 5, 2,** -**4,** -**4, 6**    Your program should loop through a plaintext message and align the key to the first 10 bytes of the message. Rotate each plaintext byte by the amount indicated by its matching key array value. Then, align the key to the next 10 bytes of the message and repeat the process.  Solution:  INCLUDE Irvine32.inc  .data  key byte -2,4,1,0,-3,5,2,-4,-4,6  input byte "abcdefghij",0  str2 byte 10 DUP(0)  var1 byte ?  .code  main proc  mov esi,offset key  mov edi,offset input  mov ecx,lengthof key  Eloop:  mov eax,[esi]  mov var1,al  cmp var1,0  jge l1  ;if cl(value of key'th index ) is negative  mov ebx,[edi]  rol bl,var1  mov al,bl  l1:  mov ebx,[esi]  ror bl,var1  mov ah,bl  inc esi  inc edi  loop Eloop  mov esi,offset str2  mov ecx,lengthof str2  mov ebx,type str2  call DumpMem  call waitmsg  exit  main endp  end main | **Estimated completion time:30mins** |

|  |  |
| --- | --- |
| Harry wants to check the relative primality of 2 numbers. For this purpose, he checks the GCD (Greatest Common Divisor) of the numbers. If GCD comes out 1 then numbers are relative prime to each other. Harry requirements are as follows:   1. Procedure **DEC\_IN** should load two registers (BX and DX) with two numbers. Numbers should be a 2 - digit decimal ranging from (01 - 99). 2. Procedure **GCD\_AB** apply the logic for GCD of two numbers. GCD of two numbers is performed by dividing the greater number (in BX) by the smaller number (in DX) till the remainder is zero. If it is zero, the divisor is the GCD if not the remainder and the divisor of the previous division are the new set of two numbers. The process is repeated by dividing greater of the two numbers by the smaller number till the remainder is zero and GCD is found. 3. Also check if,    1. The numbers are equal then GCD would be BX,    2. BX<DX then exchange the contents of 2 register. 4. Procedure **DEC\_OUT** should display the GCD on screen in decimal.     **Sample 1:**  Enter 1st Number: **20** Enter 2nd Number: **09**  GCD is: **1**  **Numbers are relative prime**  **Sample 2:**  Enter 1st Number: **09** Enter 2nd Number: **03**  GCD is: **03**  **Numbers are not relative prime**  Solution:  include Irvine32.inc  .data  str1 byte "enter the first number x ",0  str2 byte "enter the second number y ", 0  x dword 0  y dword 0  .code  main proc  mov edx,offset str1  call writestring  call readdec  mov x,eax  mov edx,offset str2  call writestring  call readdec  mov y, eax  mov eax,x  mov ebx,y  call great\_commen\_diviser  mov x,eax  call writedec  call crlf  CALL WAITMSG  exit  main endp  great\_commen\_diviser proc  mov ecx,-1  start:  mov edx, 0  idiv ebx  mov eax,ebx  mov ebx,edx  cmp ebx,0  jbe jump  loop start  jump:  ret  great\_commen\_diviser endp  end main | **time:30mins** |