HW # 8. Theme: Integer Arithmetic

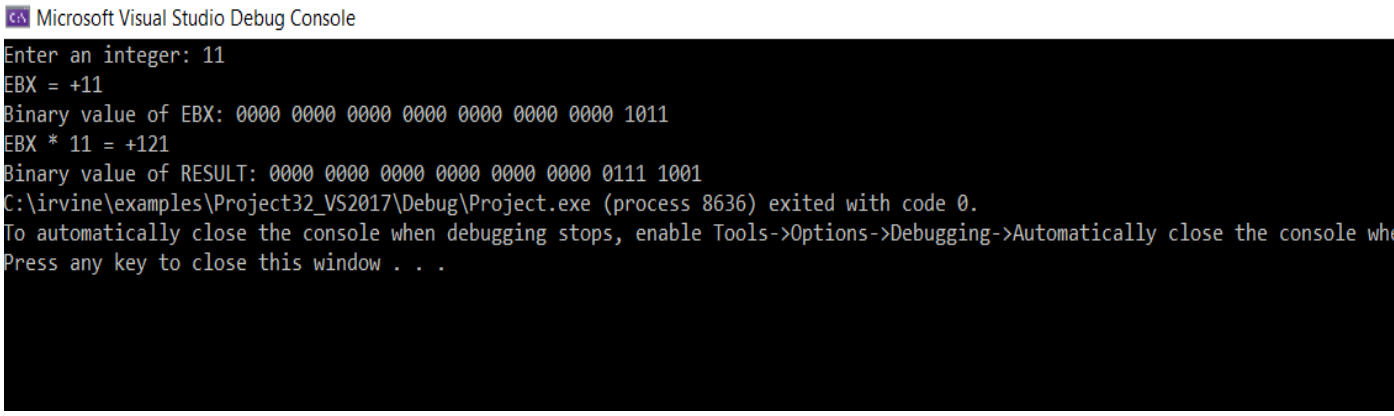
All main questions carry equal weight.

Points will be awarded to only those answers which have work clearly shown

1. In the following code sequence, show the value of AL after each shift or rotate instruction has executed. This question is to be done by hand, not by running a program.

```
mov cl, 1
mov al, 12h ;al = 0001 0010b
rol al, cl ; al = 0010 0100b
shl al, cl ; al = 0100 1000b
mov al, 34h; al = 0011 0100b
mov cl, 2
ror al, cl ; al = 0000 1101b
shr al, cl ; al = 0000 0011b
stc      ; CF = 1
mov al, 56h; al = 0101 0110b
mov cl, 1
rcl al, cl ; al = 1010 1101b
stc      ; CF = 1
mov al, 78h; al = 0111 1000b
mov cl, 1
rcr al, cl ; al = 1011 1100b
```

2. (a) Write a program which calculates EBX*11_{10} using binary multiplication.



Microsoft Visual Studio Debug Console

```
Enter an integer: 13
EBX = +13
Binary value of EBX: 0000 0000 0000 0000 0000 0000 0000 1101
EBX * 11 = +143
Binary value of RESULT: 0000 0000 0000 0000 0000 0000 1000 1111
C:\irvine\examples\Project32_VS2017\Debug\Project.exe (process 15784) exited with
To automatically close the console when debugging stops, enable Tools->Options->De
Press any key to close this window . . .
```

Microsoft Visual Studio Debug Console

```
Enter an integer: -12
EBX = -12
Binary value of EBX: 1111 1111 1111 1111 1111 1111 1111 0100
EBX * 11 = -132
Binary value of RESULT: 1111 1111 1111 1111 1111 1111 0111 1100
C:\irvine\examples\Project32_VS2017\Debug\Project.exe (process 6264) exited with
To automatically close the console when debugging stops, enable Tools->Options->
Press any key to close this window . . .
```

Microsoft Visual Studio Debug Console

```
Enter an integer: -15
EBX = -15
Binary value of EBX: 1111 1111 1111 1111 1111 1111 1111 0001
EBX * 11 = -165
Binary value of RESULT: 1111 1111 1111 1111 1111 1111 0101 1011
C:\irvine\examples\Project32_VS2017\Debug\Project.exe (process 7812) exited with c
To automatically close the console when debugging stops, enable Tools->Options->De
Press any key to close this window . . .
```

```
;hw8-2a.asm

INCLUDE Irvine32.inc

.data
string1 BYTE "Enter an integer: ", 0
message1 BYTE "EBX = ", 0
message2 BYTE "EBX * 11 = ", 0
message3 BYTE "Binary value of EBX: ", 0
message4 BYTE "Binary value of RESULT: ", 0
.code
```

```
main PROC
mov edx, OFFSET string1
call WriteString
call ReadInt
mov edx, OFFSET message1
call WriteString
call WriteInt
call Crlf
mov edx, OFFSET message3
call WriteString
call WriteBin
call Crlf
mov ebx, eax
shl ebx, 3
push ebx
mov ebx, eax
shl ebx, 1
shl eax, 0
add eax, ebx
pop ebx
add eax, ebx
mov edx, OFFSET message2
call WriteString
call WriteInt
call Crlf
mov edx, OFFSET message4
call WriteString
call WriteBin
invoke ExitProcess,0
main ENDP
end main
```

LST File:

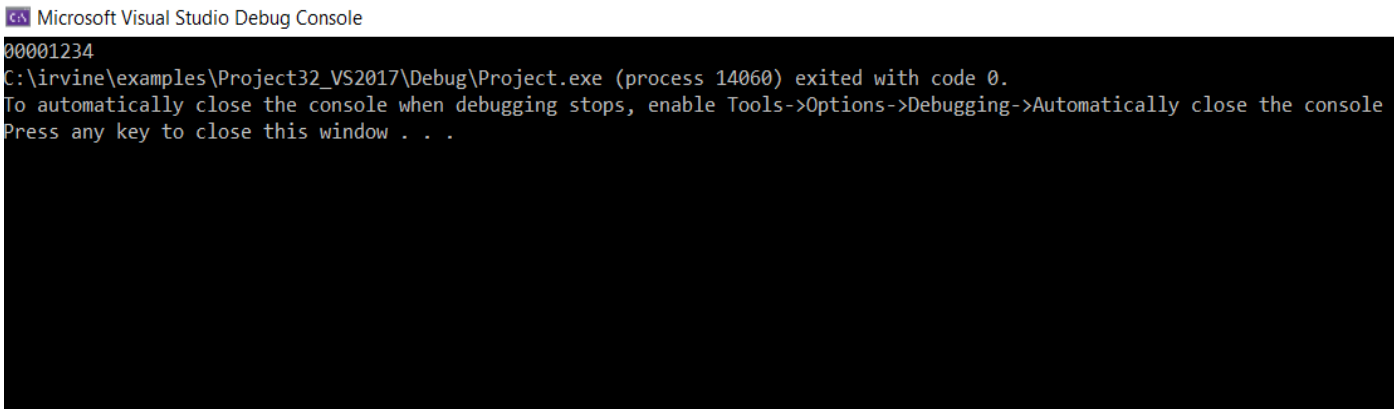
```
;hw8-2a.asm

                         INCLUDE Irvine32.inc
                        C ; Include file for Irvine32.lib             (Irvine32.inc)
                        C
                        C ;OPTION CASEMAP:NONE              ; optional: make
identifiers case-sensitive
                        C
                        C INCLUDE SmallWin.inc              ; MS-Windows prototypes,
structures, and constants
                        C .NOLIST
                        C .LIST
                        C
                        C INCLUDE VirtualKeys.inc
                        C ; VirtualKeys.inc
                        C .NOLIST
                        C .LIST
                        C
                        C
                        C .NOLIST
                        C .LIST
                        C

 00000000                .data
 00000000 45 6E 74 65 72  string1 BYTE "Enter an integer: ", 0
```

```
         20 61 6E 20 69
         6E 74 65 67 65
         72 3A 20 00
00000013 45 42 58 20 3D    message1 BYTE "EBX = ", 0
         20 00
0000001A 45 42 58 20 2A    message2 BYTE "EBX * 11 = ", 0
         20 31 31 20 3D
         20 00
00000026 42 69 6E 61 72    message3 BYTE "Binary value of EBX: ", 0
         79 20 76 61 6C
         75 65 20 6F 66
         20 45 42 58 3A
         20 00
0000003C 42 69 6E 61 72    message4 BYTE "Binary value of RESULT: ", 0
         79 20 76 61 6C
         75 65 20 6F 66
         20 52 45 53 55
         4C 54 3A 20 00
00000000                   .code
00000000                   main PROC
00000000  BA 00000000 R    mov edx, OFFSET string1
00000005  E8 00000000 E    call WriteString
0000000A  E8 00000000 E    call ReadInt
0000000F  BA 00000013 R    mov edx, OFFSET message1
00000014  E8 00000000 E    call WriteString
00000019  E8 00000000 E    call WriteInt
0000001E  E8 00000000 E    call Crlf
00000023  BA 00000026 R    mov edx, OFFSET message3
00000028  E8 00000000 E    call WriteString
0000002D  E8 00000000 E    call WriteBin
00000032  E8 00000000 E    call Crlf
00000037  8B D8            mov ebx, eax
00000039  C1 E3 03         shl ebx, 3
0000003C  53               push ebx
0000003D  8B D8            mov ebx, eax
0000003F  D1 E3            shl ebx, 1
00000041  C1 E0 00         shl eax, 0
00000044  03 C3            add eax, ebx
00000046  5B               pop ebx
00000047  03 C3            add eax, ebx
00000049  BA 0000001A R    mov edx, OFFSET message2
0000004E  E8 00000000 E    call WriteString
00000053  E8 00000000 E    call WriteInt
00000058  E8 00000000 E    call Crlf
0000005D  BA 0000003C R    mov edx, OFFSET message4
00000062  E8 00000000 E    call WriteString
00000067  E8 00000000 E    call WriteBin
                           invoke ExitProcess,0
0000006C  6A 00        *       push   +000000000h
0000006E  E8 00000000 E   *          call   ExitProcess
00000073                   main ENDP
                           end main
_Microsoft (R) Macro Assembler Version 14.28.29913.0       03/19/21 15:32:57
```

(b) Let the Consider the following value: A1BC23D4h. Let this value be stored in register EAX. Write a program that will extract the decimal digits from this value using shifts and logical instructions. Place the first two decimal numeric digits in DH and the other two into DL. Submit a screenshot of the console output of the program and the asm/lst file. Note that you are writing a program for this specific example where the letter and digit positions are known to you (you are NOT writing a generic program to separate the letters and digits) .




```
; (hw8-2b.asm)

INCLUDE Irvine32.inc

.data
binVal DWORD 0A1BC23D4h
.code
main PROC
       mov    eax,binVal                ; EAX = binary integer
       mov ebx, 00f00ff0fh
       AND eax, ebx
       rol al, 4
       rol eax, 4
       mov dl, ah
       rol eax, 16
       rol ax, 4
       rol al, 4
       mov dh, al
       movzx eax, dx
       call WriteHex
       exit
main ENDP
END main
```

```
                         ; (hw8-2b.asm)

                         INCLUDE Irvine32.inc
                        C ; Include file for Irvine32.lib            (Irvine32.inc)
                        C
                        C ;OPTION CASEMAP:NONE             ; optional: make
identifiers case-sensitive
                        C
                        C INCLUDE SmallWin.inc             ; MS-Windows prototypes,
structures, and constants
                        C .NOLIST
                        C .LIST
                        C
                        C INCLUDE VirtualKeys.inc
                        C ; VirtualKeys.inc
                        C .NOLIST
                        C .LIST
                        C
                        C
                        C .NOLIST
                        C .LIST
                        C

 00000000                .data
 00000000 A1BC23D4       binVal DWORD 0A1BC23D4h
 00000000                .code
 00000000                main PROC
 00000000  A1 00000000 R         mov    eax,binVal                ; EAX = binary
integer
 00000005  BB 0F00FF0F                  mov ebx, 00f00ff0fh
 0000000A  23 C3                AND eax, ebx
 0000000C  C0 C0 04             rol al, 4
 0000000F  C1 C0 04             rol eax, 4
 00000012  8A D4                mov dl, ah
 00000014  C1 C0 10             rol eax, 16
 00000017  66| C1 C0 04                rol ax, 4
 0000001B  C0 C0 04             rol al, 4
 0000001E  8A F0                mov dh, al
 00000020  0F B7 C2             movzx eax, dx
 00000023  E8 00000000 E        call WriteHex
                                exit
 00000028  6A 00        *       push   +000000000h
 0000002A  E8 00000000 E   *            call   ExitProcess
 0000002F                main ENDP
                         END main
_Microsoft (R) Macro Assembler Version 14.28.29913.0       03/18/21 15:20:54
hw8-2b.asm                                              Symbols 2 - 1
```

3. (a) What will be the contents of AX and DX after the following operation? What may happen if you do not set dx to 0 in the beginning? You must work this problem by hand, not by a program run.

```
mov dx, 0
mov ax, FFFAh ; ax = 15*16^3 + 15*16^2 + 15*16^1 + 10*16^0 =

                  = 15*4096 + 15*256  + 15*16   + 10*1

                  = 61440   + 3840    + 240     +  10

                  = 65530


mov cx, 5h    ; cx = 5*16^0 = 5


mul cx ; dx:ax = ax*cx = 65530 * 5 = 327650 = 0x0004FFE2;

dx = 0x0004; ax = 0xFFE2
```

DX is required along with AX to perform the MUL instruction because the resulting value may be larger than 16 bits, the size of either AX or DX alone.

By not clearing or setting the DX register to 0 before performing the MULT operation, there may be bits in dx that were not a result of the operation. Therefore, there is a risk of getting the incorrect answer by not initially clearing the DX register.

(b) When does an IDIV instruction cause an overflow? Provide an example.

If a division operand produces a quotient that will not fit into the destination operand, a divide overflow condition results.

Example:

mov ax, 1000h

cwd ; DX:AX = 00001000h

mov bl, 10h

idiv bl ; AL cannot hold 100h

The quotient 100h is too large

The above instruction generates an overflow because the quotient(100h) is too large for the 8-bit AL destination register.

(c) What will be the values of DX:AX after the following instructions execute? What might be the use of such a sequence of instructions in a 16-bit computer?

```
            mov ax, 0h
            mov dx, 0h
            sub ax, 3h  ; ax = -3 = FFFDh
            sbb dx, 0   ; DX:AX = 0000 FFFDh
```

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

The sequence of instructions above makes it possible to subtract two operands that are longer than the computer's word size(32-bits); thus, the size of these operands may be arbitrary.

4. Enter, assemble and execute the following program which implements a case table. Write a paragraph explaining how the code works. Expand the program to work with inputs 'A', 'B', 'C', 'D' and similarly 'E'. Test execute it. What is the disadvantage of manually putting a value for EntrySize and NumberofEntries instead of the way it is done in the program?

```
.data
         CaseTable BYTE 'A'       ;lookup value
                   DWORD Process_A  ;address of procedure
         EntrySize = ($ - CaseTable)
                   BYTE 'B'
                   DWORD Process_B
                   BYTE 'C'
                   DWORD Process_C
                   BYTE 'D'
                   DWORD Process_D
         NumberOfEntries = ($ - CaseTable) / EntrySize
         msgA BYTE “Process_A”, 0
         msgB BYTE “Process_B”, 0
         msgC BYTE “Process_C”, 0
         msgD BYTE “Process_D”, 0
         prompt BYTE “Press A, B, C or D:”, 0
.code
Main Proc
         Mov edx, offset prompt
         Call writestring
         Call readchar
         mov ebx,OFFSET CaseTable
         mov ecx, NumberofEntries
L1:      cmp al,[ebx]
         jne L2
         call NEAR PTR [Ebx + 1]
         call WriteString
         call Crlf
         jmp L3
```

```
L2:      add ebx, EntrySize
         loop L1
L3:      exit
main EndP

Process_A Proc
         Mov edx, offset msgA
         Ret
Process_A EndP

Process_B Proc
         Mov edx, offset msgB
         Ret
Process_B EndP

Process_C Proc
        Mov edx, offset msgC
        Ret
Process_C EndP

Process_D Proc
        Mov edx, offset msgD
        Ret
Process_D EndP
END main
```

How the code works:

Within the main procedure, the first instruction to be executed, mov edx, offset prompt, saves the address of the string prompt into the edx register. Then the next instruction, call writestring, will display the string “Press A, B, C, or D:”. The user will then be prompted to input a single character that will be stored in the al register. The offset of CaseTable defined in the data segment is stored in the ebx register. The variable NumberOfEntries, which is calculated by dividing the displacement between the offset of the variable NumberOfEntries and the offset of the variable CaseTable by the value of the variable EntrySize. The value of the variable NumberOfEntries is stored in the ecx register. The variable EntrySize is calculated by finding the displacement between the offset of the variable EntrySize and the offset of the variable CaseTable. In L1, the character input stored in the al register is compared to the first byte in the CaseTable ‘A’. If the input character stored in al is not equal to ‘A’, then jump to the target L2. L2 adds the EntrySize to the register ebx, then loops back to the first instruction in L1. L2 allows the next lookup value in the CaseTable, in this case ‘B’, to be compared for the next iteration of L1. Loop L1 will continue until the input value stored in the al register matches one of the lookup values in the CaseTable. If the operand stored in the al register does not match any of the lookup values in the CaseTable, nothing else will be printed. If the input character stored in al is equal to ‘A’, the procedure Process_A is called and the string “Process_A” is displayed. Loop L1 is also exited.

What is the disadvantage of manually putting a value for EntrySize and NumberofEntries instead of the way it is done in the program?

If the values of EntrySize and NumberofEntries were put in manually instead of the way it is done in the program, the programmer would have to manually adjust the EntrySize before each new iteration of the loop L1 to access different lookup values or entries in the case table.

```
 ; (hw8-4.asm)

INCLUDE Irvine32.inc

.data
CaseTable BYTE 'A'       ;lookup value
                  DWORD Process_A  ;address of procedure
         EntrySize = ($ - CaseTable)
                  BYTE 'B'
                  DWORD Process_B
                  BYTE 'C'
                  DWORD Process_C
                  BYTE 'D'
                  DWORD Process_D
                  BYTE 'E'
                  DWORD Process_E

         NumberOfEntries = ($ - CaseTable) / EntrySize
         msgA BYTE "Process_A", 0
         msgB BYTE "Process_B", 0
         msgC BYTE "Process_C", 0
         msgD BYTE "Process_D", 0
         msgE BYTE "Process_E", 0

         prompt BYTE "Press A, B, C, D, or E:", 0
.code
main PROC
         Mov edx, offset prompt
         Call writestring
         Call readchar
         mov ebx,OFFSET CaseTable
         mov ecx, NumberofEntries
L1:      cmp al,[ebx]
         jne L2
         call NEAR PTR [Ebx + 1]
         call WriteString
         call Crlf
         jmp L3
L2:      add ebx, EntrySize
         loop L1
L3:      exit
main ENDP
Process_A Proc
         Mov edx, offset msgA
         Ret
Process_A EndP
```

```
Process_B Proc
         Mov edx, offset msgB
         Ret
Process_B EndP

Process_C Proc
        Mov edx, offset msgC
        Ret
Process_C EndP

Process_D Proc
        Mov edx, offset msgD
        Ret
Process_D EndP

Process_E Proc
        Mov edx, offset msgE
        Ret
Process_E EndP

END main
```

LST File:

```
Microsoft (R) Macro Assembler Version 14.28.29913.0        03/20/21 14:50:09
hw8-4.asm                                                  Page 1 - 1


                            ; (hw8-4.asm)

                            INCLUDE Irvine32.inc
                           C ; Include file for Irvine32.lib             (Irvine32.inc)
                           C
                           C ;OPTION CASEMAP:NONE              ; optional: make
identifiers case-sensitive
                           C
                           C INCLUDE SmallWin.inc              ; MS-Windows prototypes,
structures, and constants
                           C .NOLIST
                           C .LIST
                           C
                           C INCLUDE VirtualKeys.inc
                           C ; VirtualKeys.inc
                           C .NOLIST
                           C .LIST
                           C
                           C
                           C .NOLIST
                           C .LIST
                           C

 00000000                   .data
 00000000 41                CaseTable BYTE 'A'       ;lookup value
 00000001  0000003B R                                DWORD Process_A  ;address of
procedure
 00000005 = 00000005                EntrySize = ($ - CaseTable)
```

```
00000005  42                                    BYTE 'B'
00000006  00000041 R                                  DWORD Process_B
0000000A  43                                    BYTE 'C'
0000000B  00000047 R                                  DWORD Process_C
0000000F  44                                    BYTE 'D'
00000010  0000004D R                                  DWORD Process_D
00000014  45                                    BYTE 'E'
00000015  00000053 R                                  DWORD Process_E
00000019
= 00000005                          NumberOfEntries = ($ - CaseTable) / EntrySize
00000019 50 72 6F 63 65             msgA BYTE "Process_A", 0
         73 73 5F 41 00
00000023 50 72 6F 63 65             msgB BYTE "Process_B", 0
         73 73 5F 42 00
0000002D 50 72 6F 63 65             msgC BYTE "Process_C", 0
         73 73 5F 43 00
00000037 50 72 6F 63 65             msgD BYTE "Process_D", 0
         73 73 5F 44 00
00000041 50 72 6F 63 65             msgE BYTE "Process_E", 0
         73 73 5F 45 00

0000004B 50 72 65 73 73             prompt BYTE "Press A, B, C, D, or E:", 0
         20 41 2C 20 42
         2C 20 43 2C 20
         44 2C 20 6F 72
         20 45 3A 00
00000000                    .code
00000000                    main PROC
00000000  BA 0000004B R             Mov edx, offset prompt
00000005  E8 00000000 E             Call writestring
0000000A  E8 00000000 E             Call readchar
0000000F  BB 00000000 R             mov ebx,OFFSET CaseTable
00000014  B9 00000005                       mov ecx, NumberofEntries
00000019  3A 03             L1:     cmp al,[ebx]
0000001B  75 0F                     jne L2
0000001D  FF 53 01                  call NEAR PTR [Ebx + 1]
00000020  E8 00000000 E             call WriteString
00000025  E8 00000000 E             call Crlf
0000002A  EB 08                     jmp L3
0000002C  81 C3 00000005    L2:     add ebx, EntrySize
00000032  E2 E5                     loop L1
00000034                    L3:     exit
00000034  6A 00         *       push   +000000000h
00000036  E8 00000000 E     *           call   ExitProcess
0000003B                    main ENDP
0000003B                    Process_A Proc
0000003B  BA 00000019 R             Mov edx, offset msgA
00000040  C3                        Ret
00000041                    Process_A EndP

00000041                    Process_B Proc
00000041  BA 00000023 R             Mov edx, offset msgB
00000046  C3                        Ret
00000047                    Process_B EndP

00000047                    Process_C Proc
00000047  BA 0000002D R            Mov edx, offset msgC
0000004C  C3                        Ret
```

```
 0000004D                   Process_C EndP

 0000004D                   Process_D Proc
 0000004D  BA 00000037 R            Mov edx, offset msgD
 00000052  C3                       Ret
 00000053                   Process_D EndP

 00000053                   Process_E Proc
 00000053  BA 00000041 R            Mov edx, offset msgE
 00000058  C3                       Ret
 00000059                   Process_E EndP

                            END main
_Microsoft (R) Macro Assembler Version 14.28.29913.0       03/20/21 14:50:09
hw8-4.asm                                                Symbols 2 - 1
```