Task#1

Write ASM instructions that calculate EAX * 25 using binary multiplication.

```
INCLUDE Irvine32.inc
.data
num DWORD ?
msg BYTE "Enter a number: ",0
msg1 BYTE "The value of eax after calculation is: ",0
cal DWORD ?
.code
main PROC
 ; EAX * (2^4 + 2^3 + 2^0)
 mov eax, 0
mov edx, OFFSET msg
call WriteString
call Crlf
call ReadInt
mov [num], eax
mov edx, eax
mov ebx, eax
shl edx, 4 ; EDX*2^4
shl ebx, 3 ; EBX *2^3
shl eax, 0 ; EAX*2^0
add eax, ebx
add eax, edx
mov [cal], eax
mov edx, OFFSET msg1
call WriteString
mov eax, [cal]
call WriteDec
call DumpRegs
exit
main ENDP
END main
```

```
Enter a number:
4
The value of eax after calculation is: 100
  EAX=00000064  EBX=00000020  ECX=007410AA  EDX=00746015
  ESI=007410AA  EDI=007410AA  EBP=00AFFC68  ESP=00AFFC5C
  EIP=007436AD  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (
```

4 * 25 = 100

Task#2

Give an assembly language program to move -128 in BX and expand EBX. Using shift and rotate instruction.

```
INCLUDE Irvine32.inc
.code
main PROC
mov bx, -128
movsx ebx, bx
shl ebx, 16
sar ebx, 16
mov eax, ebx
call DumpRegs
exit
main ENDP
END main
```

```
EAX=FFFFFF80  EBX=FFFFFF80  ECX=005D10AA  EDX=005D10AA
ESI=005D10AA  EDI=005D10AA  EBP=0095F88C  ESP=0095F880
EIP=005D3674  EFL=00000282  CF=0  SF=1  ZF=0  OF=0  AF=0  PF=0


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (proc
```

Using shift and rotate instruction:

```
INCLUDE Irvine32.inc
.code
main PROC
 mov bx, -128
 shl ebx, 16
    ror ebx, 16
    or ebx, 0FFFF0000h
    mov eax, ebx
call DumpRegs
exit
main ENDP
END main
```

```
EAX=FFFFFF80  EBX=FFFFFF80  ECX=007E10AA  EDX=007E10AA
ESI=007E10AA  EDI=007E10AA  EBP=00EFFCD8  ESP=00EFFCCC
EIP=007E3677  EFL=00000282  CF=0  SF=1  ZF=0  OF=0  AF=0  PF=0


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (proc
```

Task#3

The time stamp field of a file directory entry uses bits 0 through 4 for the seconds, bits 5 through 10 for the minutes, and bits 11 through 15 for the hours. Write instructions that extract the minutes and copy the value to a byte variable named **bMinutes**.

```
TITLE EXTRACTING MINUTES
INCLUDE Irvine32.inc
.data
   bMinutes BYTE ?
   msg2 BYTE "The minutes are: ",0
   timestamp WORD 0010011001101010b
.code
main PROC
   mov dx, [timestamp]
   mov ax, dx
   shr ax, 5
   and ax, 00111111b
   mov [bMinutes], al
   mov edx, OFFSET msg2
   call WriteString
movzx eax, [bMinutes]
call WriteDec
call DumpRegs
exit
main ENDP
END main
```

```
The minutes are: 51
  EAX=00000033  EBX=00357000  ECX=007310AA  EDX=00736001
  ESI=007310AA  EDI=007310AA  EBP=001FFB7C  ESP=001FFB70
  EIP=00733692  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (proc
```

Task#4

Write a series of instructions that shift the lowest bit of AX into the highest bit of BX without using the SHRD instruction. Next, perform the same operation using SHRD.

```
INCLUDE Irvine32.inc
.data
msg1 BYTE "using shrd : ", 0
msg2 BYTE "without using shrd : ", 0

.code
main PROC

mov eax, 0
mov ebx, 0

mov ax, 1111000011110000b
mov bx, 0000111100001111b

shr bx, 1
rcr ax, 1

mov edx, OFFSET msg2
call writestring
call writebin

call crlf
call crlf

mov edx, OFFSET msg1
call writestring

mov ax, 1111000011110000b
mov bx, 0000111100001111b

shrd ax, bx, 1
call writebin
```

```
 shrd ax, bx, 1
 call writebin

 call Crlf

 exit
 main ENDP
 END main
```

```
without using shrd : 0000 0000 0000 0000 1111 1000 0111 1000

using shrd : 0000 0000 0000 0000 1111 1000 0111 1000

C:\Users\k2308\2\source\repos\COALlab10\Debug\COALlab10.exe (process 1
```

Task#5

Implement the following C++ expression in assembly language, using 32-bit **signed** operands:

val1 = (val2 / val3) * (val1 / val2);

```
TITLE Expression
INCLUDE Irvine32.inc
.data
   val1 SDWORD 2
   val2 SDWORD 2
   val3 SDWORD 1
   quotient2 SDWORD ?
   quotient1 SDWORD ?

.code
main PROC
  mov eax, [val1]
  cdq
  mov ebx, [val2]
  idiv ebx
  mov [quotient2], eax

   mov eax, [val2]
   cdq
  mov ebx, [val3]
  idiv ebx
  mov [quotient1], eax

  mov eax, [quotient1]
  mov ebx, [quotient2]
  imul ebx
  mov [val1], eax
  call WriteDec

call DumpRegs
exit
main ENDP
END main
```

```
2
  EAX=00000002  EBX=00000001  ECX=00B410AA  EDX=00000000
  ESI=00B410AA  EDI=00B410AA  EBP=00EFFCDC  ESP=00EFFCD0
  EIP=00B436A2  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  P


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (
```

Task#6

Create a procedure **Extended_Add** procedure to add two 64-bit (8-byte) integers.

```
INCLUDE Irvine32.inc
.data
op1 QWORD 0A2B2A40674981234h
op2 QWORD 08010870000234502h
sum DWORD 3 dup(?)
.code
main PROC
mov esi, OFFSET op1
mov edi, OFFSET op2
mov ebx, OFFSET sum
mov ecx, 2
call Extended_Add
mov ecx, 3
mov esi, OFFSET sum
add esi, SIZEOF sum
sub esi, 4

l2:
mov eax, [esi]
call writehex
sub esi, 4
loop l2
exit
main ENDP
```

```
Extended_Add PROC
l1:
mov eax, [esi]
adc eax, [edi]
pushfd
mov [ebx], eax
add esi, 4
add edi, 4
add ebx, 4
popfd
loop l1

adc word ptr [ebx], 0
ret
Extended_Add ENDP
END main
```

```
0000000122C32B0674BB5736
C:\Users\k230842\source\repos\COALlab10\Debug
To automatically close the console when debug
```

Task#7

Write a procedure named **IsPrime** that sets the Zero flag if the 32-bit integer passed in the EAX register is prime. Optimize the program's loop to run as efficiently as possible. Write a test program that prompts the user for an integer, calls **IsPrime**, and displays a message indicating whether the value is prime. Continue prompting the user for integers and calling **IsPrime** until the user enters 1.

```
INCLUDE Irvine32.inc
.data
num DWORD ?
count DWORD 0
msg BYTE "Enter a number: ",0
msgprime BYTE "The number is prime.",0
msgnotprime BYTE "The number is not prime.",0
.code
main PROC
mov eax, 0
L1:
mov edx, OFFSET msg
call WriteString
call Crlf
call ReadInt
mov [num], eax
cmp [num], 1
je _exit
mov eax, [num]
call isPrime
jnz display_notprime        ; If Zero flag is not set, number is not prime
mov edx, OFFSET msgprime
call WriteString
call Crlf
jmp L1
```

```
display_notprime:
mov edx, OFFSET msgnotprime
call WriteString
call Crlf
jmp L1
_exit:
exit
main ENDP

isPrime PROC
cmp eax, 2
je prime
jl notprime
mov [count], 0
mov ecx, 2
mov ebx, eax
L2:
```

```
mov ebx, eax
L2:
mov eax, ecx
imul eax, ecx     ; Calculate ecx * ecx
cmp eax, ebx
jg check           ; If ecx^2 > number, it is prime
mov eax, ebx
cdq
div ecx
cmp edx, 0 ; if remainder is zero
jne next
inc [count]
next:
inc ecx
jmp L2
```

```
check:
cmp [count], 0
ja notprime
jmp prime

notprime:
mov eax, 1
jmp _ex
prime:
xor eax, eax ; this sets the zero flag

_ex:
ret
isPrime ENDP
END main
```

```
Enter a number:
10
The number is not prime.
Enter a number:
12
The number is not prime.
Enter a number:
2
The number is prime.
Enter a number:
3
The number is prime.
Enter a number:
4
The number is not prime.
Enter a number:
1

C:\Users\k230842\source\repos\COALl
```

Task#8

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.

```
INCLUDE Irvine32.inc
.data
key BYTE 2, 4, 1, 0, 3, 5, 2, 4, 4, 6
plaintext BYTE "Hello world", 0
encrypted BYTE 50 dup(0)
.code
main PROC
mov esi, OFFSET plaintext
mov edi, OFFSET encrypted
mov ecx, 50
mov ebx, 0
encryptionLoop:
mov al, byte ptr [esi]
cmp al, 0
je done
mov dl, byte ptr [key + ebx]
call rotateByte
mov [edi], al
inc esi
inc edi
inc ebx
cmp ebx, 10
jl encryptionLoop

mov ebx, 0
loop encryptionLoop

done:
mov edx, OFFSET encrypted
call writestring
exit
main ENDP
```

```
rotateByte proc
mov cl, dl
cmp dl, 0
jge rotateRight
neg dl

mov cl, dl
shl al, cl
rol al, cl
ret
rotateRight:
mov cl, dl
mov dl, cl
shr al, cl
mov cl, dl
rol al, cl
ret
rotateByte ENDP

END main
```