COMPOTER ORGANSATION AND

ASSEMBLY LANGUAGE. Date

BASIL ALI KHAN
20K-0477

ASSIGNMENT

Q#01

(i)

a). 0000007Ah
0111 1010b
NOT → 1000 0101b.
AL ⇒ [00000085h] Ans.

b) 00000030h 00000074h
0011 1101b 0111 0100b.
00111101b
AND → 01110100b
00110100b
AL ⇒ [00000034h] Ans.

c) 0000009Bh. 00000035h.
10011011b. 00110101b
10011011h
OR → 00110101h
10111111b
AL ⇒ [000000BFh] Ans.

d) 00000072h. 000000DCh.
0111 0010b. 1101 1100b
0111 0010b
XOR → 11011100b
10101110b
AL ⇒ [000000AEh] Ans

Date

(ii)

In mov eax, offset var1, the instruction is returning the address of a data label or offset. offset is the displacement of number of bytes from the beginning address of data segment.

In mov eax, var1, this instruction is copying the value stored in var1 in eax registers.

(iii)

1) A stack makes a convenient temporary storage area for registers when they are used.
2) When a subroutine is called the CPU saves the current subroutine return address.
3) Stack also stores local variable.
4) Stack is also helpful in passing and returning values from procedure.

(IV)

a. EAX : 00009ABCh.
b. EBX : 00009ABCh
c. ECX : 000056E8h.
d. ESP : 00000F4h.

(V)

additional instruction are :

```
push esi.
push ecx.
:
pop ecx
pop esi
```

Signature

No.

Date

(vi)

A, B, C, E are illegal instruction.

(vii)

444 bytes.

(viii)

(ix)

| | AH = 1 | AH = 2 | AH = 3 |
|---|---|---|---|
| If $V_1 = V_2$ then | | | ✓ |
| If $V_1 < V_2$ the | | ✓ | |
| If $V_1 > V_2$ then. | ✓ | | |

Signature

No.

Date

(x)

ESP : 0000 1FF 8h
EBP : 0000 1FF 8h.
EAX : 0000 0011h.
ESP : 0000 2010h.
X2 : 38h
X8 : 36h
EIP : 1150 0000h.

Q#2

i. Eax = 2
ii. Eax = 4
iii. Eax = 48
iv. Eax = 404002
v. Eax = 404006
vi. Eax = 404007

(vii)

No, because mov instruction doesnt affect any flags.

(viii)

Yes.

(ix)

No Sytax error

(x)

AL = 00000038h.

(xi)

EAX = 0006534h.

(xii)

EBX = 00030000h.

Signature

No.

Date

(xiii)

NO, instruction operand must have same size.

(xiv)

No, Both are memory location.

(xv)

EAX = J.

(xvi)

No, Both are memory location

(xvii)

No, istruction operand must have same size

(xviii)

AX = FFFFFFF6h. (-10)

(xix)

Not in syllabus.

(xx)

a. al = 0000006Ah.

b. al = 000000EAh.

(xxi)

a. CF = 0

b. OF = 0

c. SF = 1

d. ZF = 0

(xxii)

a. CF = 0

b. OF = 0

c. SF = 0

d. SF = 0

(xxiii)

a. CF = 1

b. OF = 0

c. SF = 1

d. ZF 0

(xxiv)

Eax : 0000FFFFh.

Signature

No.

For JB :

```
mov eax, 7FFFh
mov ebx, 8000h.
cmp eax, ebx
Jb L1
mov x, 2
Jmp L100
L1:
    mov x, 1
L100:
    exit
main ENDP
END MAIN
```

FOR JL :

```
mov eax, 7FFh
mov ebx, 8000h
cmp eax, ebx
Jl L1 ; mov x, 2
JMP L100
L1 :
    mov x, 1
L100 :
    exit
main ENDP
END main
```

QUESTION #03 :

(a)

```
while :
    cmp op1, op2
    Jg end
    cmp op3, op2
    Je L1
    add y, 10
    Jmp L100
    L1 :
     add y, 2
    L100 :
    Jmp while.
    end :
```

Date

(b)

```
.data
Val1 DWORD 20
Val2 DWORD 14
Val3 DWORD 12.
x.   DWORD ?
.code
main PROC
    mov eax, Val1
    mov ebx, Val2.
    cmp eax, ebx.
    Jg L1
    mov x, 2.
    Jmp L100
    L1:
      cmp ebx, Val3.
      Jg L2.
      mov x, 2.
      Jmp L100
      L2:
        mov x, 10.
      L100:
        exit
main ENDP
END main.
```

(iii)

```
.data
x DWORD 10
Y DWORD 20
Z DWORD 30
.code
main PROC.
```

Date

```
mov ax, x
mov bx, y
mov cx, z
call Minimum
call writedec
main ENDP

Minimum PROC
  cmp ax, bx
  JI  L1
   cmp bx, cx
  JI  L2          -> L3
  mov eax, cx
  Jmp L100
  L1:
    cmpax, cx
    JL  L4
    Jmp L3
  L2:
    cmp bx, cx
    JL  L5
    Jmp L3
  L4:
    mov eax, ax
    Jmp L100
  L5:
    mov eax, ax
    Jmp L100
  L100:
    retn
Minimum PROC ENDP
END main
```

Signature

No.

Date

(iv)

```
INCLUDE Irvine32.inc

.code
    Push 5
    Push 2
    call Binomial
    mov eax, edx
    call WriteDec
exit
main ENDP

Binomial PROC
    Push ebp
    mov ebp, esp
    push eax
    push ebx
    mov eax, [ebp + 8]
    mov ebx, [ebp + 12].
    cmp eax, ebx
    Ja Base1
    Cmp eax, 0
    Je Case2
    dec eax
    Sub ebx, 1
    push ebx
    push eax
    call Binomial
    push edx
    inc eax
    push ebx
    push eax
    call Binomial
    pop ebx
    add ebx, edx
    Jmp end

case1:
    mov edx, 0
    Jmp end
case2:
    mov edx, 1
end:
    pop ebx
    pop eax
    pop ebp
    ret 8
Binomial ENDP
END main.
```

Date

(b)

```
INCLUDE Irvine32.inc
.code -> main PROC
  push 5
  push 2
  call Power
  call WriteDec
exit
main ENDP

POWER PROC
  push ebp
  mov ebp, esp
  push ebx
  mov ebx, [ebp+8]
  mov ecx, [ebp+12]
  cmp ebx, 0
  Je Case
  sub ebx, 1
  push ecx
  push ebx
  call Power
  mul ecx
  Jmp end1
  case:
    mov eax, 1
  end1:
    pop ebx
    pop ebp
    ret 8
POWER endP
END main
```

Data

(V)

```
INCLUDE Irvine32.inc
.code
  call readInt
  push eax
  call Factorial
  call WriteDec
exit
main ENDP

Factorial PROC
  push ebp
  mov esp, esp
  mov eax, [esp + 8]
  cmp eax, 0
  Jg  L1
  mov eax, 1
  Jmp L2
  L1:
    dec eax
    push eax
    call Factorial
    mov ebx, [ebp + 8]
    mul ebx
  L2:
    pop ebp
    ret 4
Factorial endP
```

Date

(VI)

(Exchange Sort)

```
INCLUDE Irvine32.inc.
.data
arr WORD 10, 4, 7, 14, 3.
Count dWORD - lengthof arr.
index DWORD 0.
.code
main PROC.
mov eax, 0
mov esi, 0.
mov ecx, lengthof arr.
mov ebx, 0
mov edx, 0.
L1:
    push ecx
    mov ecx, count
    mov edi, index
    L2:
        mov dx, arr[esi].
        cmp dx, arr[edi].
        Jg swap.
        Jmp L100.
        swap:
            mov bx, arr[edi].
            mov arr[esi], bx
            mov arr[edi], dx.
    L100:
        add edi, type arr.
  Loop L2.
add index 2
dec count.
```

Date

```
add esi, type arr
pop ecx
Loop L1
exit
main ENDP
END main.
```

(vii)

(SELECTION SORT)

```
INCLUDE Irvine32.inc

.data
i DWORD ?
count DWORD @ length of arr.
.code
   mov ecx, count - 1
   Lbl: mov esi, 0
   L1:
      mov i, esi
      push ecx
      mov ecx, count
      Lbl: mov edi, esi + type arr
      L2: mov eax, arr[esi].
         cmp arr[edi], eax
         jl index
         Jmp L2
         index:
         mov i, edi
         mov edx, arr[i]
         mov arr[i], arr[esi]
         mov arr[esi], edx
```

Date

```
    add edi, type arr
  Loop L2
add esi, typ arr
pop ecx
Loop L1
exit
main ENDP
END main
```

Q#4

```
INCLUDE Irvine32.inc
.code
  mov bx, ax
  add bx, 0000111111111111b
  mov sequence_Number, bx

  mov bx, ax
  shr bx, 12
  and bx, 0000000000001111b
  mov Revision_Count, bx

  mov bx, ax
  shr bx, 15
  mov Status, bx

  mov ebx, eax
  shr ebx, 6
  mov Sensor_Data, ebx
```

(i) .data

sequence_Number WORD ?
Revision_Count WORD ?
Status HORD ?
Sensor_Data DWORD ?

Date

(ii)

```
.data
X DWORD ?

.code
main PROC.
    mov eax, X.
    mov ebx, eax
    mov ecx, eax.
    mov edx, eax.
    shl eax, 4
    shl ebx, 2
    shl ecx, 1.
    add eax, ebx.
    add eax, ecx
    add eax, x.
call writedec
exit
main ENDP
END main.
```

(iii)

- ESP : 0000 1FF8h
ESP : 0000 1FF8h.
EAX : 11h.
ESP : 0000 2010h.
$X_2$ : 38h
$X_3$ : 36h.
EIP : 11500020h.

Date

(iv)

```
INCLUDE Irvine32.inc

.data
source BYTE "This is a source string", 0
target BYTE ?
temp.  BYTE ?
.
.code
   mov esi, 0.
   mov ebx, length of source - 1.
   L1:
      push ecx
      mov edx, source[esi].
      mov ecx length of temp.
      L2:
         mov ebx, temp[edi].
         cmp ebx, edx
         Je L3.
         add edi, type source
      Loop L2.
      Jmp L4.
      L3:
         mov target[esi], edx.
      L4:
       add esi, 4
      pop ecx
 Loop L1
exit
main ENDP
END main
```

Date

(V)

```
INCLUDE Irvine32.inc

.data
string  BYTE  128 DUP(0 "),
rev     BYTE  128 DUP(0 "),

.code
main PROC
    mov edx, offset string.
    mov ecx, ~~128~~ length of string
    call readstring.
    mov ecx, eax.
    mov esi, offset string.
    L1:
      mov dl, [esi].
      cmp dl, 32
      Je L2
       add dl, 32
      mov [esi], dl
      L2:
        push [esi].
        add esi, type string.
  Loop L1
  mov ecx, eax.
  mov esi, offset rev
  L3:
    pop [esi]
    add esi, type rev.
  Loop L3
exit
End main
```

~~mov [esi] BYTE PTR~~
~~mov edx offset rev~~
~~call exit~~