Assignment No: 2

Fahid Imran

Roll No: 23i-0061

COAL

March 24, 2025

Instructor

Ms. Farheen Tabassum

Fast NUCES Islamabad

Campus

ID: 23i-0061 NAME: Fahid Imran SECTION: AI (B)

Read the Instructions Carefully

- Assignments are designed for your learning copying will lead to plagiarism. Practicing the questions yourself will help you in your exams. Fill the memory and register tables in the Word file first before running the respective programs in visual studio.
- The filled memory should represent the memory status of the executed program.
- Memory and register values should be in hexadecimal format.
- Each row in the memory table consists of 16 columns, each of 1 byte. The row indicates the starting offset. Both the starting offset and the columns are represented with hexadecimal values.
- You are supposed to debug each question in Visual Studio after filling the tables.
- Submit a **PDF** File with the naming format **23i-1234_Section_(AI,DS)**
- If you find any **ERROR** in the program highlight the error and write the justification for the error. You must understand ERROR carefully. After identifying the error, you will ignore it by commenting it out and running the program. Also, the register values corresponding to the erroneous statements must be left blank. For example:
  ; mov al, 256 ERROR: 256 is out of range for an 8-bit register like `AL`. It should be within 0-255.
- Data Declarations highlighted in yellow must be replaced as instructed below:

F1 = 'F'
F2 = 'A'
F3 = 'H'
F4 = 'I'

L1 = 'g'
L2 = 'M'
L3 = 'R'
L4 = 'A'

R = 0061
R(d) = 0061
R(h) = 03Dh
R(o) = 75o
R(b) = 1111010b

R1(d) = 00 R1(h) = 00 R1(o) = 00 R1(b) = 00
R2(d) = 61 R2(h) = 03Dh R2(o) = 75o R2(b) = 1111010b

RR(d) = 00610061
RR(h) = 094F0Dh
RR(o) = 22474154
RR(b) = 10010100111100001101

A = 41
g = 49
M = 4d

- ✓ 'F1' represents the first character of your first name, 'F2' represents the second character, 'F3' and 'F4' represent the third and the fourth characters of your first name, respectively. If your first name has less characters, then repeat the previous character(s) as needed.

- ✓ 'L1' represents the first character of your last name, 'L2' represents the second character, 'L3' and 'L4' represent the third and the fourth characters of your last name, respectively. If your last name has less characters, then repeat the previous character(s) as needed.

- ✓ 'R' represents the numeric part of your roll number. For example, if your roll number is 23i-1234, then 'R' is 1234. The base for 'R' is represented as (base), where the base can be 'h' for hexadecimal, 'd' for decimal, 'b' for binary, and 'o' for octal. You must first treat 'R' as a decimal number, then convert it into the specified base before substituting the value.

  When R is 1234, then R(d) = 1234d

  R(h) = 04D2h

  R(o) = 2322o

  R(b) =

  010011010010b

- ✓ 'R1' and 'R2' represent the first and the second part of 'R', respectively. For example, if 'R' is 1234, then 'R1' is 12 and 'R2' is 34. The base for 'R1' and 'R2' are represented as (base), where the base can be 'h' for hexadecimal, 'd' for decimal, 'b' for binary, and 'o' for octal. You must first treat 'R1' and 'R2' as decimal numbers, then convert them into the specified bases before substituting the values.

- ✓ 'RR' represents the concatenated roll number. For example, if your roll number is 23i-1234 then 'RR' is 12341234. The "base" rules defined above also apply here.

COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE
ASSIGNMENT#2 (SPRING 2025)

1. Write equivalent C code for the assembly code given below. Update the final value of registers in HEX

ASM Program

```
.data
      temp db 0
.code


main Proc
       mov ax,0
       mov ecx,5
       Outer:
       mov temp,cl
       mov ecx,3
       Inner:
             add ax,1
             Loop Inner
       mov cx,0
       mov cl,temp
       Loop Outer

main ENDP
END main
```

```
int main() {
    int Sum = 0;
    int n1 = 5;
    int n2 = 3;
    for(int i=0 ; i<n1 ; i++){

        for(int j=0 ; j<n2 ; j++){

            Sum+=1;
        }
    }
    retur 0;
}
```

2. Update Flags after executing following code.

NOTE: AND performs a Boolean(bitwise) **AND** operation between each pair of matching bits in two operands and place result in the destination. **AND** instruction always clear overflow and carry flags. It modifies Sign, Zero and Parity flags.

```
mov al,0AEH
and al,246
```

	ah	al
AX		AE
CX		a6

0AEh = 1 0 1 0 1 1 1 0

246 = 1 1 1 1 0 1 1 0

And 1 0 1 0 0 1 1 0

A 6

2

Flags	
Overflow	0
Carry	0
Sign	1
Parity	1
Zero	0

Register Values (Question No 1):

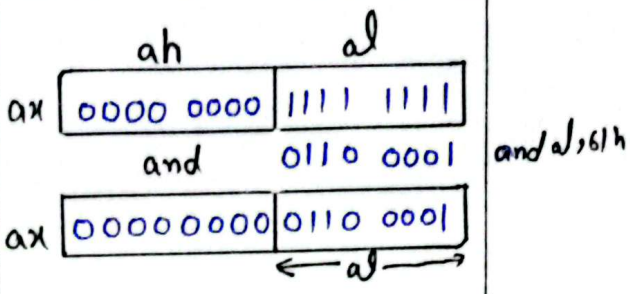
comment	ecx	ax	Temp (memory)
	0 00000005 h	0 0000 h	0 00 h
Outer 1 Inner 1	0 00000003 h	0 0001 h	0 05 h
Outer 1 Inner 2	0 00000002 h	0 0002 h	0 05 h
Outer 1 Inner 3	0 00000001 h	0 0003 h	0 05 h
Outer 1 Inner loop terminated	0 00000000 h	0 0003 h	0 05 h
mov cx,0 mov cl,temp	0 00000005 h	0 0003 h	0 05 h
	0 00000004 h	0 0003 h	0 05 h
Outer 2 Inner 1	0 00000003 h	0 0004 h	0 04 h
Outer 2 Inner 2	0 00000002 h	0 0005 h	0 04 h
Outer 2 Inner 3	0 00000001 h	0 0006 h	0 04 h
Outer 2 Inner loop terminated	0 00000000 h	0 0006 h	0 04 h
mov cx,0 mov cl,temp	0 00000004 h	0 0006 h	0 04 h
	0 00000003 h	0 0006 h	0 04 h
Outer 3 Inner 1	0 00000003 h	0 0007 h	0 03 h
Outer 3 Inner 2	0 00000002 h	0 0008 h	0 03 h
Outer 3 Inner 3	0 00000001 h	0 0009 h	0 03 h
Outer 3 Inner loop terminated	0 00000000 h	0 0009 h	0 03 h

mov cx,0 mov cl,temp	0 00000003 h	0 0009 h	0 03 h
	0 00000002 h	0 0009 h	0 03 h
Outer 4 Inner 1	0 00000003 h	0 000a h	0 02 h
Outer 4 Inner 2	0 00000002 h	0 000b h	0 02 h
Outer 4 Inner 3	0 00000001 h	0 000c h	0 02 h
Outer 4 Inner loop terminated	0 00000000 h	0 000c h	0 02 h
mov cx,0 mov cl,temp	0 00000002 h	0 000c h	0 02 h
	0 00000001 h	0 000c h	0 02 h
Outer 5 Inner 1	0 00000003 h	0 000d h	0 01 h
Outer 5 Inner 2	0 00000002 h	0 000e h	0 01 h
Outer 5 Inner 3	0 00000001 h	0 000f h	0 01 h
Outer 5 Inner loop terminated	0 00000000 h	0 000f h	0 01 h
mov cx,0 mov cl,temp	0 00000001 h	0 000f h	0 01 h
Outer Loop Terminated	0 00000000 h	0 000f h	0 01 h

3. Update memory after executing code given below

ASM Program

```
.data
      ary db 26 dup(?)
      arysize = $ - ary
      ary_copy db arysize dup(0)
      endmem db 1

.code
main Proc
       mov esi, OFFSET ary
       mov edi, OFFSET ary_copy
       mov ax, 000FFh
       mov ecx, arysize
       and al, 061h
       mov bl, al
       and bl, 0CFh ;masking
       L1:
            mov [esi], al
            mov [edi], bl
            inc esi
            inc edi
            inc al
            inc bl
       Loop L1

main ENDP
END main
```



bl → 0110 0001
and 1100 1111
bl → 0100 0001

al = 0110 0001 = 61h
bl = 0100 0001 = 41h

MEMORY

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00404000	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f	70
00404010	71	72	73	74	75	76	77	78	79	7a	41	42	43	44	45	46
00404020	47	48	49	4a	4b	4c	4d	4e	4f	50	51	52	53	54	55	56
00404030	57	58	59	5a	01											

ary
ary_copy
endmem

4. Update memory after executing code given below

ASM Program

```
.data
      ary db 26 dup(?)
      arysize = $ - ary
      ary_copy db arysize dup(0)
      endmem db 1

.code
main Proc
       mov esi, OFFSET ary
       mov edi, OFFSET ary_copy
       mov ax, 0041h
       mov ecx, arysize
       mov bl, al
       or bl, 00100000b
       L1:
            mov [esi], al
            mov [edi], bl
            inc esi
            inc edi
            inc al
            inc bl
       Loop L1
main  ENDP
END main
```

al

ax = 00 41

bl = 41

0 100 0001

OR 0010 0000

0110 0001

6 1

bl = 61h

al = 41h

MEMORY

add

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00404000	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f	50
00404010	51	52	53	54	55	56	57	58	59	5a	61	62	63	64	65	66
00404020	67	68	69	6a	6b	6c	6d	6e	6f	70	71	72	73	74	75	76
00404030	7	78	79	7a	01											

↳ endmem

5. Update Flags after executing following code

NOTE: OR performs a Boolean(bitwise) **OR** operation between each pair of matching bits in two operands and place result in the destination. **OR** instruction always clear overflow and carry flags. It modifies Sign, Zero and Parity flags.

```
mov al, 11100011b
OR al,00000100b ;Setting 3rd bit
```

Flags		
	Sign	1
	Zero	0
	Carry	0
	Overflow	0
	Parity	1
	Auxiliary	0

Rough Work

```
  1 1 1 0 0 0 1 1
  0 0 0 0 0 1 0 0      even_parity = 1
OR
  1 1 1 0 0 1 1 1
```

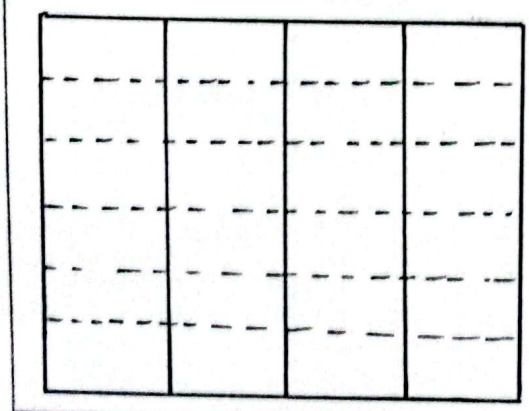
6. Update register after each line of code and update Flags after executing XOR AL,AH

```
mov ax,0A593H
XOR ax,-1
XOR ax,0
XOR ax,-1
XOR ax,ax

mov ax, 05A37H
XOR al,0

XOR ah,1
XOR al,ah
;line will check parity
;for 16 bit integer
```

	ah	al
AX	A5	93
AX	5A	6C
AX	5A	6C
AX	A5	93
AX	00	00
AX	5A	37
Al	37	
AH	5B	
AL	6C	

Sign	0
Zero	0
Carry	0
Overflow	0
Parity	1
Auxiliary	0

Rough Work Final Ax = 5B6C



Question No:- 6

ax	1010	0101	1001	0011	A593
xor	1111	1111	1111	1111	FFFF
ax	0101	1010	0110	1100	5A6C
xor	0000	0000	0000	0000	
ax	0101	1010	0110	1100	5A6C
xor	1111	1111	1111	1111	
ax	1010	0101	1001	0011	A593
xor	1010	0101	1001	0011	
ax	0000	0000	0000	0000	0000
ax	0101	1010	0011	0111	5A37
xor			0000	0000	
ax	0101	1010	0011	0111	5A37
xor	0000	0001			
ax	0101	1011	0011	0111	5B37
xor			0101	1011	
ax	0101	1011	0110	1100	5B6C

7. Write a program that finds parity of number given below? **HINT:** Use XOR and LOOP to find parity

```
4   .data
5       parity DQ 0A1B2C3D4E5F67890H
```

CODE:

```
.386
.model flat , stdcall
.stack 4096
.data
    var1 DQ 0A1B2C3D4E5F67890h
.code
    main PROC
        mov ecx, 7
        mov esi, offset var1
        mov al, Byte PTR [esi]
        inc esi
        label1:
            mov ah, Byte PTR [esi]
            XOR al, ah
            inc esi
        loop label1
    main endP
    end main
```

INSTRUCTIONS: (Question 8-11) *Perform each of the following operations on word size 2's complement numbers and update the answer and value of flags after performing the arithmetic. Perform all the steps in the "calculation" box, only filling the answer will not get any credit*

8. Update flags after arithmetic instruction? Also state which of the following jumps will taken or not taken

```
.code
        mov ax,0FFFEh
        add ax,0FC70h
        jc l1
  L1:   jz L2
  L2:   jo L3
  L3:   js L4
  L4:   jp L5
  L5:
```

	TAKEN	NOT TAKEN
Jc	Taken	
Jz		Not taken
Jo		Not taken
Js	Taken	
jp		Not taken

FLAGS	
Sign	1
Zero	0
Carry	1
Overflow	0
Parity	0

Calculations

```
  11
  FFFE
+ FC70
(1) FC6E
```

6E = 0110 1110
Parity = 0

9. Update flags after arithmetic instruction? Also state which of the following jumps will taken or not taken

```
.code
        mov ax,07B1Ah
        sub ax,0CEEBh
        jc l1
  L1:   jz L2
  L2:   jo L3
  L3:   js L4
  L4:   jp L5
  L5:
```

	TAKEN	NOT TAKEN
Jc	Taken	
Jz		Not taken
Jo	Taken	
Js	Taken	
jp		Not taken

FLAGS	
Sign	1
Zero	0
Carry	1
Overflow	1
Parity	0

Calculations

```
  0 11 1A 10
  7b1a
- ceeb
  ac2F
```

2F = 0010 1111
Parity = 0

10. Update flags after arithmetic instruction? Also state which of the following jumps will taken or not taken

```
.code
        mov cx,0FBCDh
        add cx,0E4AAh
        jnc l1 ⟶ not taken
L1:     jnz L2 ⟶ taken
L2:     jo L3 ⟶ not taken
L3:     js L4 ⟶ taken
L4:     jnp L5 ⟶ not taken
```

	TAKEN	NOT TAKEN
Jc	Taken	
Jz		Not taken
Jo		Not taken
Js	Taken	
jp	Taken	

FLAGS	
Sign	1
Zero	0
Carry	1
Overflow	0
Parity	1

Calculations

```
  1 1 1
  FBCD
+ E4AA
-------
1 E077
```

77 = 0111 0111
Parity = 1

11. Update flags after arithmetic instruction? Also state which of the following jumps will taken or not taken

```
.code
        mov bx,0FABDh
        add bx,0684Ah
        jc l1 ⟶ Taken
L1:     jz L2 ⟶ not taken
L2:     jno L3 ⟶ taken
L3:     jns L4 ⟶ taken
L4:     jp L5 ⟶ Not taken
L5:     mov ah,04ch
```

	TAKEN	NOT TAKEN
Jc	Taken	
Jz		Not Taken
Jo		Not taken
Js		Not taken
jp		Not taken

FLAGS	
Sign	0
Zero	0
Carry	1
Overflow	0
Parity	0

Calculations

```
  1 1 1
  FABD
+ 684A
-------
1 6307
```

07 = 0000 0111
Parity = 0

12. Update flags after CMP instruction? Also state which of the following jumps will taken or not taken

```
.code
        mov bx, 0A5A5h
        not bx
        cmp bx, 05A5Ah
        jz L1 ⟶ taken
L1:     je L2 ⟶ taken
L2:     jnz L3 ⟶ not taken
L3:     jne L4 ⟶ not taken
L4:
```

	TAKEN	NOT TAKEN
Jc		Not taken
Jz	taken	
Jo		Not taken
Js		Not taken
jp	taken	

FLAGS	
Sign	0
Zero	1
Carry	0
Overflow	0
Parity	1

Calculations

1010 0101 1010 0101 = A5A5

0101 1010 0101 1010 = 5A5A

not ↗ cmp 5A5A, 5A5A

```
  5A5A
- 5A5A
------
  0000
```

RENAME JUMPS: JE= JZ , JNZ= JNE

13. Update value of **ax** and **cx** registers after every iteration. Update any changes to the done to flag

```
.code
        mov ecx, 5
        mov ax, 1
L1:
        inc ax
        dec ecx
jcxz end_loop
jmp L1
end_loop:
```

	TAKEN	NOT TAKEN
Jc		
Jz		
Jo		
Js		
jp		

FLAGS	
Sign	
Zero	
Carry	
Overflow	
Parity	

Calculations

Iterations =

		1	2	3	4	5
cx	5	4	3	2	1	0
AX	1	2	3	4	5	6

Question No 13:

Iteration 1:

Ax = 1
Inc ax

```
  0001 h
+ 0001 h
  0002 h    →    0000 0010 b    →    Parity = 0
```

Ecx = 5
Dec ecx

```
  00000005 h
- 00000001 h
  00000004  →    0000 0100      →    Parity = 0
```

Iteration 2:

Ax = 2
Inc ax

```
  0002 h
+ 0001 h
  0003 h    →    0000 0011 b    →    Parity = 1
```

Ecx = 4
Dec ecx

```
  00000004 h
- 00000001 h
  00000003  →    0000 0011      →    Parity = 1
```

Iteration 3:

Ax = 3
Inc ax

```
  0003 h
+ 0001 h
  0004 h    →    0000 0100 b    →    Parity = 0
```

Ecx = 3
Dec ecx

```
  00000003 h
- 00000001 h
  00000002  →    0000 0010      →    Parity = 0
```

Iteration 4:

Ax = 4
Inc ax

```
  0004 h
+ 0001 h
```

0005 h → 0000 0101 b → Parity = 1

Ecx = 2
Dec ecx

00000002 h
- 00000001 h
00000001 → 0000 0001 → Parity = 0

Iteration 5:

Ax = 5
Inc ax

0005 h
+ 0001 h
0006 h → 0000 0110 b → Parity = 1

Ecx = 1
Dec ecx

00000001 h
- 00000001 h
00000000 → 0000 0000 → Parity = 1 & Zero = 1

Flag Values:

Flags	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
	Inc ax	Dec ecx	Inc ax	Dec ecx	Inc ax	Dec ecx	Inc ax	Dec ecx	Inc ax	Dec ecx
Sign	0	0	0	0	0	0	0	0	0	0
Zero	0	0	0	0	0	0	0	0	0	1
Carry	0	0	0	0	0	0	0	0	0	0
Overflow	0	0	0	0	0	0	0	0	0	0
Parity	0	0	1	1	0	0	1	0	1	1

INSTRUCTIONS: (Question 14) *CMP performs implied subtractions of a source operand from destination operand and updates flags, after CMP neither operands are modified.*

- *For unsigned operation you will require* **Zero** *and* **Carry Flag.** *Flags will determine relationship between destination and source given in table below*
  - **Note:** To calculate flags after **SUB** command use binary subtraction where **Carry Flag** act as borrow flag whereas **AUX** flag act as **Aux Borrow** for least nibble

CMP Results	ZF	CF
Destination < source	0	1
Destination > source	0	0
Destination = source	1	0

- *For signed operation you will require* **Zero, Overflow** *and* **Sign Flag.** *Flags will determine relationship between destination and source given in table below*
  - **Note:** To calculate flags after **SUB** command, use 2's complement subtraction where overflow is calculated using **XOR** operation

CMP Results	Flags
Destination < source	SF ≠ OF
Destination > source	SF = OF
Destination = source	ZF = 1

14. Update flags after every **CMP** instruction and calculate jumps

```
MOV al,+127
CMP al,-128
JA IsAbove
JG IsGreater
```

	TAKEN	NOT TAKEN
JA		Not Taken
JG	Taken	

Subtraction to Calculate Flags for unsigned numbers

Z's Complement addition to calculate flags for signed number

```
MOV ax,-1
CMP ax,0
JNL L1
JNLE L2
J1 L4
JB L4
```

	TAKEN	NOT TAKEN
JNL		Not taken
JNLE		Not taken
JL	Taken	
JB		Not taken

Subtraction to Calculate Flags for unsigned numbers

Z's Complement addition to calculate flags for signed number

```
MOV ax,+32
CMP ax,-35
JNG L1
JNGE L2
JGE L3 → Taken
JNAE L4 → Taken
```

	TAKEN	NOT TAKEN
JNG		Not taken
JNGE		Not taken
JL		Not taken
JB	Taken	

Subtraction to Calculate Flags for unsigned numbers

Z's Complement addition to calculate flags for signed number

Question No:- 14

(a)

num1 = 127

num2 = -128

127 = 7F h

128 = 80 h

→ Take 2's compliment of 128

FF
80
7F
+1
(80)h

num1 = 7Fh num2 = 80h

⊙ Simple Subtraction

① 7F
- 80
FF

= 1111 1111

Parity = 1

Carry = 1

OverFlow = 1

Sign = 1

Zero = 0

⊙ Using 2's Compliment Addition

take 2's Complement of 80

FF
-80
7F
+1
(80)h

7F
+ 80
FF

Unsigned Comperison

	Condition		
num1 < num2	ZF = 0	&& CF = 1	True
num1 > num2	ZF = 0	&& CF = 0	False
num1 = num2	ZF = 1	&& CF = 0	False

Signed Comparison

	Condition	
num1 < num2	SF ≠ OF	False
num1 > num2	SF = OF	True
num1 = num2	ZF = 1	False

(b)

num1 = -1

num2 = 0

```
  FFFF
- 0001
------
  FFFE
+    1
------
  FFFF
------
```

num1 = FFFF h

num2 = 0000 h

Flags

Carry = 0

Overflow = 0

Sign = 1

Zero = 0

① Simple Subtraction

```
  FFFF
- 0000
------
  FFFF
------
```

2's Complement Method

num1 = FFFF num2 = 0000

2's Complement of num2

```
  FFFF
- 0000
------
  FFFF
    +1
------
  0000
------
```

```
  FFFF
+ 0000
------
  FFFF h
------
```

Unsigned Comparison:

As ZF=0 and CF=0 hence num1 > num2

Signed Comparison:

As SF ≠ OF Hence num1 < num2

—— & (c) ——

num1 = 32 = 20h

num2 = -35

35 = 23h

```
  FFFF
- 0023
------
  FFdC
------
    +1
------
  FFdd
------
```

num1 = 0020h num2 = FFddh

○ Simple Subtraction

```
  ①0'0"20'
- FFdd
------
  0043 h
------
```

Carry = 1
Sign = 0
Overflow = 0
Zero = 0

○ 2's Complement Addition

2's Complement of num2

```
  FFFF
  FFdd
------
  0022
    +1
------
  0023
------
```

```
  0020
+ 0023
------
  0043 h
------
```

Unsigned Comparison:

As ZF = 0 and CF = 1 Hence num1 < num2 is True.

Signed Comparison:

As SF = OF hence num1 > num2

```
MOV cx,0
CMP cx,0
JG L1 → not taken 0>0
JNG L2 → taken 0≤0
JB L3 → not taken 0<0
JNB L4 → taken 0≥0
JLE L5 → taken 0≤0
```

	TAKEN	NOT TAKEN
JNG	Taken	
JNG	taken	
JB		Not Taken
JNB	Taken	
JLE	Taken	

Subtraction to Calculate Flags for unsigned numbers

2's Complement addition to calculate flags for signed number

```
MOV ax,-12
CMP cx,12
JB L1 → not taken
JNAE L2
JNGE L3
JL L4
```

	TAKEN	NOT TAKEN
JB	Not taken	
JNAE	Not taken	
JNGE		Taken
JL		Taken

Subtraction to Calculate Flags for unsigned numbers

2's Complement addition to calculate flags for signed number

```
MOV ax,12
CMP cx,-12
JA L1
JNBE L2
JNLE L3
JG L4
```

	TAKEN	NOT TAKEN
JA		Not taken
JNBE		Not taken
JNLE	Taken	
JG	Taken	

Subtraction to Calculate Flags for unsigned numbers

2's Complement addition to calculate flags for signed number

13

d

num1 = 0
num2 = 0

① Simple Subtraction

```
  0000
- 0000
------
  0000
------
```

Sign = 0
Carry = 0
Overflow = 0
Zero = 1

② Using 2's complement

Take 2's complement of num2

```
  FFFF
- 0000
------
  FFFF
    +1
------
① 0000
------
```

un Signed Comparison

As ZF = 1 and CF = 0 Hence num1 = num2

Signed Comparison

As ZF = 1 Hence num1 = num2

e

num1 = −12
num2 = 12 = C

```
  FFFF
- 000C
------
  FFF3
    +1
------
  FFF4
------
```

num1 = FFF4 h
num2 = 000C h

① Using Simple Subtraction

```
  ①
  FFF4
- 000C
------
  FFE8
------
   1
```

Carry = 0
Sign = 1
Overflow = 0
Zero = 0

① Using 2's Complement method

2's Complement of num2

```
  FFFF
- 000C
------
  FFF3
   +1
------
  FFF4
```

```
   1
  FFF4
+ FFF4
------
① ffe8
```

Unsigned Comparison :

As ZF=0 & CF=0 Hence num1 > num2

Signed Comparison:

As SF ≠ OF Hence num1 < num2

num1 = 12 = C
num2 = -12 = FFF4

① Using Simple subtraction:

```
  ①000C
- FFF4
------
  0018
------
```

Carry = 1
Sign = 0
Overflow = 0
Zero = 0

② Using 2's Complement

Take 2's Complement of num2

```
  FFFF
- FFF4
------
  000B
    +1
------
  000C
------
```

```
  000C
+ 000C
------
  0018
------
```

⊙ Unsigned Comparison

As ZF = 0 & CF = 1 So num1 < num2

⊙ Signed Comparison

As SF = OF So num1 > num2

15. Consider the following Data declaration. Fill in the given register values. **Look for errors if any.**

DATA

```
.data
  bval1 DB 034h,
     R1(b), wval2
     DW R(h) dval3
     DD RR(b) = 1001010011110000 1101
              = 94F0Dh
```

	0	1	2	3	4	5	6	7	8
0	34	00	3D	00	0D	4F	09	00	

(bval1 ← 0; wval2 → 2; dval3 → 4; memory)

CODE

```
.code
main Proc
      mov esi,OFFSET
      bval1 mov ax, WORD
      PTR bval1 mov al,
      BYTE PTR wval2 mov
      bx, WORD PTR dval3
      mov cx, WORD PTR [dval3+2]

main
ENDP END
main
```

Error: There some invalid line breaks in this code and there is required line break on some places

REGISTER

ESI	00 h
AX	0034 h
AL	3D h
BX	4F0D h
CX	0009 h

16. Consider the following Data declaration. Fill in the given register values. Look for errors if any.

DATA

```
.data
  valb LABEL Byte
     valw LABEL WORD
     vald LABEI DWORD
     valq LABEL dq   ← dq not allowed with label in MASM use Qword here.
     valt Tbyte 12345678abcdef123456h
```

CODE

```
.code
main Proc
      mov
      al,valb
      mov
      bx,valw
      mov
      ecx,vald
main
ENDP END
main
```

Error: There are some invalid line breaks in this code

REGISTER

AL	56 h
BX	3456 h
ECX	ef123456 h

ECX = ef 12 34 56 h

	0	1	2	3	4	5	6	7	8	9	a	b
0	56	34	12	ef	cd	ab	78	56	34	12		

valb, valw, vald, valq, valt

Corrected Code of Question No:- 15

```
.data
    bval1  DB   034h,
000h
    wval2  DW   03Dh
    dval3  DD   10010100111100001101b

.code
    mov   esi , offset bval1
    mov   ax ,  WORD PTR bval1
    mov   al ,  BYTE PTR wval2
    mov   bx ,  WORD PTR dval3
    mov   cx ,  WORD PTR [dval3+2]

main endp
end main
```

Corrected Code of Question No:- 16

```
.code
        main PROC
        mov al, valb
        mov bx, valw
        mov ecx, vald
main endp
end main
```

17. Consider the following Data declaration. Fill in the given Memory and register values. **Look for errors if any**

DATA

```
.data                'F' Id 75(o) = 3d
  Ary1 BYTE 'F1',29, R2(o)
  ary2 WORD 'L1',05678h, R(b)
                'J'          11110 b = 3D
```

CODE

```
.code
main Proc
     mov ESI, OFFSET Ary1
  mov
  al,[ESI]
  add
  ESI,1
  mov
  bl,[ESI]
  inc ESI
  inc BYTE PTR [ESI]
  mov cl,[ESI]

  mov ESI, OFFSET Ary2
  mov
  ax,[ESI]
  add
  ESI,2
  mov
  bx,[ESI]
  inc ESI
  inc ESI
  inc WORD PTR [ESI]
  mov cx,[ESI]

main
ENDP END
main
```

Error:

There are some incorrect line breaks at some places in code.

REGISTER

ESI	00404000 h
AL	46h
ESI	00404001 h
BL	1d h
ESI	00404002 h
CL	3Eh
ESI	00404003 h
AX	0049h
ESI	00404005 h
BX	5678h
ESI	00404007 h
CX	003Eh

MEMORY

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00404000	46	1d	3d	49	00	78	56	3D	00							
00404010																
00404																

→ Ary1 → ary2

Corrected Code of Question No1-17

```
.data
    Arg1 BYTE 'F', 29, 750
    arg2 Word 'F', 05678h, 11110lb
.code
  main PROC
      mov esi, offset Arg1
      mov al, [esi]
      add esi, 1
      mov bl, [esi]
      inc esi
      inc BYTE PTR [esi]
      mov cl, [esi]
      mov esi, offset Arg2
      mov ax, [ESI]
      add esi, 2
      mov bx, [ESI]
      inc ESI
      inc ESI
     inc WORD PTR [ESI]
     mov cx, [ESI]
     main endp
     end main
```

020																	
00404030																	

18. Consider the following Data declaration. Fill in the given register values. Look for errors if any

DATA

```
.data          'F'   3Dh
  Ary1 BYTE 'F1',2, R2(h),4,75o
  ary2 WORD 'L1',05678h, R(o),567o
  ary3 DWORD 'L2',012345678h, RR(h),567o
             'M'                094F0Dh
```

CODE

```
.code
main Proc
     mov
  ESI,0 mov
  al,ary1[ESI
  ]
     mov ESI,1 * TYPE
  ary1 mov
  bl,ary1[ESI]
     mov ESI,2 * TYPE
  ary1 mov
  cl,ary1[ESI]
```

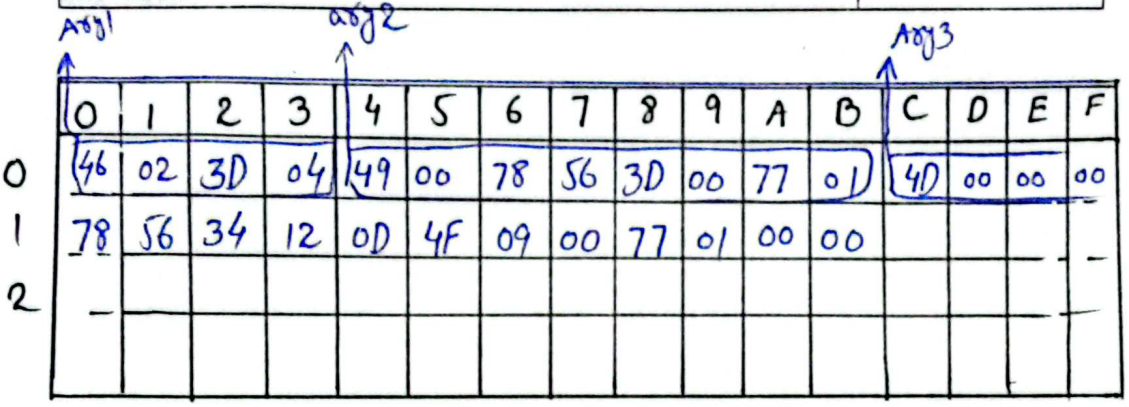
Error: There some incorrect line breaks in code. And there need line break at some placer in code.

```
  mov ESI,0
  mov ax,ary2[ESI]
     mov ESI,1 * TYPE
  ary2 mov
  bx,ary2[ESI]
     mov ESI,2 * TYPE
  ary2 mov
  cx,ary2[ESI]
```

REGISTER

ESI	0 h
AL	46 h
ESI	1 h
BL	02 h
ESI	2 h
CL	3D h
ESI	00 h
AX	0049 h
ESI	2 h
BX	5678 h
ESI	4 h
CX	003D h
ESI	0 h
EAX	0000004D h
ESI	4 h
EBX	12345678 h
ESI	8 h
ECX	00094F0D h

```
    mov ESI,0
    mov eax,ary3[ESI]
       mov ESI,1 * TYPE
    ary3 mov
    ebx,ary3[ESI]
       mov ESI,2 * TYPE
    ary3 mov
    ecx,ary3[ESI]

main ENDP
END main
```



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	46	02	3D	04	49	00	78	56	3D	00	77	01	4D	00	00	00
1	78	56	34	12	0D	4F	09	00	77	01	00	00				
2																

Corrected Code for Question No:-18

```
.data
    Arg1  BYTE   'F', 2, 3Dh, 4
    arg2  WORD   'I', 05678h, 75o, 567o
    arg3  DWORD  'M', 012345678h, 094F0Dh, 567o

.code
    main PROC
        mov  esi, 0
        mov  al, arg1[esi]
        mov  esi, 1* TYPE arg1
        mov  bl, arg1[esi]
        mov  esi, 2* Type arg1
        mov  cl, arg1[esi]
```

```
mov esi, 0
mov ax, arr2 [esi]
mov esi, 1 * TYPE arr2
mov bx, arr2[esi]
mov esi , 2 * TYPE arr2
mov cx, arr2 [esi];

mov esi, 0
mov eax, arr3 [esi]
mov esi, 1 * Type arr3
mov ebx, arr3 [esi]
mov esi , 2 * TYPE arr3
mov ecx , arr3 [esi]
main endp
end main
```