School of Computing

Faculty of Engineering

UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT NAME:	**COMPUTER ORGANIZATION AND ARCHITECTURE**
SUBJECT CODE:	**SCSR/SECR 2033**
SEMESTER:	**2019/20-2**
LAB TITLE:	**Programming 3a: Flags, OFFSET, Arrays, JMP, LOOP**
INSTRUCTION:	**Answer all questions.**
STUDENT INFO :	

Name: **SEE WEN XIANG**

Matric No: **A19EC0206** **Section:** **07**

Email: **wxsee@graduate.utm.my**

SUBMISSION DATE: **29/04/2020 (Thu)**

Part A – Programming review

Flags Affected by Arithmetic

- The ALU has a number of status flags that reflect the outcome of arithmetic (and bitwise) operations
  - based on the contents of the destination operand
- Essential flags:
  - Zero flag – set when destination equals zero
  - Sign flag – set when destination is negative
  - Carry flag – set when unsigned value is out of range
  - Overflow flag – set when signed value is out of range
- The MOV instruction never affects the flags.

Zero Flag (ZF)

- The Zero flag is set when the result of an operation produces zero in the destination operand.

```
mov ax,2
sub ax,1      ; AX = 1, ZF = 0
mov cx,1
sub cx,1      ; CX = 0, ZF = 1
mov ax,0FFFFh
inc ax  ; AX = 0, ZF = 1
inc ax  ; AX = 1, ZF = 0
```

Sign Flag (SF)

- The Sign flag is set when the destination operand is negative. The flag is clear when the destination is positive.

```
mov cx,0
sub cx,1     ; CX = -1, SF = 1
add cx,2     ; CX = 1, SF = 0
```

- The sign flag is a copy of the destination's highest bit:

```
mov al,0
sub al,1    ; AL = 11111111b, SF = 1
```

COMMENTS:

```
add al,2    ; AL = 00000001b, SF = 0
```

Overflow and Carry Flags: A Hardware Viewpoint

- How the ADD instruction modifies OF and CF:
  - OF = (carry out of the MSB) XOR (carry into the MSB)
  - CF = (carry out of the MSB)
- How the SUB instruction modifies OF and CF:
  - NEG the source and ADD it to the destination

- OF = (carry out of the MSB) XOR (carry into the MSB)
- CF = INVERT (carry out of the MSB)

MSB = Most Significant Bit (high-order bit) XOR = eXclusive-OR operation NEG = Negate (same as SUB 0,operand)

Carry Flag (CF)

- The Carry flag is set when the result of an operation generates an unsigned value that is out of range (too big or too small for the destination operand).

```
mov al,0FFh
add al,1  ; CF = 1, AL = 00
```

; Try to go below zero:

```
mov al,0
sub al,1  ; CF = 1, AL = FF
```

Overflow Flag (OF)

- The Overflow flag is set when the signed result of an operation is invalid or out of range.

; Example 1

```
mov al,+127
add al,1     ; OF = 1,   AL = 80h
```

; Example 2

```
mov al,7Fh
add al,1     ; OF = 1,   AL = 80h
```

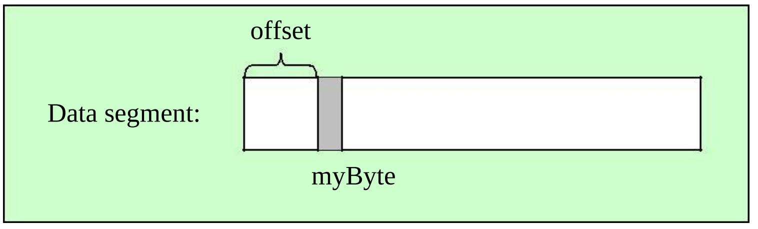
- The two examples are identical at the binary level because 7Fh equals +127. To determine the value of the destination operand, it is often easier to calculate in hexadecimal.

***NOTE: In VisualStudio Register Window during Step Over, you will find that the flag registers are presented with a different name.*

Overflow Flag (OF) = OV
Zero Flag (ZF) = ZR
Sign Flag (SF) = PL
Parity Flag (PF) = PE
Carry Flag (CF) = CY
Auxiliary Flag (AF) = AC

OFFSET Operator

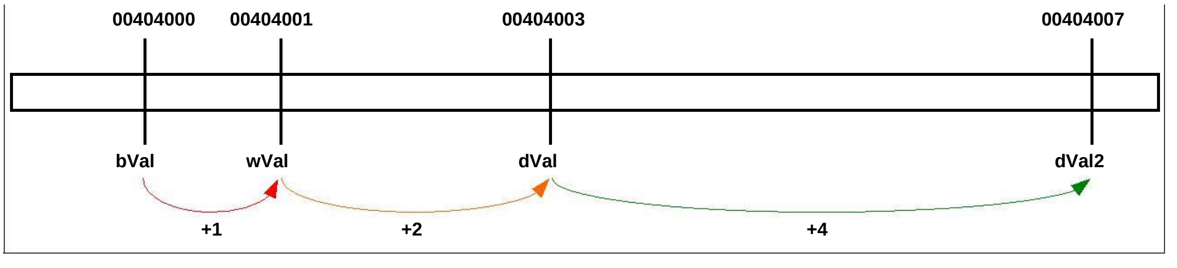
- OFFSET returns the distance in bytes, of a label from the beginning of its enclosing segment
  - Protected mode: 32 bits
  - Real mode: 16 bits
- OFFSET gives you the address where the variable (or an array) starts.



- Example: Let's assume that the data segment begins at 00404000h:

```
 .data
 bVal BYTE 10h
 wVal WORD 1000h
 dVal DWORD 10001000h
 dVal2 DWORD ?

 .code
mov esi,OFFSET bVal        ; ESI = 00404000
mov ah, bVal               ; AH = 10h
mov esi,OFFSET wVal        ; ESI = 00404001
mov ax, wVal               ; AX = 1000h
mov esi,OFFSET dVal        ; ESI = 00404003
mov eax, dVal              ; EAX = 10001000h
mov esi,OFFSET dVal2       ; ESI = 00404007
```



***TIP: Note the different outcomes in the MOV instructions with and without the use of OFFSET.*

Arrays

- Arrays are probably the most commonly used composite data type.
- Analogy:
  - An array is like a drawer that holds many items of the same type. Like a sock drawer that have 10 different pairs of socks, and you can reach these socks from that drawer.
- Defining an array:
  - Must have array name, size of each item in array, initialize (or not) the values of these items o Example:

```
Array1   byte   10h, 20h, 35h
; 3 items in array, each 1 byte in size
Array2  word 25h, 1A20h, 66h, 891h
; 4 items in array, each 2 bytes in size
Array3 word 4 dup (0)
; 4 items in array, each 2 bytes , and initialized to 0
```

Array1	**10h**	**20h**	**35h**

Array2	**0025h**	**1A20h**	**0066h**	**0891h**

- Handling an array:
  - Use register as a pointer, the method is called *indirect addressing*.
  - Traversing an array (i.e. moving through an array), the pointer must be incremented following the array type (byte [+1] or word [+2] or dword[+4]).
- Example:

```
.data

Array1 byte 10h, 20h, 35h      ;Array1 starts at address
404000
Array2 word 25h, 1A20h, 66h, 891h

.code main
PROC

; Calling array method 1
mov esi,OFFSET Array1     ; ESI = 00404010
mov al, [esi]   ; AL = 10h
add esi,1       ; ESI = 00404001
mov bl, [esi]   ; BL = 20h

mov esi,OFFSET Array2     ; ESI = 00404003
mov ax, [esi]   ; AX = 0025h
add ESI,2       ; ESI = 00404005
mov bx, [esi]   ; BX = 1A20h
```

```
; Calling array
method                  2
mov al, Array1        ; AL = 10h
mov bl, Array1+1      ; BL = 20h

mov ax, Array2        ; AX = 0025h
mov bx, Array2+2      ; BX = 1A20h
```

***TIP: You can also use TYPE <array name> to match the array type. Example: mov bl, Array1+type Array1; BL = 20h*

- Example: Sum an array

```
.data

Array1 byte 10h, 20h, 35h
Array2 word 25h, 1A20h, 66h, 891h

.code main
PROC

;Sum an array
mov al, Array1        ; AL = 10h
add al, Array1+1      ; AL = 30h
add al, Array1+2      ; AL = 65h

mov bx, Array2        ; BX = 0025h
add bx, Array2+2      ; BX = 1A45h
add bx, Array2+4      ; BX = 1AABh
add bx, Array2+6      ; BX = 233Ch
```

Jump

- To jump here means to relocate the instruction pointer to a different address, one that is not sequential (i.e. not the next one).
- Jumps can be
  - **Conditional:** jump when a condition(s) is met; if not met don't jump. Conditions can be flags, arithmetic results, etc.
    - Example: JNZ (Jump Not Zero), JE (Jump Equal), JB (Jump Below)
  - **Unconditional:** no conditions, you MUST jump
    - Example: JMP
- Let's explore unconditional jumps with JMP
- The JMP command causes unconditional transfer to a destination (label) that is usually within the same procedure.

- The command format is ***JMP destination***
- The destination is a label
- When this command is executed, the instruction pointer (EIP) will now point to the address where the label (or the destination) is.

0001	MOV AX,10
0002	
0003	**HERE:**
0004	INC AX
0005	JMP HERE
0006	ADD AX,2

  - When the command INC AX is executed, EIP points to 0005.
  - When the command JMP HERE is executed, EIP will now point to 0003 rather than 0006
  - The problem here: this is an endless loop

- Let's explore a conditional jump example with JNZ.
  - The format is the same as unconditional jump.
  - JNZ ➔ jump to a label if the Zero flag is clear [ZF = 0]
  - In the example below, once the Zero Flag is set [ZF = 1], the condition for JNZ is not met; so it will not be done. EIP will point to 0006.

0001	MOV AX,10
0002	
0003	**HERE:**
0004	DEC AX
0005	JNZ HERE
0006	ADD AX,2

- Please do explore the different conditional jumps that are available to you.
  - Conditional jumps are usually accompanied with a compare (CMP) command.
  - More of these in upcoming labs.

LOOP

- As the name implies, the LOOP instruction will repeatedly execute a block of statements.
- The number of time the looping will occur is held in a counter. In a 32-bit mode, the counter is the register ECX.
- The loop instruction decrements register ECX and compares it with 0 leaving the flags unchanged.
  - If new ECX ≠ 0, jumps to the label.
  - Else, the program execution continues with the next instruction.
- The command format is ***LOOP destination***
  - The destination is a label

- Example: try and trace the program below.

```
.code
main PROC

   mov eax,10h
   mov ecx, 4 ; ecx is the counter
L1:
   add eax,2  ; eax = eax + 2
   loop L1   ; ecx = ecx -1; go to L1 if ECX ≠ 0

      exit
 main ENDP

 END main
```

ECX		EAX
4 (initial value)		**10h (initial value)**
	add eax,2	12h
3	loop L1 → go to L1	
	add eax,2	14
2	loop L1 → go to L1	
	add eax,2	16
1	loop L1 → go to L1	
	add eax,2	18
0	loop L1 → Stop	

- You can do a nested loop instruction (if need be).
- If you need to code a loop within a loop, you must save the outer loop counter's ECX value.

Part B – Let's do a little programming

1. Given the assembly language program below, run it and list the flags' status after each instruction.

PROGRAM	OF(OV)	SF(PL)	ZF(ZR)	AF(AC)	PF(PE)	CF(CY)
	0	**0**	**1**	**0**	**1**	**0**
`mov ax,10h`	**0**	**0**	**1**	**0**	**1**	**0**
`add ax,2h`	**0**	**0**	**0**	**0**	**1**	**0**
`sub ax,15h`	**0**	**1**	**0**	**1**	**0**	**1**
`add ax,112`	**0**	**0**	**0**	**0**	**0**	**1**
`neg ax`	**0**	**1**	**0**	**1**	**1**	**1**
`mov bh,66h`	**0**	**1**	**0**	**1**	**1**	**1**
`inc bx`	**0**	**0**	**0**	**0**	**0**	**1**
`mul dh`	**1**	**0**	**0**	**0**	**1**	**1**
`sub al,3`	**0**	**0**	**0**	**1**	**1**	**0**

2. What will be the values of the Overflow flag in the program given below?

```
mov al,80h
add al,92h ;     AL = 12h   , OF = 1
mov al,-2
add al,+127 ;    AL = 7Dh   , OF = 0
```

3. Define the following arrays:
   a. A byte type array named PKP with 3 items 11, 22h and 4Ah.

```
PKP byte 11, 22h, 4Ah
```

   b. A word type array named ZOOM with 5 items 45, 45h, 444h, 4A4Bh and 44Ah.

```
ZOOM word 45, 45h, 444h, 4A4Bh, 44Ah
```

   c. A double-word type array named PADLET with 5 items initialized to 0

```
PADLET dword 5 dup(0)
```

4. Referring to the array definitions in Question 3, state the following values in the register.

a.	MOV AL, PKP	; AL = Bh
b.	MOV AL, PKP+3	; AL = 2Dh
c.	MOV AX, ZOOM	; AX = 002Dh
d.	MOV AX, ZOOM+3	; AX = 4400h
e.	MOV AX, ZOOM+4	; AX = 0444h
f.	MOV EAX, PADLET	; EAX = 00000000h
g.	MOV AH, PKP+8	; AH = 04h

h.	MOV EAX,0
	MOV AX, ZOOM+2
	MOV PADLET, EAX
		; EAX = 00000000h
		; AX = 0045h
		; PADLET = 00000045h

5. Referring to the array definitions in Question 3, write the appropriate instruction(s) to achieve the required results.

a.	MOV BL, PKP+5	; BL = 45h
b.	MOV BX, ZOOM+7	; BX = 004Ah

c.	MOV EAX, 0h
	MOV AX, ZOOM+6,
	MOV PADLET+4, EAX
		; PADLET+4 = 00004A4Bh

6. (video) Referring to the array definitions in Question 3, write a program to sum array PKP.

7. (video) Referring to the array definitions in Question 3, write a program to sum array ZOOM.

8. (video) Study the assembly instructions given below and fill in the blanks (in hexadecimal).

```
INCLUDE Irvine32.inc
.data
intArray WORD 100h,200h,300h,400h
TOTAL WORD 0

.code
main PROC
        mov edi,OFFSET intArray       ; EDI = 00404000h
        mov ecx,LENGTHOF intArray     ; ECX = 00000004h
        mov ax,0

L1:
        add ax,[edi]
        add edi,TYPE intArray ; EDI = EDI + 2
        loop L1

        mov TOTAL, ax  ; TOTAL = 0A00h

exit

main ENDP
END main
```

9. Study the assembly instructions given below and answer the following questions.

```
mov ax,20
       mov ecx,4
L1:
       inc  ax
       neg  ax
       loop L1
```

a. How many times will the loop be executed? **4**

b. What is the final result of AX in hexadecimal? **14h**

c. Fill in the table with the value of AX after each instruction in each loop.

Loop#	INC AX	NEG AX
Initially AX = 20d		
1	0015h	FFEBh
2	FFECh	0014h
3	0015h	FFEBh
4	FFECh	0014h

10. Study the assembly language code given below. What is the final value of the variable TOTAL?

```
INCLUDE Irvine32.inc
.data
total dword 0
counter dword 7

.code
main PROC

   mov eax,0
   mov ecx,counter
L1:
   add eax,10h
   loop L1

   mov total, eax

exit
main ENDP

END main
```

TOTAL = <u>00000070h</u>

11. (video) Using the LOOP instruction, write a program to achieve the following equation. What is the final result of TOTAL in hexadecimal and decimal?

TOTAL = 100h * 7h

Ans: TOTAL = <u>00000700h</u> = <u>1792d</u>