School of Computing
Faculty of Engineering
UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT NAME: **COMPUTER ORGANIZATION AND ARCHITECTURE**

SUBJECT CODE: **SECR 2033**

SEMESTER: **2019/20-2**

LAB TITLE: **Programming 5: Comparison & Conditional Jumps**

STUDENT INFO :

Name	Matric No.
SEE WEN XIANG	A19EC0206

SECTION: **07**

SUBMISSION DATE: **15/06/2020 (Mon)**

COMMENTS:

Programming 5: COMPARISON & CONDITIONAL JUMPS

Part A – Programming review

A) BOOLEAN and COMPARISON INSTRUCTIONS

Logical Instructions

The processor instruction set provides the instructions AND, OR, XOR, TEST and NOT Boolean logic, which tests, sets and clears the bits according to the need of the program. These instructions set the CF, OF, PF, SF and ZF flags.

Conditional Instructions

Sometimes a program needs to do different things depending on the result of an operation. As shown in Figure 1, if the conditions are met then process A. Otherwise, proceed with process B. This is conditional branching. This is different from unconditional branching (the JMP instruction) previously studied.

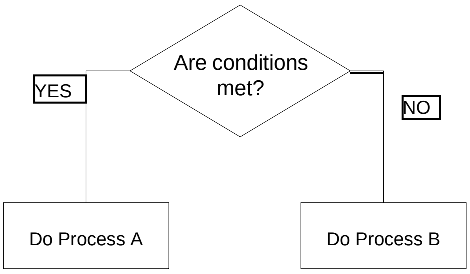


Figure 1

Compare (CMP) instruction

First, let us look at the compare (CMP) instruction. This instruction is used in to test branching conditions.

- The CMP instruction compares two operands.
- This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not.
- It does not disturb the destination or source operands (i.e. these does not change)
- The instruction format:
  - The destination operand can be either register or memory.
  - The source operand can be register, memory or immediate value.

```
CMP Destination, Source
```

```
CMP BX, 00 ; Compare the value in BX with zero
JE TARGET  ; Jump to TARGET if BX = 0
```

Sneak-peek: JE is Jump if Equal

B) CONDITIONAL JUMPS

Conditional Branching or Conditional Jump

This is performed by a set of jump instructions depending upon the condition. The conditional instructions transfer the control by breaking the sequential flow and they do it by changing the offset value in IP (Instruction Pointer). Written in the form J<condition>. Example: JE, JNZ, JL, JG

- There are different groups of conditional jump instructions:
  - Jumps based on specific flag values
  - Jumps based on equality between operands or the value of (E)CX
  - Jumps based on comparisons of unsigned operands
  - Jumps based on comparisons of signed operands
- The instruction format:

```
J<condition> TARGET
```

```
Examples:
      JE TARGET
      JNZ TARGET
      JL TARGET
```

Table 1: Jumps based on specific flag values

Instruction	Description
JZ	Jump if zero; ZF = 1
JNZ	Jump if not zero; ZF = 0
JC	Jump if carry; CF = 1
JNC	Jump if not carry; CF = 0
JO	Jump if overflow; OF = 1
JNO	Jump if not overflow; OF = 0
JS	Jump if signed; SF = 1
JNS	Jump if not signed; SF = 0
JP	Jump if parity (even); PF = 1
JNP	Jump if not parity (odd); PF = 0

Table 2: Jumps based on equality between operands or the value of (E)CX

Instruction	Description	Instruction	Description
JE	Jump if equal	JCXZ	Jump if CX= 0
JNE	Jump if not equal	JECXZ	Jump if ECX ≠ 0

Table 3: Jumps based on comparisons of unsigned operands

Instruction	Description	Instruction	Description
JA	Jump if above	JNBE	Jump if not below or equal
JAE	Jump if above or equal	JNB	Jump if not below
JB	Jump if below	JNAE	Jump if not above or equal
JBE	Jump if below or equal	JNA	Jump if not above

Note: These are only meaningful when comparing unsigned values

Table 4: Jumps based on comparisons of signed operands

Instruction	Description	Instruction	Description
JG	Jump if above	JNLE	Jump if not less or equal
JGE	Jump if above or equal	JNL	Jump if not less
JL	Jump if less	JNGE	Jump if not greater or equal
JLE	Jump if less or equal	JNG	Jump if not greater

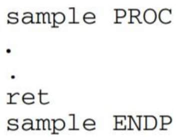
Note: These are only meaningful when comparing signed values

C) DEFINING and USING PROCEDURES

PROC Directive

Creating Procedures

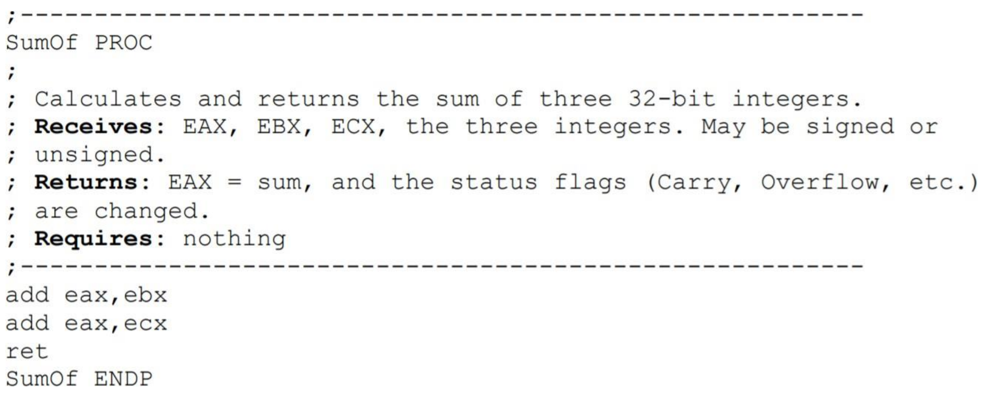
- o Large problems can be divided into smaller tasks to make them more manageable
- o A procedure is the ASM equivalent of a Java or C++ function
- o Following is an assembly language procedure named sample:



```
sample PROC
.
.
ret
sample ENDP
```

- Documenting Procedures
    - o A description of all tasks accomplished by the procedure
    - o **Receives**: A list of input parameters; state their usage and requirements
    - o **Returns**: A description of values returned by the procedure
    - o **Requires**: Optional list of requirements called preconditions that must be satisfied before the procedure is called

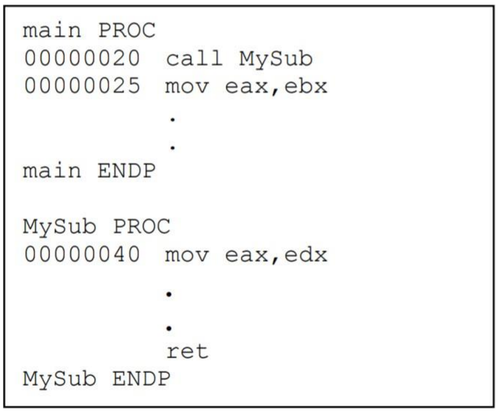
      Note: If a procedure is called without its preconditions satisfied, it will probably not produce the expected output



```
;---------------------------------------------------------
SumOf PROC
;
; Calculates and returns the sum of three 32-bit integers.
; Receives: EAX, EBX, ECX, the three integers. May be signed or
; unsigned.
; Returns: EAX = sum, and the status flags (Carry, Overflow, etc.)
; are changed.
; Requires: nothing
;---------------------------------------------------------
add eax,ebx
add eax,ecx
ret
SumOf ENDP
```

RET Instructions

o The RET instruction returns from a procedure back to the next instruction after CALL instruction



```
main PROC
00000020  call MySub
00000025  mov eax,ebx
          .
          .
main ENDP

MySub PROC
00000040  mov eax,edx
          .
          .
          ret
MySub ENDP
```

0000025 is the offset of the instruction immediately following the CALL instruction

00000040 is the offset of the first instruction inside MySub

Part B – Let's do a little programming by example

You are given a few examples here. Try them out.

Example 1

Convert the character in AL to upper case.
*(**Note: 'A' = 41h; 'a' = 61h; 'Z' = 5Ah; 'z' = 7Ah*)

```
MOV AL, 'a'        ; AL = 01100001B = 'a'
AND AL, 11011111B  ; AL = 01000001B = 'A'
```

Does the same code work for 'z'?

Example 2

Increment AX by 1 until reaches the value of 10. This is essentially doing a loop using a CMP command.

```
;using CMP MOV
EAX,0
L1:
INC AX
CMP AX, 10
JL L1
MOV TOTAL, AX
```

```
;using LOOP
MOV EBX,0 MOV
ECX,10
L2:
INC BX
LOOP L2
MOV TOTALS, BX
```

Note: the result of both TOTAL and TOTALS are the same.

Example 3

Some conditional jumps examples.

```
MOV AX,4
CMP AX,4  ; compare AX with 4
JE L1     ; if AX = 4 then jump to L1

MOV BX,0AAAAH ; do this if AX ≠ 4
JMP HERE  ; use this to guide the program sequence
L1:
MOV BX, 0BBBBH ; do this if AX = 4
HERE:

CALL DUMPREGS
```

```
MOV   AX,TOTAL   ; say TOTAL can be 2 or 4
SUB   AX,2
JZ    L2         ; if ZF = 1,jump to L2

MOV   BX,0AAAAH  ; this is done if TOTAL = 4
JMP   HERE
L2:              ; this is done if TOTAL = 2
MOV BX,0BBBBH
HERE:
CALL DUMPREGS
```

Example 4

A look into signed and unsigned comparisons. The same value compared as signed and unsigned will yield different results. JA is a jump based on unsigned comparison while JG is a jump based on signed comparison. In the example below, JA will not go to L3 (unsigned 7Fh is smaller than unsigned 80h) but JG will jump to L4 (signed 7Fh is larger than signed 80h).

```
MOV AX,7FH
MOV BX,80H
CMP AX,BX
JA L3    ; jump based on unsigned comparison

MOV CX,0AAAAH
JMP HERE
L3:
MOV CX, 0BBBBH
HERE:

CALL DUMPREGS

MOV AX,+127 ;signed version of 7FH
MOV BX,-128 ;signed version of 80H
CMP AX,BX
JG L4   ; jump based on signed comparison

MOV DX,0DDDDH
JMP SINI
L4:
MOV DX,0EEEEH
SINI:

CALL DUMPREGS

EXIT
```

Part C – Let's do a little programming on your own

1. In the following instruction sequence, show the value of AL for each line of code. Write the value in hexadecimal.

Instructions	Value of AL (H)
MOV AL,01100001B AND AL,00011101B	61H 01H
MOV AL,12H AND AL,3BH	12H 12H
MOV AL,00001111B OR AL,72H	0FH 7FH
MOV AL,83H XOR AL,26H	83H A5H

2. Write instructions in assembly language code that:

   a. Jumps to label L1 if either bit 4, 5 or 6 is set in the BL register.

```
Ans:  test    bl, 01110000b
      jnz     L1
```

   b. Jumps to label L1if bits 4, 5, and 6 are all set in the BL register.

```
Ans:  push    bx
      and     bl, 01110000b
      cmp     bl, 70h
      pop     bx
      jz      L1
```

   c. Jumps to label L2 if AL has even parity.

```
Ans:  or      al, al
      jpe     L2
```

   d. Jumps to label L3 if EAX is negative.

```
Ans:  cmp     eax, 0
      jl      L3
```

   e. Jumps to label L4 is the expression (EBX – ECX) is greater than zero.

```
Ans:  cmp     ebx, ecx
      jg      L4
```

3. Analyse the following code segment and answer the following questions.

```
CMP EAX, 20
JG L1
JL L2
L1:
MOV EBX, 1
JMP OUTT
L2:
MOV EBX, 0
OUTT:
```

a. If EAX=25, which conditional jump is taken. Please explain your answer. What is the final value of EBX?

Ans: L1 conditional jump is taken.

As EAX is greater than 20.

The final value of EBX is 1 in decimal.

b. Why is the JMP OUTT instruction needed for this code segment? Please elaborate your answer.

Ans: If not have JMP OUTT instruction, this code will go in L2.
The final value of EBX will be affected.

4. Copy the assembly programming code below. Complete the three procedures (i),(ii) and (iii) according to the comments given.

```
TITLE MASM Template                               (main.asm)
; Description:
;
; Revision date:
INCLUDE Irvine32.inc
INCLUDE Macros.inc

;---------------------------------------------------------------------------
; Receives: SumProc, a summation procedure
;        ECX as n, to calculate 1+2+...+n
;---------------------------------------------------------------------------

mCallSumProc MACRO SumProc:REQ
   push    ecx              ; decrements ESP and copies ECX into stack
   call    GetMseconds      ; get start time
   mov     esi,eax
   call    SumProc
   mWrite  "&SumProc: "     ; mWrite macro displays string on console
   call    WriteDec
   call    crlf

   call    GetMseconds      ; get start time
   sub     eax,esi
   call    WriteDec         ; display elapsed time
   mWrite <' millisecond(s) used', 0Dh,0Ah, 0Dh,0Ah>
   pop ecx
ENDM

.code
main PROC
   call    Clrscr
   mWrite "To calculate 1+2+...+n, please enter n (1~4294967295): "
   call    ReadDec               ; read value from user
   mov     ecx, eax
   call    crlf

   mCallSumProc Using_LOOP
   mCallSumProc Using_DEC_JNE
   mCallSumProc Using_DEC_JECXZ_JMP
   call WaitMsg
   exit
main ENDP
```

```
;---------------------------------------------------------------------------
; (i) Receives: ECX, as n, an integer to calculate 1+2+...+n
;    Returns:  EAX, the sum of 1+2+...+n
;---------------------------------------------------------------------------
Using_LOOP PROC
      mov eax,0

L1:
      add eax, ecx
      loop L1
   ret
Using_LOOP ENDP

;---------------------------------------------------------------------------
; (ii) Receives: ECX, as n, an integer to calculate 1+2+...+n
;    Returns:  EAX, the sum of 1+2+...+n
;---------------------------------------------------------------------------
Using_DEC_JNE PROC
     mov eax,0

L2:
     add eax, ecx
     sub ecx,1
     cmp ecx, 0
     jne L2
   ret
Using_DEC_JNE ENDP

;---------------------------------------------------------------------------
; (iii)Receives: ECX, as n, an integer to calculate 1+2+...+n
;    Returns:  EAX, the sum of 1+2+...+n
;---------------------------------------------------------------------------
Using_DEC_JECXZ_JMP PROC
     mov eax,0

L3:
     add eax, ecx
     sub ecx,1
     jecxz _Exit
     jmp L3

_Exit:
   ret
Using_DEC_JECXZ_JMP ENDP

END main
```