Lecture : 5

jmp, loop INSTRUCTIONS AND BUILT-IN PROCEDURES

Objectives:

- jmp Instruction
- loop Instruction
- Built-in Procedures

Section 01 | jmp Instruction

Assembly language programs use conditional instructions to implement high-level statements such as IF statements and loops. Each of the conditional statements involves a possible transfer of control (jump) to a different memory address. A transfer of control, or branch, is a way of altering the order in which statements are executed. There are two basic types of transfers:

Unconditional Transfer: Control is transferred to a new location in all cases; a new address is loaded into the instruction pointer, causing execution to continue at the new address. The JMP instruction does this.

Conditional Transfer: The program branches if a certain condition is true. A wide variety of conditional transfer instructions can be combined to create conditional logic structures. The CPU interprets true/false conditions based on the contents of the ECX and Flags registers.

The jmp instruction causes an unconditional transfer to a destination, identified by a code label that is translated by the assembler into an offset. The syntax is

```
jmp destination
```

When the CPU executes an unconditional transfer, the offset of destination is moved into the instruction pointer, causing execution to continue at the new location.

CMP Instruction

The CMP instruction compares two operands. It is generally used in conditional execution. 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. It is used along with the conditional jump instruction for decision making. Syntax is:

```
CMP destination, source
```

Mnemonic	Description
JE	Jump if equal
JG/JNLE	Jump if greater/Jump if not less than or equal
JL/JNGE	Jump if less/Jump if not geater
JGE/JNL	Jump if greater or equal/Jump if less
JLE/JNG	Jump if less or equal/Jump if not greater
JNE	Jump if not equal

Some conditional jump instructions can also test values of the individual CPU flags.

Mnemonic	Description	Flags
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

Example: Printing numbers from 1 to 8

```
INCLUDE Irvine32.inc

.code
main PROC
    mov eax, 1

    start1:
        cmp eax, 9
        je end1
        call WriteDec
        call Crlf
        add eax, 1
        jmp start1

    end1:
```

```
        exit
main ENDP
END main
```

Section 02 | loop Instruction

The loop instruction, formally known as Loop According to ECX Counter, repeats a block of statements a specific number of times. ECX is automatically used as a counter and is decremented each time the loop repeats. Its syntax is:

```
loop destination
```

The execution of the loop instruction involves two steps: First, it subtracts 1 from ECX. Next, it compares ECX with zero. If ECX is not equal to zero, a jump is taken to the label identified by destination. Otherwise, if ECX equals zero, no jump takes place, and control passes to the instruction following the loop.

Example 01: Printing numbers from 1 to 10

```
INCLUDE Irvine32.inc

.code
main PROC
    mov eax, 1
    mov ecx, 10

    start1:
        call WriteDec
        call Crlf
        add eax, 1
        loop start1

        exit
main ENDP
END main
```

Example 02: Printing an Array

```
INCLUDE Irvine32.inc

.data
    array DWORD 100, 200, 300, 400, 500
.code
main PROC
    mov esi, OFFSET array
    mov ecx, LENGTHOF array

    print_array:
```

```
        mov eax, [esi]
        call WriteDec
        call Crlf
        add esi, TYPE array
        loop print_array
        exit
main ENDP
END main
```

Nested Loop

When creating a loop inside another loop, special consideration must be given to the outer loop counter in ECX. You can save it in a variable.

Section 03 | Procedures in Irvine32 Library

1. `Clrscr`
    - Clears the console window and locates the cursor at the above left corner.
2. `Crlf`
    - Writes the end of line sequence to the console window.
3. `WriteBin`
    - Writes an unsigned 32-bit integer to the console window in ASCII binary format.
4. `WriteChar`
    - Writes a single character to the console window.
5. `WriteDec`
    - Writes an unsigned 32-bit integer to the console window in decimal format.
6. `WriteHex`
    - Writes a 32-bit integer to the console window in hexadecimal format
7. `WriteInt`
    - Writes a signed 32-bit integer to the console window in decimal format.
8. `WriteString` (EDX= OFFSET String)
    - Write a null-terminated string to the console window.
9. `ReadChar`
    - Waits for single character to be typed at the keyboard and returns that character.

10. `ReadDec`
    - Reads an unsigned 32-bit integer from the keyboard.
11. `ReadHex`
    - Reads a 32-bit hexadecimal integers from the keyboard, terminated by the enter key.
12. `ReadInt`
    - Reads a signed 32-bit integer from the keyboard, terminated by the enter key.
13. `ReadString` (EDX=OFFSET, ECX=SIZEOF)
    - Reads a string from the keyboard, terminated by the enter key.
14. `Delay` (EAX)
    - Pauses the program execution for a specified interval (in milliseconds).
15. `Randomize`
    - Seeds the random number generator with a unique value.
16. `DumpRegs`
    - Displays the EAX, EBX, ECX, EDX, ESI, EDI, ESP, EIP and EFLAG registers.
17. `DumpMem` (ESI=Starting OFFSET, ECX=LengthOf, EBX=Type)
    - Writes the block of memory to the console window in hexadecimal.
18. `getDateTime`
    - Gets the current date and time from system
19. `GetMaxXY` (DX=col, AX=row)
    - Gets the number of columns and rows in the console window buffer.
20. `GetTextColor` (Background= Upper AL, Foreground= Lower AL)
    - Returns the active foreground and background text colors in the console window.
21. `Gotoxy` (DH=row , DL=col)
    - Locates the cursor at a specific row and column in the console window. By default X coordinate range is 0-79,and Y coordinate range is 0-24.
22. `MsgBox` (EDX=OFFSET String, EBX= OFFSET Title)
    - Displays a pop-up message box.
23. `MsgBoxAsk` (EDX=OFFSET String, EBX= OFFSET Title)
    - Displays a yes/noquestion in a pop-up message box. (EAX=6 YES, EAX=7 NO)

24. `SetTextColor` (EAX= Foreground + (Background*16))
    - Sets the foreground and background colors of all subsequent text output to the console.
25. `WaitMsg`
    - Display a message and wait for the Enter key to be pressed.

Example Programs

Example 01

`WriteDec`: The integer to be displayed is passed in EAX
`WriteString:` The offset of string to be written is passed in EDX `WriteChar`
The character to be displayed is passed in `AL`

```
INCLUDE Irvine32.inc

.data
     Dash BYTE " - ", 0
.code
     main PROC
          mov ecx, 255
          mov eax, 1
          mov edx, OFFSET Dash

          L1:
               call WriteDec ; EAX is a counter
               call WriteString ; EDX points to string
               call WriteChar ; AL is the character
               call Crlf
               inc al ; next character
               Loop L1
          exit
     main ENDP
END main
```

Example 02

DumpMem: Pass offset of array in ESI, length of array in ECX & type in EBX.
ReadInt: Reads the signed integer into EAX.
WriteInt: Signed integer to be written is passed in EAX.
WriteHex: Hex value to be written is passed in EAX
WriteBin: Binary value to be written is passed in EAX

```
INCLUDE Irvine32.inc

.data
     COUNT = 4
     arrayD SDWORD 12345678h, 1A4B2000h, 3434h, 7AB9h
     prompt BYTE "Enter a 32-bit signed integer: ", 0
.code
     main PROC
          ; Display an array using DumpMem.
          mov esi, OFFSET arrayD          ; starting OFFSET
          mov ebx, TYPE arrayD            ; doubleword = 4 bytes
          mov ecx, LENGTHOF arrayD        ; number of units in arrayD
          call DumpMem                    ; display memory
          call DumpRegs

     ; Ask the user to input a sequence of signed integers

          call Crlf ; new line

          mov ecx, COUNT
          L1:
               mov edx, OFFSET prompt
               call WriteString
               call ReadInt               ; input integer into EAX
               call Crlf                  ;new line

               ; Display the integer in decimal, hexadecimal, and binary
               call WriteInt              ; display in signed decimal
               call Crlf
               call WriteHex              ; display in hexadecimal
               call Crlf
               call WriteBin              ; display in binary
               call Crlf
               call Crlf
               Loop L1 ; repeat the loop
          exit
     main ENDP
END main
```

Example 03

SetTextColor: Background & foreground colors are passed to EAX

```
INCLUDE Irvine32.inc

.data
	str1 BYTE "Sample string in color", 0dh, 0ah, 0
.code
	main PROC
		mov eax, yellow + (blue*16)
		call SetTextColor
		mov edx, OFFSET str1
		call WriteString
		call DumpRegs
	exit
	main ENDP
END main
```

Example 04

MsgBoxAsk: (EDX=OFFSET String, EBX= OFFSET Title)
Displays a yes/no question in a pop-up message box. (EAX=6 YES, EAX=7 NO)

```
INCLUDE Irvine32.inc

.data
	caption  BYTE "Survey Completed ", 0
	caption2 BYTE "Thanks for completing Survey. "
		 BYTE "Would you like to recieve the results?", 0
	result	 BYTE "Thanks for participating!", 0
.code
	main PROC
		mov ebx, 0
		mov ebx, OFFSET caption
		mov edx, OFFSET caption2
		call MsgBoxAsk

		mov edx, OFFSET result
		call WriteString

	exit
	main ENDP
END main
```