

Faculty of Computing

Year 1 Semester 1 (2024)

IIT1140 – Fundamental of Computing

Lab Sheet 02

Objectives:

1. *To understand the Assembly programming concepts.*
 2. *Apply the assembly syntaxes for branching and looping to implement a working program.*
-

Branching Instructions in Assembly Language

Several branching instructions accomplish conditional execution in assembly language. These instructions can change the flow of control in a program. We can simulate **if-else** conditions supported by higher-level programming languages using **CMP** and **branching** instructions. **Branching** instruction is observed in two scenarios namely Unconditional Jump and Conditional Jump.

1. Unconditional Jump

This is performed by the JMP instruction. Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps.

The JMP instruction provides a label name where the flow of control is transferred immediately. The syntax of the JMP instruction is,



JMP label

Example:

```
1 Label1;  
2 MOV dl, 'a' // Initializing dl to a  
3 MOV ah, 02 // Initializing ah to 2  
4 ADD dl, ah // Add dl, ah  
5 JMP Label1 // repeats the statements
```

Figure 2.1

Once the **JMP** instruction is executed, the control will transfer to the line which **Label1** specified. There is no condition evaluated in JMP statement. This is commonly used in loops, subroutine calls, and error-handling routines.

2. Conditional jump

This is performed by a set of jump instructions depending upon the condition. A conditional jump transfers the execution to a specified location only if a certain condition is met, based on the status flags set by previous instructions (e.g., zero flag, carry flag). There are numerous conditional jump instructions depending on the condition and data. The syntax of this instruction is,

***J*<condition> label**

2.1 Conditional jump instructions used on signed data used for arithmetic operations.

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JG , JNLE	Jump if Greater (>). Jump if Not Less or Equal (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL , JNGE	Jump if Less (<). Jump if Not Greater or Equal (not >=).	SF <> OF	JNL, JGE
JGE , JNL	Jump if Greater or Equal (>=). Jump if Not Less (not <).	SF = OF	JNGE, JL
JLE , JNG	Jump if Less or Equal (<=). Jump if Not Greater (not >).	ZF = 1 or SF <> OF	JNLE, JG

Figure 2.2

2.2 Conditional jump instructions used on unsigned data used for logical operations.

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JA , JNBE	Jump if Above (>). Jump if Not Below or Equal (not <=).	CF = 0 and ZF = 0	JNA, JBE
JB , JNAE, JC	Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry.	CF = 1	JNB, JAE, JNC
JAE , JNB, JNC	Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry.	CF = 0	JNAE, JB
JBE , JNA	Jump if Below or Equal (<=). Jump if Not Above (not >).	CF = 1 or ZF = 1	JNBE, JA

Figure 2.3

2.3 Conditional jump instructions have special uses and check the value of flags.

Instruction	Description	Condition	Opposite Instruction
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

Figure 2.4

This is used for decision-making processes in the code, such as if-else statements, loops, and comparisons.

Below is a summary of the conditional and unconditional jump instructions.

Feature	Unconditional Jump (JMP)	Conditional Jump (e.g., JE, JNE)
Control Transfer	Always	Only if the specified condition is met
Usage Scenarios	Loops, subroutines, errors	If-else, comparisons, decision making
Dependence on Flags	No	Yes, depends on the status flags set by previous instructions
Predictability	Predictable, always jumps	Conditional, may or may not jump
Complexity	Simple	Slightly more complex due to condition checks

3. CMP Instruction

The **CMP** instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other to compare 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. The syntax of the CMP instruction is,

CMP destination, source

CMP compares two numeric data fields. The destination operand could be either in the register or in memory. The source operand could be a constant (immediate) data, register, or memory.

Example:

```
1  CMP DX, 00;    //Compare the DX value with zero
2  JE  L7;        //If yes, then jump to label L7
```

Figure 2.5

Activity 1:

- a) Consider the Assembly code given in figure 2.6. Explain the purpose of each statement.

```
1  MOV AL, 80h
2  MOV BL, 7Fh
3  CMP AL, BL
4  JG EXIT
```

Figure 2.6

Line 1:

Line 2:

Line 3:

Line 4:

- b) Will the “JG” instruction in the following code result in a jump? (Yes/ No):

.....

Activity 2:

Refer to the Assembly code given in figure 2.7. Will the program jump to the "**GreaterThan**" label? Explain the reason for your answer.

```
1  MOV AX, 10
2  MOV BX, 5
3  CMP AX, BX
4  JG GreaterThan
```

Figure 2.7

.....

.....

.....

.....

Activity 3:

Consider the Assembly code given in figure 2.8 and briefly explain it.

```
MOV AX, 10
ADD AX, 5
JMP Exit

Loop:
    ; Code to be executed repeatedly

Exit:
    ; Code to execute after the loop
```

Figure 2.8

.....

.....

.....

.....

Looping Instructions in Assembly Language

Loops can also be used in assembly language, but unlike higher-level languages, it doesn't offer various loop types. Although the emu8086 emulator supports five loop syntaxes: LOOP, LOOPE, LOOPNE, LOOPNZ, and LOOPZ, they often lack the flexibility needed for many situations. Instead, we can create custom loops such as for, while, do-while using condition and jump statements.

LOOP instructions are used to execute a statement or set of statements for a specific number of times. The syntax is;

LOOP <Label_Name>

For Loop

A for loop contains three key parts: an initialization section where loop variables are set, a loop condition section determining if the loop should continue, and an increment/decrement section updating loop variables before the next iteration.

Activity 4:

The Assembly code to print a square of star (*) pattern series of 5 rows is given in figure 2.9. Execute this and understand the procedure.

```
*****
*****
*****
*****
*****

MAIN PROC
    MOV CX,5
    MOV BX,5

; Outer Loop (L1)
L1:
    PUSH CX
    MOV CX,5

; Inner Loop (L2)
L2:
    MOV AH,2
    MOV DL,'*'
    INT 21H

    LOOP L2

    MOV AH,2
    MOV DL,10
    INT 21H
    MOV DL,13
    INT 21H

    POP CX
    LOOP L1

EXIT:
    MOV AH,4CH
    INT 21H
    MAIN ENDP

END MAIN
```

Figure 2.9

Activity 5: Write the Assembly code to print a square pattern of 4 rows as given below.

```
++++  
++++  
++++  
++++
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....