JMP AND LOOP INSTRUCTIONS

- By default, the CPU loads and executes programs sequentially, however, control may be transferred to a new location in the program.

- A *transfer of control*, or *branch*, is a way of altering the order in which statements are executed, there are two basic types:

1. **Unconditional Transfer**: No condition is involved, control is transferred to a new location in all cases.

2. **Conditional Transfer:** The program branches if a certain condition is true (based on status of flags).

JMP Instruction

The `JMP` instruction causes an unconditional transfer to a destination, identified by a code label.

```
JMP destination
```

offset of *destination* is moved into the instruction pointer, causing execution to continue at the new location

```
top:    INC AX
        MOV BX, AX
        jmp top
```

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.

```
LOOP destination
```

The loop destination must be within -128 to +127 bytes of the current location counter.

- –128 bytes is the largest backward jump from current instruction +127 bytes is the largest forward jump.

The execution of the LOOP instruction involves two steps:

1. First, it subtracts 1 from ECX.
2. Next, it compares ECX to zero. If ECX is not equal to zero, a jump is taken to the label identified by *destination*. Otherwise, no jump takes place, and control passes to the instruction following the loop.

```
mov ax,0
mov ecx,5

L1:    inc ax
       loop L1
       mov bx,ax
```

YOUR TURN . . .

What will be the value of BX?

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
    mov bx, ax
```

NESTED LOOPS

- 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:

```
.data
      count DWORD ?
.code
      mov ecx,100             ; set outer loop count
L1:
      mov count,ecx           ; save outer loop count
      mov ecx,20              ; set inner loop count
L2:
       .
      loop L2                 ; repeat the inner loop
      mov ecx,count           ; restore outer loop count
      loop L1                 ; repeat the outer loop
```