PROGRAM FLOW

by controlling program flow through various command, we can take decisions inside the program based on conditions. One such command is **JMP** , that can skip and revert back to code segments using **labels.** Labels are defined with a colon after their name. Such as, label1: . they can be declared on the same or different line before an instruction. JMP syntax looks like , **JMP** ***label*****:**

```
01 org 100h
02 MOV AX, 20
03 MOV BX, 4
04
05 JMP calc:
06
07 back:
08 ret
09
10 calc:
11 add AX, BX
12 JMP back:
```

Above is an example of **unconditional jump**, but there are instructions that act only when a condition is in act.
These instructions are divided in three groups, first group just test single flag, second compares numbers as signed, and third compares numbers as unsigned. Changes in the flag registers provide conditions for various jumps.**JMP** instruction they can only jump **127** bytes forward and **128** bytes backward .Even though some instructions do the same thing, different names are used to make programs easier to understand , to code and most importantly to remember.

From page 26: “
very offset dissembler has no clue what the original instruction was look like that's why it uses the most common name. if you emulate this code you will see that all instructions are assembled into **JNB**, the operational code (opcode) for this instruction is **73h** this instruction has fixed length of two bytes, the second byte is number of bytes to add to the **IP** register if the condition is true. because the instruction has only 1 byte to keep the offset it is limited to pass control to -128 bytes back or 127 bytes forward, this value is always signed
”
Didn’t understand this part.

Below is an example for **conditional JMP**:

```
include "emu8086.inc"

org 100h

mov AL, 0
mov BL,5

cmp AL,BL

JE equal

putc 'n'
JMP stop


equal:
putc 'y'

stop:
ret
```

Loop

loop instructions are used to simplify the decrementing, testing and branching portion of the loop.The functionalities of loop instructions can be achieved by using JMP and CMP.all loop instructions use **CX** register to count step. It is possible to use nested loop. Following is example of nested loop:

```
include 'emu8086.inc'
org 100h

mov BX,0

mov CX,5

k1:
  add BX,1
  mov AL,'1'
  mov AH,0eH
  int 10H
  push CX
  mov CX,5
  putc AL
    k2:
      add BX,1
      mov AL,'2'
      mov AH,0eH
      int 10H
      push CX
      mov CX,5
      putc AL
        k3:
          add BX,1
          mov AL,'3'
          mov AH,0eH
          int 10H
          loop k3
      pop CX
      loop k2
    pop CX
    loop k1

ret
```

I know this code loops through the labels, but I m not sure how.I have difficulties, navigating through the emulator.
"it is possible store original value of cx register using **push cx** instruction and return it to original when the internal loop ends with **pop cx.**"
"when the address of desired location is too far assemble automatically assembles reverse and long jump instruction, making total of 5 bytes instead of just 2, it can be seen in disassembler as well. "
Didn't understand these parts from page 29.
Didn't understand page 30-31.