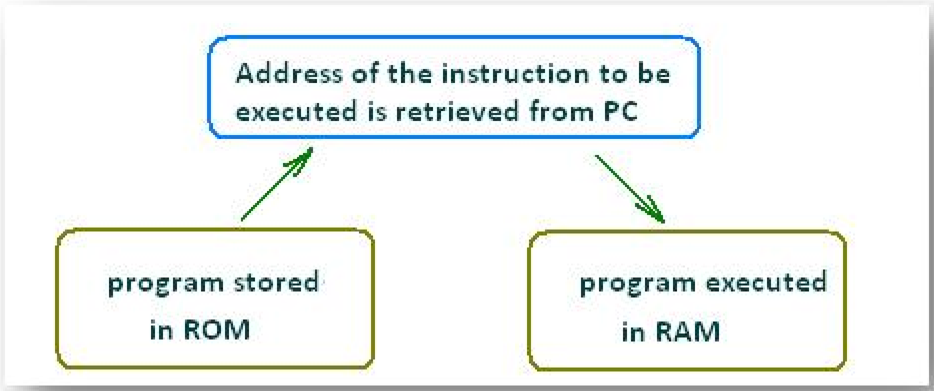
JUMP, LOOP AND CALL INSTRUCTIONS

After you have understood the tutorial on Introduction to assembly language which includes simple instruction sets like input/output operations, now it's time to learn how to create loops, function calls and jumps while writing a code in assembly language.

Let us first discuss an important concept that relates RAM & ROM. You can skip this section if you wish but it is an important to understand registers which helps in building up understanding of architecture of microcontrollers.

How a program which is burned in ROM gets executed in RAM?
The program you write is burned in ROM and this program is executed in RAM. The data burned is in the form of logics or binary digits (0 & 1) also called an Opcode or machine code. The program written in assembly language is converted to opcode by assembler.
Each line of the assembly code is assigned a unique opcode by the assembler as shown below. These opcodes are then stored in ROM one after another. **There is a register called Program Counter (PC)** which always points to the current opcode being executed. When the power is switched on it sets itself to zero and it keeps on incrementing itself as opcodes are executed one after another. This information is used by RAM to extract correct opcode from ROM which is then executed. This process goes on line by line till the whole program is executed.


Address of the instruction to be executed is retrieved from PC
program stored in ROM
program executed in RAM

Let’s take an example:-

```
PC                            Mnemonic, Operand       Opcode (Machine code)
0000                             ORG      0H
0000                             MOV      R0, #0              7800
0002                             MOV      A, #55H                                    7455
0004                             JZ       NEXT                 6003
0006                             INC      R0                  08
0007          AGAIN:               INC      A                   04
0008                             INC      A                   04
0009          NEXT:                ADD      A, #77H              2477
000B                              JNC      OVER                5005
000D                             CLR      A                    E4
000E                              MOV      R0, A                F8
000F                              MOV      R1, A                F9
0010                             MOV      R2, A                FA
0011                             MOV      R3, A                FB
0012         OVER:                ADD,     R3                   2B
0013                             JNC      AGAIN                50F2
0015          HERE:               SJMP     HERE                   80FE
0017                             END
```

We can clearly see that the PC increases with execution of program. The PC starts with zero when ‘ORG 00H’ line is executed and then in subsequent lines PC keeps on incrementing as machine codes are executed one after another.

Program Counter is a 2 byte or 16 bit register. Therefore we cannot have internal ROM of more than the number this register can hold (i.e. not exceeding the FFFF hex value).

LOOP AND JUMP INSTRUCTIONS

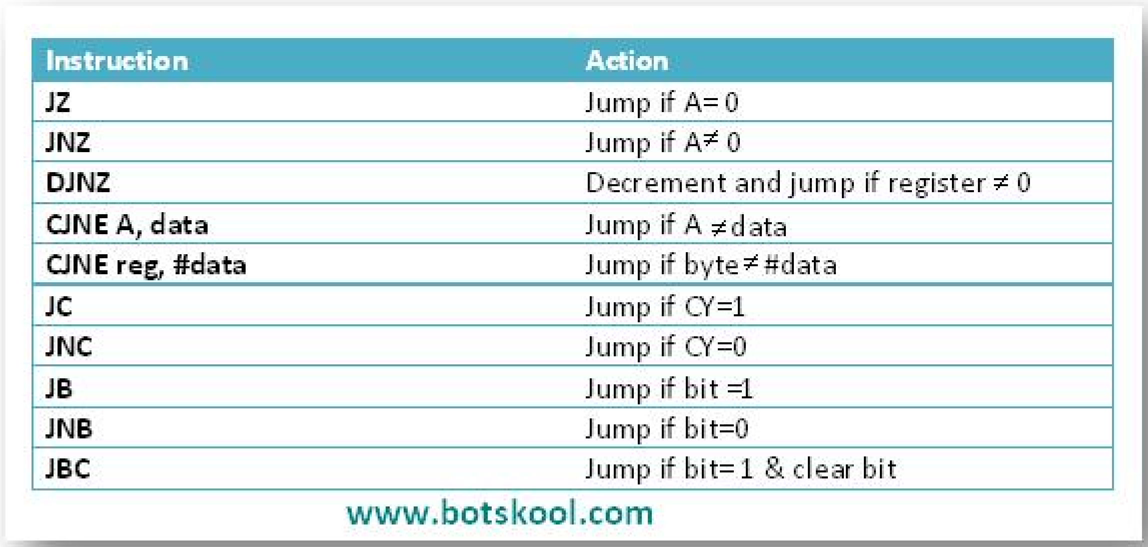
Let us start with a simple example that will help you to learn how to create loops in assembly. In the following code the instruction DJNZ is used to reduce the counter and is repeated till the counter becomes zero.

Eg-1:

```
                ORG    0H
                MOV    A, #0              ; clear A
                MOV    R1, #10            ; load counter R1 =10
AGAIN:           ADD    A, # 05            ; add five to register A
                DJNZ   R1, AGAIN          ; repeat until R1=0 (10 times)
                MOV    R3, A              ; save A in R3
                END
```

In this code R1 acts as a counter. The counter value is initialized i.e. 10 HEX is loaded to R1. In each iteration, the instruction DJNZ decrements R1 by one until it becomes zero. This loop adds 5 HEX to A every time it runs. After ten iterations R1 becomes zero and the instructions below it are executed.

Note: - Some Jump statements can only be performed on some special register A (or bit CY) as mentioned in the table below.



Instruction	Action
JZ	Jump if A= 0
JNZ	Jump if A≠ 0
DJNZ	Decrement and jump if register ≠ 0
CJNE A, data	Jump if A ≠data
CJNE reg, #data	Jump if byte≠#data
JC	Jump if CY=1
JNC	Jump if CY=0
JB	Jump if bit =1
JNB	Jump if bit=0
JBC	Jump if bit= 1 & clear bit

www.botskool.com

Nested loops:

```
            ORG 0H
            MOV    A, #55H          ; A= 55 hex
            MOV    R1, #100         ; the outer counter R1 =100
NEXT:         MOV    R2, # 20         ; the inner counter
AGAIN:        CPL     A, # 05         ; add five to register A
            DJNZ   R2, AGAIN         ; repeat until R1=0 (100 times)
            DJNZ   R1, NEXT          ; repeat till 20 times (outer loop)
            END
```

SJMP refers to short jump and LJMP refers to long jump. All the conditional jumps are short jumps.

SJMP: This instruction is of two bytes in which first one is opcode & second is the address. The relative address of the instruction called should be in between -127 to 127 bytes from the current program counter (PC).

LJMP: This instruction is of three bytes in which the first is the opcode and the second & third are for address. The relative address of the instruction can be anywhere on the ROM.

So it is clear from the above examples that we can use different jump instructions with a condition or counter called conditional loop. And when we create loop inside an existing loop it is called nested loop.

CALL INSTRUCTIONS:

Example:

LCALL (long call)

```
                ORG      0H
BACK :            MOV     A, #55H                       ; load A= 55 hex value
                MOV     P1, A                         ; issue value of register A to port1
                LCALL  DELAY                          ; to call DELAY function created below
                MOV A, #0AAH                         ;load AAH hex value to A
                MOV P1,A                             ;issue value of register A to port 1
                LCALL   DELAY                         ; to call DELAY function as created
        below
                SJMP     BACK                          ; keep doing this
        ; ________ this is the delay subroutine

DELAY:
                  MOV        R5, #0FFH               ; R5= 255 hex, the counter
AGAIN:              DJNZ     R5, AGAIN                ; stay here until R5 becomes zero
                  RET                                 ; return to caller
                  END
```

In this code we keep on toggling the value of the register of port 1 with two different hex values and a DELAY subroutine is used to control how fast the value is changing. Here in DELAY subroutine the program is kept busy by running an idle loop and counting 256 counts. After the DELAY subroutine is executed once the value of port 1 is toggled and this process goes on infinitely.

By using DELAY we can create PWM (Pulse Width Modulation) to control motors or LED blinking for further details view our tutorial on Input/ output instructions in Assembly Language[coming soon].

We can also use ACALL i.e. absolute call for calling a subroutine that is within 2K byte of PC.

Source : http://www.botskool.com/tutorials/electronics/8051/jump-loop-and-call-instructions