Program Flow Instructions

General Introduction

- Program flow instructions modify the value of PC upon execution.
- Divided in two groups:
  - Subroutine instructions: **call, return, return from interrupt**
  - Jump or branch instructions: Conditional and unconditional jumps

Program Flow: Subroutine Instructions

CALL   and RETURN

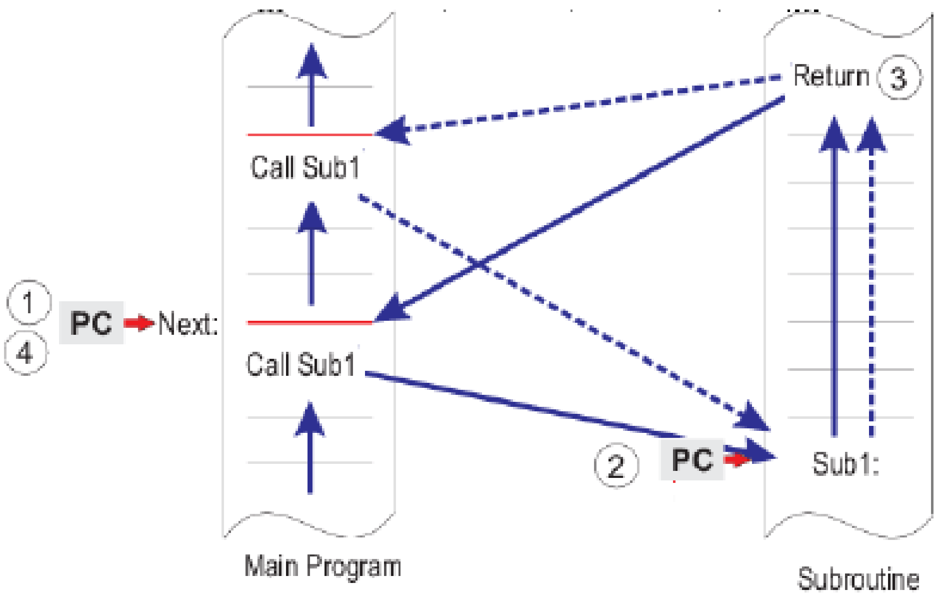
Subroutines and Procedures

- Subroutines (also called functions or procedures) are pieces of executable code written and stored apart from the main code
- They are to be executed when invoked from main code or other subroutine, but flow must return to original “normal” flow
- Subroutine may be located anywhere in the source, before or after main code.

Instructions: Call and Return

- **call dest**: Pushes PC and loads PC with dest, which points to address of first instruction of subroutine (entry line)
- **ret :** pops PC

Subroutine process



1. Just before CALL execution PC points to next memory location after CALL.
2. Upon execution, the content of PC is pushed onto stack, and PC loaded with address of subroutine entry line
3. Subroutine is executed until instruction **RET** (return) is found.
4. Execution of RET pops PC, restoring the address of the instruction just after CALL

--- This happens every time CALL is executed

Important remark: Subroutine must be designed so that when RET is encountered SP is pointing to the right location

Program Flow: Jumps

General Concepts

- Jumps simply change the flow, without implying any return.
- Unconditional jumps change the value of PC.
- Conditional jumps change the value of PC only when certain flag conditions are met
  - Normally, a conditional jump is preceded by one instruction affecting flags.
- Jumps are limited in memory space, microcontroller dependent
  - MSP430 jumps are limited within +- 1024 spaces

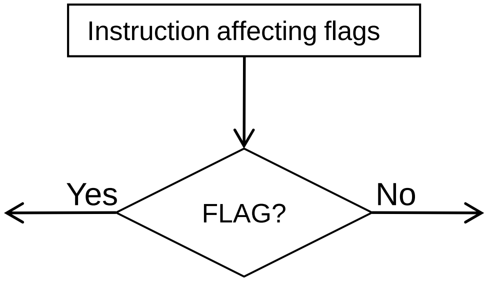
General Concepts (2)

- Conditional jumps are used to emulate control structures of the type, IF-THEN, IF-ELSE-THEN, FOR LOOPS, WHILE LOOPS, etc.

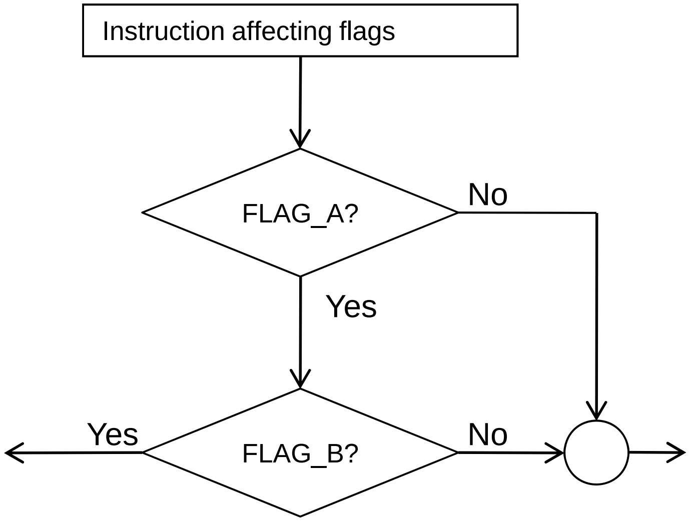
- A conditional jump test if a flag or a relation among flags is set or not set.

Principle governign Conditional Jump and the Control Structures (cont)

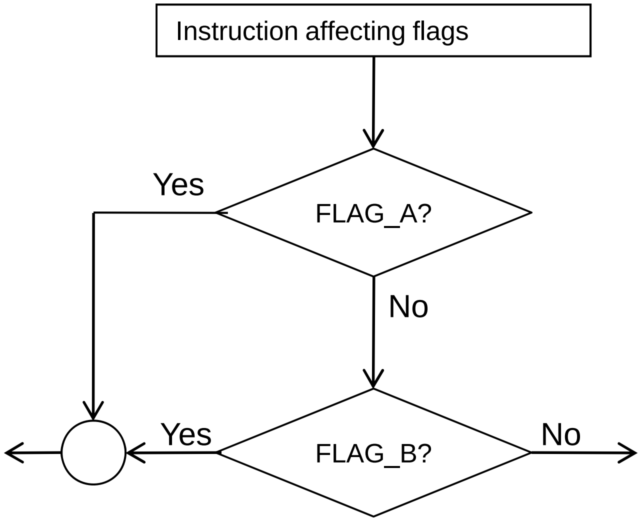
- Single Flag(s) condition



- Multiple Flags:
FLAG_A AND FLAG_B



- Multiple Flags:
FLAG_A OR FLAG_B



Unconditional Jump and the Branch Instruction

- **jmp label**: loads PC with label

- **br dest:** branch to dest, is the same that *mov dest,PC.*

Basic conditional jumps (Not all present in all CPU’s)

- **jz:** jump if zero (jumps if Z=1) ** (in MSP430)
- **jnz:** jump if not zero (jumps if Z=0) **
- **jc:** jump if carry (jumps if C=1)**
- **jnc:** jump if no carry (jumps if C=0)**
- **jv:** jump if overflow (jumps if V=1)
- **jnv:** jump if no overflow (jumps if V=0)
- **jn:** jump if negative (jumps if N=1)**
- **jp:** jump if not negative (jumps if N=0)

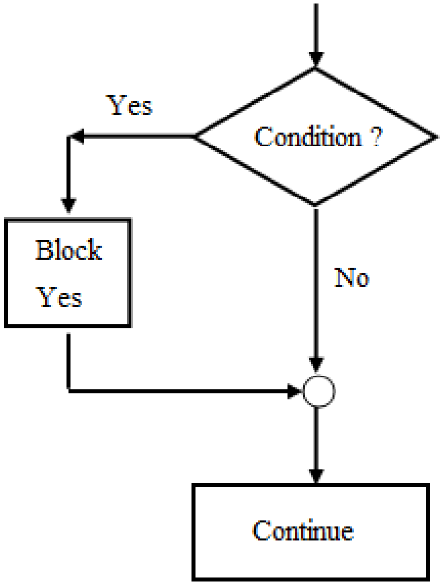
Conditional Jump (cont): Other mnemonics and jumps

- These are used when comparing numbers
- **jeq:** jump if equal (= **jz**)
- **jne:** jump if not equal (= **jnz**)
- **jhs:** jump if higher or same, for unsigned numbers (= **jc**)
- **jlo:** jump if lower, for unsigned numbers (= **jnc**)
- **jge:** jump if greater or equal, for signed numbers (jumps if N=V)**
- **jl:** jump if less, for signed numbers (jumps if N≠V)

Conditional and loop structurs

Code structure in assembly

IF-THEN



(a) Flow Diagram If_Then Structure

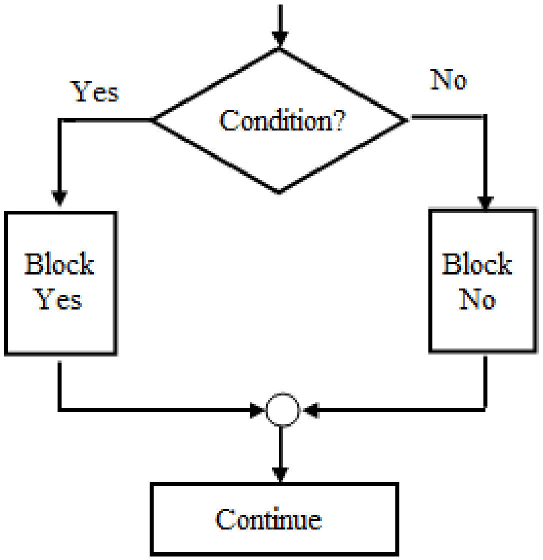
```
 Test_Ints:       -------
                jump_if_No  Continue

Block_Yes:        -----
                  ------

End_Block:      -------
Continue:       -------
```

(b) Negative_jump code structure

```
Test_Ints:       ------
                 jump_if_Yes    Block_Yes
                 jmp         Continue

Block_Yes:        -----
                  ------

End_Block:       -------
Continue:        -------
```

(c) Positive_Jump code structure

IF_ELSE



(a) Flow Diagram IF_ELSE Structure

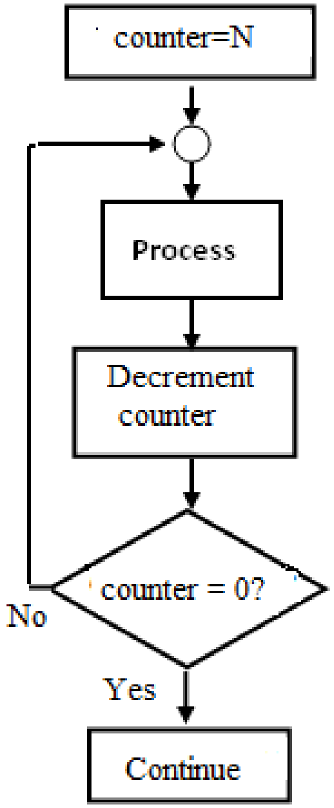
```
Test_Inst:       -----------
                jump_if_NO  Block_NO

Block_Yes:      ---------
                ---------

End_Yes:        jmp Continue
Block_NO        ----------
                ----------


End_NO          ----------
Continue:       -----------
```

(b) Negative_jump code structure

```
Test_Inst:       -----------
                jump_if_YES  Block_YES
                jump_if_NO   Block_NO
Block_Yes:      ---------
                ---------

End_Yes:        jmp Continue
Block_NO        ----------
                ----------


End_NO          ----------
Continue:       -----------
```

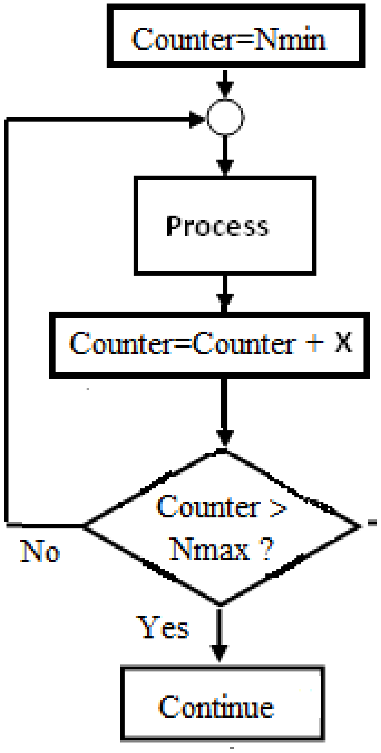
(c) Positive_Jump   code structure

FOR-LOOPS



```
            mov   #N,counter
FOR_Loop:   ---------------
            -------------
            dec   counter
End_FOR     jnz   FOR_Loop

Continue:   -------
```

(a) For counter= N to 1, step -1



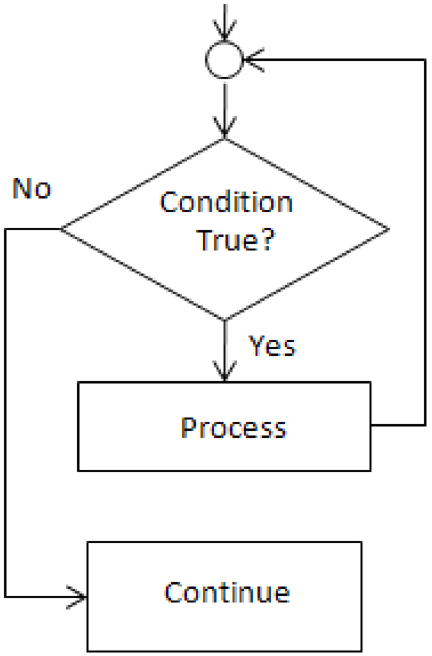
```
            mov   #Nmin, Counter
FOR_Loop:   ------------
            ------------

            add   #X, Counter
            cmp   #Nmax,Counter
IF_Lower:   jlo   FOR_Loop
IF  Equal:  jeq  FOR_Loop
IF_Greater: -----------
            ------------
```

(b) For Counter = Nmin to Nmax, step X

WHILE LOOP

```
WHILE  <condition is TRUE> DO
            Loop Process
END_WHILE
```

(a)



(b)

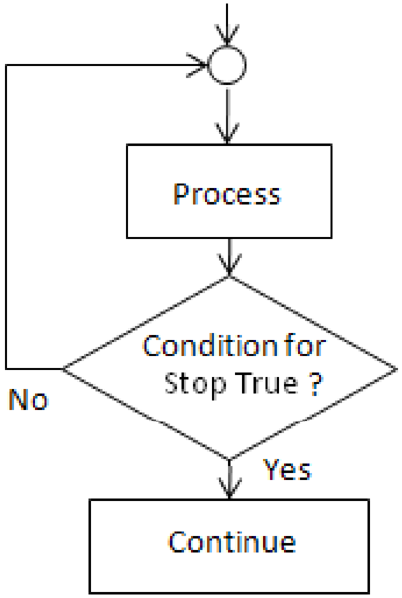
```
                --------------
                --------------
WhileTest:      <Instruction for testing parameter>
                 <Jump_if_False  to Continue>
While_Loop:      -------------
                 -------------
                 <Process, changing parameter>
EndWhile:      jmp  WhileTest

Continue:       -----------
                ------------
                -------------
```

(c)

REPEAT-UNTIL LOOP

```
REPEAT     Loop Process
UNTIL <Stop Condition True>
END UNTIL
```

(a)

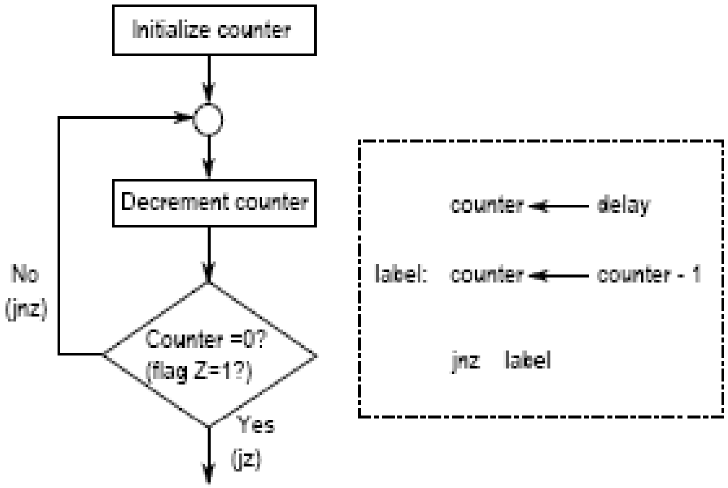


(b)

```
RepeatLoop:    -------------
               -------------
              <Process, changing parameter>
              ---------------
RepeatTest:   <Instruction for testing parameter>
EndRepeat:    <Jump_if_False  to  RepeatLoop>

Continue:     -----------
             ------------
             -------------
```

(c)

Examples: Delay loop and iteration loop

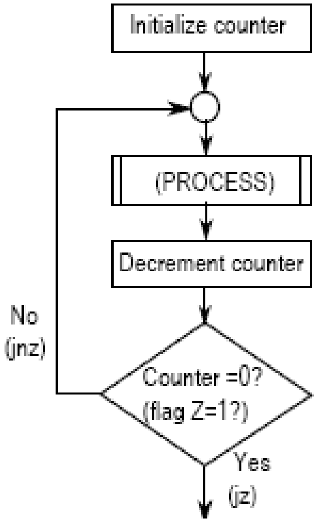


```
        counter ←— delay
label:  counter ←— counter - 1
        jnz   label
```

(a)

Example:

```
          mov  #delay,R15
DelLoop:   dec   R15
          jnz   DelLoop
```



```
        counter ←— delay
label:  [ PROCESS ]
        counter ←— counter - 1
        jnz   label
```

Example: Extended delay loop

```
          mov  #delay,R15
DelLoop:  nop
          dec   R15
          jnz   DelLoop
```