# ECE 3120
# Computer Systems
# Programming Loops

---

Manjeera Jeedigunta
http://blogs.cae.tntech.edu/msjeedigun21
Email: msjeedigun21@tntech.edu
Tel: 931-372-6181, Prescott Hall 120

- Prev:
  - Write programs to do arithmetic
- Today:
  - Loops

# Program Loops

Types of program loops: *finite* and *infinite* loops

## Looping mechanisms:

1. **do** *statement S* **forever**

2. **For** i = *n1* **to** *n2* **do** *statement S* or **For** i = *n2* **downto** *n1* **do** *statement S*

3. **While** *C* **do** *statement S*

4. **Repeat** *statement S* **until** *C*

Program loops are implemented by using the conditional branch instructions and the execution of these instructions depends on the contents of the CCR register.

# Do Statement S forever

- Infinite loop
- Possible to add “If C then exit”

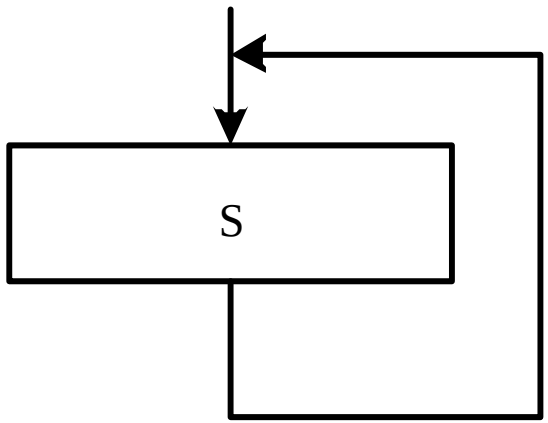



Figure 2.4 An infinite loop

# For i=n1 to n2 Do S

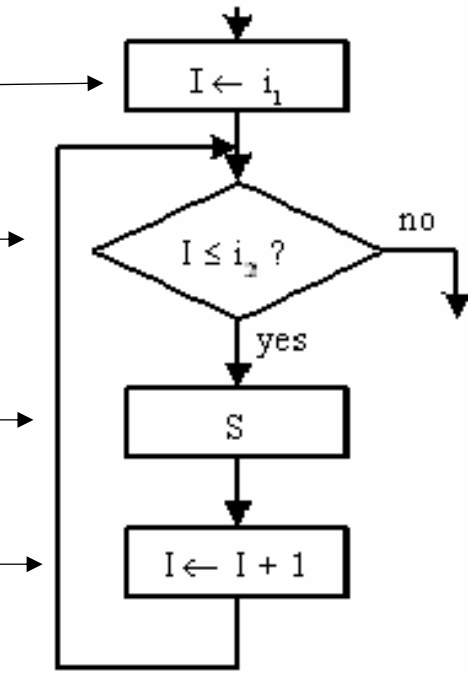
- i = loop counter
- S = Statement

1) Initialize the loop counter

2) Compare the loop counter with the limit

3) Perform the operations in S if loop counter within the limit

4) Increment the loop counter ‘i’ and go to step 2




(a) For $I = i_1$ to $i_2$ DO S

# For i=n2 to n1 Do S

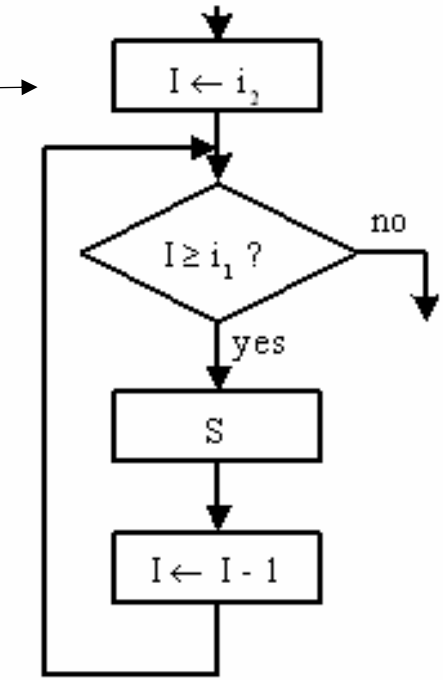
- i = loop counter
- S = Statement

1) Initialize the loop counter

2) Compare the loop counter with the limit

3) Perform the operations in S if loop counter within the limit

4) Decrement the loop counter ‘i’ and go to step 2




(b) For I = $i_2$ downto $i_1$ DO S

# While C Do S

- Logical expression C is evaluated
  - Only while C if true, S will be executed

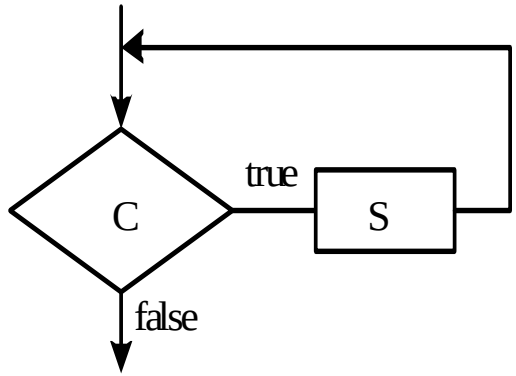



Figure 2.6 The While ... Do looping construct

1) Initialize the logical expression C

2) Evaluate the logical expression C

3) If C is true perform the functions specified by S,
go back to 2, if not exit

4) Exit

# Repeat S until C

1) Initialize loop counter

2) Execute statement S

3) Evaluate C, if true go to step 2, else exit

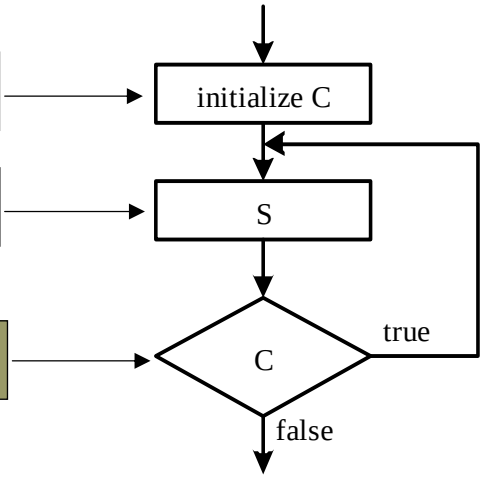



Figure 2.7 The Repeat ... Until looping construct

# Condition Code Register

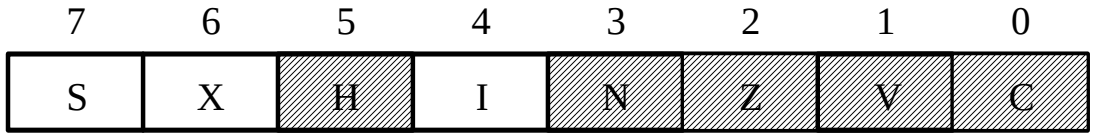



Figure 2.8 Condition code register

**C: carry**
**V: overflow**
**Z: zero**
**N: Negative**
**H: half-carry**

# Branch Instructions

- Four types of branch instructions:
  - **Unary (unconditional) branch:** always execute
  - **Simple branches:** branch is taken when a specific bit of CCR is in a specific status
  - **Unsigned branches:** branches are taken when a comparison or test of unsigned numbers results in a specific combination of CCR bits
  - **Signed branches:** branches are taken when a comparison or test of signed quantities results in a specific combination of CCR bits

## **Three categories** of Branches

- **Short Branches:** in the range of -128 ~ +127 bytes
- **Long Branches:** in the range of 64KB
- **bit-conditional branches**

Table 2.2 Summary of short branch instructions

| Unary Branches | | |
|---|---|---|
| Mnemonic | Function | Equation or Operation |
| BRA | Branch always | 1 = 1 |
| BRN | Branch never | 1 = 0 |
| **Simple Branches** | | |
| Mnemonic | Function | Equation or Operation |
| BCC | Branch if carry clear | C = 0 |
| BCS | Branch if carry set | C = 1 |
| BEQ | Branch if equal | Z = 1 |
| BMI | Branch if minus | N = 1 |
| BNE | Branch if not equal | Z = 0 |
| BPL | Branch if plus | N = 0 |
| BVC | Branch if overflow clear | V = 0 |
| BVS | Branch if overflow set | V = 1 |
| **Unsigned Branches** | | |
| Mnemonic | Function | Equation or Operation |
| BHI | Branch if higher | C + Z = 0 |
| BHS | Branch if higher or same | C = 0 |
| BLO | Branch if lower | C = 1 |
| BLS | Branch if lower or same | C + Z = 1 |
| **Signed Branches** | | |
| Mnemonic | Function | Equation or Operation |
| BGE | Branch if greater than or equal | N ⊕ V = 0 |
| BGT | Branch if greater than | Z + (N ⊕ V) = 0 |
| BLE | Branch if less than or equal | Z + (N ⊕ V) = 1 |
| BLT | Branch if less than | N ⊕ V = 1 |

Table 2.3 Summary of long branch instructions

| Unary Branches | | |
|---|---|---|
| Mnemonic | Function | Equation or Operation |
| LBRA | Long branch always | 1 = 1 |
| LBRN | Long branch never | 1 = 0 |
| **Simple Branches** | | |
| Mnemonic | Function | Equation or Operation |
| LBCC | Long branch if carry clear | C = 0 |
| LBCS | Long branch if carry set | C = 1 |
| LBEQ | Long branch if equal | Z = 1 |
| LBMI | Long branch if minus | N = 1 |
| LBNE | Long branch if not equal | Z = 0 |
| LBPL | Long branch if plus | N = 0 |
| LBVC | Long branch if overflow is clear | V = 0 |
| LBVS | Long branch if overflow set | V = 1 |
| **Unsigned Branches** | | |
| Mnemonic | Function | Equation or Operation |
| LBHI | Long branch if higher | C + Z = 0 |
| LBHS | Long branch if higher or same | C = 0 |
| LBLO | Long branch if lower | C = 1 |
| LBLS | Long branch if lower or same | C + Z = 1 |
| **Signed Branches** | | |
| Mnemonic | Function | Equation or Operation |
| LBGE | Long branch if greater than or equal | N ⊕ V = 0 |
| LBGT | Long branch if greater than | Z + (N ⊕ V) = 0 |
| LBLE | Long branch if less than or equal | Z + (N ⊕ V) = 1 |
| LBLT | Long branch if less than | N ⊕ V = 1 |

# Compare and Test Instructions

- Condition flags need to be set up before conditional branch instruction should be executed.
- The 68HCS12 provides a group of instructions for testing the condition flags.

Table 2.4 Summary of compare and test instructions

| Compare instructions | | |
|---|---|---|
| Mnemonic | Function | Operation |
| CBA | Compare A to B | (A) - (B) |
| CMPA | Compare A to memory | (A) - (M) |
| CMPB | Compare B to memory | (B) - (M) |
| CPD | Compare D to memory | (D) - (M:M+1) |
| CPS | Compare SP to memory | (SP) - (M:M+1) |
| CPX | Compare X to memory | (X) - (M:M+1) |
| CPY | Compare Y to memory | (Y) - (M:M+1) |
| Test instructions | | |
| Mnemonic | Function | Operation |
| TST | Test memory for zero or minus | (M) - $00 |
| TSTA | Test A for zero or minus | (A) - $00 |
| TSTB | Test B for zero or minus | (B) - $00 |

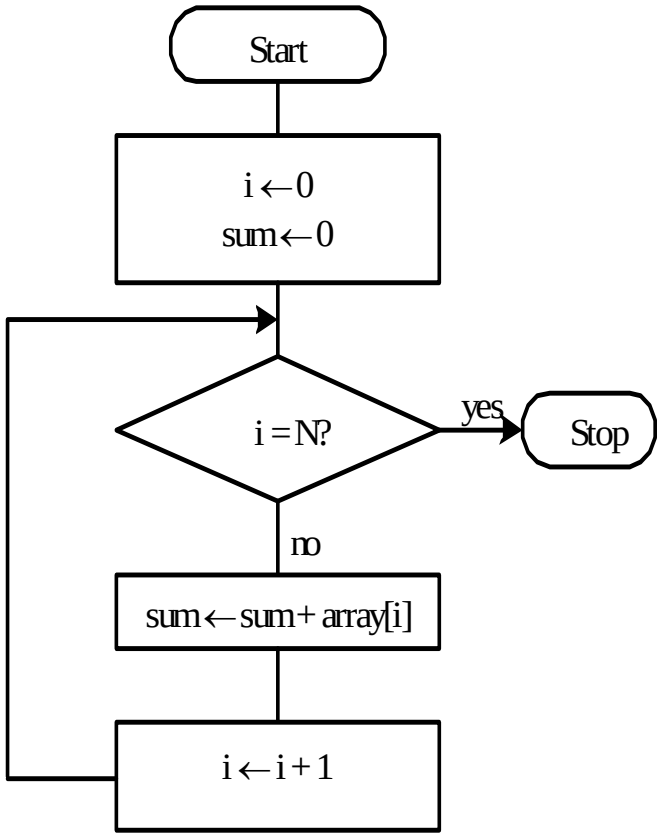
# Decrementing & Incrementing Instructions

- DEC, DECA,DECB,DES,DEX,DEY
- INC,INCA,INCB,INS,INX,INY

- ldaa i
- adda #1
- staa i

**Example 2.14'** Write a program to add an array of N 8-bit numbers and store the sum at memory locations $1800~$1801. Use the **For** i = n1 **to** n2 **do** looping construct.

**Solution**:




i = loop counter

N = no.of elements in the array

Figure 2.9 Logic flow of example 2.14

# Loop Primitive Instructions

- 68HCS12 provides a group of instructions that either decrement or increment a loop count to determine if the looping should be continued.
- The range of the branch is from $80 (-128) to $7F (+127).

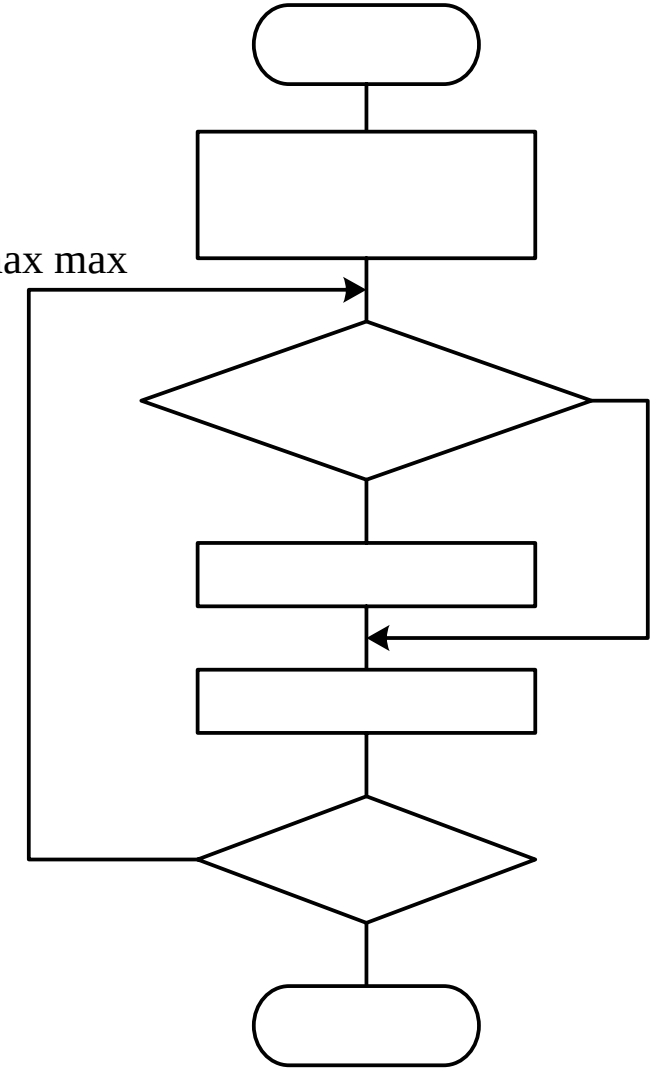
Table 2.5 Summary of loop primitive instructions

| Mnemonic | Function | Equation or Operation |
|---|---|---|
| DBEQ cntr, rel | Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP) | counter ← (counter) - 1<br>If (counter) = 0, then branch<br>else continue to next instruction |
| DBNE cntr, rel | Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP) | counter ← (counter) - 1<br>If (counter) ≠ 0, then branch<br>else continue to next instruction |
| IBEQ cntr, rel | Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP) | counter ← (counter) + 1<br>If (counter) = 0, then branch<br>else continue to next instruction |
| IBNE cntr, rel | Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP) | counter ← (counter) + 1<br>If (counter) ≠ 0, then branch<br>else continue to next instruction |
| TBEQ cntr, rel | Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP) | If (counter) = 0, then branch<br>else continue to next instruction |
| TBNE cntr, rel | Test counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP) | If (counter) ≠ 0, then branch<br>else continue to next instruction |

Note. 1. **cntr** is the loop counter and can be accumulator A, B, or D and register X, Y, or SP.
2. **rel** is the relative branch offset and is usually a label

**Example 2.15'** Write a program to find the maximum element from an array of N 8-bit elements using the **repeat S until C** looping construct.

```
N         equ     20
          org     $1800
max_val   ds.b    1
          org     $1000
          ldaa    array       ; set array[0] as the temporary max max
          staa    max_val     ;          "
          ldx     #array+N-1  ; start from the end of the array
          ldab    #N-1        ; set loop count to N - 1
loop      ldaa    max_val
          cmpa    0,x
          bge     chk_end
          ldaa    0,x
          staa    max_val
chk_end   dex
          dbne    b,loop      ; finish all the comparison yet?
forever   bra     forever
array     db      1,3,5,6,19,41,53,28,13,42,76,14
          db      20,54,64,74,29,33,41,45
          end
```



# Bit Condition Branch Instructions

[<label>]  BRCLR  (opr) (msk) (rel)  [<comment>]
[<label>]  BRSET  (opr) (msk) (rel)  [<comment>]

where

**opr** specifies the memory location to be checked and must be specified using either the direct, extended or index addressing mode.

**msk** is an 8-bit mask that specifies the bits of the memory location to be checked. The bits of the memory byte to be checked correspond to those bit positions that are 1s in the mask.

**rel** is the branch offset and is specified in the relative mode.

For example, in the sequence

```
loop        inc  count
            …
            brset  $66,$e0,loop
            …
```

the branch will be taken if the most significant three bits at $66 are all ones.

# Instructions for Variable Initialization

1. [<label>] CLR opr [<comment>]

   where **opr** is specified using the *extended* or *index* addressing modes. The specified memory location (1 bye) is cleared.

2. [<label>] CLRA [<comment>]

   Accumulator A is cleared to 0

3. [<label>] CLRB [<comment>]

   Accumulator B is cleared to 0

# Next…

- Shift & Rotation
- Read Chapter 2.7