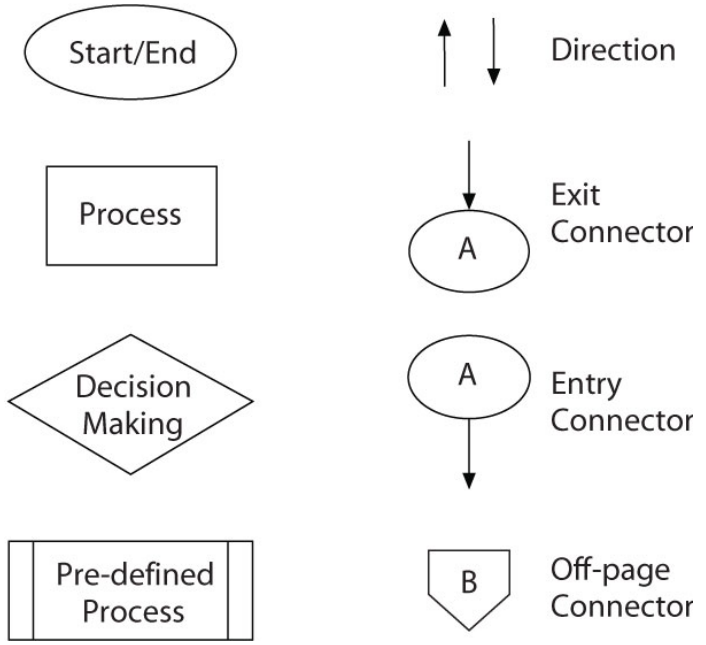
Chapter 4

Programming and Problem Solving

2/14/2019

Flowcharting

- Flowchart
  - A graphical representation of processes (tasks) to be performed and the sequence to be followed in solving computational problem

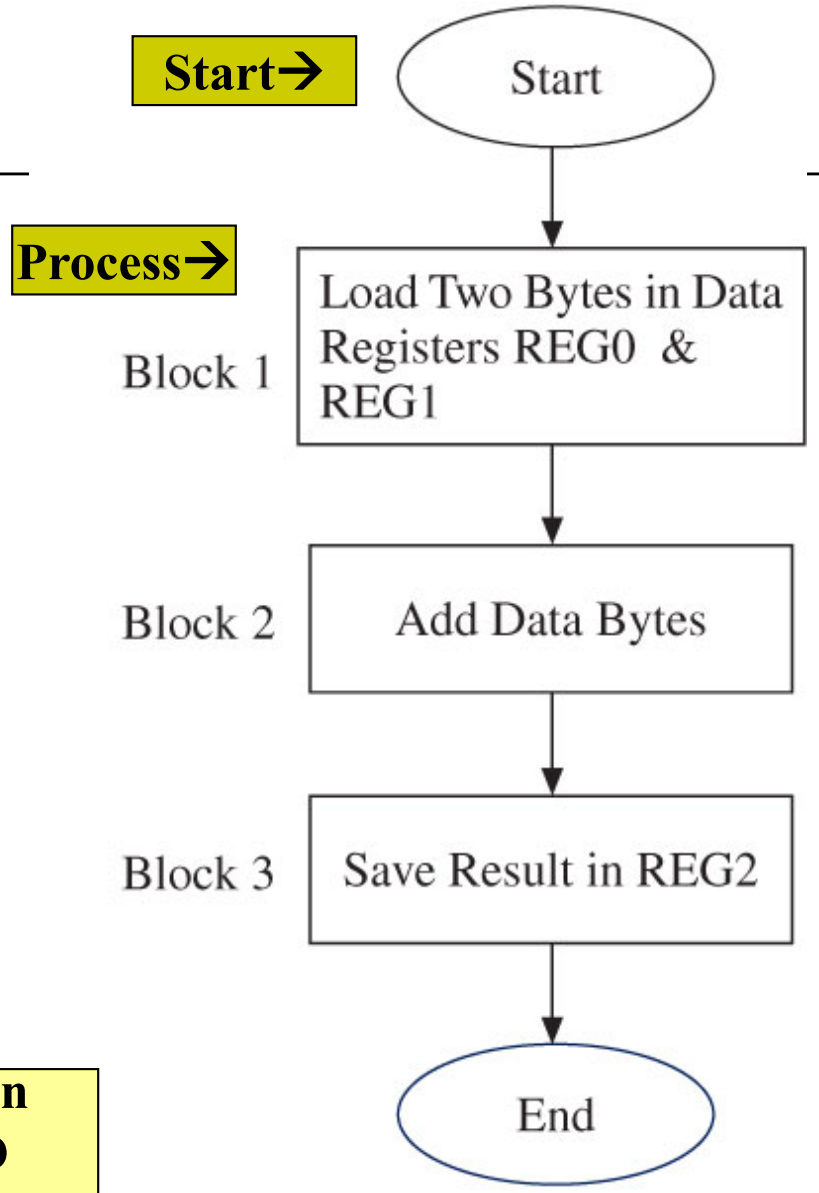


Example 4.1 (1 of 2)

- Write instructions to load two bytes (37H and 92H) in data registers REG0 and REG1. Add the bytes and store the sum in REG2.
- Steps
  - Load the two bytes in data registers REG0 and REG1.
  - Add the bytes.
  - Save the sum in data register REG2.

Example 4.1 (2 of 2)

```
ORG     0x20
REG0    EQU     0x00
REG1    EQU     0x01
REG2    EQU     0x02

MOVLW 0x37
MOVWF REG0,0
MOVLW 0x92
MOVWF REG1,0
ADDWF  REG0,0
MOVWF REG2, 0
SLEEP
```



Change the program: if the sum is larger than 50H then the result should be in REG3 AND REG2=0; OTHERWISE, the result should be in REG2 AND REG3=0;

What does it do? Is it correct?

```
        ORG   0x20
        REG0  EQU       0x00
        REG1  EQU       0x01
        REG2  EQU       0x02
        REG3  EQU       0x03

        COMPREG         EQU       0x10
        CONST           EQU       0x50

        MOVLW           CONST
        MOVWF           COMPREG,0

        MOVLW           0x37
        MOVWF           REG0,0
        MOVLW           0x92
        MOVWF           REG1,0
        ADDWF           REG0,0    ;the result is in W

        CPFSLT  COMPREG,0
        BRA     WR_REG3
        BRA     WR_REG2
WR_REG3:
        MOVWF           REG3, 0
        BRA     DONE_PROG
WR_REG2:
        MOVWF           REG2, 0

DONE_PROG:
        SLEEP
```

Draw the flowchart after you complete the program!

Find: Reg0, Reg1, Reg2, RegA, RegB, Reg10, Reg11, W

Find: Reg0, Reg1, Reg2, RegA, RegB, Reg10, Reg11, W

Steps in Writing and Executing Assembly Language Program

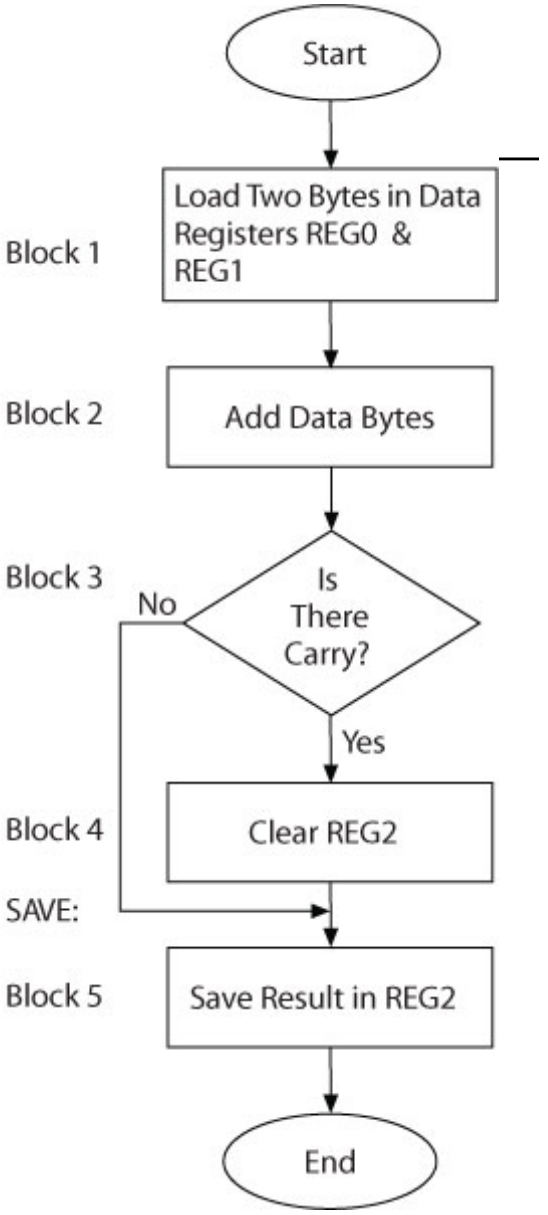
- Analyze the problem.
- Draw a flowchart.
- Convert the flowchart in mnemonics.
- Look up Hex code and assign memory addresses.
- Enter the Hex code into memory of a lab training board.
- Execute the program.
- Debug the program if necessary.

Illustrative Program: Addition With Carry Check

- Write instructions to load two bytes, Byte1 (F2H) and Byte2 (32H), in data registers REG0 and REG1 respectively and add the bytes.
- If the sum generates a carry, clear the data register REG2; otherwise, save the sum in REG2.

Illustrative Program: Addition With Carry Check

- Write instructions to load two bytes, Byte1 (F2H) and Byte2 (32H), in data registers REG0 and REG1 respectively and add the bytes.
- If the sum generates a carry, clear the data register REG2; otherwise, save the sum in REG2.

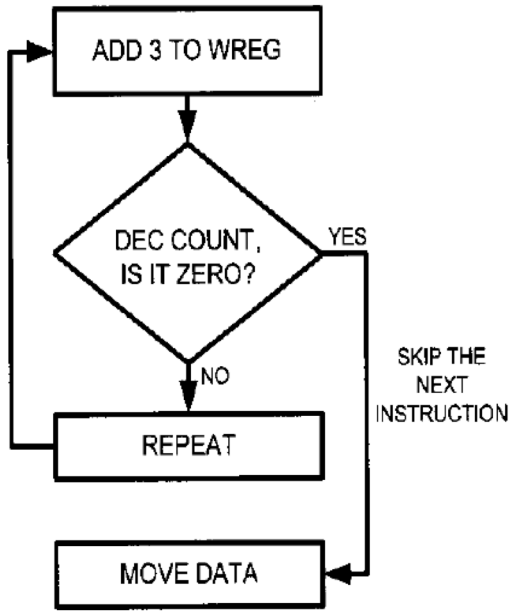


Creating Loops - Example

Write a program to (a) clear WREG, and (b) add 3 to WREG ten times and place the result in SFR of PORTB. Use the DECFSZ instruction to perform looping.

Solution:

```
;this program adds value 3 to WREG ten times

COUNT   EQU 0x25            ;use loc 25H for counter

        MOVLW   d'10'       ;WREG = 10 (decimal) for counter
        MOVWF   COUNT       ;load the counter
        MOVLW   0           ;WREG = 0
AGAIN   ADDLW   3           ;add 03 to WREG (WREG = sum)
        DECFSZ  COUNT,F     ;decrement counter, skip if count = 0
        GOTO    AGAIN       ;repeat until count becomes 0
        MOVWF   PORTB       ;send sum to PORTB SFR
```



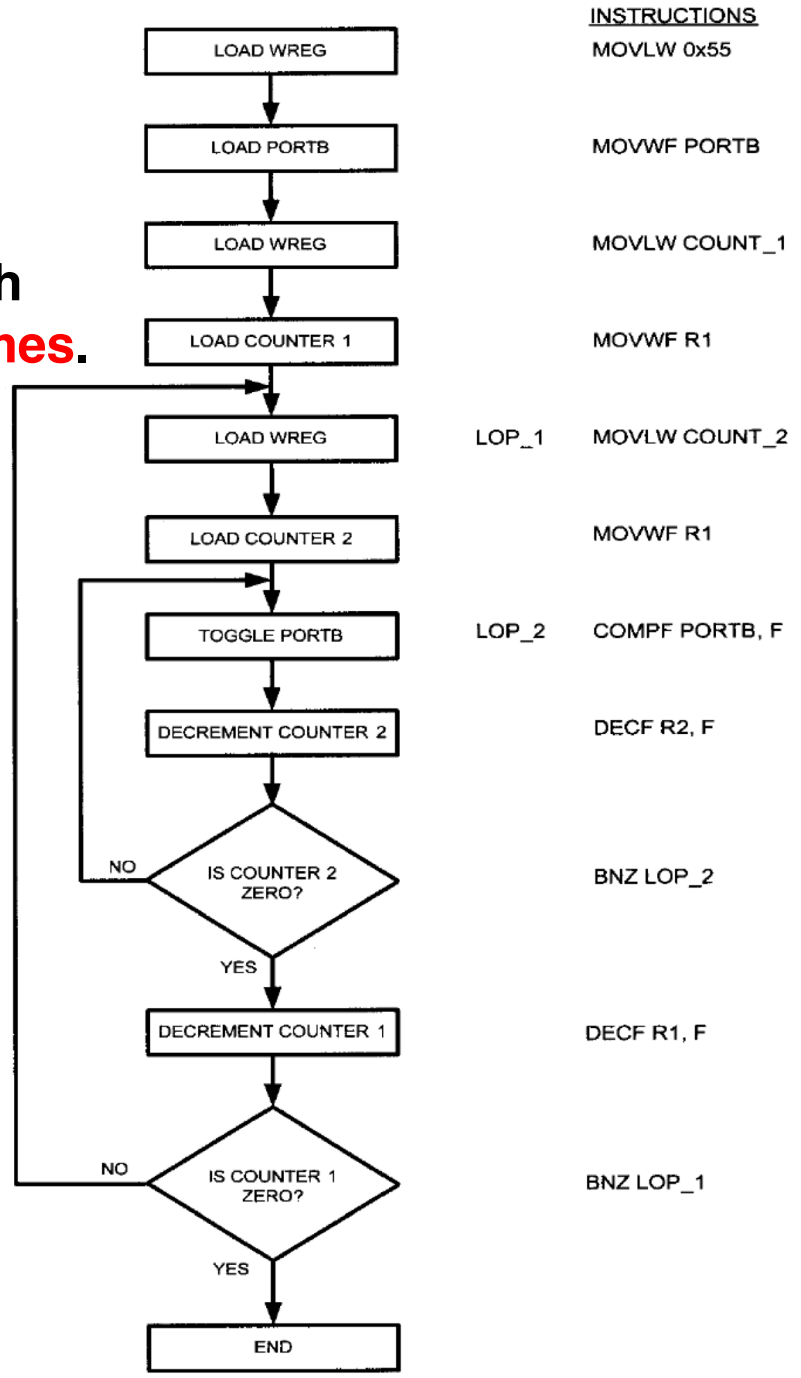
What is the maximum number of loops you can have?

Nested Loop Example

Write a program to (a) load PORTB register with value of 55H and (b) complement port B 700 times.

```
        R1 EQU 0x25
        R2 EQU 0x26
        COUNT_1 EQU d'10'
        COUNT_2 EQU d'70'
        MOVLW 0x55          ;WREG = 55h
        MOVWF PORTB         ;PORTB = 55h
        MOVLW COUNT_1       ;WREG = 10, outer loop count value
        MOVWF R1            ;load 10 into loc 25H (outer loop count)
LOP_1   MOVLW COUNT_2       ;WREG = 70, inner loop count value
        MOVWF R2            ;load 70 into loc 26H
LOP_2   COMPF PORTB, F      ;complement Port B SFR
        DECF R2, F          ;dec fileReg loc 26 (inner loop)
        BNZ LOP_2           ;repeat it 70 times
        DECF R1, F          ;dec fileReg loc 25 (outer loop)
        BNZ LOP_1           ;repeat it 10 times
```

MEMORY LOCATION	VALUE	
25	10	R1
26	70	R2



Assembler Directives

- Assembly Language Format:
  - Label Opcode Operand Comment
  - STATRT: MOVLW 0xF2 ;Load Operation
- Directives, also called pseudocodes, are instructions to the assembler
  - Define constants, labels, where to assemble a program, reserves memory for data
  - Different types:
    - Object File (e.g., CODE, UDATA)
    - Control (e.g., #define, #include, EQU, ORG)
    - List (e.g., nolist, page, space)
    - Data (e.g., data, db, de, dw – all have to do with declaration)
- Directives do not translate to machine language → do not require memory assignment (come for free!)
  - Example BYTE EQU 0x02
    - Label BYTE is being equated to value 2Hex
  - Example ORG 20H
    - Assemble the program starting at location 20H

Assembler Directives - Examples

Directives	Examples	Description
ORG: Originate	ORG 000020H	Assemble the program starting at location 000020_H
END: End	END	End of assembling
EQU: Equate	COUNT EQU 0x20	The label COUNT is equal to 20_H
SET: Define	REG10 SET 0x10	Define register REG10 as a variable
#INCLUDE	#include <P18F452.inc>	Header file for PIC 18F452
DB: Data Byte	DB 0x32, 0xF2, 0x45	Store next bytes in consecutive memory locations
DW: Data Word	DW 0x167F, 0x12A2	Store next two words, each requiring two memory locations—low-order byte first followed by high-order byte

Directive Examples

```
38      ; Main Program
39      ;---------------------
40
41          ORG     0x20                ;Start program listing at 0x20 in the memory
42          DW  0x2301, 0x8899          ;Declare two consecutive words
43          DW  0x0A                    ;Declare a word for HEX value 'A'
44          DB  0x22, 0x33              ;Declare two consecutive bytes
45          DA  "I love embedded system" ;Store string in program memory
46
47          DB  0xF2, 0xF3              ;Declare two consecutive bytes
48
49        ORG       0x80                ;Start program listing at 0x80 in the memory
50      BEGIN2
51          CLRF    RESULT            ;Make sure the RESULT is cleared
52
```

Program

Address	00	02	04	06	08	0A	0C	0E	ASCII
0000	0000	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0010	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0020	2301	8899	000A	3322	2049	6F6C	6576	6520	.#...."3 I love e
0030	626D	6465	6564	2064	7973	7473	6D65	F3F2	mbedded system..
0040	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0050	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0060	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0070	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0080	6A30	0E12	6E20	0E02	6E20	0E12	6E30	0E02	0j.. n.. n..0n..
0090	2630	E102	6A20	6A20	EF4C	F000	FFFF	FFFF	0&.. j j L.....

Format of Radixes

- Hexadecimal
  - 0x0F
  - H`4F`
  - 4F
  - 4FH
- Decimal
  - D`200`
- Binary
  - B`1001`
- ASCII
  - `This stuff are interesting!`

Radix	Example representation
Decimal	D'255'
Hexadecimal	H'8d' or 0x8d
Octal	O'574'
Binary	B'01011100'
ASCII	'G' or A'G'

How to start a GOOD program:

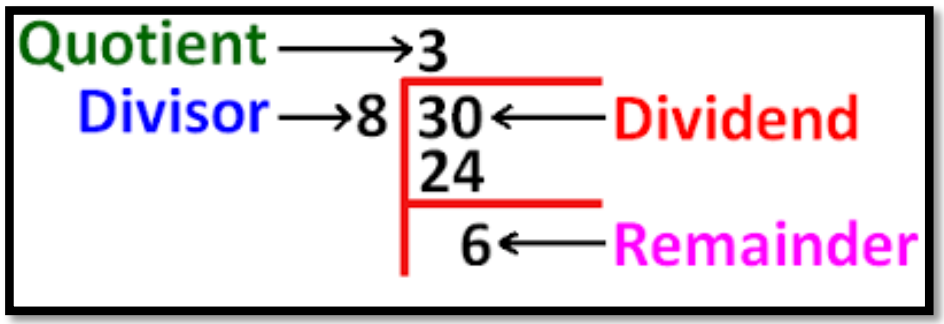
```
;---------------------
; Title: Program_name
;---------------------
;Program Detail:
;---------------------
; Purpose: What does it do?
; Inputs: What are the inputs to the program?
; Outputs: What are the output of the Program?
; Date: date and time
; Compiler: Simulator Version 6.8
; Author: name of the author
; Versions:
;           V1 - What is the version numer and what is the change for each version?
;           V2 - What is the version numer and what is the change for each version?

;---------------------
; File Dependencies: these are the listing and header files you need to run the
;   program
;---------------------
List P=18F452 f=inhx32
#include <p18F452.inc>

;---------------------
; Mail Program
;---------------------
REG0    EQU 0X00                ;Define RG0
BUFFER  EQU 0X10

        ORG 0X20                ;Start program listing from Reg. 0x20 in the memory

START:  CLRF    REG0            ;Make sure the label is clearly separated
        LFSR    FSR0,BUFFER
        MOVFF   BUFFER,W
NEXT:   MOVF    POSTINC0,W
        BZ      FINISH
        CPFSGT  REG0,0
        BRA     NEXT
        MOVWF   REG0
        BRA NEXT
FINISH: NOP
        END                     ; End of the program
```

Division Example Using Subtraction

- 30/8?
  - 30-8 = 22
  - 22-8 = 14
  - 14-8 = 6
  - 6-8 = 0
  - → 8-6=2 **Remainder**

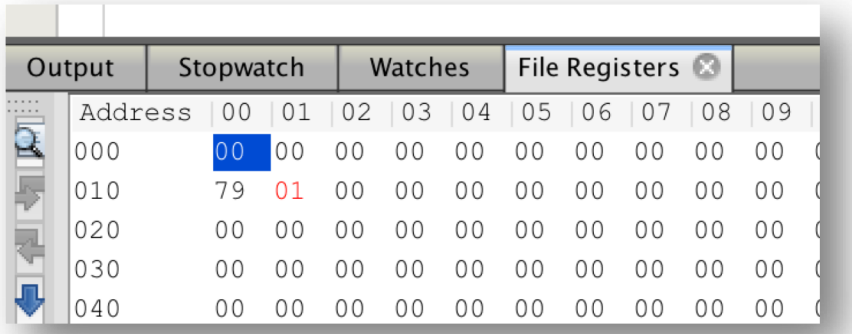
Quotient=1+1+1+1-1 = 3

Note: 2x10/8 → 25
The Quotient: 3.25



Division Example Using Subtraction

```
54    ;//** I N I T I A L I Z A T I O N **********/
55    ; Dividend/Divisor = Quotient & Remainder
56
57    REMAINDER       EQU             0x11
58    QUOTIENT        EQU             0x10
59    DIVIDEND        EQU             D'243'
60    DIVISOR         EQU             D'2'
61
62        ORG 0x20
63
64        CLRF        QUOTIENT
65        MOVLW       DIVIDEND
66        MOVWF       REMAINDER
67        MOVLW       DIVISOR
68    B1
69        INCF        QUOTIENT,F
70        SUBWF       REMAINDER,F     ;F=F-W
71        BC          B1              ;Keep doing it until C = 0
72        DECF        QUOTIENT,F
73        ADDWF       REMAINDER,F     ;Remainder
74        END
```



Division Example Using Subtraction Using Subroutines: CALL

```
57  REMAINDER       EQU             0x11
58  QUOTIENT        EQU             0x10
59  DIVIDEND        EQU             D'243'
60  DIVISOR         EQU             D'2'
61
62      ORG 0x20
63
64      CLRF        QUOTIENT
65      MOVLW       DIVIDEND
66      MOVWF       REMAINDER
67      CALL        DIVIDEIT
68      MOVLW       0x0
69
70      ORG     0xF0
71  DIVIDEIT
72      MOVLW       DIVISOR
73  B1
74      INCF        QUOTIENT,F
75      SUBWF       REMAINDER,F     ;F=F-W
76      BC          B1              ;Keep doing it until C = 0
77      DECF        QUOTIENT,F
78      ADDWF       REMAINDER,F     ;Remainder
79      RETURN
80      END
```

Cannot be an END!
You can use
STOP GOTO STOP

What is the problem with this Code?

```
REMAINDER   EQU             0xF0
QUOTIENT    EQU             0xF1
DIVIDEND    EQU             0x23
DIVISOR     EQU             0x12


    ORG 0x20
    CLRF        QUOTIENT
    MOVLW       DIVIDEND
    MOVWF       REMAINDER
    MOVLW       DIVISOR
    CALL        MYDIVIDE
    MOVLW       0x02
    END

    ORG 0x24
    MOVLW       0x0
MYDIVIDE
B1
    INCF        QUOTIENT,F
    SUBWF       REMAINDER,F
   BC           B1
   ADDWF        QUOTIENT,F
 RETURN
    END
```