MECE336 Microprocessors I Programming

Dr. Kurtuluş Erinç Akdoğan

kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

CONTENT

- In this lecture we will learn about:
  - how to visualise a program, and represent it diagrammatically;
  - how to use program branching

A Very First Program

- The program starts with a header made up of five comment lines, each starting with a semicolon.
- org directive is used to define the start address as Reset Vector address (*when there are program blocks at different locations, start address is controlled by the programmer*)
- The program which follows uses only three instructions. It first clears the W register.
- The following instruction has been given the label loop. It adds the number 8, embedded into the instruction as a 'literal' value, to the Wregister.
- The following **goto** instruction, using the label loop as its operand, causes the program to return to the add instruction, which it does repeatedly.
- The W register therefore repeatedly increments by the value 8.
- The end of the program is defined with an end directive.

```
;*******************************************************************
;Very first program
;This program repeatedly adds a number to the Working Register.
;TJW 1.11.08                                            Tested 1.11.08
;*******************************************************************
;
; use the org directive to force program start at reset vector
       org    00
;program starts here
       clrw                  ;clear W register
loop   addlw 08              ;add the number 8 to W register
       goto  loop
       end                   ;show end of program with "end" directive
```

A larger program – using data memory and moving data

-The rule:

$x_n = x_{n-1} + x_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

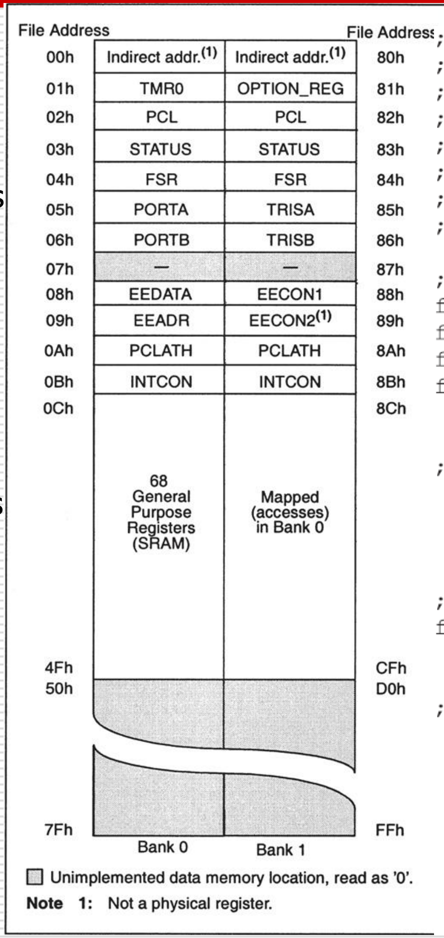
-Four memory locations are needed, three to hold the most recent numbers in the series and one to hold temporary data.

- The memory map shows that memory locations in the address range 0CH to 4FH are available.

-In this program the locations from 20H to 23H have been arbitrarily chosen.

-Labels corresponding to memory location addresses have been defined using the equ directive, for example:

- Wherever the word **fib0** is used after this line, itwill be replaced by the number **20H**.




```
;*********************************************************************
;Fibo_simple
;In a Fibonacci series each number is the sum of the two previous ones,
;e.g. 0,1,1,2,3,5,8,13,21....
;This program calculates Fibonacci numbers within an 8-bit range.
;Program intended for simulation only, hence no input/output.
;TJW 6.11.08                                Tested by simulation 6.11.08
;*********************************************************************

;these memory locations hold the Fibonacci series
fib0    equ  20   ;lowest number
fib1     equ  21 ;middle number
fib2     equ  22 ;highest number
fibtemp equ  23 ;temporary location for newest number

          org 00
;preload initial values
              clrf fib0           ;clear location fib0
              movlw 1             ;move value 1 to W register
              movwf fib1          ;move W register to fib1
              movwf fib2          ;move W register to fib2
;
forward       movf  fib1,0        ;move the contents of fib1 to W register
              addwf fib2,0        ;add W reg to fib2
              movwf fibtemp       ;move new number formed to fibtemp
;now shuffle numbers held, discarding the oldest (ie fib0)
              movf  fib1,0        ;move fib1 to W register
              movwf fib0          ;move W register to fib0
              movf  fib2,0        ;move fib2 to W register
              movwf fib1          ;move W register to fib1
              movf  fibtemp,0     ;move fibtemp to W register
              movwf fib2          ;move W register to fib2
              goto  forward
              end
```

```
fib0      equ   20      ;lowest number
```

Basic Instructions: Move

- **movwf f :** This moves the contents of the W register to the memory location f.
- **movf f,d :** This instruction moves the contents of the memory location f to theWregister, if the d bit is set to 0; if it is set to 1 then the contents of f are just returned to f (but the Z bit may still change).
- **movlw k :** This instruction moves the literal value k, an 8-bit number which accompanies the instruction, into the W register.

A larger program – using data memory and moving data

-The program starts by preloading the three first numbers in the series, 0,1,1, into the reserved memory locations.
-Location fib0 is simply cleared using a **clrf** instruction.
-The value 1 is loaded into fib1 and fib2.
-The number must first be moved into the Wregister with a **movlw** instruction, before being transferred to the memory location with a **movwf** instruction.
- Starting at the label **forward**, the program starts calculating the next value in the series by adding the two most recent numbers.
-The instruction set does not allow the direct addition of two memory locations. One location therefore, **fib1**, is moved first to the W register. This is done using a **movf** instruction, with the **d** bit set to 0.
-The W register is then added to **fib2**. Because the **d** bit is set to 0 again, the result is saved in the W register.
-The next instruction moves it to **fibtemp**.
-The program then shuffles the numbers held in the memory locations, retaining the three most recent values and discarding the oldest.
-Using a **goto** instruction, the program then loops back to **forward**, and starts to calculate a new member of the series.

```
;***************************************************************************
;Fibo_simple
;In a Fibonacci series each number is the sum of the two previous ones,
;e.g. 0,1,1,2,3,5,8,13,21....
;This program calculates Fibonacci numbers within an 8-bit range.
;Program intended for simulation only, hence no input/output.
;TJW 6.11.08                                       Tested by simulation 6.11.08
;***************************************************************************

;these memory locations hold the Fibonacci series
fib0    equu  20   ;lowest number
fib1     equ  21 ;middle number
fib2     equ  22 ;highest number
fibtemp equ   23 ;temporary location for newest number

           org 00
;preload initial values
              clrf fib0           ;clear location fib0
              movlw 1             ;move value 1 to W register
              movwf fib1          ;move W register to fib1
              movwf fib2          ;move W register to fib2
;
forward       movf  fib1,0        ;move the contents of fib1 to W register
              addwf fib2,0        ;add W reg to fib2
              movwf fibtemp       ;move new number formed to fibtemp
;now shuffle numbers held, discarding the oldest (ie fib0)
              movf  fib1,0        ;move fib1 to W register
              movwf fib0          ;move W register to fib0
              movf  fib2,0        ;move fib2 to W register
              movwf fib1          ;move W register to fib1
              movf  fibtemp,0     ;move fibtemp to W register
              movwf fib2          ;move W register to fib2
              goto  forward
              end
```

Basic Instructions: Bit-wise Operations

- **bcf f,b:** Set bit b (between 0 and 7) in memory location f to logic 0. This is called clear.
- **bsf f,b:** Set bit b in memory location f to logic 1.

Status Register

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)

01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 **$\overline{\text{TO}}$**: Time-out bit

1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{\text{PD}}$**: Power-down bit

1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit

1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit Carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)

1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)

1 = A carry-out from the Most Significant Bit of the result occurred
0 = No carry-out from the Most Significant Bit of the result occurred

Note: A subtraction is executed by adding the twos complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

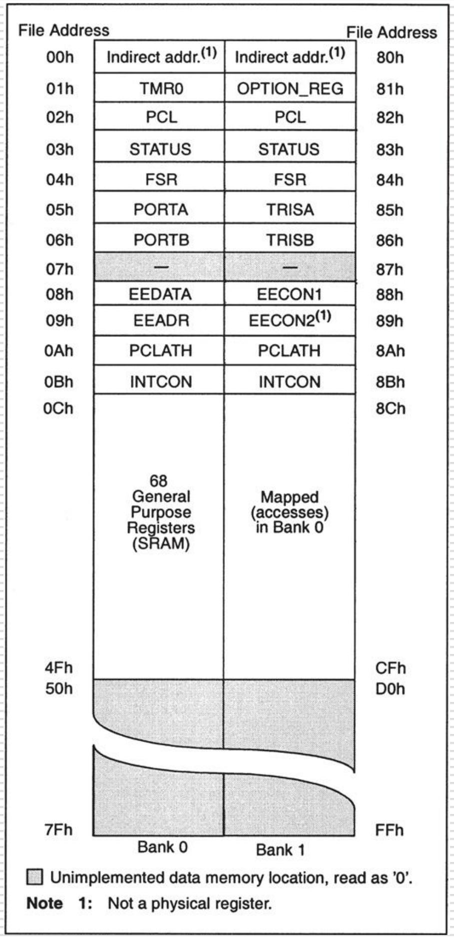
Programming for a target piece of hardware – a simple datatransfer program

-This program just uses the **Status register**, **Ports A and B**, and their control registers **TRISA** and **TRISB**. Labels for these are therefore defined, taking memory addresses directly from the memory map

-The program starts with an initialisation section

-As SFRs are placed in RAM memory bank 1, it is necessary first of all to set bit 5 of the Status register to 1.

-This is done in the first program line, labelled start, using the **bsf** instruction.

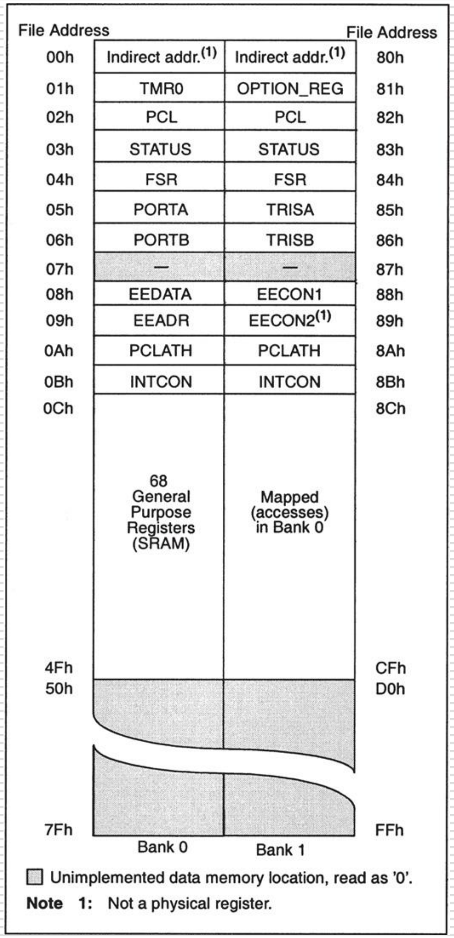



```
;***********************************************************************
;Ping-pong data move
;This program moves push button switch values from Port A to the
;leds on Port B
;TJW 21.2.05                                          Tested 22.2.05
;***********************************************************************
;
;Configuration Word: WDT off, power-up timer on,
;                       code protect off, RC oscillator
;
;specify SFRs
status   equ      03
porta    equ      05
trisa    equ      05
portb    equ      06
trisb    equ      06
;
       org    00
;Initialise
start  bsf    status,5      ;select memory bank 1
       movlw  B'00011000'
       movwf  trisa         ;port A according to above pattern
       movlw  00
       movwf  trisb         ;all port B bits output
       bcf    status,5      ;select bank 0
;
;The "main" program starts here
       clrf  porta          ;clear all bits in ports A
loop   movf   porta,0       ;move port A to W register
       movwf  portb         ;move W register to port B
       goto   loop
       end
```

```
start      bsf   3,5;  select memory bank 1
```

Programming for a target piece of hardware – a simple datatransfer program

-To be an output a **port pin** must have a 0 in its corresponding **TRIS** register bit. It must have a 1 for the bit to be an input. Therefore we must send the word 00011000 to TRISA.
-A similar process is followed for setting up **Port B**.




```
;*************************************************************************
;Ping-pong data move
;This program moves push button switch values from Port A to the
;leds on Port B
;TJW 21.2.05                                              Tested 22.2.05
;*************************************************************************
;
;Configuration Word: WDT off, power-up timer on,
;                              code protect off, RC oscillator
;
;specify SFRs
status   equ      03
porta    equ      05
trisa    equ      05
portb    equ      06
trisb    equ      06
;
        org    00
;Initialise
start   bsf    status,5      ;select memory bank 1
        movlw  B'00011000'
        movwf  trisa         ;port A according to above pattern
        movlw  00
        movwf  trisb         ;all port B bits output
        bcf    status,5      ;select bank 0
;
;The "main" program starts here
        clrf   porta         ;clear all bits in ports A
loop    movf   porta,0       ;move port A to W register
        movwf  portb         ;move W register to port B
        goto   loop
        end
```

```
start      bsf    3,5;  select memory bank 1
```

The main idea – building structure into programs

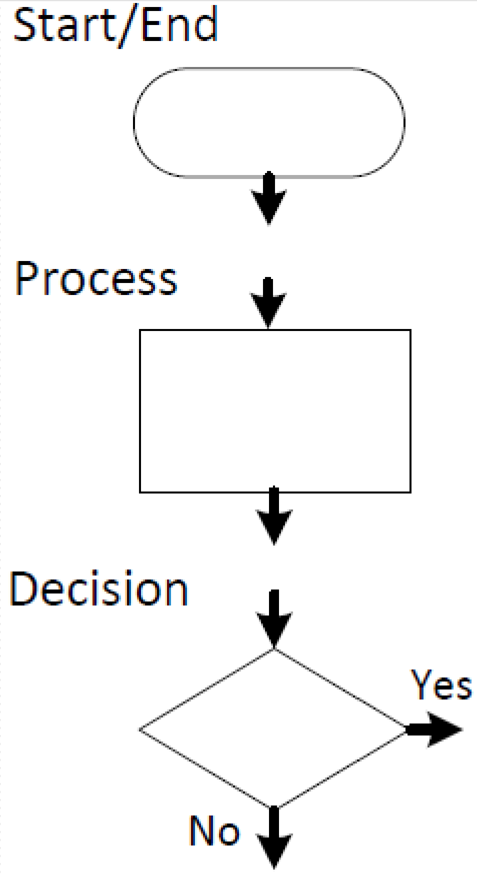
- When we actually design a program, it is important to think about and plan its structure, before starting to write the code.
- Otherwise it leads to unstructured ‘spaghetti’ programs ie code which has no structure, with branches going anywhere, and which is incomprehensible to any but the programmer, and incomprehensible even to him/her after a week.
- Therefore it is essential to plan a programme structure.
- We must consider means of representing the program diagrammatically.

Flow Charts: Components

- Shown as oval or rounded rectangle
- Represents the start or end of a process
- Example content: Start, End

- Shown as rectangle
- Used to show that some operation is
- performed
- Example: "Add 1 to X", "Save X"

- Shown as diamond
- Represents a true/false (Yes/No) decision
- Example: "Is X > 0?"

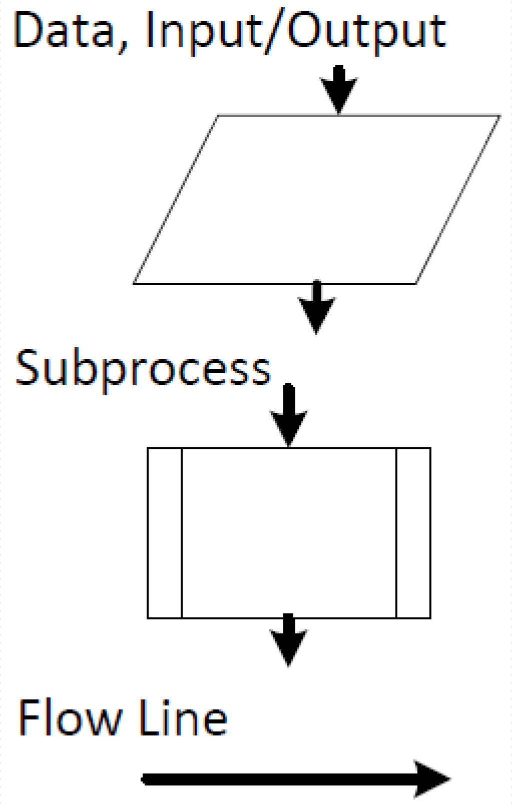


Flow Charts: Components

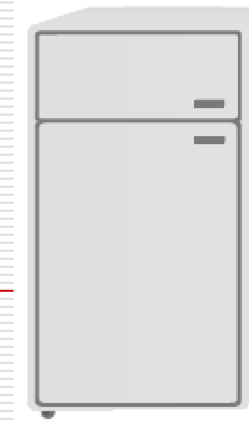
- Shown as a parallelogram
- Represents receiving data, displaying data
- Examples: Get X from the user, display X

- Shown as rectangle with double lines
- Represents a complex processing step
- with a separate owchart
- Example: Subroutine

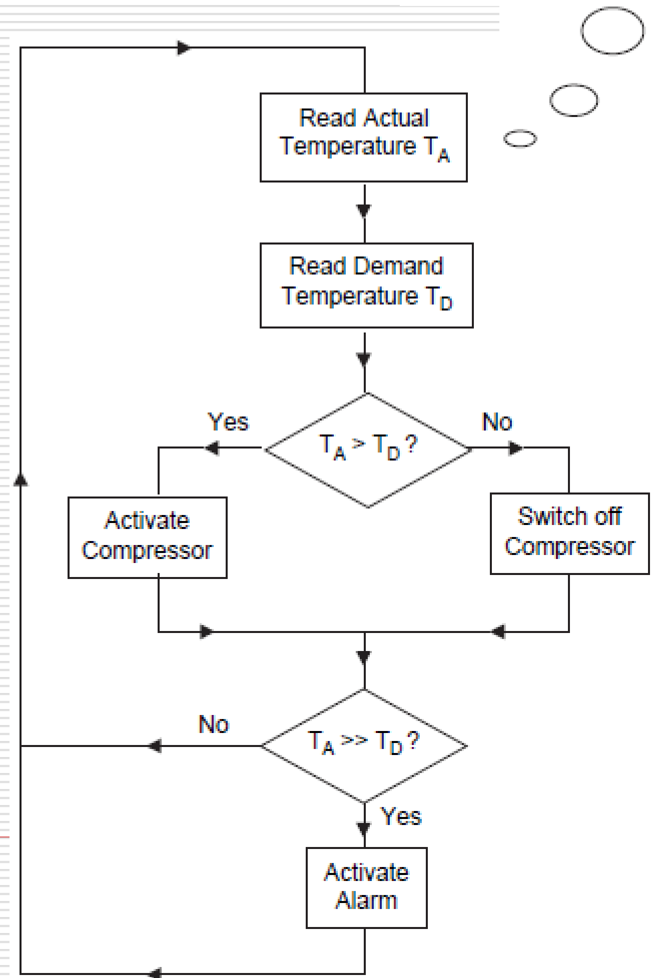
- Arrow from one symbol to another symbol
- Represents that control passes to the
- symbol the arrow points to



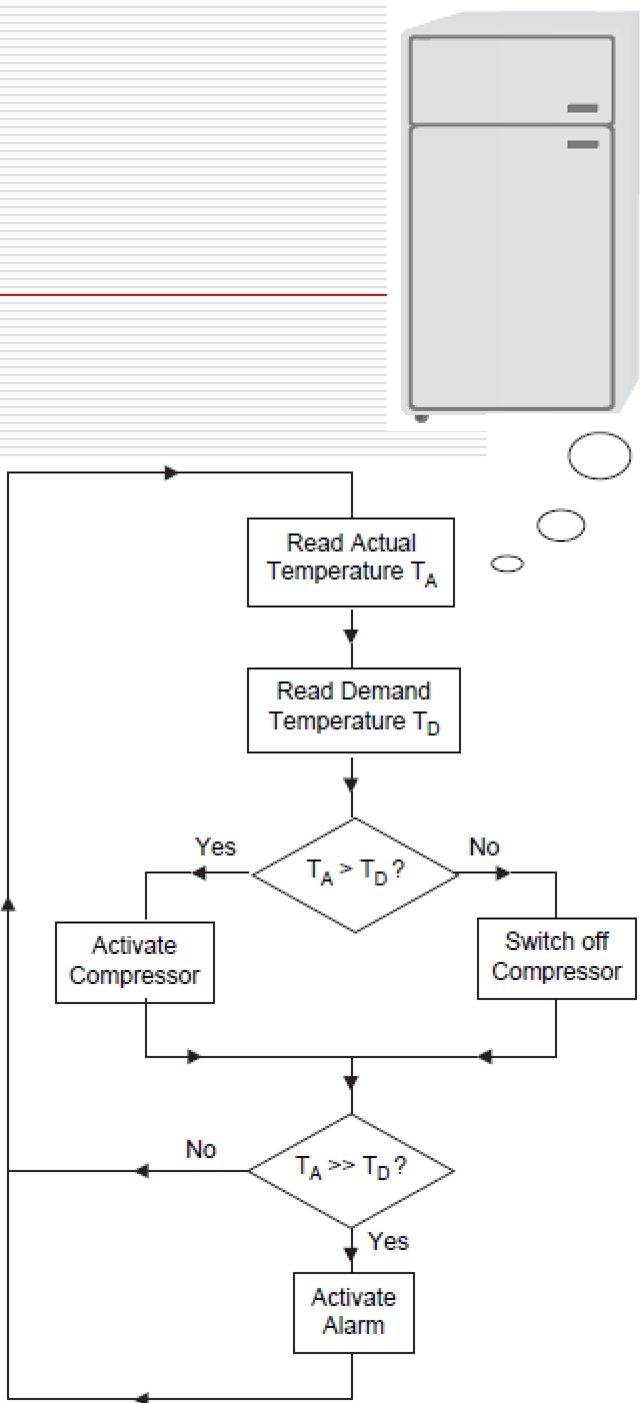
A Refrigerator Controller



- The user has a single control, an adjustable potentiometer that allows him/her to set a desired temperature.
- Within the fridge there is a temperature sensor.
- Temperature is controlled by switching the compressor on or off – the temperature will fall when it is running.
- The program reads both the actual and demand temperatures and determines which is higher.
- If it is the actual temperature, then the compressor is switched on.
- If the difference between the two is very great, then an alarm will sound.

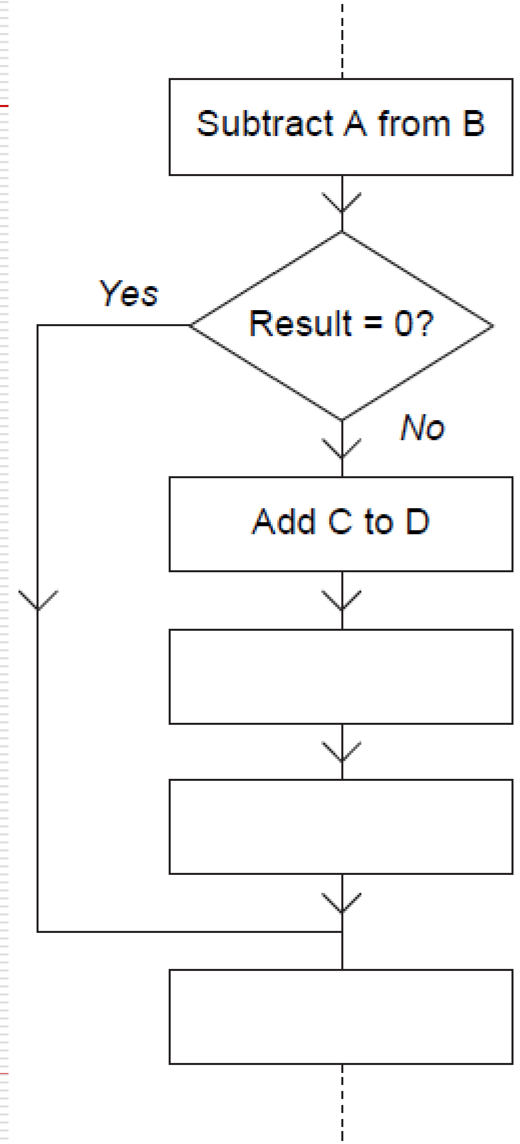


- The flow diagram shows this action, using just the two symbols.
- Notice how each diamond decision symbol contains a question within it with a yes/no answer.
- Its two exit points then correspond to the two possible answers.
- It can be seen that this example program will loop indefinitely.
- This is a common embedded system program structure and is sometimes called a ‘super loop’.



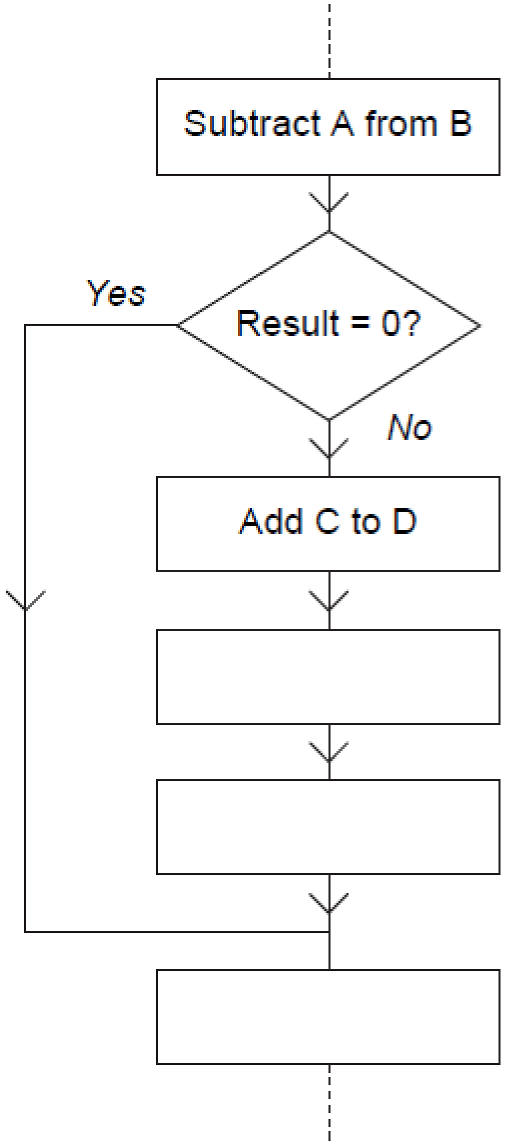
Conditional branching and working with bits

- One of the most important features of any microprocessor or microcontroller program is its ability to make 'decisions', i.e. to act differently according to the state of logical variables.
- **Microprocessors** generally have within their instruction sets a number of instructions which allow them to test a particular bit, and
  - either **continue** program execution if a condition is not met
  - or **branch** to another part of the program if it is.
- These variables are often bit values in condition code or Status registers.



Conditional branching and working with bits

- The PIC 16 Series microcontrollers are a little unusual when it comes to conditional branching as they do not have branch instructions as such.
- They have instead four conditional ‘skip’ instructions.
- These test for a certain condition, skipping just one instruction if the condition is met and continuing normal program execution if it is not.
- The most versatile and generalpurpose of these are the instructions:
- **btfsc f,b**: tests bit b in memory location f and skips just one instruction if the bit is clear (i.e. at Logic 0).
- **btfss f,b**: does a similar thing but skips if the tested bit is set (i.e. at Logic 1).



Example: Condition

- Port A input goes low when the button is pressed.
- The program needs to ‘set’ the output bit (to light the LED) if the input is low, and ‘clear’ it if it is high.
- This implies a selection process – in a high-level language we might call this an ‘if…else’ structure.
- The simple skip instruction is not able to do this on its own.
- One way to do this is to ‘preset’ the output bit with one value and then change it if we find it has been set wrong.

```
;The “main” program starts here
       movlw  00          ;clear all bits in port A and B
       movwf  porta
       movwf  portb
loop   bcf    portb, 3    ;preclear port B, bit 3
       btfss  porta, 3
       bsf    portb, 3    ;but set it if button pressed
;
       bcf    portb, 4    ;preclear port B, bit 4
       btfss  porta, 4
       bsf    portb, 4    ;but set it if button pressed
       goto   loop
       end
```

4 more Arithmetic Instructions

- ADDWF fileReg, d
  - Add the contents of WREG and a file register
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register
- SUBWF fileReg, d
  - Subtracts W content from f register.
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register
- INCF fileReg, d
  - Increment the content of f register.
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register
- DECF fileReg, d
  - Decrement the content of f register.
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register

If a subtract occurs and the result is positive, then the Carry bit is 'set'. If the result is negative, then the Carry bit is 'clear'.

Fibonnacci Program Extended Version

-A counter has been included to show how many numbers in the series have been calculated.
-The program tests for range overflow by checking the Carry bit after each addition.
-When the 8-bit range is exceeded, it reverses the series by subtracting.
-You will notice that **c** and **z** are defined as labels in the opening equates section.
-The program starts as before by preloading the first three numbers in the series into the memory store.
-It starts moving up the series from the label **forward**.
-The two most recent numbers are added and the **Carry** bit then checked.
-If it is set, the 8-bit range has been exceeded and the program will need to reverse.
-Assuming **Carry** was not set, the program then increments the **counter** and shuffles the numbers in the memory store, discarding the oldest.
-The program then loops up to **forward**.
-If, however, the **Carry** had been set, the program branches to reverse. Now it works down the series by **subtraction**.
-It tests the **counter** number to determine when it should return to **forward**.

```
status equ 03
c       equ 0
z       equ 2
;these memory locations hold the three highest values of the Fibonacci series
fib0    equ  10     ;lowest number (oldest when going up,
                    ;newest when reversing down)
fib1    equ  11     ;middle number
fib2    equ  12     ;highest number
fibtemp equ  13     ;temporary location for newest number
counter equ  14     ;indicates value reached, opening value is 3

   org 00
;preload initial values
       movlw 0
       movwf fib0
       movlw 1
       movwf fib1
       movwf fib2
       movlw 3
       movwf counter;we have preloaded the first three numbers,
                           ;so start count at 3
;
forward movf  fib1,0
       addwf fib2,0
       btfsc status,c      ;test if we have overflowed 8-bit range
       goto  reverse       ;here if we have overflowed, hence reverse down
       movwf fibtemp       ;latest number now placed in fibtemp
       incf  counter,1
;now shuffle numbers held, discarding the oldest
       movf  fib1,0        ;first move middle number, to overwrite oldest
       movwf fib0
       movf  fib2,0
       movwf fib1
       movf  fibtemp,0
       movwf fib2
       goto  forward
;when reversing down, subtract fib0 from fib1 to form new fib0
reverse movf  fib0,0
       subwf fib1,0
       movwf fibtemp       ;latest number now placed in fibtemp
       decf  counter,1
;now shuffle numbers held, discarding the oldest
       movf fib1,0         ;first move middle number, to overwrite oldest
       movwf fib2
       movf fib0,0
       movwf fib1
       movf fibtemp,0
       movwf fib0
;test if counter has reached 3, in which case return to forward
       movf  counter,0
       sublw 3
       btfsc status,z
       goto  forward
       goto  reverse
;
       end
```

More Instructions: Rotate

- RLF fileReg, d
  - Rotate left the content of memory location f using carry.
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register
- RRF fileReg, d
  - Rotate right the content of memory location f using carry.
  - Destination, d
    - If d=0, result is placed in WREG
    - If d=1, result is placed in file register
- These commands will move a bit in a register one place to the left (RLF) or the right (RRF) in a register. For example, if we had 00000001 and we used RLF, then we would have 00000010. Now, what happens if we have 10000000 and carried out the RLF instruction? Well, the 1 will be placed in the carry flag. If we carried out the RLF instruction again, the 1 will reappear back at the beginning. The same happens, but in reverse, for the RRF instruction. The example below demonstrates this for the RLF instruction, where I have shown the 8 bits of a register, and the carry flag :

	C 76543210
	0 00000001
RLF	0 00000010
RLF	0 00000100
RLF	0 00001000
RLF	0 00010000
RLF	0 00100000
RLF	0 01000000
RLF	0 10000000
RLF	1 00000000
RLF	0 00000001