

Projeto de Sistemas Microcontrolados (DCA0444)

## **Assembly PIC – *Midrange Family***

Aula 3

Diogo Pinheiro Fernandes Pedrosa  
[diogo@dca.ufrn.br](mailto:diogo@dca.ufrn.br)

UFRN – CT – DCA

Natal, RN.

# Instruções

- `clrf f`
  - Limpa o conteúdo no *f* da memória (o endereço deve ser válido).
- `addwf f, d`
  - Adiciona o conteúdo do registrador W ao conteúdo presente no endereço *f*. O local onde o resultado será armazenado vai depender do bit *d*. Os bits Z, C e DC do registrador de status podem ser afetados.

# Instruções

- **addlw *k***
  - Adiciona o valor de uma constante **k**, de 8 bits, no registrador de trabalho.
- **bcf *f*, *b***
  - Torna o bit **b** da palavra de 8 bits armazenada no endereço **f** igual a 0 lógico.
- **goto *k***
  - Desvia a execução das instruções para a instrução assinalada com a etiqueta **k**.

# Instruções

- **movwf *f***
  - Move o conteúdo presente no registrador W para o endereço de memória *f*.
- **movf *f*, *d***
  - Move o conteúdo presente no endereço da memória *f* para o registrador W (se *d* = 0) ou para o próprio endereço *f* (se *d* = 1).
- **movlw *k***
  - Move o valor *k* (8 bits) para o registrador W.

# Instruções

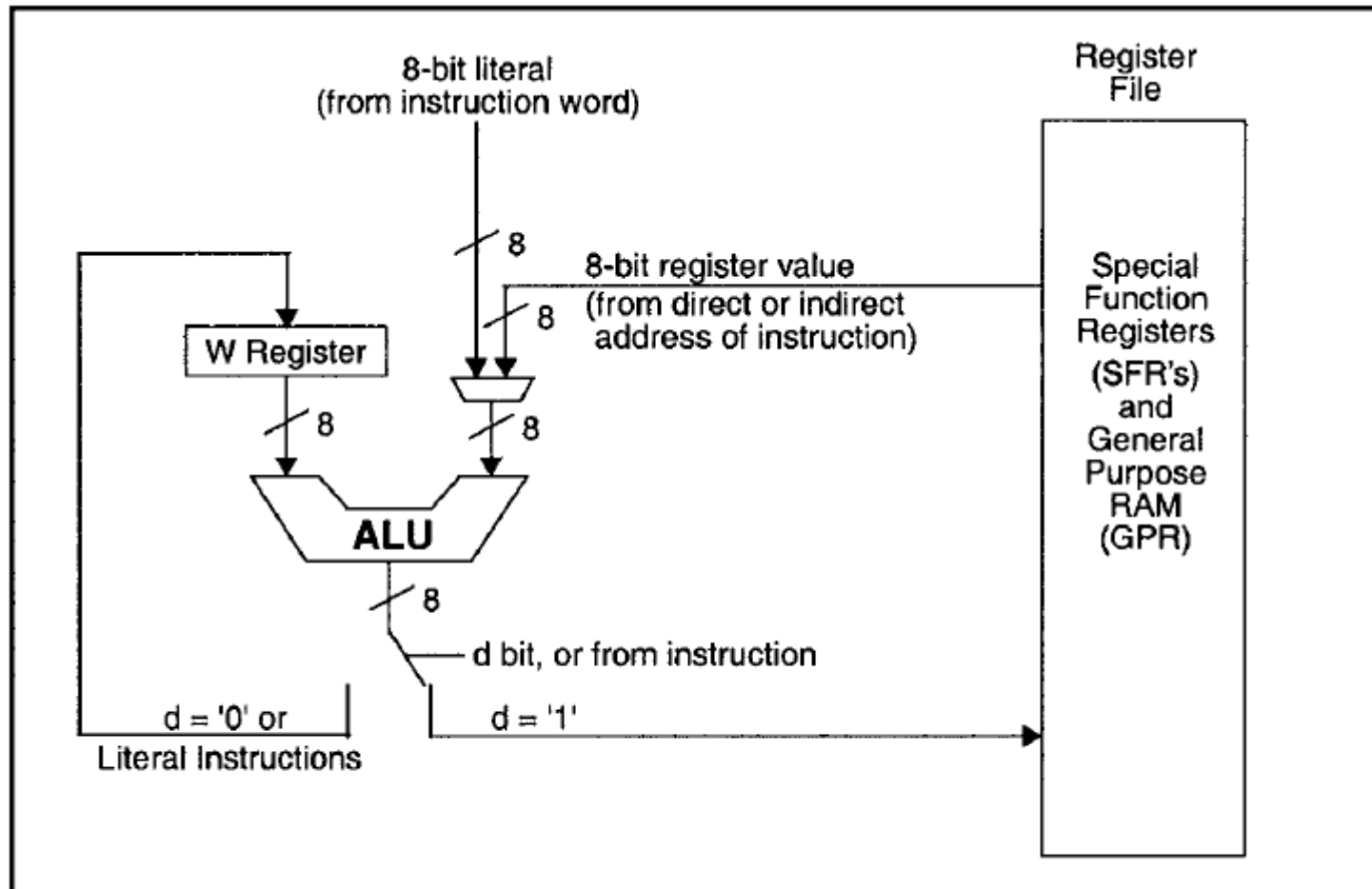


Figure 4.4: Block diagram of the PIC 16 Series ALU

# Instruções

TABLE 4.1 Some common MPASM Assembler directives

Assembler directive	Summary of action
<b>list</b>	Implement a listing option <sup>*</sup>
<b>#include</b>	Include another file within the source file; the included file is embedded within the source file
<b>org</b>	Set program origin; this defines the start address where code which follows is placed in memory
<b>equ</b>	Define an assembly constant; this allows us to assign a value to a label
<b>end</b>	End program block

<sup>\*</sup> Listing options include setting of radix and microcontroller type.

TABLE 4.2 Number representation in MPASM Assembler

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

# Instruções

```
fib0 equ 20  
fib1 equ 21  
fib2 equ 22  
fibtemp equ 23
```

```
org 00
```

```
clrf fib0  
movlw 1  
movwf fib1  
movwf fib2
```

```
forward movf fib1, 0  
         addwf fib2, 0  
         movwf fibtemp
```

```
movf fib1, 0  
movwf fib0  
movf fib2, 0  
movwf fib1  
movf fibtemp, 0  
movwf fib2  
goto forward  
end
```

Uso de etiquetas 'fib0', 'fib1', etc., para referenciar endereços de memória. Foram escolhidos os endereços de 20 até 23 (em hexadecimal).

Informa ao montador onde o programa deve ser armazenado. Neste caso, o endereço inicial é 00.

# Instruções

```
fib0 equ 20  
fib1 equ 21  
fib2 equ 22  
fibtemp equ 23
```

```
org 00
```

```
clrf fib0  
movlw 1  
movwf fib1  
movwf fib2
```

```
forward movf fib1, 0  
addwf fib2, 0  
movwf fibtemp
```

```
movf fib1, 0  
movwf fib0  
movf fib2, 0  
movwf fib1  
movf fibtemp, 0  
movwf fib2  
goto forward  
end
```

Inicia a série de Fibonacci com os endereços citados:

fib0 ← 0  
fib1 ← 1  
fib2 ← 1 (fib2 = fib1 + fib0)

Move o conteúdo de fib1 para o registrador de trabalho.

Soma o conteúdo de fib2 com o conteúdo do registrador de trabalho.

Move o conteúdo do registrador de trabalho para o endereço 'temporário'.

Esses são os procedimentos para fazer fib1 + fib2



# Instruções


```
fib0 equ 20
fib1 equ 21
fib2 equ 22
fibtemp equ 23

org 00

clrf fib0
movlw 1
movwf fib1
movwf fib2

forward movf fib1, 0
        addwf fib2, 0
        movwf fibtemp

movf fib1, 0
movwf fib0
movf fib2, 0
movwf fib1
movf fibtemp, 0
movwf fib2
goto forward
end
```



Move novamente o conteúdo de fib1 para o registrador de trabalho.

Move o conteúdo do registrador de trabalho para fib0 (implementação em *assembly* de  $\text{fib0} \leftarrow \text{fib1}$ ).

Move o conteúdo de fib2 para o registrador de trabalho.

Move o conteúdo do registrador de trabalho para fib1 (implementação em *assembly* de  $\text{fib1} \leftarrow \text{fib2}$ ).

Move novamente o conteúdo de fibtemp (que tem a soma dos números anteriores da série) para o registrador de trabalho.

Move o conteúdo do registrador de trabalho para fib2 (implementação em *assembly* de  $\text{fib2} \leftarrow \text{fibtemp}$ ).

Sustentação do laço com goto!

# Teste

- Para quem tem o MPLAB...
  - Ler as seções 4.4 até 4.7 (capítulo 4 do livro texto *Designing Embedded Systems with PIC Microcontrollers*) e reproduzir a simulação do programa da série de Fibonacci.

# Instruções

- Programa exemplo para transferência de dados para os portos...

```
include "p16f84.inc"
```

```
status equ 0x03  
porta  equ 0x05  
trisa  equ 0x05  
portb  equ 0x06  
trisb  equ 0x06
```

**bsf *f*, *d***

Seta o bit **d** do conteúdo armazenado em **f** com valor lógico 1.

```
org 00
```

```
start bsf status, 5  
      movlw b'00011000'  
      movwf trisa  
      movlw 00  
      movwf trisb  
      bcf status, 5
```

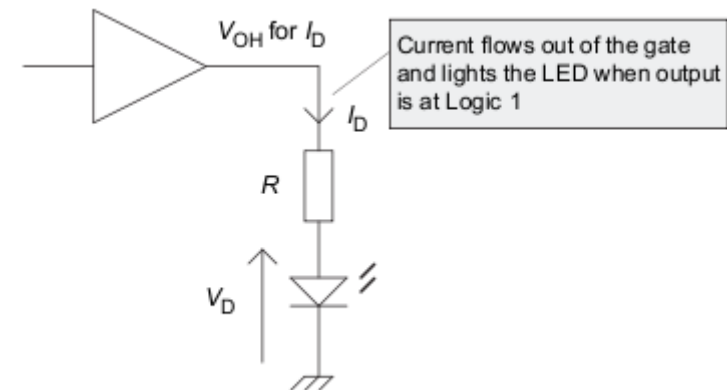
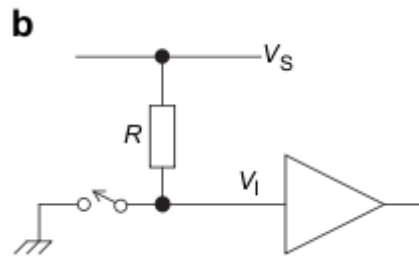
**bcf *f*, *d***

Seta o bit **d** do conteúdo armazenado em **f** com valor lógico 0.

```
      clrf porta  
loop  movf porta, 0  
      movwf portb  
      goto loop  
end
```

# Mais um teste...

- Reproduzir este programa em assembly em um simulador (Proteus, Multisim, Ktechlab,...).
- Adicionar botões para os pinos de entrada no porto A e leds nos pinos do porto b.



# Programas em Assembly

- Ideias para programar melhor:
  - Diagramas de estado.
    - Abstrato → mais difícil de traduzir diretamente para o assembly.

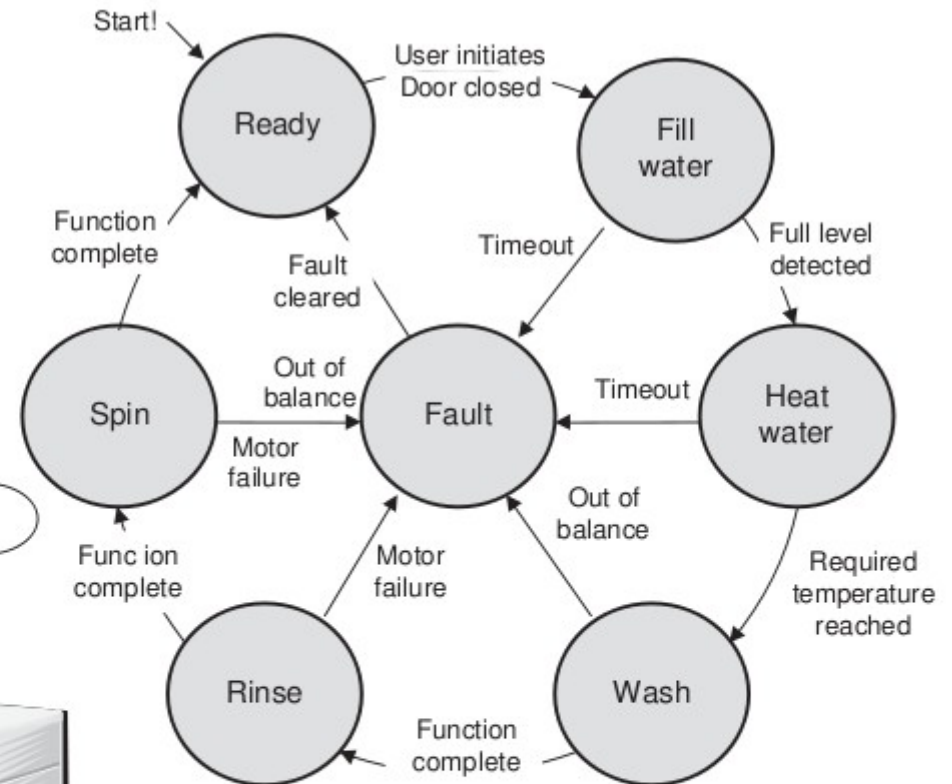


Figure 5.2: A washing machine control program – visualised as a state diagram

# Programas em Assembly

- Ideias para programar melhor:
  - Estruturar o programa como fluxograma.

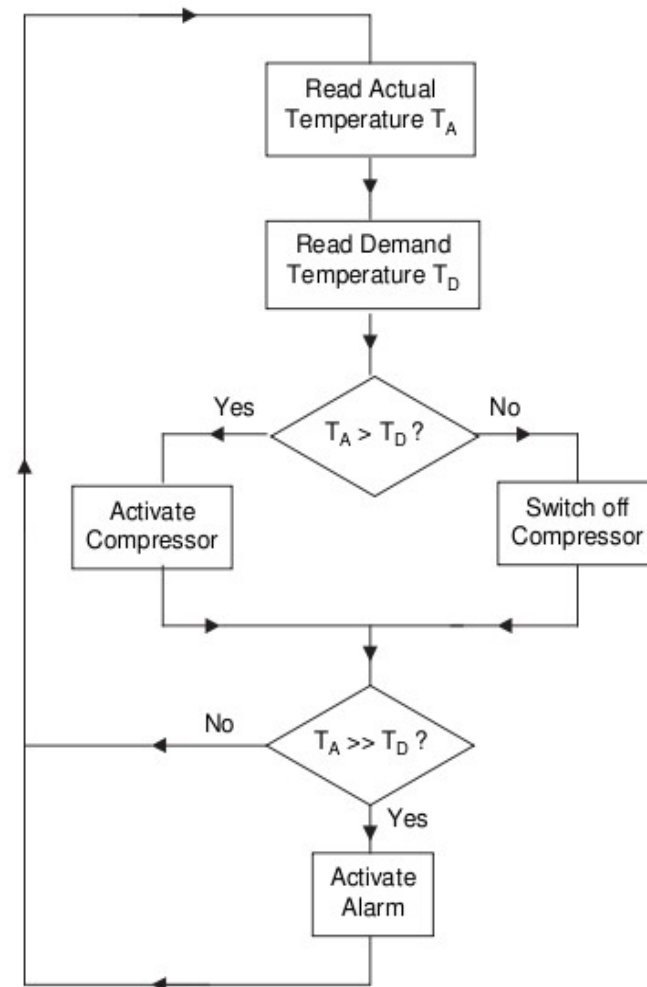


Figure 5.1: Flow diagram of simple refrigerator controller

# Programas em Assembly

- Decisões:
  - Instruções que testam um *bit* em particular e, dependendo do resultado do teste, ou o programa continua sendo executado normalmente ou há um desvio de execução das instruções.

**btfsc *f*, *b***

Testa o bit **b** do conteúdo armazenado no endereço **f**. Se esse bit for 0, então há o 'salto' de apenas uma instrução.

**btfss *f*, *b***

Testa o bit **b** do conteúdo armazenado no endereço **f**. Se esse bit for 1, então há o 'salto' de apenas uma instrução.

# Programas em Assembly

- Exemplo:

```
;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
```



# Programas em Assembly

- Exemplo:

```
;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
```

# Programas em Assembly

- Mais instruções (aritméticas):

*addwf e addlw*

*subwf e sublw*

*incf e decf*

Modificam os três bits menos significativos do registrador de status (Z, DC e C).

# Programas em Assembly

```
;*****
;Fibonacci_full
;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,
;first going up and then down.
;Program intended for simulation only, hence no input/output.
;The program demonstrates addition, subtraction, compare.
;TJW 17.3.05.                      Tested by simulation 18.3.05
;*****

;no i/o ports used
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
```

# Programas em Assembly

```
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
```

# Programas em Assembly

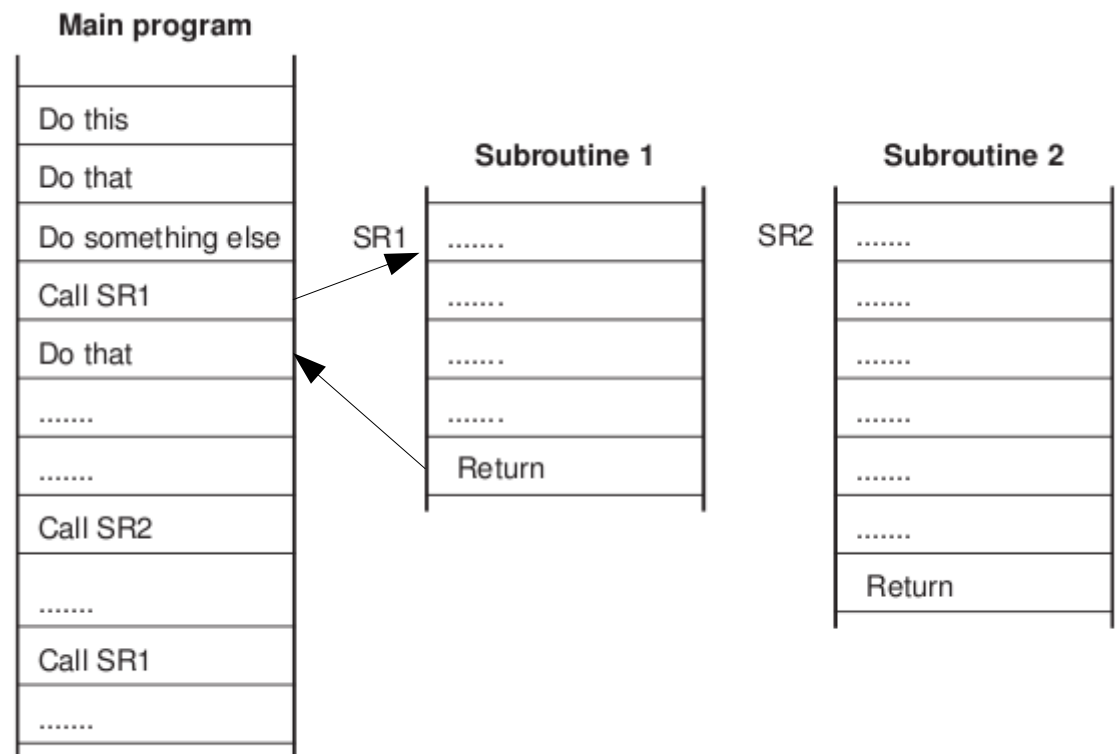
```
;when reversing down, subtract fib0 from fib1 to form new fib0
reverse movf  fib0,0
        subwf fib1,0
        movwf fibtemp      ;latest number now placed in fibtemp
        decf  counter,1
;now shuffle numbers held, discarding the oldest
        movf fib1,0        ;first move middle number, to 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
```

# Mais testes...

- Reproduzir e simular esse programa assembly no MPLAB.

# Subrotinas

- Indicações para uso:
  - Código mais organizado.
  - Uso constante de um mesmo conjunto de instruções.



# Subrotinas

- Instruções PIC 16F:

- *call* e *return*



Necessita de etiqueta (label de 8 bits)

**OBS.:** a pilha somente tem 8 posições.

Quando *call* é utilizada, o endereço da instrução que vem logo após *call* (ou seja, o conteúdo do PC – contador do programa) é armazenada na pilha. O endereço da primeira instrução da subrotina é gravada no PC.

Quando a subrotina é finalizada por *return*, o conteúdo do topo da pilha (endereço da instrução que vem após *call*) é 'desempilhado' e armazenado no PC.

As instruções *call* e *return* devem sempre serem usadas juntas.



# Subrotinas

```
...
(initial program sections omitted)
...
;
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 the series
          movwf fibtemp       ;latest number now placed in fibtemp
          incf  counter,1
;now shuffle numbers held, discarding the oldest
          call shuffle_up
          goto  forward
;when reversing down, we will 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
          call shuffle_down
;test if counter has reached 3, in which case return to forward
          movf  counter,0
          sublw 3
          btfsc status,z
          goto  forward
          goto  reverse
```

# Subrotinas

```
;*****  
;Subroutines  
;*****  
;Shuffles numbers in series, moving fib1 to fib0, fib2 to fib1, fibtemp to fib2  
shuffle_up movf  fib1,0    ;first move middle number, to overwrite oldest  
            movwf fib0  
            movf  fib2,0  
            movwf fib1  
            movf  fibtemp,0  
            movwf fib2  
            return  
;Shuffles numbers in series, moving fib1 to fib2, fib0 to fib1, fibtemp to fib0  
shuffle_down movf fib1,0    ;first move middle number, to overwrite oldest  
            movwf fib2  
            movf fib0,0  
            movwf fib1  
            movf fibtemp,0  
            movwf fib0  
            return  
end
```

# Gerando atraso de tempo

- Subrotinas podem ser usadas para gerar *delays...*
- Ideia:
  - Usar uma certa posição da memória para armazenar um valor que servirá como 'contador' – valor decrementado, em um laço, até chegar a zero.
  - O tempo gasto para isso depende de: do valor inicial armazenado na memória e do tempo gasto no laço.

# Gerando atraso de tempo

- Para que o *delay* seja preciso...
  - Saber frequência do PIC (usando cristal, mais estável).
    - Atrasos não tão precisos podem ser gerados usando outros osciladores (circuito RC externo, por exemplo).
  - Lembrete: cada ciclo de instrução gasta 4 ciclos do oscilador.

# Gerando atraso de tempo

- Exemplo...

```
;Delay of 5ms approx. Instruction cycle time is 5us.
delay5  movlw      D'200'          ;200 cycles called, each taking 5x5=25us
        movwf      delcntrl
del1    nop                ;1 inst. cycle
        nop                ;1 inst. cycle
        decfsz     delcntrl,1      ;1 inst. cycle, when no skip
        goto del1              ;2 inst. cycles
        return
```

**Obs.:** solução apropriada para atrasos pequenos (ordem de *dezenas* de milissegundos).

## decfsz *f*, *d*

Decrementa o conteúdo presente no endereço *f*. Se esse conteúdo for zero, salta-se uma instrução. O bit *d* direciona para onde o resultado do decremento será armazenado.

*nop* → *no operation* (sem execução na ULA, mas gasta-se os quatro ciclos de *clock*)

# Gerando atraso de tempo

- Para obter atrasos maiores (da ordem de *centenas* de milissegundos)...
- Colocar laço de atraso dentro de outro laço de atraso!

```
;Flashing LEDs 1.
;This program continuously outputs a series of LED patterns,
;using ping-pong hardware. LED patterns are listed within
;the program.
;TJW 9.11.08                                     Tested in simulation 9.11.08
;*****
;Clock is 800kHz
;Configuration Word: WDT off, power-up timer on,
;                      code protect off, RC oscillator
;
;specify SFRs
pcl      equ 02
status  equ 03
trisa    equ 05
portb    equ 06
trisb    equ 06
;
delcntr1 equ 11
delcntr2 equ 12
;
```

# Gerando atraso de tempo

```
        org      00
;Initialise
start bsf      status,5      ;select memory bank 1
        movlw   B'00011000' ;set port A right for hardware,
        movwf   trisa        ;even tho not used in this program.
        movlw   00
        movwf   trisb        ;all port B bits output
        bcf     status,5      ;select bank 0
;
;The "main" program starts here
loop movlw   B'01010101'
        movwf   portb        ;set up new output pattern
        call    delay
        movlw   B'10101010'
        movwf   portb        ;set up new output pattern
        call    delay
        goto    loop         ;loop again
```

# Gerando atraso de tempo

```
;*****  
;Subroutine  
;*****  
;Introduces delay of 500ms approx, for 800kHz clock  
delay movlw D'100'  
        movwf delcntr2 ;will do the outer loop 100 times  
outer  movlw D'200'  
        movwf delcntr1 ;will do the inner loop 200 times,  
                        ;at 5cycles = 25us, this is 5ms  
inner  nop                ;1 cycle  
        nop                ;1 cycle  
        decfsz delcntr1,1 ;normally 1cycle  
        goto    inner      ;2 cycles  
        decfsz delcntr2,1  
        goto    outer  
        return  
;  
        end
```



# Instruções Lógicas

**andwf *f*, *d***

AND entre conteúdo de W e conteúdo de *f*.

**iorwf *f*, *d***

OR inclusivo entre W e *f*.

**xorwf *f*, *d***

OR exclusivo entre W e *f*.

**andw *k***

AND entre conteúdo de W e *k*.

**iorlw *k***

OR inclusivo entre W e valor literal *k*.

**xorlw *k***

OR exclusivo entre W e valor *k*.

**rlf *f*, *d***

Deslocamento de 1 bit para esquerda

**rrf *f*, *d***

Deslocamento de 1 bit para direita.

# Instruções Lógicas

```
;Clock is 800kHz
;Configuration Word: WDT off, power-up timer on,
;                      code protect off, RC oscillator
;
;specify SFRs and bits
c      equ      0
pcl    equ      02
status equ      03
trisa  equ      05
portb  equ      06
trisb  equ      06
;
delcntr1 equ 11      ;used as counter in delay subroutine
delcntr2 equ 12      ;used as counter in delay subroutine
flags    equ 13      ;bit 0 will be set once initial pattern is op
;
                org    00
;Initialise
start bsf      status,5      ;select memory bank 1
        movlw  B'00011000'   ;set port A right for hardware,
        movwf  trisa         ;even tho not used in this program.
        movlw  00
        movwf  trisb         ;all port B bits output
        bcf    status,5      ;select bank 0
        clrf   flags
```

# Instruções Lógicas

```
;
;The "main" program starts here
;insert below one of pattern_XOR, pattern_RRF, pattern_IOR to be the
;target of this subroutine call
loop  call  pattern_IOR ;select SR to be called here
      call  delay
      goto  loop
;
```

# Instruções Lógicas

```
;*****  
;Subroutines  
;*****  
;Changes led pattern, using XOR instruction  
pattern_XOR btfsc flags, 0  
    goto    patt_XOR1    ;here if first visit to SR  
    bsf     flags,0  
    movlw   B'10101010'  
    movwf   portb        ;set up initial output pattern  
    return  
patt_XOR1 movf portb,0    ;here if 2nd or later visit to SR  
    xorlw   B'11111111'  
    movwf   portb  
    return  
  
;Changes led pattern, using rrf instruction  
pattern_RRF btfsc flags,0  
    goto    patt_RRF1    ;here if first visit to SR  
    bsf     flags,0  
    movlw   B'10000000'  
    movwf   portb        ;set up initial output pattern  
    return  
;here if 2nd or later visit to SR  
patt_RRF1 bcf status,c    ;clear carry flag  
    rrf     portb,1  
    btfsc   status,c      ;has pattern reached carry flag?  
    bsf     portb,7       ;if yes, reset msb  
    return
```

# Instruções Lógicas

```
;Changes led pattern, using OR and rrf instructions
pattern_IOR btfsf flags,0
    goto    patt_IOR1    ;here if first visit to SR
    bsf     flags,0
    clrf    portb        ;set up initial output pattern
    return
patt_IOR1 rrf portb,0     ;here if 2nd or later visit to SR
    iorlw   B'10000000'  ;add another 1 bit to pattern
    btfsf   status,c     ;has pattern reached carry flag?
    clrw    ;if yes, clear W
    movwf   portb
    return

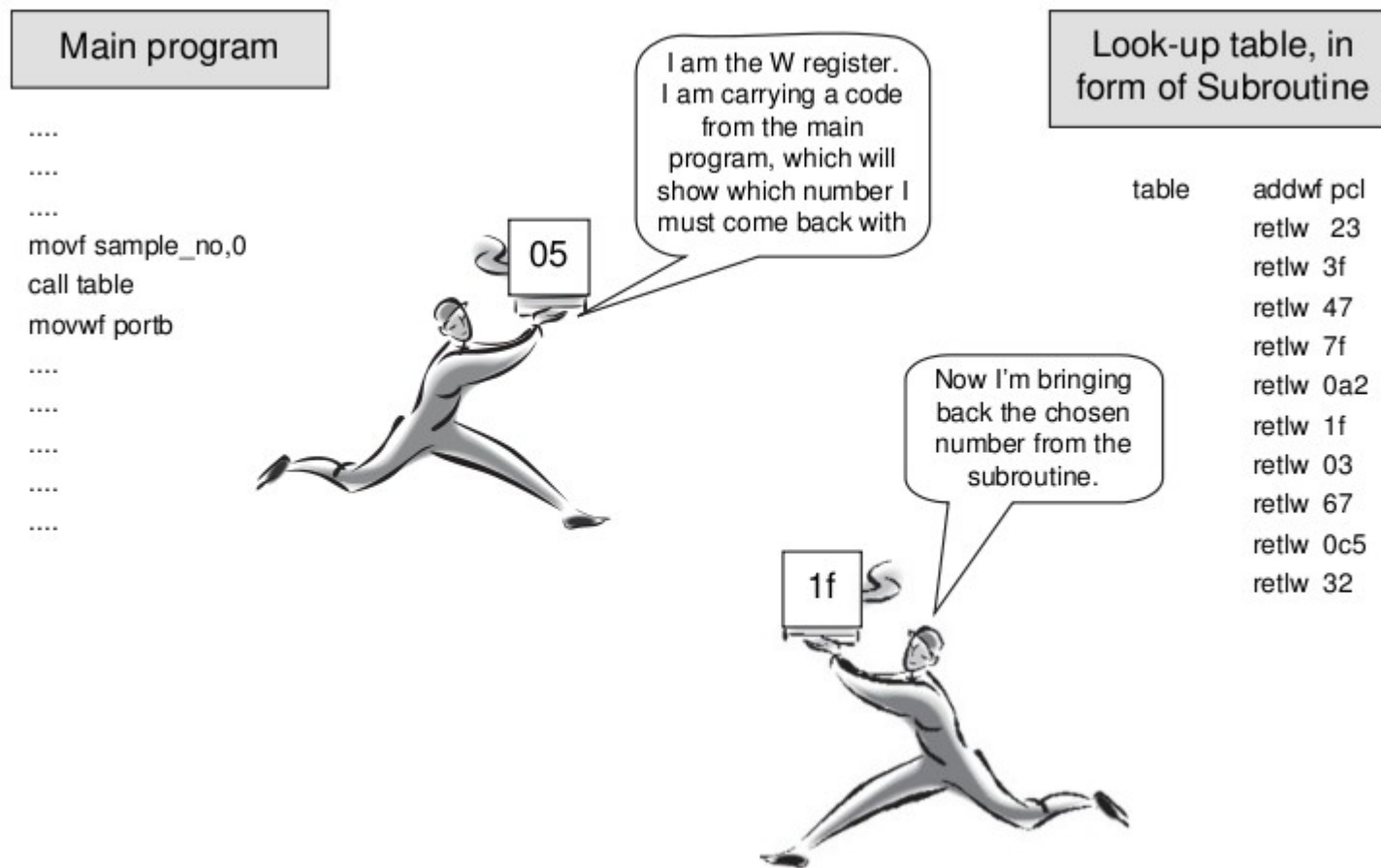
;Introduces delay of 500ms approx, for 800kHz clock
...
    (delay subroutine omitted)
...
    end
```

# Tabelas *Look-up*

- Consiste em um bloco de dados que é armazenado na memória do programa, e que é acessado e utilizado por ele.
- É o elemento utilizado para a inserção de um padrão de valores em algum endereço ou registrador.

# Tabelas *Look-up*

- Ideia geral → subrotina!



# Tabelas *Look-up*

- Somente duas instruções na subrotina são executadas.
  - `addwf pcl` → soma o conteúdo do registrador W ao `pcl` (byte menos significativo do PC).
  - `retwl` → instrução de retorno de subrotina. Retorna o valor literal indicado ao registrador W.

Ver exemplo!



# Tabelas *Look-up*

File Address		File Address	
00h	Indirect addr. <sup>(1)</sup>	Indirect addr. <sup>(1)</sup>	80h
01h	TMR0	OPTION_REG	81h
	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2 <sup>(1)</sup>	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch			8Ch
68 General Purpose Registers (SRAM)		Mapped (accesses) in Bank 0	
4Fh			CFh
50h			D0h
Bank 0		Bank 1	
7Fh			FFh

MSB is 'bank select bit' (Status register).

Unimplemented data memory location, read as 'U'.

**Note 1:** Not a physical register.

# Tabelas *Look-up*

```
;*****  
;Flashing LEDs 3.  
;This program continuously outputs a series of LED patterns,  
;using simulation or ping-pong hardware.  
;TJW 5.3.05.           Tested in simulation 11.3.05.  
;*****  
;Clock is 800kHz  
;Configuration Word: WDT off, power-up timer on,  
;                      code protect off, RC oscillator  
;  
;specify SFRs  
pcl      equ 02  
status   equ 03  
porta    equ 05  
trisa    equ 05  
portb    equ 06  
trisb    equ 06  
;  
pointer  equ 10  
delcntr1 equ 11  
delcntr2 equ 12  
;  
          org 00
```

# Tabelas *Look-up*

```
;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
      movlw    00                  ;clear all bits in port A
      movwf    porta
      movwf    pointer            ;also clear pointer
loop   movf     pointer,0          ;move pointer to W register
      call     table
      movwf    portb              ;move W register, updated from table SR, to port B
      call     delay
      incf     pointer,1
      btfsc    pointer,3          ;test if pointer has incremented to 8
      clrf     pointer            ;if it has, clear pointer to start over
      goto     loop
;
```


# Tabelas *Look-up*

```
;*****  
;Subroutines  
;*****  
;Introduces delay of 500ms approx, for 800kHz clock  
...  
    (delay subroutine omitted)  
...  
  
;Holds Lookup Table  
table    addwf pcl  
         retlw 23  
         retlw 3f  
         retlw 47  
         retlw 7f  
         retlw 0a2  
         retlw 1f  
         retlw 03  
         retlw 67  
;  
        end
```

# Tabelas *Look-up*

- Dica: pode-se usar um 'and' para 'limpar' a variável pointer...

```
...  
loop  movf  pointer,0      ;move pointer to W register  
      call table  
      movwf portb          ;move W register, updated from table SR, to port B  
      call delay  
      incf pointer,0       ;increment pointer, place result in W reg  
      andlw 07  
      movwf pointer  
      goto loop  
...
```



# Complexidade dos Programas

- Diretiva `#include`: cada arquivo “.inc” possui várias definições de endereços de memória.

```
;specify SFRs
```

```
timer    equ    01
```

```
status   equ    03
```

```
porta    equ    05
```

```
trisa    equ    05
```

```
portb    equ    06
```

```
trisb    equ    06
```

```
intcon   equ    0B
```

```
#include p16f84A.inc
```

# Complexidade dos Programas

- Macros: recurso utilizado pelo programador para agrupar instruções e, com isso, obter mais flexibilidade na escrita dos programas.

```
;now ready for action
;macro to move a literal value to a file
movlf macro const,address
    movlw const
    movwf address
endm

;macro to branch if a specified bit is set
bfbset macro file,bit,target
    btfsc file,bit
    goto target
endm

;macro to branch if a specified bit is clear
bfbclr macro file,bit,target
    btfss file,bit
    goto target
endm

wait    movlf 04,porta    ;at rest, "out of play"
        movlf 00,portb    ;all play leds off
;both paddles must initially be clear before play allowed to commence
        bfbclr porta,4,wait ;go to wait if right paddle pressed
        bfbset porta,3,wait ;go to wait if left paddle pressed
;
```