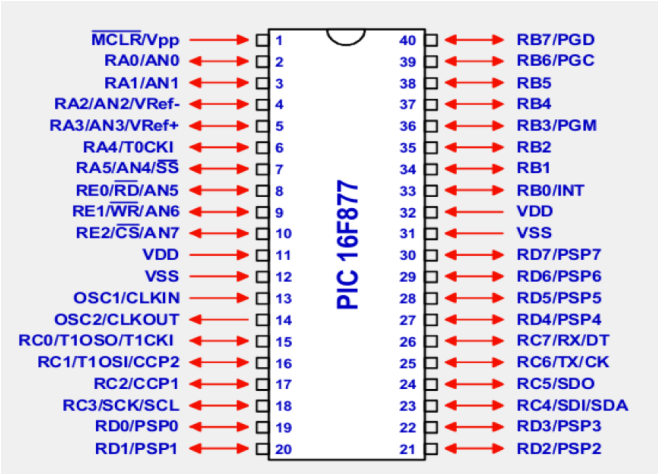
le PIC16f877

cours n°2
LI326

Boitier pic16f877 40 broches



Architecture

Caractéristiques pic16f877

Fonctionne **à 20 MHz** maximum.

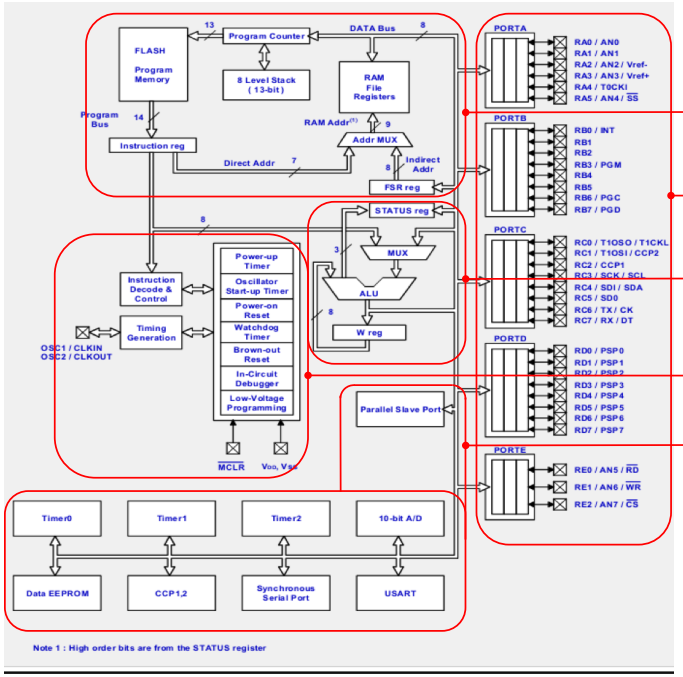
- mais utilise 4 cycles par instruction donc
  5 milions d'instructions par seconde ou **200 nanosecondes par instruction**

Possède :

- 35 instructions (composant RISC),
- 8Ko de mémoire Flash interne pour le programme,
- 368 octets de RAM, ce sont plutôt des registres (FILE)
- 256 octets de d'EEprom,
- 3 compteurs / timer de 8 ou 16 bits (PWM)
- 1 Watchdog,
- 8 entrées analogiques 10bits / comparateur de tension
- 1 port série (RS232), 1 port I2C / SPI
- 15 sources d'interruption,
- 33 entrées/sorties numériques configurables individuellement, disposés en 5 ports nommés de A à E,
- un mode SLEEP.

Architecture

Architecture pic16f877



mémoires
programme et données

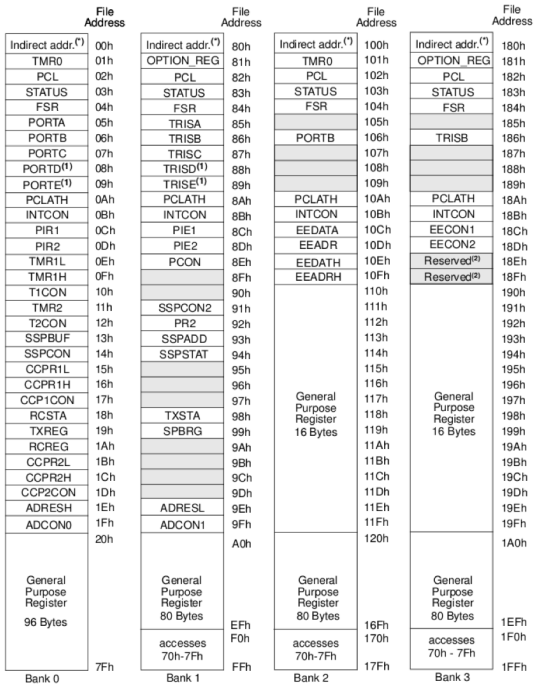
ports d'entrées-sorties

unité de calcul

unité de contrôle

périphériques

Architecture



Données Registres

- 4 bancs
- SFR
  Special File Register
  pour les périphériques
- GPR
  General Purpose Reg
  pour les programmes
- certains registres
  adresses multiples

  $0 = $80 = $100 = $180
  $70 = $F0 = $170 = $1F0
  ...

Mémoire

Assembleur à une adresse

Cas général d'une instruction (dit 3 adresses)
register_dest = operande1 OPER operande2

Pour le PIC
une des operandes est **W**
Le registre de destination est une des opérandes

exemple MIPS :
add $rd, $rs, $rt c.-à-d. $rd = $rs + $rs (p.ex. add $6,$4,$3)

en PIC (notez que la destination est à la fin) :
addwf 23,1 reg23 = reg23 + W
addwf 23,0 W = reg23 + W

Pour certaines instructions on n'a qu'une seule opérande

Assembleur

Assembleur PIC

Un programme en assembleur comporte une *instruction* par ligne.
Une ligne se découpe en quatre colonnes

1. la 1ère colonne commence en tout début de ligne et contient au plus un mot appelé symbole ou étiquette associé à l'adresse de la case mémoire de l'instruction ou de la donnée courante. On peut aussi attribuer une valeur quelconque à une étiquette.
2. la 2ème colonne commence après un espace ou une tabulation. elle contient une instruction, une macro-instruction ou une directive.
3. la 3ème colonne est séparée par un espace ou une tabulation contient les arguments séparés par des virgules de la 2ème colonne.
4. la 4ème colonne commence par un ; s'achève au retour chariot contient un commentaire.

Table 29-1: Midrange Instruction Set

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

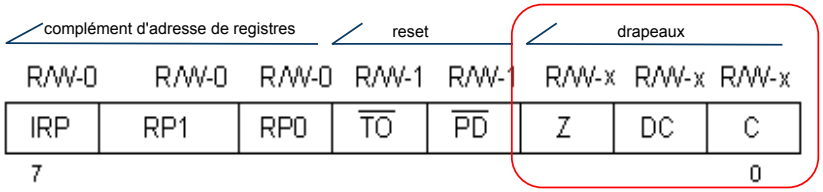
Instructions

Assembleur

Exercices pour comprendre

$20 = 0	clrf 0x20
$20 = 1	clrf 0x20 incf 0x20,F
$20 = $20 + 1	incf 0x20
$20 = $20 + $21	movf 0x21, W addwf 0x20,F
$20 = $21 - 2	movlw 2 addwf 0x21,W movwf 0x20
$20 = $20 - $21	movf 0x20,W subwf 0x21,F

Registre status



R = Readable bit **W** = Writable bit

U = Unimplemented bit, read as '0' - n = Value at power-on reset

Le registre status est modifié par la plupart des instructions

Branchements (principe)

Comparaisons

goto adresse

branch_if_cond operande1, operande2, adresse

Pour le PIC

goto adresse (ATTENTION l'adresse n'est pas complète)

skip_if_notcond

goto adresse

exemple MIPS :

```
beq   $4,$5,label
```

en PIC :

```
movf   23,0
xorwf  22,0
btfsc  status,2
goto   label
```


Branchements (exemples)

Égalité registre/registre et registre/immédiat

```
; si (REG1 == REG2) goto LAB
; beq REG1, REG2, LAB
  movf   REG1, W
  xorwf  REG2, W
  btfsc  STATUS, Z
  goto   LAB

; si (REG1 != REG2) goto LAB
; bne REG1, REG2, LAB
  movf   REG1, W
  xorwf  REG2, W
  btfss  STATUS, Z
  goto   LAB

; si (REG == IMM) goto LAB
; beq REG, IMM, LAB
  movlw  IMM
  xorwf  REG, W
  btfsc  STATUS, Z
  goto   LAB

; si (REG1 != IMM) goto LAB
; bne REG1, IMM, LAB
  movlw  IMM
  xorwf  REG, W
  btfss  STATUS, Z
  goto   LAB
```

Comparaisons registre/registre sur des nombres non signés

```
; si (REG1 < REG2) goto LAB
; blt REG1, REG2, LAB
  movf   REG2, W
  subwf  REG1, W
  btfss  STATUS, C
  goto   LAB

; si (REG1 > REG2) goto LAB
; bgt REG1, REG2, LAB
  movf   REG1, W
  subwf  REG2, W
  btfss  STATUS, C
  goto   LAB

; si (REG1 <= REG2) goto LAB
; ble REG1, REG2, LAB
  movf   REG1, W
  subwf  REG2, W
  btfsc  STATUS, C
  goto   LAB

; si (REG1 >= REG2) goto LAB
; bge REG1, REG2, LAB
  movf   REG2, W
  subwf  REG1, W
  btfsc  STATUS, C
  goto   LAB
```

Exercices pour comprendre

$20 = SOMME de 0 à 100

```
      $21 = 100
      $20 = 0
do    $20 = $20 + $21
      $21 = $21 - 1
      Si $21 != 0 goto do
```

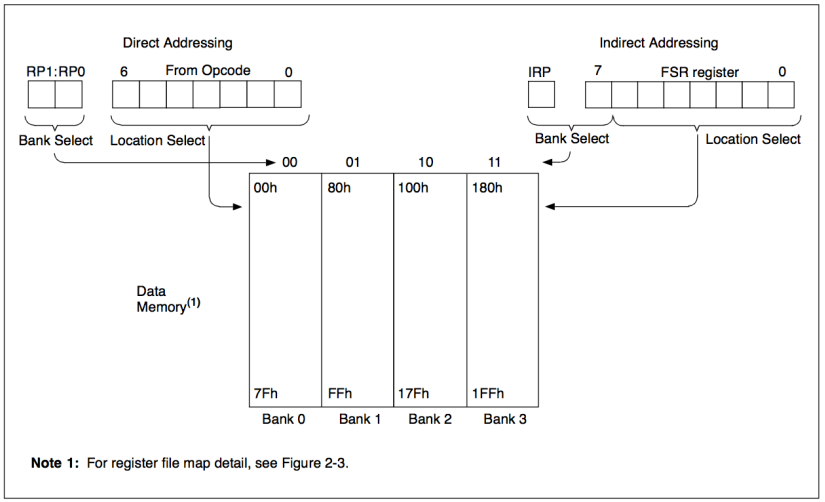
```
      movlw   100
      movwf   0x21
      clrf    0x20
do
      movf    0x21,W
      addwf   0x20,F
      decfsz  0x21
      goto    do
```

$20 = PGCD ($70, $71)

```
      goto L3
L0    si $70 > $71 goto L2
L1          $70 = $70 - $71
       goto L3
L2          $71 = $71 - $70
L3    si $70 != $71 goto L0
```

Calcul des operandes

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING

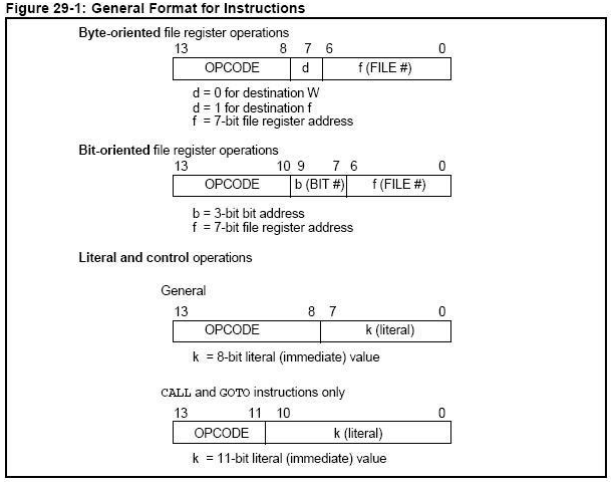


Note 1: For register file map detail, see Figure 2-3.

Assembleur

Codage des instructions

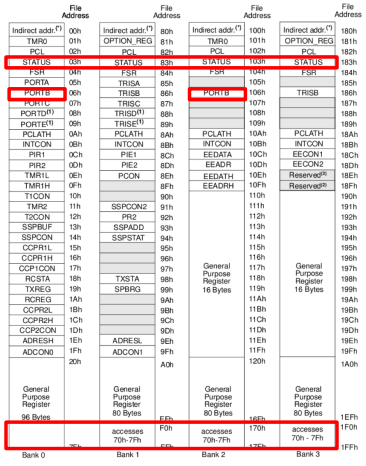
Figure 29-1: General Format for Instructions

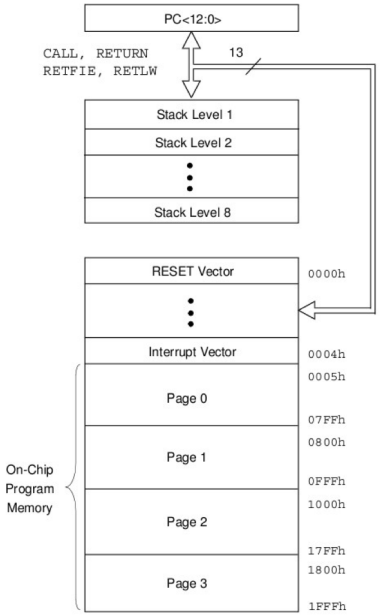


Il manque 2 bits pour adresser les registres 7 au lieu de 9
il manque 2 bits pour fabriquer les adresses d'instructions

Accès aux registres

- GPR standard
  - Ecrire dans les bits RP0, RP1 du registre STATUS
    - (bcf | bsf) STATUS, (rp0 | rp1)
    - ou banksel REGISTRE
  - Accéder au registre
- GPR répliqué dans les 4 bancs
  - Accéder au registre
- GPR répliqué dans les 2 bancs
  - Ecrire dans les bits RP0 du registre STATUS
    - (bcf | bsf) STATUS, rp0
    - ou banksel REGISTRE
  - Accéder au registre





Programme

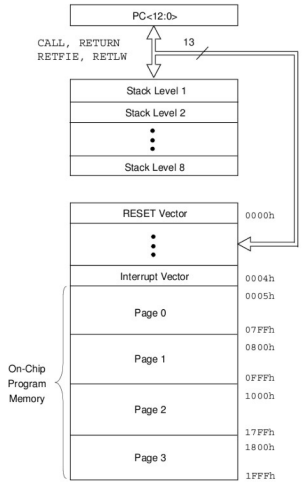
- 4 pages de programmes
- Pile hardware pour les adresses
- 2 adresses pour le traitement des événements matériels
  - reset
  - interruption

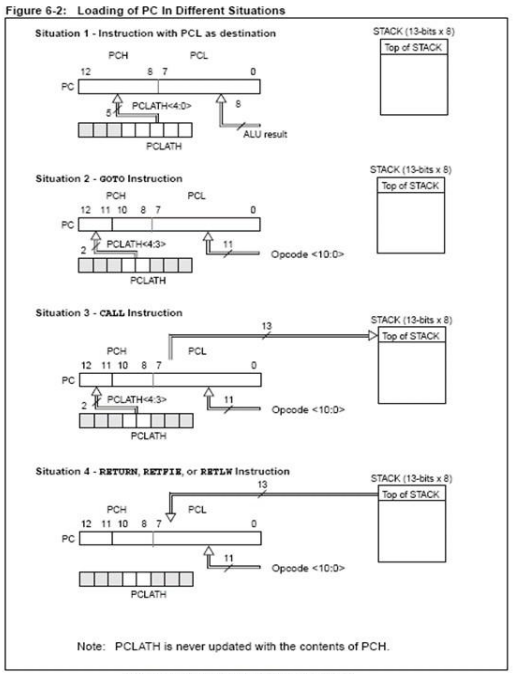
Mémoire

Saut à une instruction quelquonque

- GOTO
  - Ecrire dans les bits de PCLATH le numéro du banc
  - goto address
- Exemple
  - goto 0x1000

```
bcfPCLATH,3
bsfPCLATH,4
goto   0x1000
```





Courtesy Microchip Technology Inc.

Calcul du PC

- Le PC est sur 13 bits
- Les 8 bits de poids faible sont accessibles en lecture/écritures
- Les 5 bits de poids fort sont accessible en écriture seule

Assembleur

Hello World

(façon microcontrôleur)

GPIO principe

Les ports à usage général sont des entrées / sorties (GPIO)

Pour chaque broche, nous avons deux bits de commande
- 1 bit pour définir de sens : entrée (1), sortie (0)
- 1 bit pour lire la donnée (si entrée) ou l'écrire (si sortie)

Les registres de direction sont nommés TRISx
Les registres de données sont nommés PORTx

Par exemple :
TRISD / PORTD sur 8 bits

Hello World !

```
        list    p=16f877          ; definit le processeur cible
        include "p16f877.inc"     ; declaration des noms de registres

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC & _LVP_OFF

        ORG     0

initialisation
        bcf     STATUS,RP1        ; STATUS(RP1) <- 0
        bsf     STATUS,RP0        ; STATUS(RP0) <- 1 permet de désigner le banc 1
        bcf     TRISD,0           ; place le bit 0 du port D en sortie
        bcf     STATUS,RP0        ; STATUS(RP0) <- 0 revient sur le banc 0

main
        movlw   1                 ;
        movwf   PORTD             ; met 1 sur le bit 0 sur port D
loop
        xorwf   PORTD,f           ; inverse la valeur du bit 0
        goto    loop              ; on boucle

        END                       ; directive de fin de programme
```

programme

Directives assembleur

LIST <expression> [, <expression>] *
- n=nnn Sets the number of lines per page
- p= <symbol> Sets the current processor
- r= [oct | dec | hex] Sets the radix

INCLUDE "p16f877.inc"
- déclaration des noms de registres
- macro-définitions
- définition de fonctions

ORG <expression>
- définition de l'adresse à partir de laquelle le programme va être placé.

<symbol> EQU <expression> (cf CONSTANT)
- associe définitivement <expression> à <symbol>

<symbol> SET <expression> (cf VARIABLE)
- associe temporairement <expression> à <symbol>

__CONFIG <expression>
- Definition du registre de configuration du pic
- _CP_OFF : le code peut être relu
- _WDT_OFF: pas de timer watch dog

END
- fin du programme

p16f877.inc (extrait)

```
W               EQU  H'0000'
F               EQU  H'0001'

INDF            EQU  H'0000'
TMR0            EQU  H'0001'
PCL             EQU  H'0002'
STATUS          EQU  H'0003'
FSR             EQU  H'0004'
PORTA           EQU  H'0005'
PORTB           EQU  H'0006'
PORTC           EQU  H'0007'

C               EQU  H'0000'
DC              EQU  H'0001'
Z               EQU  H'0002'
NOT_PD          EQU  H'0003'
NOT_TO          EQU  H'0004'
IRP             EQU  H'0007'

RP0             EQU  H'0005'
RP1             EQU  H'0006'
```

Outils de développement

- **éditeur**
  gvim-emacs-kate-gedit
- **assembleur**
  gpasm
- **simulateur**
  gpsim
- **programmateur**
  picprog
  (ou un bootloader)