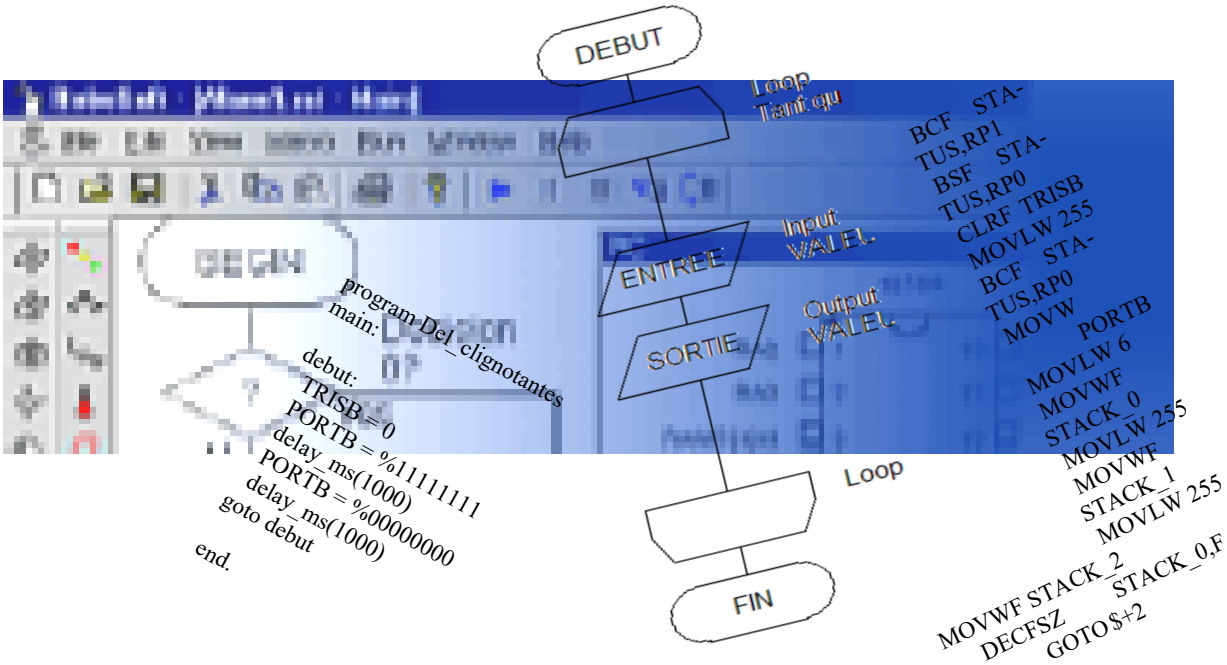


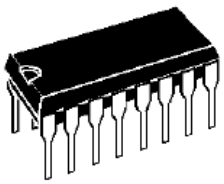
INTÉGRER L'ÉLECTRONIQUE ET L'INFORMATIQUE INDUSTRIELLE DANS LES PROJETS DE BTS CIM

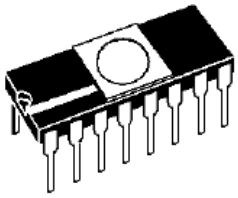
La programmation des microcontrôleurs

MICROCONTRÔLEURS

Qu'est-ce qu'un microcontrôleur?

Un microcontrôleur (MCU) pourrait être très sommairement défini comme un système de contrôle complet, dédié à une application particulière et pour cela doté de fonctions logiques et de la plupart des autres éléments nécessaires à son fonctionnement, d'où un nombre très restreint de composants périphériques.





Voici le *block diagram* (l'organisation interne) typique d'un microcontrôleur

ROM	RAM
CPU	EEPROM
TIMER	A/D CONVERTER
I/O PORT	SERIAL INTERFACE

ROM (*Read Only Memory*): c'est la mémoire "morte", qui contient le programme, donc les instructions à accomplir, et éventuellement des données (*data*).

> La ROM est parfois remplacée par de la mémoire flash, qui peut être effacée électriquement, ce qui permet de modifier le programme. Une mémoire flash peut être effacée des milliers de fois!

RAM (*Random Access Memory*): c'est la mémoire dite "vive", qui permet de stocker des données pendant l'éxécution du programme.

EEPROM (*Electrically Erasable Programmable Read Only Memory*): cette mémoire peut être effacée et reprogrammée, comme la flash. Elle sert à sauvegarder des données, en cas par exemple de coupure de courant.

CPU (*Central Processing Unit*): c'est le "cerveau" du système. Ce microprocesseur lit et éxécute les instructions du programme stocké en mémoire.

Timer: il sert de base de temps interne au système, génère des signaux, compte des évènements.

> Certains MCU sont dotés d'un *watchdog timer* (chien de garde): ce dispositif, s'il n'est pas réinitialisé par le programme à intervalles prédéfinis, considère qu'il y a problème au niveau logiciel et provoque un reset matériel (*hardware reset*).

I/O ports (*Input/output ports*): la plupart des MCU possèdent plusieurs ports d'entrée/sortie, qui permettent de "communiquer" avec le système (par l'intermédiaire, par exemple, d'un clavier) . .

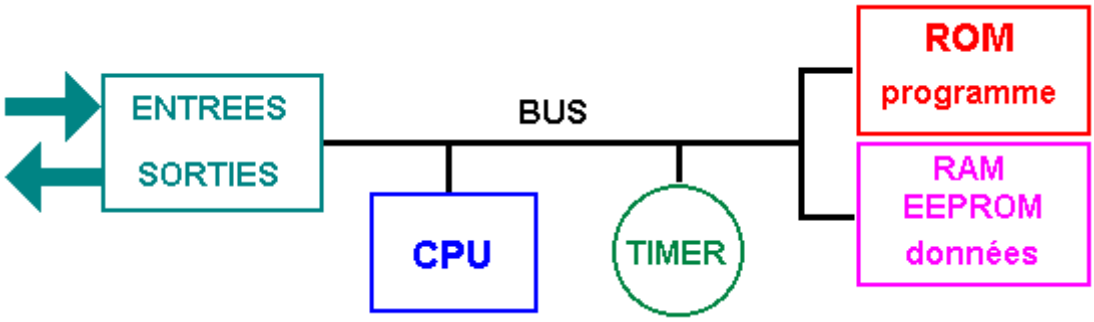
Serial interface: elles servent à échanger des données avec le monde extérieur. On en trouve de deux types: asynchrone (*serial communication interface*, SCI ou UART) ou synchrone (*serial peripheral interface*, SPI).

> Ces interfaces sont utilisées, par exemple, pour relier le MCU à un PC . Dans le cas d'une transmission synchrone, chaque bit est synchronisé par un signal d'horloge. Dans le cas d'une transmission asynchrone, chaque octet est précédé d'un bit de départ et se termine par un bit de fin, qui permettent de synchroniser émetteur et récepteur.

A/D converter: convertissseur analogique/numérique

Comment fonctionne un microcontrôleur?

Le schéma ci-dessous résume le fonctionnement interne d'un microcontrôleur: à droite, les mémoires contenant instructions et données; au centre l'horloge et le processeur, qui participent à l'exécution des instructions du programme; à gauche, les entrées et sorties, permettant au microcontrôleur de communiquer avec le monde extérieur (recevoir des instructions, transmettre des données...).

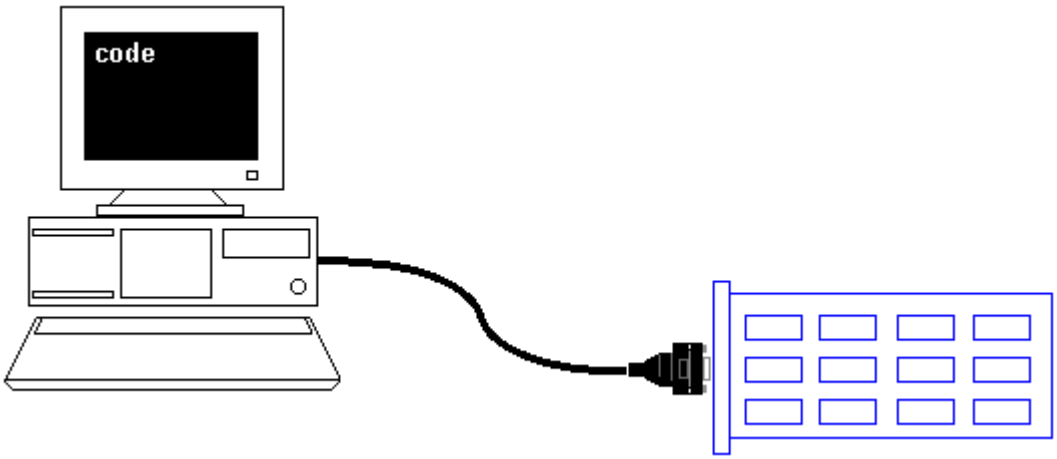


Cette manière de présenter les choses n'a sans doute qu'un seul mérite: une (relative) lisibilité... Dans la réalité, un microcontrôleur se révèle nettement plus complexe, surtout s'il est doté de caractéristiques plus sophistiquées.

La programmation d'un microcontrôleur

Compte tenu de ce qui a été dit plus haut, il parait évident qu'un MCU n'est pas un circuit comme les autres: on ne peut pas se contenter de le souder sur la carte. D'ailleurs, on évite de le souder: il vaut mieux l'insérer dans un support, ce qui permet au besoin de le retirer, puis de le remettre en place ou de le remplacer.

Pour accomplir sa tâche, un MCU doit être **programmé** à cet effet: il faut au préalable implanter dans sa mémoire un programme, c'est-à-dire une série d'instructions, que le processeur éxécutera. Cette étape implique deux types d'outils, appelés de développement: des outils logiciels (*software tools*) et matériels (*hardware tools*).



La programmation proprement dite est réalisée sur ordinateur, grâce à un environnement de programmation, comprenant un langage (souvent le C), un compilateur, un linker, un debugger... Il s'agit d'écrire le code, ou suite d'instructions qui devront être éxécutées, dans les limites bien sûr des capacités du MCU (mémoire, vitesse d'horloge...).

Une fois le programme écrit, il convient de le tester, à l'aide d'un simulateur (ou émulateur), qui reproduit en temps réel le comportement d'un MCU donné, et du debugger. Lorsque tout va bien, il ne reste plus qu'à transférer le programme dans la mémoire du MCU.

La frontière entre électronique et informatique est ici, on le voit, pour ainsi dire inexistante...

A quoi servent les microcontrôleurs?

Le microcontrôleur apparait donc comme un système extrêmement complet et performant, capable d'accomplir une ou plusieurs tâches très spécifiques, pour lesquelles il a été programmé.

Ces tâches peuvent être très diverses, si bien qu'on trouve aujourd'hui des microcontrôleurs presque partout: dans les appareils électro-ménagers (réfrigérateurs, fours à micro-ondes...), les téléviseurs et magnétoscopes, les téléphones sans fil, les périphériques informatiques (imprimantes, scanners...), les voitures (airbags, climatisation, ordinateur de bord, alarme...), les avions et vaisseaux spatiaux, les appareils de mesure ou de contrôle des processus industriels, etc.

La force du microcontrôleur, qui lui a permis de s'imposer de manière si envahissante en si peu de temps, c'est sa spécialisation (il remplit une ou quelques tâches bien définies, c'est tout), sa très grande fiabilité, son coût modique (pour les modèles produits en grande série, notamment pour l'industrie automobile).

Ajoutons que par rapport au microprocesseur, toujours assoiffé de performances pures, le modeste microcontrôleur se cantonne dans un rôle obscur et ingrat, mais essentiel, tout en visant le meilleur rapport qualité/prix

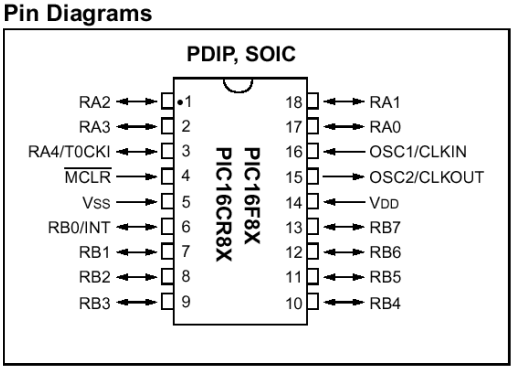
Programmation des PIC 16F84 PIC 16F628 , PIC 16F88 ..

INTRODUCTION :

Les PIC 16F84, 628, 88 sont des microcontrôleurs 8 bits d'architecture de type RISC. RISC de l'anglais *Reduced Instructions Set Computer*, signifie " Calculateur à jeu réduit d'instructions ".

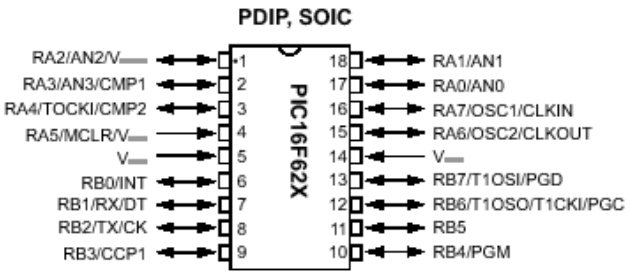
PIC 16F84

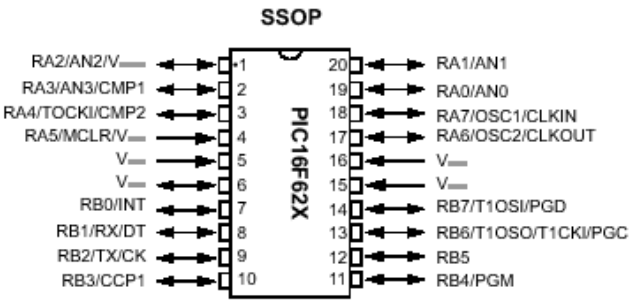
Pin Diagrams



On trouve sur le marché 2 familles opposées, les RISC et les CISC (Complex Instructions Construction Set). Chez les CISC, on diminue la vitesse de traitement, mais les instructions sont plus complexes, plus puissantes, et donc plus nombreuses. Il s'agit donc d'un choix de stratégie.

PIC 16F628





Tous les PICs Mid-Range ont un jeu de 35 instructions, et stockent chaque instruction dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en 1 cycle. On atteint donc des très grandes vitesses, et les instructions sont de plus très rapidement assimilées. L'exécution en un seul cycle est typique des composants RISC.

L'horloge fournie au PIC est pré divisée par 4 au niveau de celle-ci. C'est cette base de temps qui donne le temps d'un cycle.Si on utilise par exemple un quartz de 4MHz , on obtient donc 1000000 de cycles/seconde, or, comme le PIC exécute pratiquement 1 instruction par cycle, hormis les sauts, cela vous donne une puissance de l'ordre de 1MIPS (1 Million d'Instructions Par Seconde).

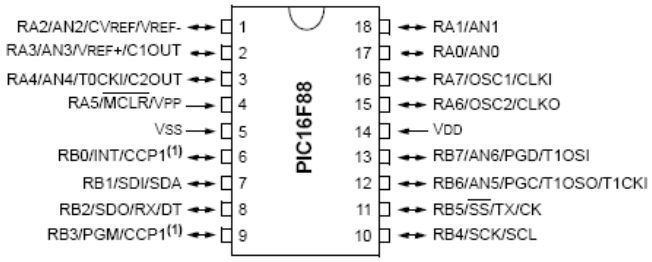
Les pics peuvent monter à 20MHz. .

PIC 16F88

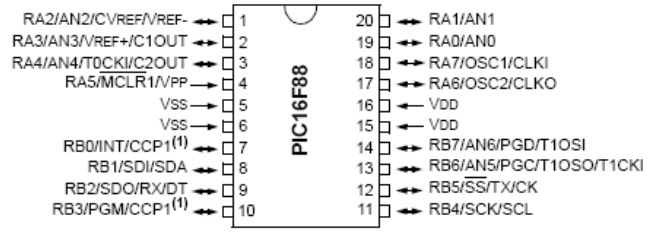
Le Pic 16F88 possède:

- 16 entrées sorties dont 7 entrées analogiques sur 10 bits.
- Oscillateur intégré jusqu'à 8 MHZ.
- Reset interne ou externe.
- Instructions d'écriture dans la mémoire Flash.
- Mode basse consommation
- 4096 mots de mémoire flash
- 368 octets de mémoire RAM
- Peut être utilisé en mode stand alone.

18-Pin PDIP, SOIC



20-Pin SSOP

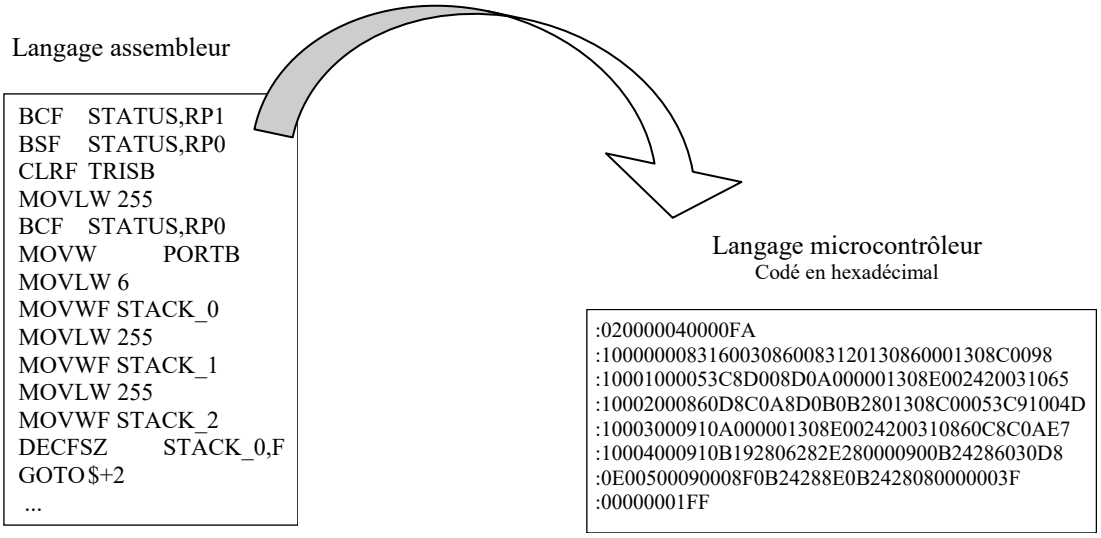


LES INSTRUCTIONS DU PIC 16 F88 :

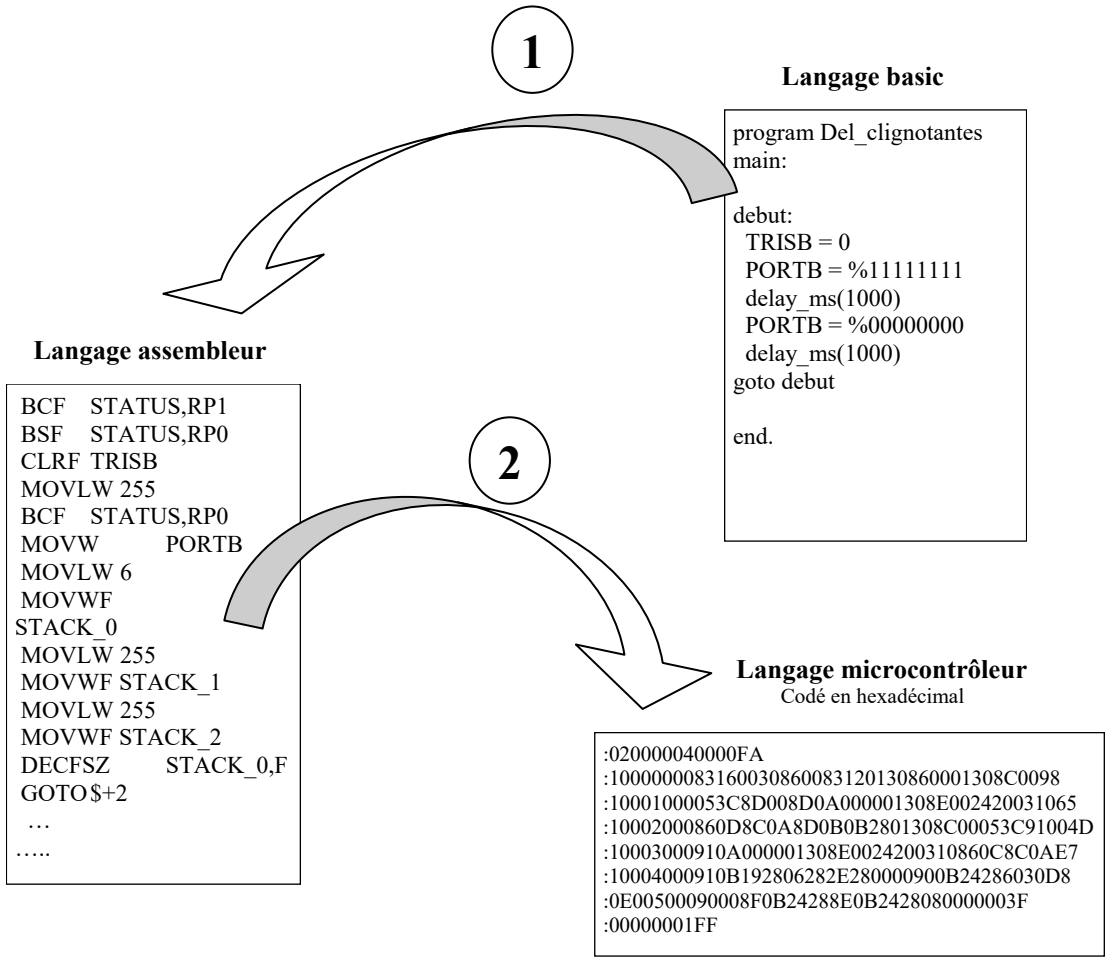
TABLE 16-2: PIC16F87/88 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode 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	lfff	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	lfff	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	

Vous l'aurez compris la programmation d'un PIC consistera donc simplement à écrire les bonnes instructions dans les bons emplacements mémoires !!
Pour un spécialiste les 35 instructions deviennent rapidement familières et l'écriture d'un programme en assembleur (utilisation des mnémoniques cf. tableau ci-dessus) devient habituel.



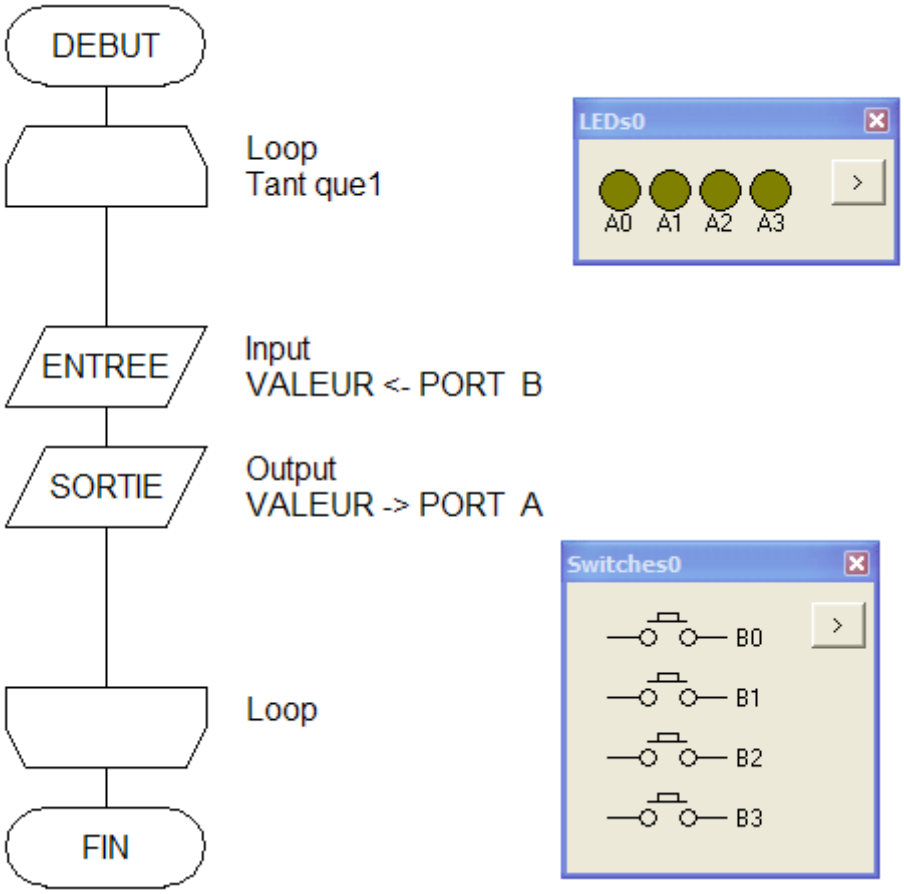
Toutefois il existe une alternative à la programmation directe dans le langage du microcontrôleur ou en assembleur celle d'utiliser un compilateur.
Il existe des logiciels en C, en Pascal ou en basic qui génèrent du code assembleur et donc du langage microcontrôleur.



L'autre alternative est d'utiliser la programmation par ***Algorigramme.***

LOGICIEL FLOWCODE:

1°) Création de l'algorigramme



2°) Traduction en langage C (transparent pour l'utilisateur)

```
//Definir pour microcontrôleur

//Fonctions PIC
#pragma CLOCK_FREQ 20000000
#define P16F84A
#include <system.h>
#define MX_EE
#define MX_EE_TYPE1
const char MX_EE_SIZE = 64;


//Déclarations de fonction Macro


//Déclarations de Variable
char FCV_VALEUR;

//Implémentations Macro
```

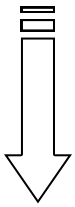
```
void main()
{
//Initialisation PIC>

//Code d'initialisation d'Interruption
option_reg = 0xC0;

        while( 1 )
        {
                TRISB = TRISB | 0xff;
                FCV_VALEUR = PORTB;

                TRISA = 0x00;
                PORTA = FCV_VALEUR;

        }
mainendloop: goto mainendloop;
```

3°) Conversion en langage Assembleur

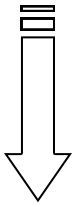
```
; This file was generated by C2C-plus compiler version 5.5e

        include "C:\PROGRA~1\MPLABI~1\MCHIP_~1
\P16F84A.inc"
        ;Variables ***************************************
_FCV_VALEUR                 equ 0x0c ;1
_indf                       equ 0x00 ;1
_tmr0                       equ 0x01 ;1
_pcl                        equ 0x02 ;1
_status                     equ 0x03 ;1
_fsr                        equ 0x04 ;1
_porta                      equ 0x05 ;1
_portb                      equ 0x06 ;1
_eedata                     equ 0x08 ;1
_eeadr                      equ 0x09 ;1
_pclath                     equ 0x0a ;1
_intcon                     equ 0x0b ;1
_option_reg                 equ 0x81 ;1
_trisa                      equ 0x85 ;1
_trisb                      equ 0x86 ;1
_eecon1                     equ 0x88 ;1
_eecon2                     equ 0x89 ;1
        ORG 0
        goto start__code

        ORG 4
_interrupt
        bcf INTCON, T0IF            ;clear interrupt flag
        retfie

start__code
_main__code
        movlw D'192'
        bsf STATUS, RP0
        movwf _option_reg
label_0000
        bsf STATUS, RP0
        movf TRISB, W
        iorlw D'255'
        movwf TRISB
        bcf STATUS, RP0
        movf PORTB, W
        movwf _FCV_VALEUR
        bsf STATUS, RP0
        clrf TRISA
        movf _FCV_VALEUR, W
        bcf STATUS, RP0
        movwf PORTA
        goto label_0000
label_0001
_mainendloop
        goto _mainendloop
_main__end
const_ptr_data
        END
```

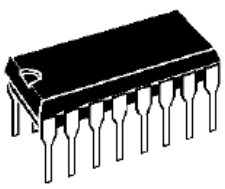
4°) Compilation en code hexadécimal

```
:020000000628D0
:080008000B110900C030831642
:1000100081008316060 8FF3886008
31206088C00CC
:0E002000831685010C08831285000
928162816
:00000001FF
```



5°) TRANSFERT vers le microcontrôleur.



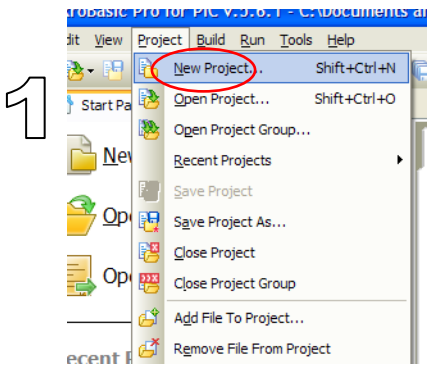


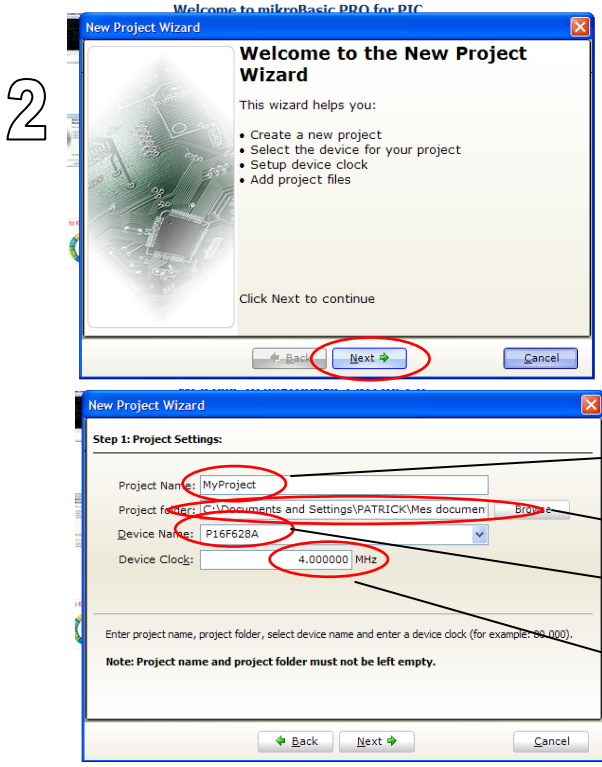
PROGRAMMER un microcontrôleur en MikroBasic Pro

Pour écrire un programme en Mikrobasic il faut au préalable suivre les 10 étapes décrites ci-dessous :

1°) Project ...New project

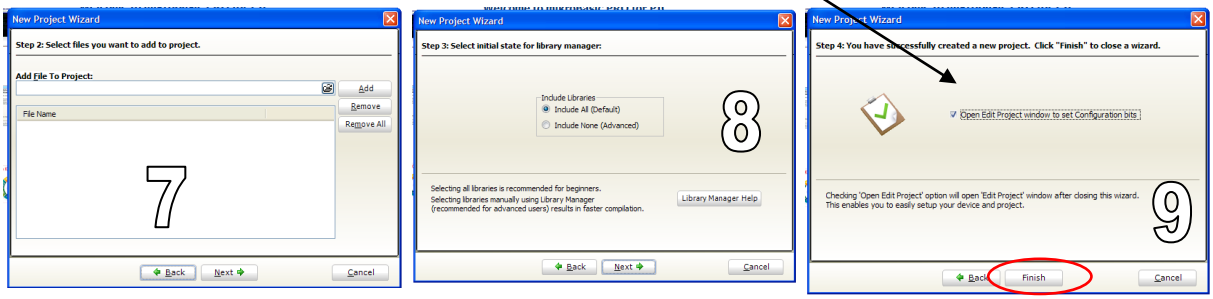
2°) Poursuivre avec l'assistant de nouveau projet



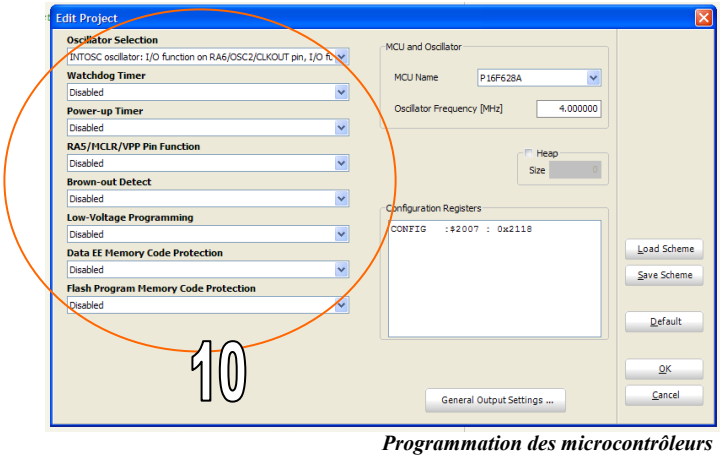


3°) Taper le nom du projet.
4°) Indiquer le chemin ou sera stocké le projet.
5°) Choisir le Microcontrôleur.
6°) Indiquer la vitesse d'horloge. 4Mhz en horloge interne.

Validez les dans 2 fenêtres suivantes (Next) puis ensuite dans la troisième cochez *« open edit Project window.... »* puis finish.



10°) Choisir les réglages définis ci contre. (*A adapter pour chaque microcontrôleur*)



```
                                    program Tete
                                    '
                                    symbol en1=portb.3
                                    symbol in1=portb.2
Attributs des                       symbol in2=portb.1
'alias' simplifiant                 symbol Relais=portB.4
la lecture du code                  symbol capteurAv=portA.1
                                    symbol capteurAr=portA.0
                                    symbol Led1=portA.2
                                    symbol Led2=portA.3
                                    symbol Radar=portB.0
Déclaration des
variables                           dim x,i as byte

                                    sub procedure initialisation
Sous programme                          TRISA=%00000011
d'initialisation                        TRISB  = 1
                                        PORTB  = 8
                                        cmcon=7
                                    end sub

                                    sub procedure avant
Sous programme                          en1=1
Marche avant                            in1=0
moteur                                  in2=1
                                    end sub

                                    sub procedure arriere
Sous programme                          in1=1
Marche arrière                          in2=0
moteur                              end sub

                                    sub procedure pause (dim t as byte)
Sous programme                          for i=1 to t
De pause avec une                        delay_ms(1000)
variable passée à                       next i
l'appel                                 Led1=0
                                        Led2=0
                                    end sub

                                    sub procedure arret
Sous programme                          in1=0
Marche arrêt                            in2=0
moteur                              end sub

                                    sub procedure Clignote
                                        For x = 1 to 10
Sous programme                          Led1=1
clignotement Led                        Led2=1
                                        delay_ms(500)
                                        Led1=0
                                        Led2=0
                                        delay_ms(500)
                                        NEXT x
                                    end sub
                                    '_____________Programme Principal

                                    main:
                                    initialisation

                                    debut:

                                    while radar=0
                                    Wend

                                    Avant

                                    While capteurAv=1
                                    Wend

                                    Arret

                                    Clignote

                                    pause (5)
                                    Arriere

                                    While capteurAr=1
                                    Wend

                                    Arret

                                    pause (5)

                                    GOTO DEBUT
                                    end.
```

Développement avec le logiciel MikroBasic

3.1 Les principales instructions

Les opérateurs arithmétiques élémentaires :

- + : addition
- - : soustraction
- * multiplication
- / : division
- div : exécute la division et restitue la partie entière
- mod : exécute la division et restitue le reste de la division

Les opérateurs logiques élémentaires :

- & : (AND)
- ! : (OR)
- ^ : (XOR)
- not : inverseur
- >> x : où x indique le nombre de décalages à droite successifs dans un mot binaire
- << x : où x indique le nombre de décalages à gauche successifs dans un mot binaire.

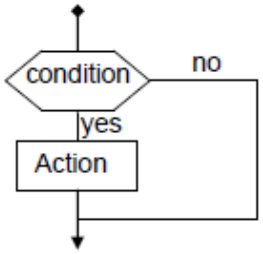
Les principales instructions :

Syntaxe	Commentaires	Exemple
program **…** **end.**	**Program** précise à la première ligne le nom du fichier en basic. La fin du programme est repérée par **end** avec un point	***program essai*** ***end.***
dim **as byte (integer,…)**	Permet de déclarer **les variables** utilisées dans le programme en précisant leur type : • byte : octet (8 bits) • integer : 16 bits • word : nom alphanumérique	**dim i, j, k as byte** **dim counter as word** **dim tab as longint[100]**
const	Déclare une **donnée constante** de type numérique ou caractère	**const MIN = 1000** **const SWITCH = "n"** **const vals as byte[12] = (31,12,17)**
symbol	Déclaration d'alias	**symbol t1s = delay_ms(1000)** **symbol led = PortB.3**
sub **…** **end sub**	Déclaration des **sous programmes** (procédures ou fonctions) pour une meilleure structure du programme. S'écrivent avant le programme principal	**sub procedure calcul** **n = a * (b +3)** **end sub**
main :	**Etiquette** de début de programme principal, toujours suivi de deux points	**main :**

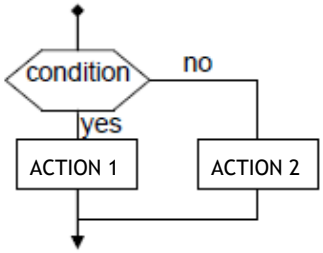
if ... then ...
(else) ...
end if
	Structure de contrôle pour réaliser un **test** à l'aide d'une expression booléenne. Exécute un traitement si condition vraie (ou éventuellement un autre si faux)	**if plus = 1 then**
		i = i+1
		else i=i*2
		end if

while ... **...** **wend**	Pour répéter un traitement tant qu'une condition est vraie. Rem : s'utilise aussi pour créer une boucle sans fin	**while i < 4** **i = i+1** **wend** **while true** **...** **wend**
Do **...** **Loop until**	Répeter une action jusqu'à ce que la condition soit vraie.	**Do** **i=i+1** **Loop until i=100**
Select case case 0 **.....** **case 3** **.....** **case else** **.....** **end select**	Suivant que la variable vaut 0 ou 3 ou autre alors faire	**Select case j** **case 0** **portB=%00001111** **case 3** **portB=%01111111** **case else** **portB=0** **end select**
for ... to ... **next**	Permet de réaliser une itération à l'aide d'une variable	**for i = 0 to 4** **portB = i** **next i**
delay_us (n) **delay_ms (m)**	Fonctions prêtes à l'emploi pour réaliser une temporisation de n microsecondes ou m millisecondes	**delay_ms (500)**
goto	Renvoi inconditionnel à une ligne de programme définie par une étiquette. L'étiquette est indiquée par son nom suivi de deux points (:) **A éviter autant que possible.**	**goto main**

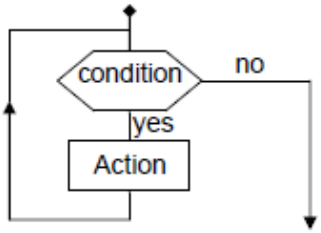
Iterations:

<table>
<tr>
<td> 
</td>
<td>

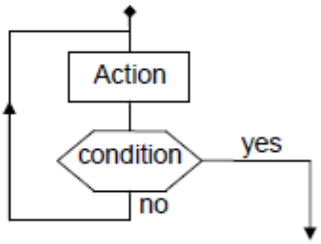
```
if (condition) then
   Action
End if
```

</td>
</tr>
<tr>
<td> 
</td>
<td>

```
if (condition) then
   Action 1
   else
   Action 2
End if
```

</td>
</tr>
<tr>
<td> 
</td>
<td>

```
while (condition)
   Action
wend
```

</td>
</tr>
<tr>
<td> 
</td>
<td>

```
do
   Action
Loop until (condition)
```

</td>
</tr>
</table>

Paramètres d'initialisation des microcontrôleurs

PIC 16F628

CMCON=7 'Désactive les comparateurs
PortA Utilisé en numérique
TRISA=0.à.255 'suivant utilisation*
TRISB=0.à.255 'suivant utilisation*

*chaque bit utilisé comme entrée prend la valeur 1 ou 0 comme sortie
EX: TRISA=%00001111 ou 15_{10} ou F_{16}

PIC 16F877

PIC 16F88

OSCCON=$7E 'horloge à 8MHz
ANSEL=0 (valeur à adapter *)
CMCON=7 'Désactive les comparateurs
'analogiques sur Port A'
TRISA=0.à.255 'suivant utilisation**
TRISB=0.à.255 'suivant utilisation**

*sélection des entrées utilisées en analogique
Si pas d'entrée analogique Ansel=0

**chaque bit utilisé comme entrée prend la valeur 1 ou 0 comme sortie
EX: TRISA=%00001111 ou 15_{10} ou F_{16}

PIC 12F675

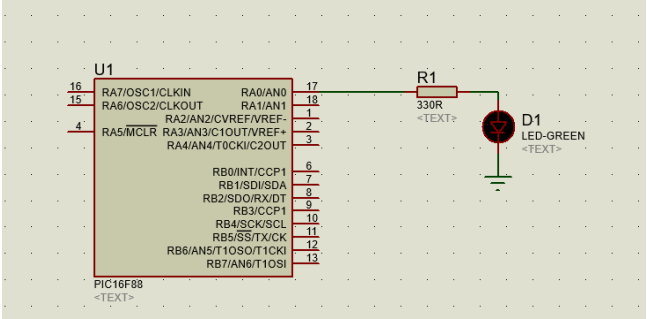
PIC 12F683

PIC 10F200

PIC 10F220

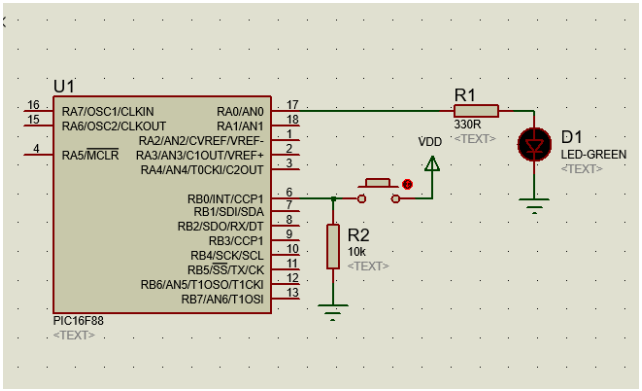
PROGRAMME

Matériel: Microcontrôleur PIC16F88, résistance 330Ω, LED.



Programme: La LED s'allume au démarrage du programme

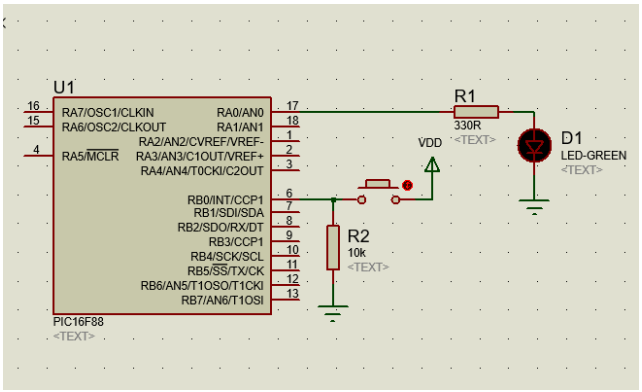
Matériel: Microcontrôleur PIC16F88, résistance 330Ω, LED, Bouton poussoir, résistance 10KΩ.



Programme:

1) La LED s'allume lorsque le bouton poussoir est enfoncé.
2) La LED s'allume lors d'une impulsion sur le bouton poussoir et s'éteint après 5 secondes.

Matériel: Microcontrôleur PIC16F88, résistance 330Ω, LED, Bouton poussoir, résistance 10KΩ.

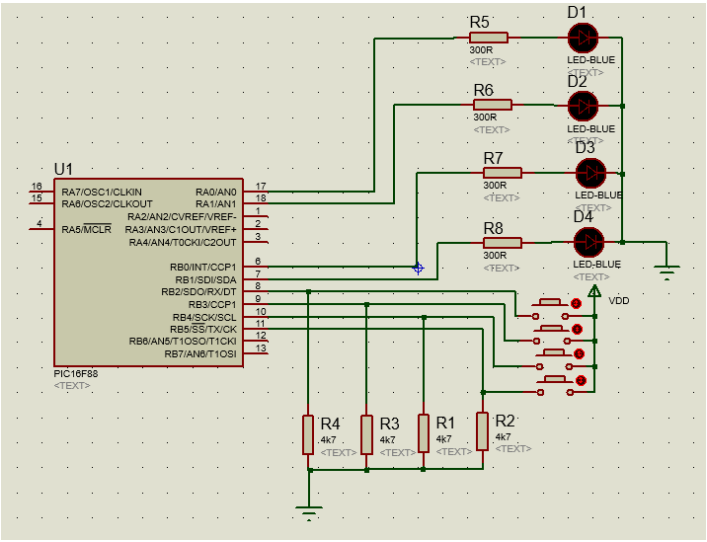


Programme:

1) La LED clignote 4 fois lors d'une impulsion sur le bouton poussoir .

PROGRAMME

Matériel: Microcontrôleur PIC16F88, 4 résistances 330Ω, 4 LED, 4 bouton poussoir (Button),4 résistances 4.7kΩ



Programme:

- Le bouton poussoir 1 allume la DEL 1.
- Le bouton poussoir 2 allume la DEL 1 et la DEL 2.
- Le bouton poussoir 3 allume la DEL 1, DEL2, DEL3.
- Le bouton poussoir 4 allume toutes les DELs.
- Le bouton 1 et 4 enclenchés simultanément éteignent les DELs.

Programme:

- Le bouton poussoir 1 permet de faire clignoter toutes les DELs simultanément.
- Le bouton poussoir 2 permet de réaliser un chenillard de la DEL 1 vers la DEL4.
- Le bouton poussoir 2 permet de réaliser un chenillard de la DEL 4 vers la DEL1.
- Le bouton poussoir 4 permet d'éteindre toutes les DELs.

Programme:

- Le bouton poussoir 1 permet de réaliser un chenillard de la DEL 1 vers la DEL4.
- Le bouton poussoir 2 permet d'accélérer le chenillard
- Le bouton poussoir 2 permet de ralentir le chenillard.
- Le bouton poussoir 4 permet d'éteindre toutes les DELs.