Introduction aux microcontrôleurs et codage de base des informations

Cours n°1
LI326

Généralités

Définition wikipedia

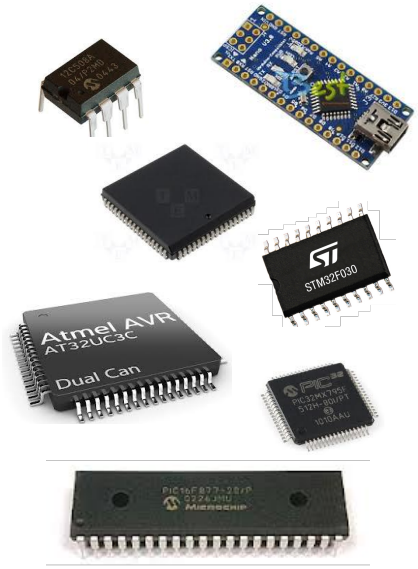
Un microcontrôleur
(en notation abrégée µc, ou uc ou encore MCU en Anglais)

est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties.

Les microcontrôleurs se caractérisent par

- un plus haut degré d'intégration,
- une plus faible consommation électrique,
- une vitesse de fonctionnement faible (de quelques mHz à 1GHz)
- et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels.

A quoi ressemble un µC ?



Il existe une infinité de modèles pour coller au plus près des applications embarquées.

Nombreux constructeurs

- Microchip
- Atmel
- Siemens/Infineon
- Intel
- Freescale
- STMicroelectronics
- Analog Devices
- Texas Instruments
- Cypress
- Philips
- ARM

Chacun dispose de plusieurs familles avec des dizaines de modèles

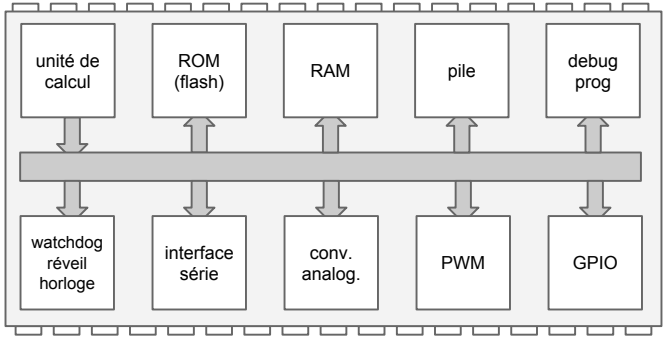
A quoi sert un µC ?

- Les micro-contrôleurs sont partout :
  - électroménager (machine à laver, four, …)
  - multimédia (radio, HP, …)
  - automobile (moteur, habitacle, …)
  - équipemement informatique (modem, souris, imprimante, …)
  - industrie (robot, automate, …)
  - bref partout où il y a un besoin "d'intelligence"

- Les alternatives sont :
  - Logique cablée avec des portes discrètes
  - Logique cablée dans un circuit reprogrammable (FPGA)
  - Circuiterie analogique ou électromécanique

La tendance est au tout programmable
et dans un avenir proche au tout connecté !

Que contient un µC ?

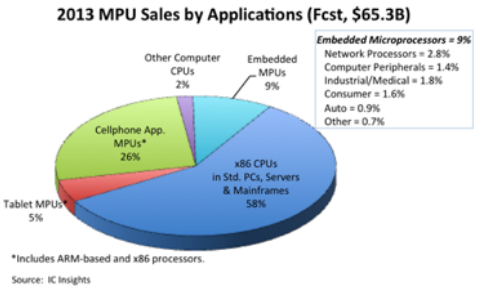


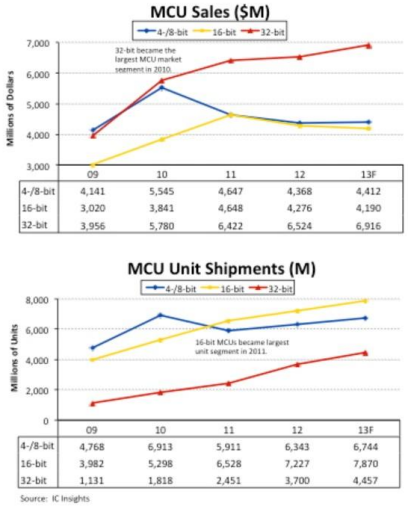
La dimension et le nombre des composants varient de manière continue entre le microcontrôleur 4 bits et le système intégré multicoeurs utilisés dans les équipements nécessitant beaucoup de calcul et d'entrée sortie (équipement réseau par exemple)

Marché : quelques données

La part des microcontrôleurs 32 bits devient prédominante en raison principalement de l'usage en informatique mobile.

Mais attention, en nombre de pièces donc d'usage c'est encore les 4/8/16 qui dominent.





source IC Insights

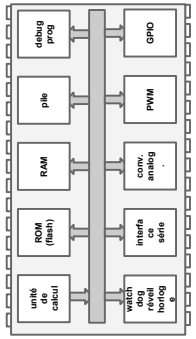
Que connecte-on à un µC ?

- Entrées capteurs

  lumière, micro, hygromètre, mouvement, détecteur de gaz, proximité, torsion, pression, thermomètre, accéléromètre, GPS, ...

- IHM

  clavier, télécommande, surface tactile, voix, ...



- Sorties actionneurs

  moteurs CC ou PàP, relais, HP, leds, ….

- IHM

  écran LCD, buzzer, voix, ...

- Il existe une très grande variété de capteur et d'actionneurs mais le nombre d'interface est plus réduit en passant par des protocoles de communication standard : rs232, CAN, I2C, USB, …
- Les capteurs et actionneurs contiennent souvent des µC pour simplifier le travail de l'application ou réduire la quantité d'information à échanger

Quel langage utilise-t-on pour les µC ?

Les langages de programmation dépendent de la taille et de l'usage. Lorsque les systèmes sont petits,

- la taille des programmes est importante car elle influence le prix,
- la performance influence la consommation donc l'autonomie.
  - Assembleur de moins en moins.
  - C ou C-like.
  - Basic.
  - Langages ad-hoc dérivés des langages d'automate.

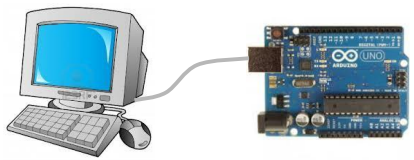
Pour les systèmes 32bits, on trouve aussi des langages interprétés

- Java
- Lua

Module LI326

Comment programme-t-on un µC ?

Le programme est écrit, compilé, mis-au-point sur une machine de développement puis chargé dans le micro-contrôleur.



Le programme peut être exécuté

- sur un simulateur (ex: gpsim)
- sur le matériel avec la possibilité d'interrompre l'exécution sur une condition (événement, état d'un registre, …) et de lire l'état de la mémoire depuis la machine de programmation (ex: jtag, icd)

Objectif du module

- Comprendre les principes de bases de la conception autour d'un microcontrôleur.
- Les concepts sont vus sur le µC PIC16F877 de Microchip parce qu'il est complet (mais ce n'est pas le plus simple).
- La programmation se fait en assembleur pour retirer toutes les abstractions utilisées par les programmeurs.
- Les outils de conception sont sur Linux.
- Les périphériques seront programmés au signal près.

11 séances

1	Introduction aux microcontroleurs	TD calcul, structures de données
2	Bases de l'assembleur PIC et environnement programmation	Premiers programmes de calcul
3	Programmation en assembleur PIC : directives, macro-instructions, boucles, switch-case.	Programmes évolués sans interruption: boucles, macro-instructions
4	Entrées-sorties numériques simples et bibliothèques de fonctions : leds et bouton poussoir idéal.	Leds + bouton poussoir, principes fonctions simples
5	Gestion des évènements et programmation par composants: interruptions, timer, reset, watchdog, sleep, automates.	Leds + bouton poussoir réel automates
6	Communication série RS232 : bootloader et terminal de commandes.	terminal pour contrôler le clignotement des leds
7	Périphériques basiques : clavier matriciel et afficheur LCD.	Digicode programmable
8	Périphériques sur 1 fil : clavier, commande PWM et échange de données.	clavier sur 1 fils, sortie PWM pour régler la luminosité d'une led
9	Conversion analogique numérique	Serrure analogique programmation du PWM
10	bus I2C : commande de périphériques standardisés.	Télémètre à ultra-son
11	Compteurs et protocole IR	Serrure à télécommande

PIC16F877

Principes de fonctionnement du module

Le but est que vous compreniez tout :

- Les 5 premières séances se feront sur un simulateur
- Les 6 suivantes sur des vrais composants

Toutes les séances sont notées :

- La présence 20%
- L'investissement et les résultats 20%

L'examen compte pour 60%

Microchip PIC16F877

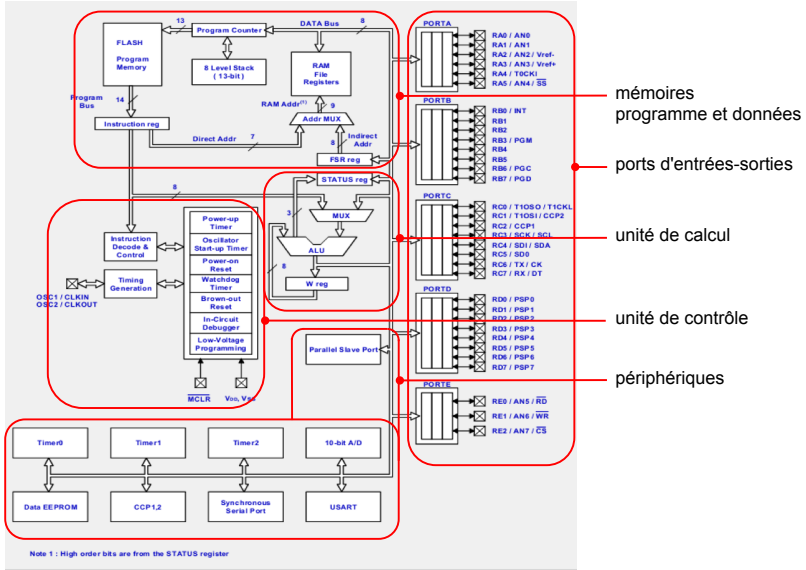
Fonctionne **à 20 MHz** maximum.

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

Caractérisitiques :

- 35 instructions (composant RISC),
- 8Kinstructions 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 interne



Les différentes mémoire

Le PIC16F877 dispose de plusieurs mémoires avec des usages spécifiques :

- Une mémoire flash pour le programme, mais qui peut contenir des données statiques qui peuvent être modifiée à l'exécution.
- Une mémoire pour les données pour le calcul.
  Ce sont plutôt des registres. Ils sont volatiles.
- Une mémoire eeprom pour les données non-volatiles.
- Une pile pour les adresses de retour des fonctions et des routines d'interuption.
- Des registres de contrôle des périphériques.

Quatre espaces d'adressage:

- Mémoire des registres accédée en direct avec les instructions
- Mémoire de programme accédée en lecture/écriture indirectement par des registres et des séquences de programmes précises
- Mémoires eeprom accédée indirectement par les registres
- Mémoire de la pile non accessible.

Les autres membres de la famille

Microchip propose des centaines de modèles répartis dans 9 familles

8 bits

- PIC10/PIC12 très petits
- PIC16 milieu de gamme
- PIC17/PIC18 haut de gamme (75 inst, usb, ethernet, C)

16 bits

- PIC24 compatible avec GCC
- dsPIC30, dsPIC33 pour le traitement du signal

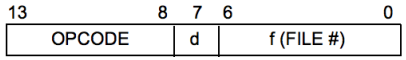
32 bits

- PIC32 mips4k

Calcul avec le PIC

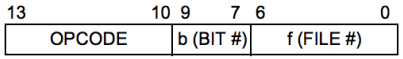
Format des instructions

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

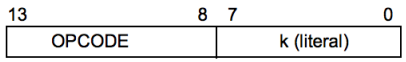
Bit-oriented file register operations



b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General



k = 8-bit immediate value

Les instructions font 14 bits

- 7 bits pour l'adresse du registre opérande
- 1 bit pour l'adresse du registre destination

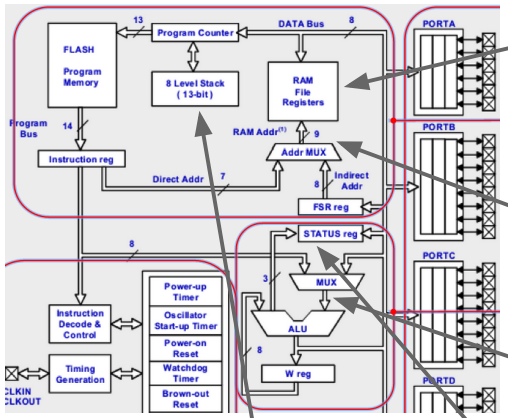
Un type d'instructions "*bit*"

- 3 bits pour désigner le numéro du bit

Un type d'instruction "*literal*"

- 8 bits pour la valeur

Architecture interne : ALU



- La mémoire de travail du PIC16F877 est constituée de registres
- Il y a 512 adresses mais seulement 368 registres différents

- 2 modes d'adressages
  - direct (dans l'instruction)
  - indirect (par un registre unique : FSR
- 9 bits = 512 adresses

- 2 opérandes
  1. registre ou valeur
  2. registre W
- 1 résultat : W ou registre

- La pile n'est pas adressable

- Un registre de status va recevoir les drapeaux de calcul

Format des instructions

- Les instructions n'ont qu'une seule opérande
  - En principe : resultat **=** opérande **opération** opérande
    par exemple : Reg1 = Reg2 + Reg3
  - En PIC : W = Reg3 + W
    W = Reg2 + W
    Reg1 = W
- L'état de l'opération est stocké dans le registre status
  - Z = 1 si le résultat est NUL
  - C = 1 valeur de la retenue sortante (bit n°8)
  - DC = 1 valeur de la retenue du bit n°3

Registre status

STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

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	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, 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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
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: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's 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.

Exemples d'instruction

ADDLW **Add Literal and W**

Syntax:	[*label*] ADDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) + k → (W)
Status Affected:	C, DC, Z
Encoding:	11 | 111x | kkkk | kkkk
Description:	The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.
Words:	1
Cycles:	1

- addlw 46
  - W = W + 46
- addlw 34'
  - W = W + 0x34

ADDWF **Add W and f**

Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) + (f) → (destination)
Status Affected:	C, DC, Z
Encoding:	00 | 0111 | dfff | ffff
Description:	Add the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

- addwf R1,W
  - W = R1 + W
- addwf R1,F
  - R1 = R1 + W

Le jeu d'instructions

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	

Seulement
35 instructions

mais ce n'est
pas un gage de
simplicité pour le
programmeur

Pour des instructions complètes

Pour effectuer : Reg1 = Reg2 + Reg3

Il faut une séquence d'instructions :

```
movf   R3,W
addwf  R2,W
movwf  R1
```

Exemples :

R1 = R1 - R2 :

R1 = L :

R1 = R1 - L :

R1 = L - R1 :

Pour des opérations sur 16 bits (1)

- Un registre de 16 bits, c'est 2 registres de 8bits consécutifs
  - si R1 est une adresse de registre
    alors R1+1 est l'adresse du registre suivant

- On choisit la stratégie little endian
  - R1 contient les bits de poids faibles
  - R1+1 contient les bits de poids forts

- illustration sur l'incrément
  - R1 = R1 + 1

```
incf    R1,F
```

  - $R1_{16} = R1_{16} + 1$

```
incf    R1,F
btfsc   STATUS,Z
incf    R1+1
```

Pour des opérations sur 16 bits (2)

- illustration sur une addition
  - R1 = R2 + L

```
movlw   L
addwf   R2,W
movwf   R1
```

  - $R1_{16} = R2_{16} + L_{16}$

```
movlw   HIGH (L16)
addwf   R2+1,W
movwf   R1+1
movlw   LOW (L16)
addwf   R2,W
movwf   R1
btfsc   STATUS,C
incf    R1+1,F
```

Premier TD

Le but de la première séance sera d'appendre à faire des calculs avec le PIC par une série de petits exercices utilisant les instructions de base.