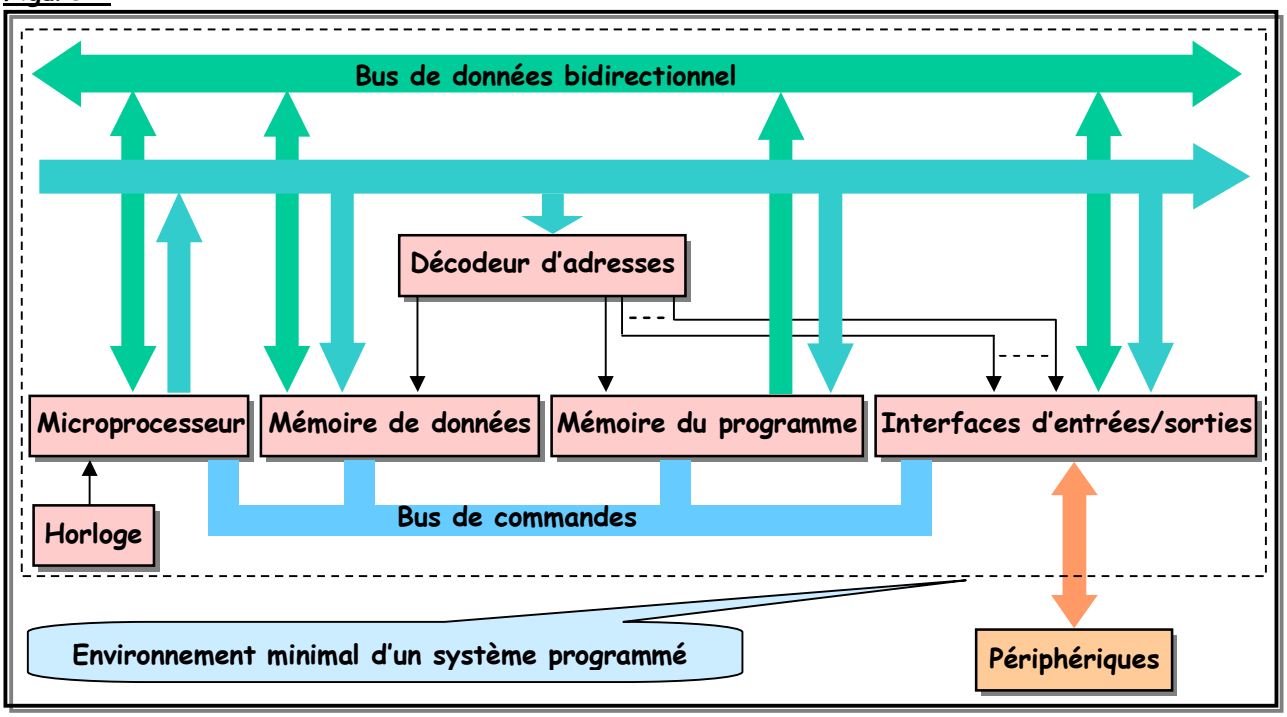
1. Introduction

Le contrôle d'un processus industriel fait appel à un dispositif de gestion et de traitement des informations. IL est souvent réalisé par un système électronique programmé.

2. Architecture d'un système programmé

La structure matérielle d'un système programmé est essentiellement organisée autour d'un microprocesseur. Cependant, ce dernier ne peut fonctionner sans un environnement minimal.
Une telle structure est représentée à la figure 1.

Figure 1



21. Bus de données, d'adresses et de commandes

211. Notion de bus

Un bus est un ensemble de fils destiné à la communication entre les différents composants du système minimal. On distingue 3 groupes de bus : bus de données, bus d'adresses et bus de commandes.

212. Bus de données : Data bus

C'est un bus bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement et inversement. Son nombre de lignes est égal à la capacité de traitement du µP : 8, 16, 32 ou 64 bits.

213. Bus d'adresses : Adress bus

C'est un bus unidirectionnel qui permet la sélection du destinataire des informations à traiter dans un espace mémoire qui peut avoir 2^n emplacements où **n** est le nombre de conducteurs du bus d'adresses.

314. Bus de commandes : Control bus

C'est un bus constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur le bus de données et d'adresse.

Exemple : Le µP 6809 est doté d'un :
☑ Bus de données de 8 bits de D_0 à D_7.
☑ Bus d'adresses de 16 bits de A_0 à A_{15}.
☑ Bus de commandes de 10 bits : On cite à titre d'exemple les broches Read/$\overline{\text{Write}}$, Halt et Reset.

22. Microprocesseur : CPU

C'est l'unité centrale de traitement (**UCT** ou **CPU**). Il exécute séquentiellement le programme (ensemble d'instructions) situé en mémoire **ROM**. L'exécution d'une instruction est cadencée par une horloge.

23. Mémoires

231. Mémoire du programme : mémoire morte

C'est une mémoire permanente et à lecture seule. Elle est destinée à stocker le programme.
Il existe différents types, à savoir : **ROM**, **PROM**, **EPROM** ou **UV EPROM**, **EEPROM** et **Flash EPROM**.

232. Mémoire de données : mémoire vive

C'est une mémoire volatile et à lecture/écriture. Elle est chargée de sauvegarder temporairement les données du programme (variables d'entrées, variables intermédiaires, variables de sorties, etc…).
Il s'agit de la **RAM**.

24. Interfaces ou ports d'entrées/sorties

L'interfaçage est un ensemble matériel permettant d'assurer la liaison entre le µP et les périphériques. On distingue les interfaces parallèles et les interfaces séries.

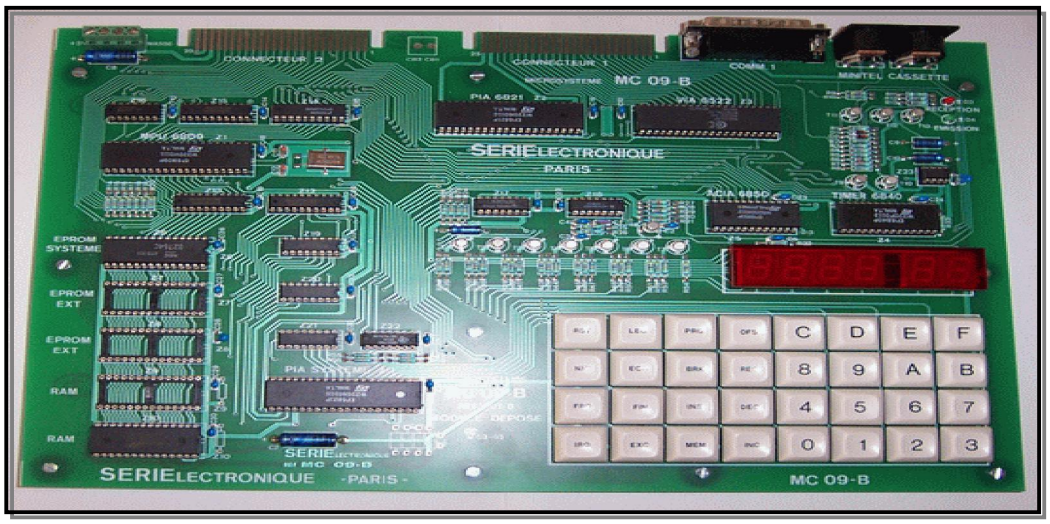
25. Périphériques

C'est un ensemble matériel extérieur au système programmé. On cite à titre d'exemple les interfaces homme/machine (clavier, afficheurs, interrupteurs, boutons poussoirs, etc…), capteurs et actionneurs.

26. Décodeur d'adresses

Le décodeur d'adresses est chargé d'aiguiller les données présentes sur le bus de données. En effet, le µP peut communiquer avec les différents composants de son environnement. Ceux-ci sont tous reliés sur le même bus de données et afin d'éviter des conflits, un seul composant doit être sélectionné à la fois. Ainsi, à une adresse présentée par le µP, devra répondre un seul composant du système (RAM, ROM, Interface).
🔔 **Remarque** : Lorsqu'un composant n'est pas sélectionné, il est mis en état haute impédance.

3. Kit dadactique d'un système programmé à base du microprocesseur 6809



Fonction Traiter

1. Introduction

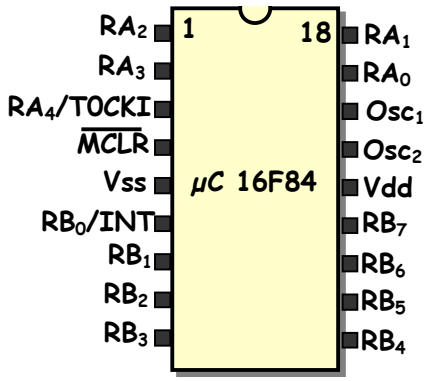
Le **PIC** (Programmable **I**nterface **C**ontroler) **16F84** est un microcontrôleur produit par la société **MicroChip**. C'est un composant qui regroupe dans un même boîtier tous les éléments vitaux d'un système programmé : CPU, RAM, ROM, Interfaces d'entrées/sorties, etc.

2. Architecture du PIC 16F84

21. Organisation externe

Le µC 16F84 est un circuit intégré de 18 broches : voir figure 1.

Figure 1



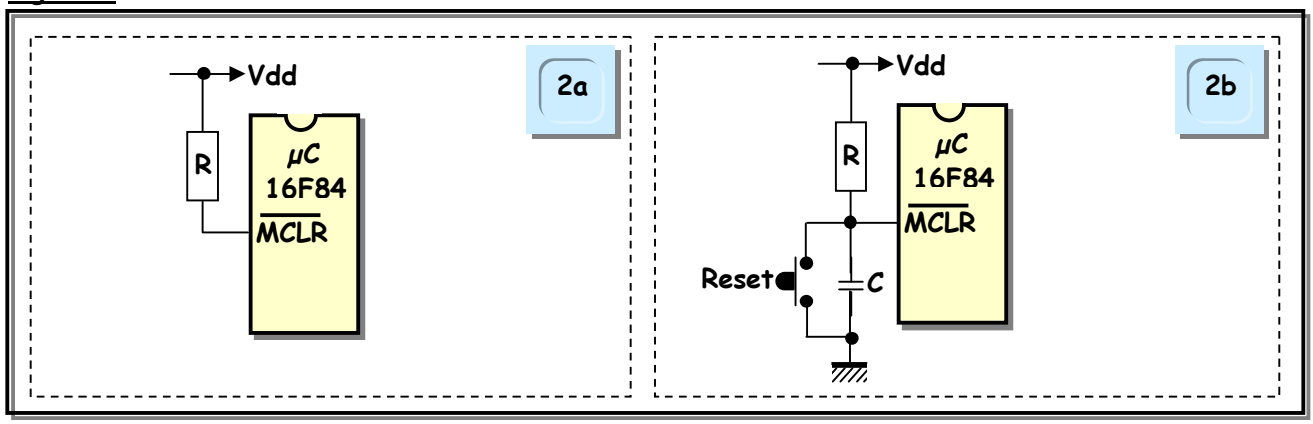
- **Vss** et **Vdd** : broches d'alimentation.
- **Osc_1** et **Osc_2** : broches du circuit horloge.
- **$\overline{MCLR}$** : Reset (Master Clear).
- **RA_0..RA_4** : 5 entrées/sorties du port A.
- **RB_0..RB_7** : 8 entrées/sorties du port B.
- **TOCKI** : entrée d'horloge externe du timer.
- **INT** : entrée d'interruption externe.

La tension nominale de l'alimentation du µC 16F84 est +5 V : Vdd=+5 V et Vss=0 V.

Le **Reset** du µC 16F84 peut avoir plusieurs causes :

☑ Une mise sous tension **POR** (**P**ower **O**n **R**eset) : voir figure 2a
☑ Une mise à 0 de la broche **$\overline{MCLR}$** (Reset manuelle) : voir figure 2b.
☑ Un débordement du timer du chien de garde **WDT** : voir plus tard.

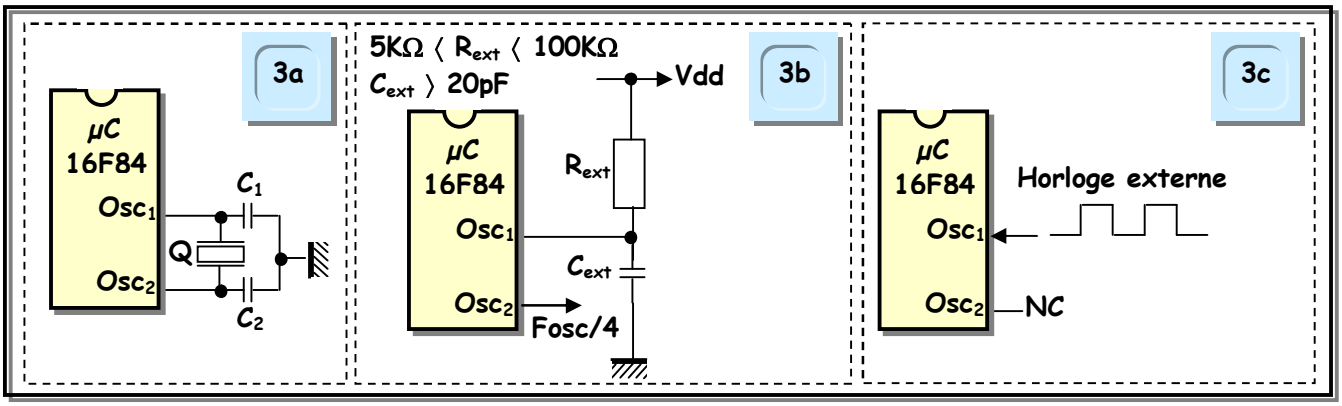
Figure 2



Le fonctionnement du µC 16F84 nécessite une horloge qui rythme l'exécution des instructions du programme. On distingue trois modes d'horloge :

☑ **Horloge interne à quartz** (Figure 3a) : Avec l'oscillateur à quartz, on peut avoir des fréquences allant de 4 MHz jusqu'à 20 MHz selon le type du µC.
☑ **Horloge interne à circuit RC** (Figure 3b) : Avec un oscillateur à circuit RC, la fréquence de l'oscillation dépend de la tension Vdd et des éléments R_{ext} et C_{ext}.
☑ **Horloge externe** (Figure 3c) : Application d'un signal horloge externe.

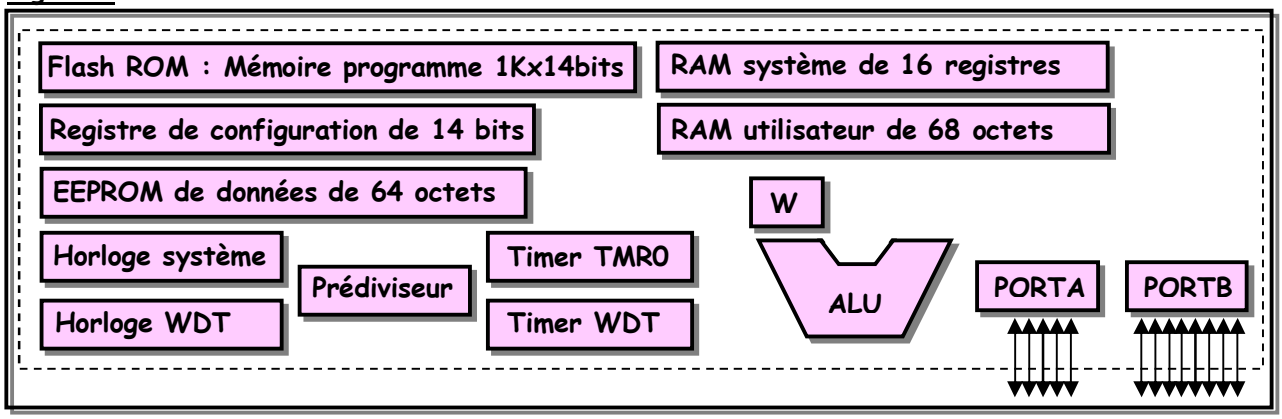
Figure 3



22. Organisation interne

La structure interne simplifiée du µC 16F84 peut être donnée par la figure 4.

Figure 4



221. Horloge système

L'horloge système ou horloge instruction, est la base du temps interne qui cadence le fonctionnement du µC. Quelque soit l'oscillateur utilisé, l'horloge système s'obtient en divisant la fréquence de l'oscillateur par **4**.
Exemple : Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1 µs.

222. Mémoire Flash ROM

Cette mémoire de **1024** mots de **14** bits stocke le programme. Elle est permanente et reprogrammable à souhait. Chaque position contient une instruction. Suite à un Reset ou lors d'une mise sous tension, le µC commence l'exécution du programme à l'adresse **0**. De plus, lorsqu'il y a une interruption, il va à l'adresse **4**.

223. Registre de configuration

Pendant la phase de la programmation du µC, on programme aussi un registre de configuration logé dans la mémoire EEPROM. Ce registre est un mot de **14** bits qui permet de :
☑ Choisir le type de l'oscillateur pour l'horloge.
☑ Valider ou non le timer du watchdog WDT.
☑ Autoriser ou non une temporisation à la mise sous tension.
☑ Interdire ou non la lecture des mémoires de programme et de données.

Registre de configuration ⇒

CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	$FOSC_1$	$FOSC_0$

☑ Bits **$FOSC_0$** et **$FOSC_1$** : Sélection du type d'oscillateur pour l'horloge.
$FOSC_1FOSC_0$=11 ⇒ Oscillateur à circuit RC jusqu'à 4 MHz.
$FOSC_1FOSC_0$=10 ⇒ Oscillateur HS, quartz haute fréquence, jusqu'à 20 MHz.
$FOSC_1FOSC_0$=01 ⇒ Oscillateur XT, quartz standard jusqu'à 4 MHz.
$FOSC_1FOSC_0$=00 ⇒ Oscillateur LP, quartz basse fréquence, jusqu'à 200 KHz.

☑ Bit **WDTE** : Validation du timer du watchdog WDT.
WDTE=1 ⇒ WDT validé et WDTE=0 ⇒ WDT inhibé.

☑ Bit **PWRTE** : Validation d'une temporisation à la mise sous tension.
Le µC possède un timer permettant de retarder de 72 ms le lancement du programme après la mise sous tension. Ce délai maintient le µC à l'arrêt et permet ainsi à la tension d'alimentation de bien se stabiliser.
PWRTE=1 ⇒ le µC démarre tout de suite et PWRTE=0 ⇒ le µC attend 72 ms.

☑ Bits **CP** : Protection en lecture des mémoires de programme et de données.
CP=1 ⇒ pas de protection et CP=0 ⇒ protection activée.

Exemple : On désire configurer le registre pour répondre aux critères suivants : oscillateur à quartz de 4 MHz, le timer du watchdog n'est pas autorisé, une attente de 72 ms est souhaitée et le µC n'est pas protégé en lecture.

Configuration requise ⇒

1	1	1	1	1	1	1	1	1	1	0	0	0	1

⇔ **3FF1** en hexadécimal

224. Mémoire EEPROM de données

Cette mémoire est constituée de **64** octets que l'on peut lire et écrire depuis un programme. Ces octets sont conservés même après une coupure de l'alimentation et sont très utiles pour conserver des paramètres semi permanents : code d'accès, version du programme, message d'accueil, valeur invariable, etc.

225. Mémoire RAM

La mémoire RAM est constituée de deux parties : voir figure 5.

☑ Une zone mémoire de **68** octets réservée à l'utilisateur pour stocker ses variables. Elle adressable de l'@ 0x**0C** à l'@ 0x**4F** ou de l'@ 0x**8C** 0x**CF**.
☑ Un fichier de **16** registres de contrôle répartis en **2** banques :
➢ Une banque **0** adressable de l'@ 0x**00** à l'@ 0x**0B**.
➢ Une banque **1** adressable de l'@ 0x**80** à l'@ 0x**8B**.
L'accès à un registre s'effectue en se plaçant dans la banque où il se trouve. Ceci est réalisé en positionnant le bit **RP0** du registre d'état **STATUS** :

Registre STATUS ⇒

		RP0	TO	PD	Z	DC	C

RP0=0 ⇒ banque 0 et RP0=1 ⇒ banque 1.

Figure 5

@	Banque 0	Banque 1	@
00	INDF	INDF	80
01	TMR0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C ⋮ 4F	Mémoire utilisateur	Idem	8C ⋮ CF

Fonction Traiter

226. ALU et le registre W

Le registre W, qui n'a pas d'adresse, est un registre de travail de **8** bits.
L'ALU est une unité arithmétique et logique de **8** bits qui réalise les opérations entre **W** et n'importe quel autre registre **f** ou constante **k**. Le résultat de l'opération peut être placé soit dans W soit dans f.
L'ALU est associée au registre d'état **STATUS** par les bits **Z**, **C** et **DC** :

Registre STATUS ⇒

		RP0	TO	PD	Z	DC	C

☑ **C** : Ce bit flag passe à 1 lorsqu'il y a une retenue sur un octet.
☑ **DC** : Ce bit flag passe à 1 lorsqu'il y a une retenue sur un quartet.
☑ **Z** : Ce bit flag passe à 1 quand le résultat d'une opération est nul.

227. Port d'entrées/sorties PORTA

Le port A désigné par **PORTA** est un port bidirectionnel de **5 bits** (RA_0 à RA_4). La configuration de direction pour chaque bit du port est déterminée avec le registre **TRISA** :
☑ Bit i de TRISA=0 ⇒ bit i du PORTA est configuré en sortie.
☑ Bit i de TRISA=1 ⇒ bit i du PORTA est configuré en entrée.
Les broches RA_0 à RA_3 sont des entrées/sorties compatibles TTL alors que la broche RA_4 peut être utilisée soit comme entrée/sortie normale du port A, soit comme entrée horloge externe pour le Timer TMR0. Le choix se fait à l'aide du bit **TOCS** du registre **OPTION** :
☑ TOCS=0 ⇒ RA_4 est une entrée/sortie normale.
☑ TOCS=1 ⇒ RA_4 est une entrée horloge externe pour le timerTMR0.

De plus, RA_4 est une sortie à drain ouvert. Donc, il ne faut pas oublier de mettre une résistance externe vers Vdd.
Chaque broche du port A configurée en sortie peut fournir un courant de 20 mA au maximum, mais tout le port A configuré en sortie ne peut pas débiter un courant total supérieur à 50 mA.
Chaque broche du port A configurée en entrée peut accepter un courant de 25 mA au maximum, mais tout le port A configuré en entrée ne peut pas accepter un courant total supérieur à 80 mA.

228. Port d'entrées/sorties PORTB

Le port B désigné par **PORTB** est un port bidirectionnel de **8 bits** (RB_0 à RB_7). La configuration de direction se fait à l'aide du registre **TRISB** (identique à celle du PORTA : voir TRISA). Toutes les broches sont compatibles TTL.
Les entrées du port B peuvent être pourvues de résistances de rappel à Vdd. Cette option est validée par le bit **RBPU** du registre **OPTION** :

Registre OPTION ⇒

RBPU	INTEDG	TOCS	TOSE	PSA	PS_2	PS_1	PS_0

RBPU=0 ⇒ les broches configurées en entrées sont connectées à des résistances de rappel à Vdd et les broches configurées en sorties sont dépourvues automatiquement de ces résistances de rappel.
En entrée, la broche RB_0 appelée aussi INT peut déclencher l'interruption externe INT et une quelconque des broches RB_4 à RB_7 peut déclencher l'interruption RBI.

Chaque broche du port B configurée en sortie peut fournir un courant de 20 mA au maximum, mais tout le port B configuré en sortie ne peut pas débiter un courant total supérieur à 100 mA.
Chaque broche du port B configurée en entrée peut accepter un courant de 25 mA au maximum, mais tout le port B configuré en entrée ne peut pas accepter un courant total supérieur à 150 mA

🔔 **Remarque** : Au Reset du µC, les ports PORTA et PORTB sont configurés en entrée.

Fonction Traiter

1. Introduction

Le µC 16F84 possède **un jeu de 35 instructions**. Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (OC) ainsi que l'opérande. A part les instructions de saut, toutes les instructions sont exécutées en un cycle d'horloge. Sachant que l'horloge fournie au µC est prédivisée par 4, si on utilise par exemple un quartz de 4 MHz, on obtient donc 1000000 cycles/seconde, cela nous donne une puissance de l'ordre de 1 **MIPS** (1 **M**illion d'**I**nstructions **P**ar **S**econde). Avec un quartz de 20 MHz, on obtient une vitesse de traitement d'autant plus rapide.

2. Organisation des instructions

21. Instructions « orientées octets » (adressage direct) : Figure 1

Ce sont des instructions qui manipulent les données sous forme d'octets. Elles sont codées de la manière suivante :

☑ 6 bits pour l'instruction : c'est logique, car comme il y a 35 instructions, il faut 6 bits pour pouvoir les coder toutes.
☑ 1 bit (d) pour indiquer si le résultat obtenu doit être conservé dans le registre de travail (accumulateur) W de l'unité de calcul (W pour Work) ou sauvé dans un registre f (f pour file).
☑ Reste 7 bits pour encoder l'adresse de l'opérande, mais 7 bits ne donnent pas accès à la mémoire RAM totale, c'est ainsi qu'on on utilise le bit RP0 du registre STATUS pour compléter le 8ème bit.

🕭 **Remarque** : (W, f ?d) signifie que le résultat est stocké soit dans W si d=0, soit dans f si d=1.

Figure 1

Instructions opérant sur un registre (adressage direct)		indicateurs	Cycles
ADDWF f,d	W + f → (W, f ?d)	C, DC, Z	1
ANDWF f,d	W And f → (W, f ?d)	Z	1
CLRF f	Effacer f	Z	1
CLRW	Effacer W	Z	1
CLRWDT	Effacer watchdog timer	TO, PD	1
COMF f,d	Complémenter f → (W, f ?d)	Z	1
DECF f,d	Décrémenter f → (W, f ?d)	Z	1
DECFSZ f,d	Décrémenter f → (W, f ?d), sauter si 0		1(2)
INCF f,d	Incrémenter f → (W, f ?d)	Z	1
INCFSZ f,d	Incrémenter f → (W, f ?d), sauter si 0		1(2)
IORWF f,d	W Or f → (W, f ?d)	Z	1
MOVF f,d	f → (W, f ?d)	Z	1
MOVWF f	W → f		1
RLF f,d	Rotation à gauche de f à travers C → (W, f ?d)	C	1
RRF f,d	Rotation à droite de f à travers C → (W, f ?d)	C	1
SUBWF f,d	f - W → (W, f ?d)	C, DC, Z	1
SWAPF f,d	Permuter les deux quartets de f → (W, f ?d)		1
XORWF f,d	W Xor f → (W, f ?d)	Z	1

22. Instructions « orientées bits » : Figure 2

Ce sont des instructions destinées à manipuler directement les bits d'un registre d'une case mémoire.
Elles sont codées de la manière suivante :

☑ 4 bits pour l'instruction.
☑ 3 bits pour indiquer le numéro du bit à manipuler (de 0 à 7).
☑ 7 bits pour indiquer l'opérande.

Figure 2

Instructions opérant sur un bit d'un registre		indicateurs	Cycles
BCF f,b	Mettre à 0 le bit b du registre f		1
BSF f,b	Mettre à 1 le bit b du registre f		1
BTFSC f,b	Tester le bit b de f, sauter une instruction si 0		1(2)
BTFSS f,b	Tester le bit b de f, sauter une instruction si 1		1(2)

23. Instructions opérant sur une donnée (adressage immédiat) : Figure 3

Ce sont des instructions qui manipulent des données qui sont codées dans l'instruction directement.
Elles sont codées de la manière suivante :

☑ L'instruction est codée sur 6 bits.
☑ Elle est suivie d'une valeur immédiate codée sur 8 bits (donc de 0 à 255).

Figure 3

Instructions opérant sur une donnée (adressage immédiat)		indicateurs	Cycles
ADDLW k	W + k → W	C, DC, Z	1
ANDLW k	W And k → W	Z	1
IORLW k	W Or k → W	Z	1
MOVLW k	k → W		1
SUBLW k	k - W → W	C, DC, Z	1
XORLW k	W Xor k → W	Z	1

24. Instructions de saut et appel de procédures : Figure 4

Ce sont des instructions qui provoquent une rupture dans la séquence de déroulement du programme.
Elles sont codées de la manière suivante :

☑ Les instructions sont codées sur 3 bits.
☑ La destination est codée sur 11 bits.

Figure 4

Instructions générales		indicateurs	Cycles
CALL L	Branchement à un sous programme de label L		2
GOTO L	Branchement à la ligne de label L		2
NOP	Pas d'opération		1
RETURN	Retourner d'un sous programme		2
RETFIE	Retourner d'une procédure d'interruption		2
RETLW k	Retourner d'un sous programme avec k dans W		2
SLEEP	Mettre le µC en mode de veille	TO, PD	1

1. Introduction

Les étapes nécessaires permettant de voir un programme s'exécuter sur un µC 16F84 sont :

1. Ecrire un programme en langage assembleur dans un fichier texte et le sauvegarder avec l'extension **asm**.

2. Compiler ce programme avec l'assembleur **MPASM** fourni par MicroChip. Le résultat est un fichier avec l'extension **hex** contenant une suite d'instruction compréhensible par le µC.

3. Transférer le fichier **hex** dans la mémoire programme du µC (Flash ROM) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de MicroChip ou tout autre programmateur acheté ou réalisé par soit même.

4. Mettre le µC dans son montage final, mettre sous tension et vérifier le fonctionnement.
MicroChip propose gratuitement l'outil de développement **MPLAB** qui regroupe l'éditeur de texte, le compilateur MPASM, un outil de simulation et le logiciel de programmation.

On utilisera MPLAB pour écrire, compiler et éventuellement simuler les programmes, ensuite on les transférera à l'aide du programmateur **JDM** piloté par le logiciel **ICPROG**, les deux sont disponibles gratuitement sur le Web.

2. Directives de MPASM

Les directives de l'assembleur sont des instructions qu'on ajoute dans le programme et qui seront interprétées par l'assembleur MPASM. Ce ne sont pas des instructions destinées au µC.

21. Directives les plus utilisées

☑ **LIST** : Permet de définir un certain nombre de paramètres comme le processeur utilisé (p), la base par défaut pour les nombres (r) ainsi que d'autres paramètres.
Exemple : LIST p=16F84, r=dec.

☑ **#INCLUDE** : Permet d'insérer un fichier source. Par exemple le fichier **p16F84.inc** contient la définition d'un certain nombre de constantes comme les noms des registres ainsi que les noms de certains bits du µC 16F84.
Exemple : #INCLUDE "p16f84.inc" ou #INCLUDE ⟨p16f84.inc⟩.

☑ **__CONFIG** : Permet de définir les 14 fusibles de configuration qui seront copiés dans le registre de configuration lors de l'implantation du programme dans le µC.
Exemple : __CONFIG B'11111111111001' ou __CONFIG H'3FF9'.
🔔 **Remarque** : Si le fichier p16f84.inc a été inséré, on peut utiliser les constantes prédéfinies :
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF

☑ **EQU** : Permet de définir une constante.
Exemple : XX EQU 0x20.
Chaque fois que le compilateur rencontrera la valeur XX, il la remplacera par la constante 0x20. Ça peut être une constante s'il s'agit d'une instruction avec adressage immédiat, ou d'une adresse s'il s'agit d'une instruction avec adressage direct.

☑ **#DEFINE** : Définit un texte de substitution.
Exemple : #DEFINE monbit PORTA,1
Chaque fois que le compilateur rencontrera monbit, il le remplacera par PORTA,1.

☑ **ORG** : Définit la position dans la mémoire programme à partir de laquelle seront inscrites les instructions qui suivent cette directive ORG.
Exemple : ORG 0 ; adresse du début du programme principal.
ORG 4 ; adresse du début du sous-programme d'interruption.

Fonction Traiter

☑ **CBLOCK …ENDC** : Définit un bloc de variables logé dans la zone mémoire RAM accessible par l'utilisateur.

Exemple :
```
CBLOCK 0x0C        ; zone mémoire RAM accessible par l'utilisateur débute à l'adresse 0x0C
Var1 : 1           ; zone de 1 octet
Var2 : 8           ; zone de 8 octets
ENDC
```

☑ **END** : Indique la fin du programme.

22. Format des nombres

L'assembleur reconnaît les nombres en décimal, hexadécimal, binaire ou octal. Pour préciser la base il faut utiliser les préfixes précisés dans le tableau de la figure 1. On peut à l'aide de la directive LIST définir un format par défaut. Si par exemple on place l'instruction LIST r=dec au début du programme, tous les nombres sans préfixe seront interprétés en décimal.

Figure 1

Base	Préfixe	Exemple : 36
Décimal	D'nnn'	D'36'
	.nnn	.36
Hexadécimal	H'nn'	H'24'
	0xnn	0x24
	nnh	24h
Binaire	B'...'	B'00100100'
Octal	O'nnn'	O'44'

23. Structure d'un programme écrit en assembleur

Un programme écrit en assembleur doit respecter une certaine syntaxe et un certain nombre de règles afin qu'il soit facile à lire et à débuguer :

1. Quelques lignes de commentaire précisant la fonction du programme.

2. Tout ce qui commence à la première colonne est considéré comme une étiquette (Label) permettant de faire des sauts et des appels aux sous-programmes.

3. Tout ce qui suit un point virgule est considéré comme un commentaire non interprété par le compilateur.

4. Un programme apparaît donc comme un texte écrit sur 3 colonnes :

☑ La colonne de gauche contient les étiquettes.
☑ La colonne du milieu contient les instructions.
☑ La colonne de droite contient les commentaires.

5. Si le programme utilise des interruptions, mettre à l'adresse 0, adresse du branchement du µC lors d'un Reset ou d'une mise sous tension, une instruction de branchement au début du programme principal.

Exemple :
```
ORG 0
GOTO  début
```

6. Ecrire le sous-programme d'interruption à l'adresse 4 ou mettre à cette adresse un branchement au même sous-programme.

Exemple :
```
ORG 4
Ecrire le sous-programme d'interruption ici.
RETFIE
```

🔔 **Remarque** : Si le programme ne gère pas les interruptions, on peut se passer des étapes 5 et 6.

7. Ecrire les sous-programmes (s'il y en a). Chaque procédure commence par une étiquette qui représente son nom, et se termine par l'instruction Return.

8. Ecrire le programme principal (commençant par l'étiquette début : si les étapes 4 et 5 sont présentes).

9. Terminer avec la directive END.

Exemple : La figure 2 montre un extrait du programme écrit en assembleur.

Figure 2

Etiquette	Assembleur	Commentaire
	LIST p=16F84	; indique le µC utilisé
	#INCLUDE <p16F84.inc>	; fichier contenant les noms des registres internes.
	__CONFIG H'3FF1'	; configuration des fusibles
		; **Initialisation**
	ORG 0	; début du programme à l'adresse 0
	BSF STATUS,5	; sélectionner la banque 1
	BCF OPTION_REG,7	; initialiser le registre option
	BCF TRISA,2	; configurer la broche 2 du port A en sortie
	BSF TRISB,2	; configurer la broche 2 du port B en entrée
	BCF STATUS,5	; repasser en banque 0
	BCF PORTA,2	; initialiser la broche RA_2 à 0

1. Timer TMR0

C'est un compteur 8 bits ayant les caractéristiques suivantes :

☑ Il est incrémenté en permanence soit par l'horloge interne (mode timer) soit par une horloge externe appliquée à la broche RA_4 du port PORTA (mode compteur).
Le choix de l'horloge se fait à l'aide du bit **TOCS** du registre **OPTION** et la sélection du front actif de l'horloge externe s'effectue par le bit **TOSE** du même registre :

Registre OPTION ⇒

RBPU	INTEDG	**TOCS**	**TOSE**	PSA	PS_2	PS_1	PS_0

➢ TOCS=0 ⇒ TMR0 utilise l'horloge interne et TOCS=1 ⇒ TMR0 utilise l'horloge externe (broche RA_4).
➢ TOSE=0 ⇒ front montant et TOSE=1 ⇒ front descendant.

☑ Quelque soit l'horloge choisie, on peut la passer dans un prédiviseur de fréquence programmable dont le rapport est fixé par les bits PS_0, PS_1 et PS_2 du registre **OPTION** (voir tableau de la figure 1a). L'affectation ou non du prédiviseur se fait à l'aide du bit **PSA** du même registre.

Registre OPTION ⇒

RBPU	INTEDG	TOCS	TOSE	**PSA**	PS_2	PS_1	PS_0

➢ PSA=0 ⇒ le prédiviseur est affecté au timer TMR0
➢ PSA=1 ⇒ le prédiviseur est affecté au timer WDT.
🕭 Remarque : Pour ne pas avoir une prédivision de l'horloge du timer TMR0 il suffit d'affecter le prédiviseur au timer du chien de garde WDT.

☑ Au débordement de TMR0 (FF → 00), le drapeau **TOIF** du registre **INTCON** est placé à 1. Ceci peut déclencher l'interruption **TOIE** si celle-ci est validée (voir plus tard).

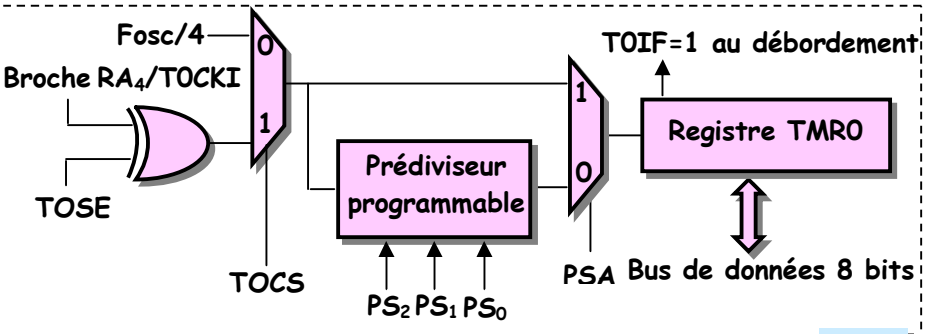
☑ Le contenu du timer TMR0 est accessible dans le fichier de registres aussi bien en lecture qu'en écriture.

La figure 1b explique le fonctionnement du registre TMR0.

Figure 1

PS_2	PS_1	PS_0	Prédivisions
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

1a



1b

Exemple : Supposons que le timer TMR0 utilise l'horloge interne à base d'un quartz de 4 MHz, Calculons alors le temps de débordement du timer correspondant à une prédivision de 2 : En effet, le timer TMR0 est un compteur de 8 bits qui compte 256 périodes d'horloge. Puisque la période d'horloge est multipliée par 2 alors le temps de débordement du timer est égal à 256*2*1 µs soit 512 µs.
Avec un prédiviseur de 256 on aura un temps de débordement égal à 256*256*1 µs soit 65536 µs.
Si ce temps est insuffisant on pourrait utiliser un compteur supplémentaire.

2. Watchdog Timer WDT

C'est un compteur 8 bits incrémenté en permanence (même si le µC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT Time Out), deux situations sont possibles :

☑ Si le µC est en fonctionnement **normal**, le WDT Time Out provoque un **Reset**. Ceci permet d'éviter le blocage du µC en cas de plantage du programme comme par exemple une boucle sans fin, un Reset répété,…
☑ Si le µC est en mode **sleep**, le WDT Time Out provoque un **Wake-Up**, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode sleep. Cette situation est souvent exploitée pour réaliser des temporisations.

L'horloge du WDT est ajustée pour que Le Time Out arrive toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passer le signal Time Out dans un prédiviseur programmable partagé avec le timer TMR0 (voir tableau de la figure 2a).

L'usage du WDT doit se faire avec précaution pour éviter la réinitialisation inattendue et répétée du programme. Pour éviter un WDT time Out lors de l'exécution d'un programme, on a deux possibilités :

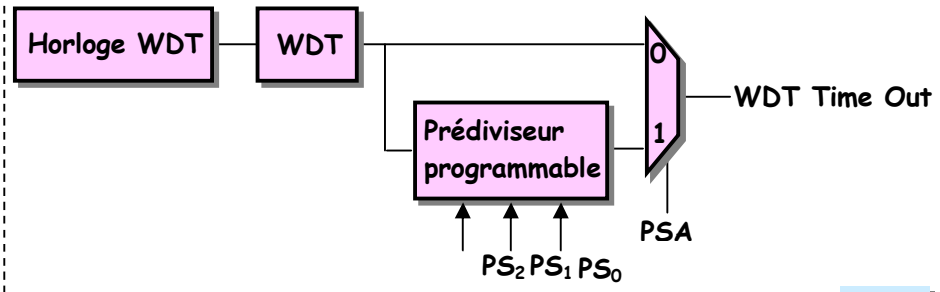
☑ Inhiber le WDT de façon permanente en mettant à 0 le bit **WDTE** dans le registre de configuration.
☑ Remettre le WDT à 0 périodiquement dans le programme principal à l'aide de l'instruction **CLRWDT** pour éviter qu'il ne déborde. On ne doit pas, en particulier, loger cette instruction dans un sous programme d'interruption.

La figure 2b explique le fonctionnement du timer WDT.

Figure 2

PS_2	PS_1	PS_0	Prédivisions
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

2a



2b

Exemple : Le temps de débordement du timer WDT est estimé à 18 ms sans utiliser une prédivision. Cependant on pourrait prolonger ce temps en utilisant le prédiviseur. Avec une prédivision de 128 le débordement du timer WDT aura lieu tout les 128*18 ms soit 2304 ms.

🔔 **Remarque** : Le constructeur du µC 16F84 donne trois temps pour le débordement du timer WDT :

☑ Le temps nominal ou typique de 18 ms : c'est le temps de réaction normal que met en général le watchdog pour provoquer un Reset du µC en cas de plantage du programme.
☑ Le temps minimal de 7 ms : c'est le délais maximum dont on dispose entre deux instructions **CLRWDT** pour éviter un Reset non désiré du µC.
☑ Le temps maximum de 33 ms : c'est le temps de réaction du watchdog dans le cas le plus défavorable en fonction du composant et des conditions d'utilisation. MicroChip garantit que le Reset s'effectuera au maximum en 33 ms.

1. Interruption

Une interruption est un événement inattendu qui provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption. A la fin de cette procédure, le µC reprend le programme à l'endroit où il s'était arrêté.

Le µC 16F84 possède **4 sources d'interruption**. A chaque interruption sont associés deux bits :

☑ Un bit de validation : permet d'autoriser ou non l'interruption.
☑ Un bit de signalisation ou drapeau (flag) : permet au µC de savoir de quelle interruption il s'agit.

Tous ces bits sont logés dans le registre **INTCON** à part le drapeau **EEIF** de l'interruption **EEI** qui se trouve dans le registre **EECON1**.

Registre INTCON ⇒

GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

☑ **GIE** : ce bit autorise ou interdit globalement toutes les interruptions.
GIE=1 ⇒ toutes les interruptions sont autorisées.

☑ **EEIE** : ce bit permet de valider l'interruption de fin d'écriture en EEPROM.
EEIE=1 ⇒ interruption validée.

☑ **TOIE** : ce bit valide l'interruption générée par le débordement du timer TMR0.
TOIE=1 ⇒ interruption validée.

☑ **INTE** : ce bit valide l'interruption sur la broche RB_0.
INTE=1 ⇒ interruption validée.

☑ **RBIE** : ce bit valide l'interruption sur l'une des broches RB_4 à RB_7.
RBIE=1 ⇒ interruption validée.

☑ **TOIF** : ce bit signale le débordement du timer TMR0.
TOIF=1 ⇒ débordement du timer TMR0.

☑ **INTF** : ce bit signale qu'une interruption a été provoquée par la broche RB_0.
INTF=1 ⇒ interruption générée.

☑ **RBIF** : ce bit signale qu'une interruption a été provoquée par l'une des broches RB_4 à RB_7.
RBIF=1 ⇒ interruption générée.

Registre EECON1 ⇒

			EEIF	WRERR	WREN	WR	RD

☑ **EEIF** : ce bit signale le déclenchement d'une interruption de fin d'écriture en EEPROM.
EEIF=1 ⇒ interruption générée.

2. Déroulement d'une interruption

Lorsque l'événement déclencheur d'une interruption intervient, alors son drapeau est positionné à 1. Si de plus l'interruption correspondante a été validée, elle est alors déclenchée : le µC arrête ce qu'il est en train de faire et va exécuter la procédure d'interruption qui se trouve à **l'adresse 4** en exécutant les étapes suivantes :

☑ L'adresse contenue dans le PC (Program Counter) est sauvegardée dans la pile, puis remplacée par la valeur 4 qui correspond à l'adresse de la routine d'interruption.
☑ Le bit GIE est positionné à 0 pour inhiber toutes les interruptions afin que le µC ne soit pas dérangé pendant l'exécution de la procédure d'interruption.
☑ A la fin de la procédure d'interruption (instruction **RETFIE**) :

➢ Le bit GIE est repositionné à 1 en autorisant ainsi un autre événement.
➢ Le contenu du PC est rechargé à partir de la pile ce qui permet au programme de reprendre là où il s'est arrêté.

🔔 **Remarques :**

☑ Le drapeau reste à 1 même après le traitement de l'interruption. Par conséquent, il faut toujours le remettre à 0 à la fin de la routine d'interruption sinon l'interruption sera déclenchée de nouveau juste après l'instruction RETFIE.
☑ Seul le PC est empilé automatiquement. Si cela est nécessaire, les registres **W** et **STATUS** doivent être sauvegardés temporairement en RAM utilisateur puis restaurés à la fin de la routine d'interruption pour que le µC puisse reprendre le programme dans les mêmes conditions où il l'a laissé.

3. Sources d'interruption

31. Interruption INT : Entrée RB_0 du PORTB

Cette interruption est provoquée par un changement d'état sur l'entrée RB_0 du PORTB quand elle est programmée en entrée. Elle est gérée par les bits :

☑ **INTE** et **INTF** du registre **INTCON**.
☑ **INTEDG** du registre **OPTION** : ce bit permet de choisir le front du déclenchement de l'interruption.
INTEDG=1 ⇒ front montant et INTEDG=0 ⇒ front descendant.

Registre OPTION ⇒

RBPU	INTEDG	TOCS	TOSE	PSA	PS_2	PS_1	PS_0

32. Interruption RBI : Entrées RB_4 à RB_7 du PORTB

Cette interruption est provoquée par un changement d'état sur l'une des entrées RB_4 à RB_7 du PORTB. Le front n'a pas d'importance.
Les bits associés sont **RBIE** et **RBIF** du registre **INTCON**.

33. Interruption TOI : Débordement du Timer TMR0

Cette interruption est provoquée par le débordement du timer TMR0.
Les bits associés sont **T0IE** et **T0IF** du registre **INTCON**.

34. Interruption EEI : Fin d'écriture dans l'EEPROM

Cette interruption est déclenchée à la fin d'une écriture réussie dans l'EEPROM de données.
Les bits associés sont **EEIE** du registre **INTCON** et **EEIF** du registre **EECON1**.

Fonction Traiter