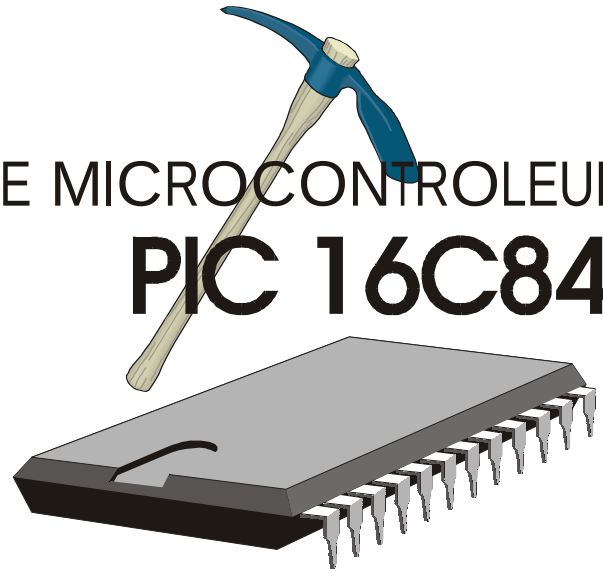
UNIVERSITE PIERRE ET MARIE CURIE

INSTITUT DE SCIENCES ET TECHNOLOGIE

MCR
Mesure Controle Régulation

INSTITUT de SCIENCES et TECHNOLOGIE



LE MICROCONTROLEUR
PIC 16C84

J AUVRAY
1998

LES MICROCONTROLEURS PIC

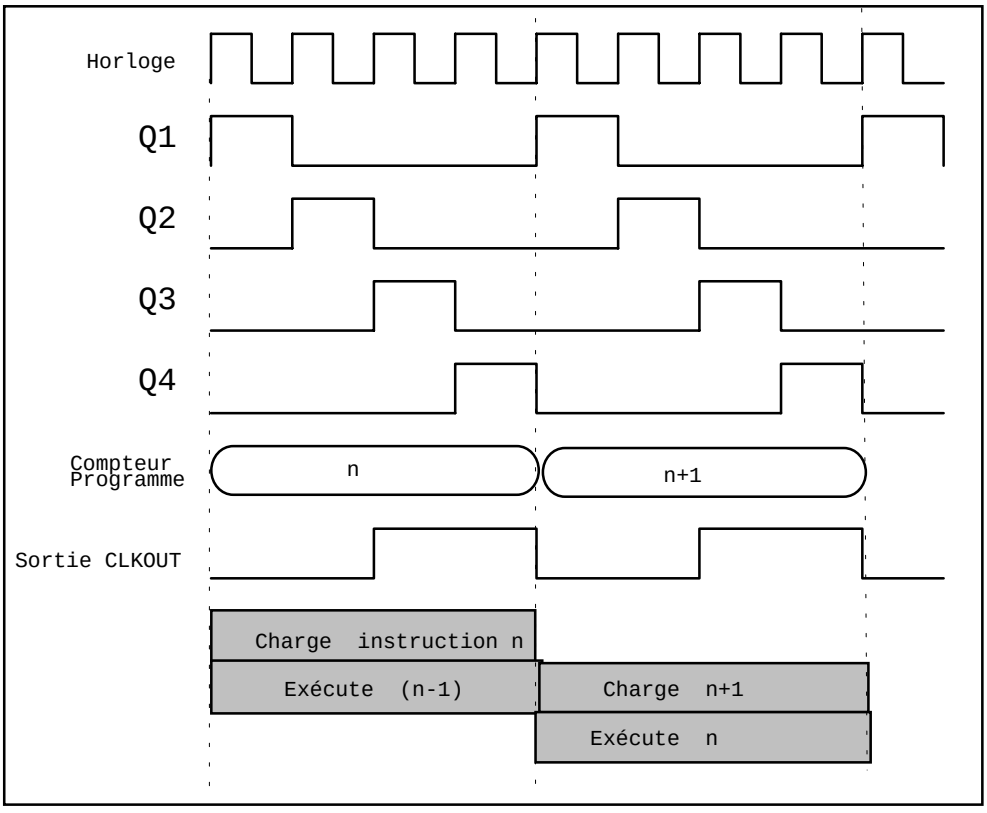
Les microcontroleurs PIC de MICROCHIP se distinguent des microcontroleurs classiques (8051 ou 68HC11) par les deux aspects suivants .

Ce sont des microprocesseurs **de structure Harvard** :

C'est à dire que les mots codes et les données sont stockées dans deux mémoires différentes. Les mots codes sont longs 12 à 14 bits suivants les modèles et écrits dans une ROM interne non accessible de l'extérieur. Il en résulte que toutes les instructions peuvent être codées sur un seul mot d'où une exécution rapide, mais en contre partie aucun fonctionnement avec ROM extérieure n'est possible. L'utilisation d'un simulateur de ROM lors de la mise au point du programme n'est pas possible. Il n'existe pas de boîtiers ROMLESS (sans ROM interne) mais seulement des circuits avec ROM interne (Programmables par masque à la fabrication) PROM interne effaçable par UV (UVPROM) si une fenêtre est prévue pour l'irradiation de la puce , ou non (OTPROM) si cette fenêtre n'existe pas. Le circuit 1684 qui nous occupera plus particulièrement se distingue par une mémoire de programme de type EEPROM ou Flash (16F84) effaçable électriquement.

Ce sont des **microprocesseurs RISC** (Reduce Instructions Set Computers)

Leurs instructions sont en nombre réduit ,mais efficaces. De plus ils possèdent une architecture push pull permettant pendant un cycle d'horloge , l'exécution d'une instruction et simultanément le chargement de l'instruction suivante La figure suivante montre comment la fréquence de l'oscillateur est divisée par 4 de façon à définir au cours de chaque cycle machine 4 phases Q1 Q2 Q3 Q4 au cours desquelles sont réalisées les différentes taches nécessaires .
On notera la sortie CLKout qui délivre un signal de fréquence F/4 (En mode RC)



La société MICROCHIP commercialise de nombreux circuits de cette famille qui se distinguent par le nombre de leurs bornes d'entrée- sortie, la taille de leur mémoire de programme et les périphériques intégrés.

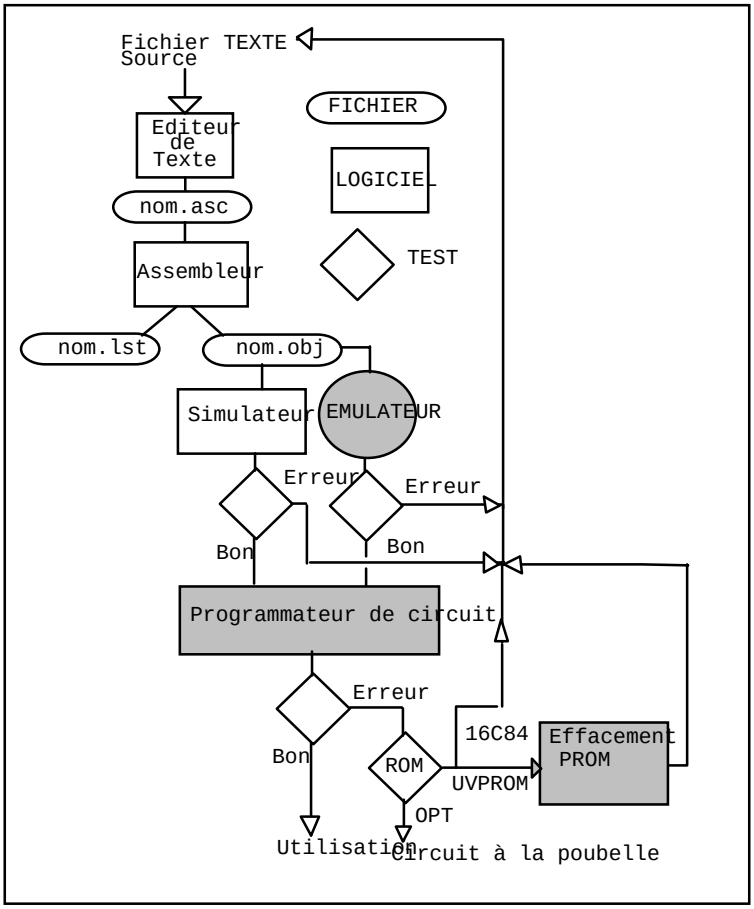
Les principaux membres de la famille sont décrits dans le tableau suivant . Ce tableau est incomplet et de nouveaux circuits apparaissent chaque année.

Nom	ROM programme	Mots bits	RAM interne	I/O	Broches [1]	Timer	Interruptions	Autres
16C54	512mots PROM	12	25 octets	8+4	18	1	0	
16C55	1k PROM [2]	12	25	8+8+4	28	1	0	
16C56	1k PROM[2]	12	25	8+4	28	1	0	
16C57	2k PROM[2]	12	72	8+8+4	28	1	0	
16C64	2K PROM[2]	14	128	33	40	3	8	Sortie série synchrone +I2C +CAN 8 bits
16C71	1K PROM[2]	14	36	13	18	1	4	
16C74	4K PROM [2]	14	192	33	40	3	12	Sortie série synchrone +I2C+RS232 +CAN 8 bits
16C84	1K EEPROM	14	36	8+5	18	1	4	

[2] PROM interne 12 ou 14 bits effaçable (UVPROM) ou non (OTP)
[1] Pour boîtier DIL

L'impossibilité d'utiliser une ROM extérieure et en particulier un émulateur de PROM complique la mise en oeuvre de ces circuits. Le logiciel écrit en assembleur doit être testé avec un logiciel de simulation , puis la PROM interne gravée grâce à un programmateur de PIC et enfin le circuit mis sous tension. Si le logiciel comporte des erreurs la PROM doit être effacée (Il faut donc utiliser un boîtier à fenêtre , coûteux) et les tests repris. La procédure est lourde et les circuits soumis à de multiples effacements ont une durée de vie réduite . L'utilisation d'un émulateur est souhaitable, mais un tel système est coûteux (plus de 5000F) et fragile. (La sonde est fragile)

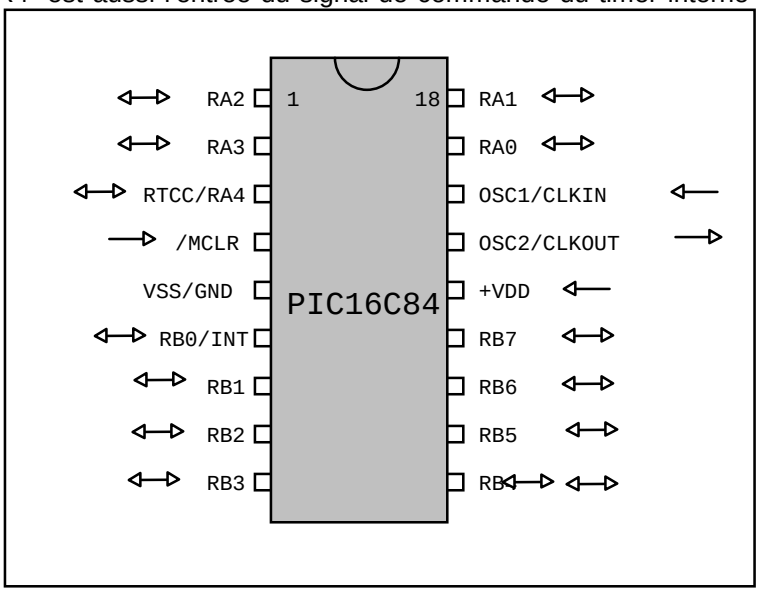
Le 16C84 permet de retrouver la souplesse qu'apportait l'émulateur de ROM avec le 8031. L'EEPROM (ou ROM Flash pour le 16F84) peut être programmée très rapidement 'sur cite' par un programmateur de coût négligeable , les circuits supportent au moins 1000 cycles d'effacement ce qui est bien suffisant pour mettre au point un programme déjà conséquent Nous avons choisi pour cette raison d'utiliser ce boîtier , malgré sa puissance relativement faible
Il est en effet semblable au 16C54 mais connaît la notion d' interruption , possède une pile nettement plus étendue autorisant jusqu'à 8 niveaux de sous programmes contre 2 pour le 54 , et inclus une EEPROM de données de capacité 64 octets.



ARCHITECTURE MATERIELLE

Les boîtiers des 16C54 et 16C84 sont identiques .
Outre les deux bornes d'alimentation VDD-VSS=5V typiques on notera la présence de :
- 1 port B de 8 bits utilisable en entrée ou sortie (Broches RB0-7)
- 1 port de 5 fils RA0-4 , le fil R4 est aussi l'entrée du signal de commande du timer interne (RTCC) pour le 16C54 il n'a que cette fonction.
- 2 bornes pour réaliser l'oscillateur :

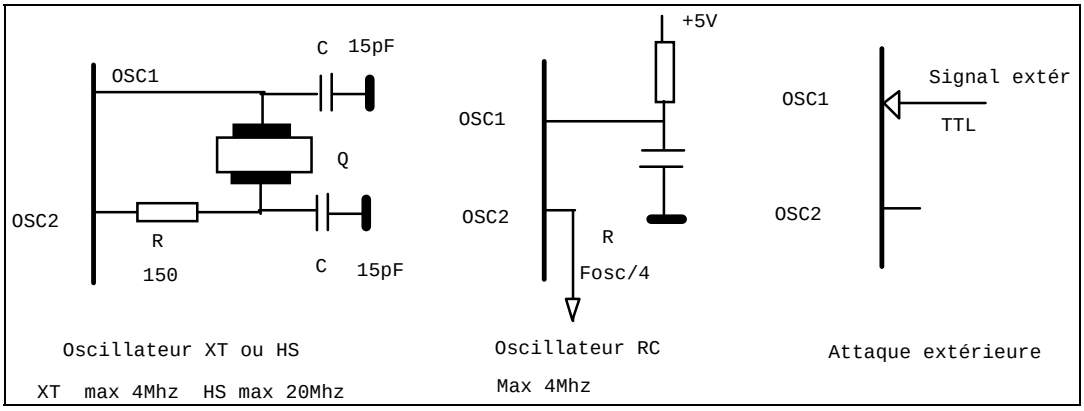
Cet oscillateur peut être un oscillateur à quartz ou de type RC. Un signal logique extérieur peut également être injecté sur OSC1,
. Les 16C5x peuvent fonctionner jusqu'à 20mhz mais il faut acheter le boîtier qui convient au type d'oscillateur que l'on veut réaliser.



(Le 16C54 existe par exemple en 5 versions :
16C54-RC pour oscillateur RC
16C54-XT quartz jusqu'à 4Mhz
16C54-HS quartz jusqu'à 20Mhz
16C54LP Faible puissance 2 à 6V d'alimentation 200kHz max
16C54-JW version UVPROM)

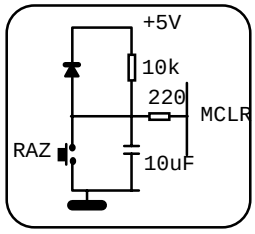
Le 16C84 n'existe qu'en une seule version (Il existe en fait une version standard 4Mhz max et une version HF qui fonctionne jusqu'à 20 Mhz) et accepte l'un ou l'autre des schémas il faut seulement indiquer au moment de la programmation quel est le schéma utilisé. (Programmation des fusibles , voir plus loin)

Ces différents schémas sont indiqués ci dessous .



On notera qu'avec la version RC ou en utilisant un oscillateur extérieur la fréquence d'horloge peut descendre jusqu'a zéro, le circuit ne possède pas de mémoires dynamiques internes

- La broche MCLR est utilisée pour le RAZ et également dans le cas du 16C84 pour la programmation de la mémoire de programme (EEPROM) .
Le schéma de remise à zéro est classique (figure ci contre)



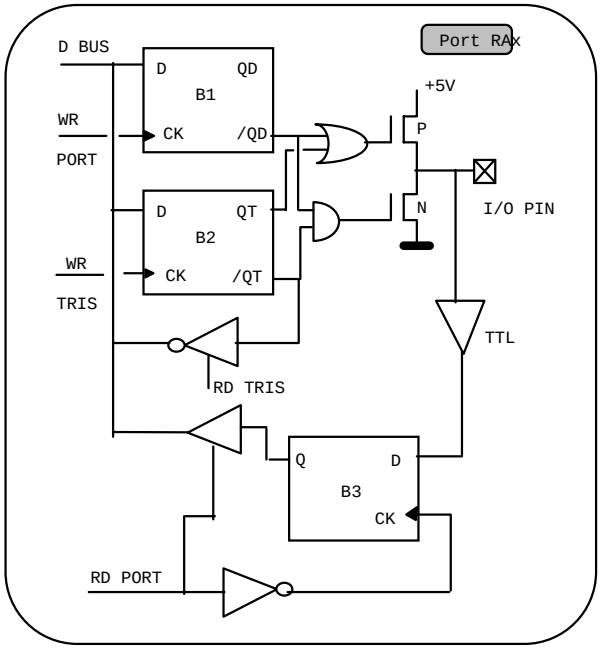
LES PORTS D'ENTREE SORTIE

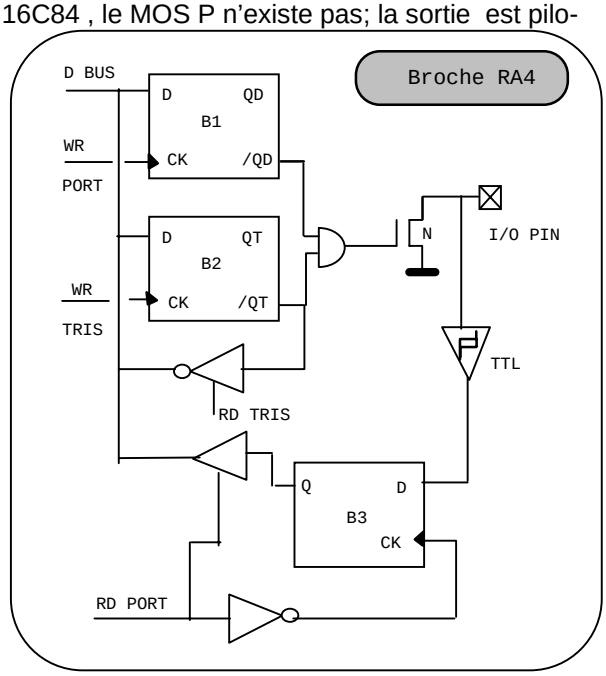
Le 16C84 possède 2 ports d'entrée sortie A et B ayant respectivement 5 (4 pour le 16C54) et 8 fils. Le sens de transfert de chaque fil est déterminé par un bit situé dans le registre TRIS correspondant (TRISA ou TRISB)

La structure de chaque borne du **port A** est représentée dans la figure ci jointe . Les 'latches' B1 et B2 correspondent respectivement à l'un des bits des registres PORT et TRIS

Lorsque le fil est programmé en entrée la sortie QT de B2 est au 1 , les deux MOS canal P et N voient leur grille portée respectivement à 1 et 0, ils sont bloqués et l'état de la borne lue par le buffer TTL est déterminée par le circuit extérieur relié à cette borne. En sortie au contraire QT=0 et la tension sur les grilles des MOS est entièrement déterminée par l'état de la bascule B1.Si par exemple QD=1 , le MOS P est conducteur (0 sur sa grille) et le MOS N bloqué, la broche de sortie est portée au niveau haut.

Cette structure est celle des bornes RA0 à RA3 , pour RA4 qui n'est présente que sur le 16C84 , le MOS P n'existe pas; la sortie est pilotée par le seul MOS N et une résistance extérieure de pull up est nécessaire. (Figure ci contre) Cette broche RA4 est aussi l'entrée du TIMER interne (Fil RTCC du 16C54) .





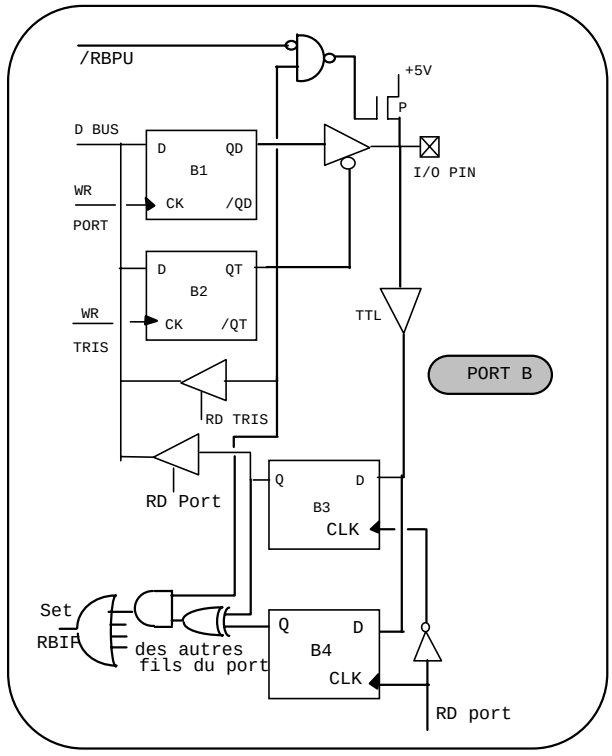
dans ce cas fixé par un buffer 3 états relié à la sortie QD de la bascule B1. (Figure ci contre). .

Le MOS de pull up peut également être bloqué par la mise à 0 du bit RBPU du registre OPTION (OPTION-7)

La bascule B4 et le OU Exclusif associé n'existent que pour les bornes PB4 à PB7.La bascule B4 mémorise l'état de l'un de ces 4 fils au moment du coup d'horloge précédent. Le OU exclusif compare ainsi deux états successifs et un signal d'interruption RBIF est généré s'ils sont différents c'est a dire si i l'état de l'un des fils PB4-7 à été modifié. Pour le 16C84 la moitié haute du port B joue le rôle d'entrée d'interruption (Voir plus loin).

Le **PORT B** a une structure différente .Il existe un seul MOS P qui joue le rôle de résistance de pull up (Courant de 100uA équivalent à une résistance de 50kΩ)

Ce MOS est bloqué si le fil est programmé en sortie. Le niveau de sortie est

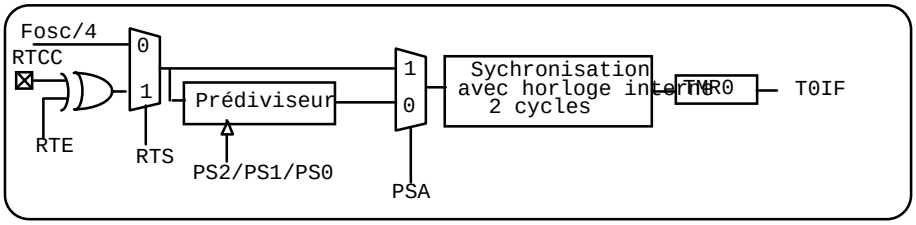


LE TIMER INTERNE

Le 16C84 (comme le 16C54) possède un timer interne dont le fonctionnement est déterminé par le contenu du registre OPTION et qui active une interruption (Bit TOIF) au dépassement (over-flow) .

Les événements qui font avancer le compteur RTCC sont soit un signal synchrone du compteur interne et de fréquence Fosc/4 (si le bit RTS du registre OPTION est au 0, c'est le mode TIMER) , soit une transition sur la broche RTCC/RA4 .(Si RTS=1, c'est le mode COMPTEUR).Le bit PSA permet ce faire intervenir ou non un diviseur supplémentaire dont le taux de division est déterminé par les bits PS2-PS1-PS0 . Pour synchroniser les signaux d'entrée et l'horloge interne , deux cycles machine sont nécessaires, il existe donc un retard entre l'arrivée de l'événement sur la broche RTCC et l'avance du compteur.

Le diviseur préalable (prescaler) est utilisé également par le chien de garde intégré qui sera décrit plus loin.



Les bits du registre OPTION sont les suivants , ils sont tous mis au 1 par un RESET .:

OPTION	/RBPU	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0
81H	7	6	5	4	3	2	1	0

OPTION.7 /RBPU Au zéro il met en place le MOS de pull up du port B
OPTION.6 INTEDG Définit le sens du front (0=⇓ , 1=⇑) qui provoquera une interruption sur la borne INT =RB0 (si cette interruption est activée)
OPTION.5 RTS Définit la source utilisée par le timer (1= borne RA4 0= Fosc/4)
OPTION.4 RTE Défini le sens de la transition sur RA4 qui fait avancer le compteur (0=⇓ , 1=⇑)
OPTION.3 PSA Si PSA=1 le prédiviseur est utilisé par le TIMER , au 0 il est affecté au chien de garde.
OPTION.2-1-0 PS2-PS1-PS0 Définit le taux de division du pré compteur conformément au tableau ci dessous .

PS2	PS1	PS0	Taux de division pour le TIMER	pour le WAT-CHDOG
0	0	0	2	1
0	0	1	4	2
0	1	0	8	4
0	1	1	16	8
1	0	0	32	16
1	0	1	64	32
1	1	0	128	64
1	1	1	256	128

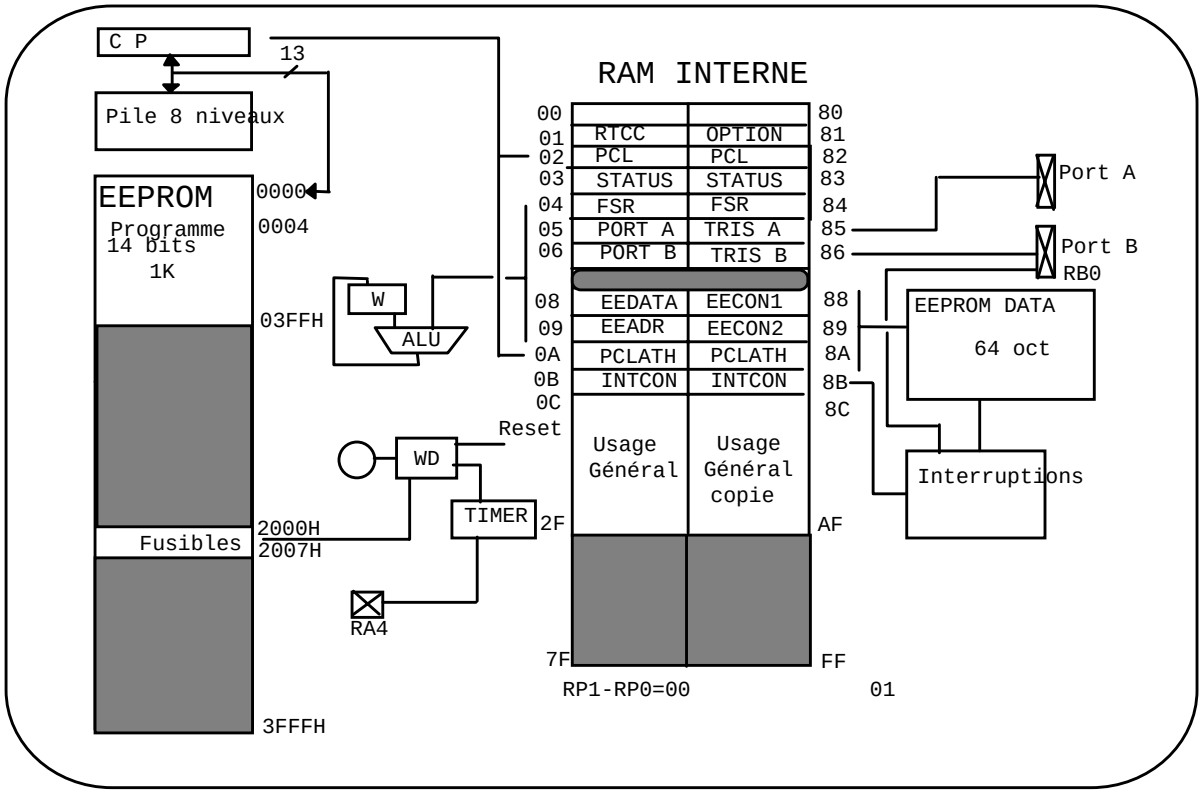
Le registre RTCC (appelé parfois TMR0) peut être lu ou écrit .
Cependant :

Pour toute écriture dans ce registre le comptage ne redémarre que deux cycles machine plus tard .(2µS pour un quartz à 4 Mhz) de plus, le pré diviseur est remis à zéro .

Pour tester l'état du compteur il est possible de lire RTCC par une instruction MOVF TMR0,W sans perturber le déroulement du comptage .

LA STRUCTURE INTERNE

Les éléments constituant du boîtier sont représentés sur la figure ci dessous :
Comme dans les microprocesseurs classiques , contrairement au 8051, l'accumulateur **qui s'appelle ici W** ne fait pas partie du champ mémoire..



L'unité arithmétique et logique (ALU) travaille avec les octets provenant de W et le contenu des registres placés dans la RAM interne que l'on appelle ici registres f . L'ALU reçoit également des nombres entiers (de 0 à FF) ,ici un entier est appelé LITERAL .

Le boîtier contient 52 octets de RAM interne dont 12 sont les registres de commande et 36 disponibles pour stocker des résultats numériques . L'adresse contenue dans le code des instructions n'a que 7 bits (Permettant d'accéder à des cases mémoire entre 00 et 7F , dont seules celles comprises entre 00 et 2F existent réellement) Le 8eme bit nécessaire pour accéder aux cases 80 à FF est le bit RP0 du registre d'ÉTAT.chaque valeur de ce bit définit une PAGE. On notera que certains registres ont deux adresses ,une dans chaque page.

Les mots de programme ont une longueur de 14 bits ils sont contenus dans une EEPROM adressée par un bis de 13 bits , en réalité seuls 13 bits sont gérés par le compteur programme , 8 dans le registre PCL et les 5 autres fournis par le registre PCLATH . Ces 13 bits permettent d'accéder à 8k de ROM dont 1k seulement sont implantés dans le boîtier. Les bits 11 et 12 sont ignorés .Il existe cependant 8 octets situés entre 2000H et 2007H non accessibles directement car ils mettent en oeuvre le fil A13 non géré par le compteur programme . Ces 8 octets contiennent les FUSIBLES qui déterminent le type d'oscillateur utilisé et la mise en oeuvre ou non du chien de garde. Ils ne sont accessibles que par le logiciel de gravure dans l'EEPROM.

A ceci s'ajoutent 64 octets dans une EEPROM de données , accessibles par une procédure spéciale.

Compte tenu du faible nombre d'octets de RAM disponibles , la pile utilisée pour la gestion des sous programmes n'est pas implantée en RAM . Une pile spéciale de 8 niveaux est prévue. Ainsi 8 niveaux de sous programmes sont admis au maximum ,c'est deja pas mal, mais les instructions PUSH et POP n'existent pas .

Les registres en RAM

Nous avons déja décrit le registre OPTION qui intervient dans le pilotage du TIMER , passons en revue les autres .
A l'adresse 01 **RTCC** est le compteur utilisé par le TIMER et le chien de garde
A l'adressé 02 ou 82H on trouve le compteur programme **PCL** ou plus exactement l'octet bas de l'adresse dans la ROM de programme . Les 5 bits supplémentaires sont contenus dans le registre **PCLATH** d'adresse 09 (ou 89H) .
Le registre d'état **STATUS** est situé à l'adresse 03 (ou 83H) , son contenu est le suivant :

STATUS 03

IRP	RP1	RP0	/T0	/PD	Z	DC	C
7	6	5	4	3	2	1	0

STATUS.7 IRP est un bit destiné à permettre l'accès à plus de 256 mots de RAM par adressage indirect (voir plus loin) ,il n'est pas utilisé par le 16C84
STATUS 6-5 RP1-RP0 Sont les deux bits de sélection de la PAGE en RAM , seul RP0 est utilisé par le 16C84 .Le second peut être utilisé comme bit d'usage général mais cela est déconseillé pour des raisons de compatibilité des logiciels avec d'autres boitiers de la famille 16CXX.
STATUS.4 /TO (Time Out) Est mis à 1 lors de la mise sous tension ou de la remise à zéro du chien de garde , il est basculé à 0 si le chien de garde déborde .
STATUS.3 /PD (Power Down) Est mis à 1 à la mise sous tension , ao 0 par une instruction SLEEP
STATUS.2 Z (Zéro) Mis à 1 si le résultat d'une opération est nulle .
STATUS.1 DC (Digit Carry) C'est la retenue intermédiaire de l'arithmétique DCB
STATUS .0 C Est la retenue en addition ou soustraction.

Le registre FSR est utilisé en adressage indirect, voir plus loin .
L'adresse 5 est le PORT A et 85H le registre de direction TRIS A de ce port (1=entrée 0=sortie)
Le PORT B est localisé dans la case suivante 06 ou 86H
Il n'y a pas de case mémoire à l'adresse 07 (ou 87H)
Les octets suivants sont consacrès à la gestion de l'EEPROM de données et aux interruptions.

Les interruptions

Le 16C84 dispose de 4 sources d'interruptions :

- Une source externe via la broche RB0/INT
- Le débordement du TIMER
- un changement de l'état des bornes 4 à 7 du port B
- La programmation de l' EEPROM de données

Ces interruptions sont définies et autorisèes à partir du degistre INTCON d'adresse 0B ou 8B .

INTCON 0B

GIIE	EEIE	RTIE	INTE	RBIE	RTIF	INTF	RBIF
7	6	5	4	3	2	1	0

INTCON.7 GIE est le bit d'autorisation de toutes les interruptions ,au 0 aucune interruption n'est active, c'est son état au RESET.
INTCON.6 EEIE valide l'interruption associée à l'EEPROM de données
INTCON.5 RTIE Mis à 1 il autorise les interruptions associées au débordement du compteur du TIMER (RTCC)
INTCON.4 INTE C'est la validation de l'interruption extérieure reçue sur RB0
INTCON.3 RBIE Valide ou non les interruptions provoquées par un changement d'état sur les fils 7 à 4 du port B.
INTCON.2 INTf S'il est mis à 1 ce bit indique qu'une interruption à été décelée sur l'entrée INT (RB0) . Ce bit détecte cette interruption même si elle n'est pas autorisée.

INTCON.0 RBIF Indique un changement d'état sur les 4 fils RB7-4 .

Ne pas oublier le bit INTEDG = OPTION.6 qui détermine le sens de la transition active sur l'entrée d'interruption extérieure RB0.

A propos des interruptions il faut ajouter les remarques suivantes :

Chaque bit de validation d'une interruption (RBIF,INTF,...) doit être remis à 0 par logiciel dans le programme de traitement de cette interruption.

Lorsqu'une interruption survient le bit de validation général GIE est automatiquement mis à 0 par le processeur de façon à ne pas perturber l'exécution de l'interruption en cours. Il est remis au 1 lors de l'exécution de l'instruction de retour RETFIE

toutes les interruptions utilisent la même adresse 0004 dans la ROM de programme .

L'EEPROM de données

Elle ne fait pas partie du champ mémoire normal et n'est accessible que par une procédure spéciale mettant en oeuvre les registres EEDATA EEADR EECON1 et 2

EECON1				EEIF	WRERR	WREN	WR	RD
88H	7	6	5	4	3	2	1	0

EECON1;4	EEIF	EEProm interrupt Flag est mis à 1 lorsqu'une écriture est terminée Une interruption lui est associée, il ne faut pas oublier que l'écriture d'un octet dans l'EEPROm prend 10mS .
EECON1.3	WREER	Mis à 1 si une erreur s'est produite pendant l'écriture
EECON1.2	WREN	Ce bit doit être mis à 1 pour autoriser une écriture , il est au 0 au reset
EECON1.1	WR	Doit être forcé à 1 pour écrire une donnée, est remis à 0 par le logi ciel lorsque l'écriture est achevée
EECON1.0	RD	Doit être mis à 1 pour lire un octet, est remis automatiquement au 0 par le circuit.

Le registre EECON2 n'a pas d'existence réelle, il n'est évoqué que lors d'une écriture pour sécuriser l'EEPROM.

Procédure de lecture d'un octet en EEPROM

1° Écrire l'adresse dans EEARD
2° Mettre à 1 le bit RD=EECON1.0
3° Lire le contenu de EEDATA

Procédure d'écriture :

1° Charger l'adresse dans EEADR
2° Charger la donnée à écrire dans EEDATA
3° Exécuter le programme de sécurisation suivant:

Charger 55 dans W
Transférer W dans EECON2 (adresse 89H)
Charger AA dans W
Transférer W dans EECON2
Monter à 1 le bit WR de EECON1

Il n'existe pas de procédure d'effacement car toute écriture dans une case mémoire EEPROM efface son contenu précédent .

Le chien de Garde (WATCHDOG)

Un chien de garde est un dispositif matériel et logiciel destiné à se prémunir contre les 'plantages ' accidentels du programme , dus à une impulsion sur l'alimentation par exemple . Il s'agit d'un compteur interne commandé par un oscillateur indépendant qui au débordement (le passage de FF à 00 pour un compteur 8 bits) provoque un RESET du CPU. Pour éviter ce RESET le programme doit périodiquement remette à zéro le compteur du chien de façon qu'un débordement ne se produise jamais. Si le programme se 'plante' cette fonction de remise à zéro n'est plus effectuée et un RESET est commandé.

Les microcontroleurs 16CXX possèdent un tel dispositif qui peut être activé ou non par un fusible programmable par le programmateur de PIC. (Il n'est pas accessible normalement)

Le fonctionnement est piloté :

Par le fusible WDT cité plus haut
Par le bit PSA du registre OPTION
Par l'instruction CLRWDT qui effectue un RESET du compteur du WD et doit être activée périodiquement dans votre programme.

Entre deux débordements il s'écoule normalement 18mS environ , si cette durée est trop courte elle peut être allongée jusqu'à 2,5 secondes en mettant en oeuvre le prédiviseur utilisé également par le TIMER , pour cela il suffit de basculer le bit PSA à 1 . On notera qu'il n'est pas possible d'utiliser le prédiviseur simultanément pour le TIMER et le WD .

L'ASPECT LOGICIEL

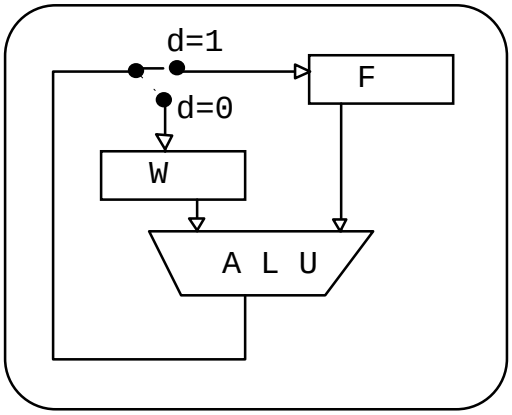
LES MODES D'ADRESSAGE

L'absence de toute mémoire extérieure et le fait que les codes machine contiennent le plus souvent l'adresse du registre interne concerné , réduisent à sa plus simple expression les modes d'adressage nécessaires.

L'adressage immédiat qui correspond à la manipulation d'un nombre , ici un octet, entre les registres internes est le plus simple d'entre eux. Ici pour respecter la terminologie de MICROCHIP un nombre s'appelle un LITERAL il est généralement noté k .

L'adressage direct consiste à indiquer en clair sur 7 bits l'adresse du registre considéré. Le huitième bit qui permet d'accéder à la page 1 est le bit RP0 du registre STATUS. La manipulation de ce bit est nécessaire pour changer de PAGE. L'assembleur désigne par f les registres internes .Dans la description des mots codes fffffff représente le mot de 7 bits qui est l'adresse interne en RAM du registre considéré.

La majorité des opérations s'effectuent par l'intermédiaire du registre de travail (accumulateur) W , mais ici ,et c'est une souplesse intéressante par rapport aux microprocesseurs classiques , le résultat de l'opération n'est pas toujours stockée dans W ; un bit noté d détermine si le résultat doit être chargé dans W (d=0) ou dans le registre f en cause (d=1) . Ceci est illustré par la figure suivante.



Il existe également un adressage direct de bits ,par exemple :
BCF f,3 met à zéro (BitClear) le bit 3 du registre f
BSF f,3 met ce même bit à 1
Dans la description qui va suivre des codes machine bbb représente l'adresse du bit dans l'octet.

L'adressage indirect est ici un peut particulier car il met en oeuvre le registre d'adresse 04 FSR et le registre INDF d'adresse 0 qui n'a pas d'existence réelle.
L'instruction MOVWF;INDF transfert le contenu de W dans la case mémoire dont l'adresse est l'octet situé dans le registre FSR. *C'est en langage 8051 l'équivalent de MOV @FSR,W*

LES INSTRUCTIONS

Les microcontroleurs PIC ont une structure RISC ,leurs instructions sont peu nombreuses, 35 pour le 16C84 .

Une particularité importante est l'absence d'instructions de saut conditionnel. Le déroulement du programme peut cependant être modifié par le résultat d'un calcul. Il existe des instructions de test et saut si zéro mais ce saut est limité à l'instruction suivante. Par exemple :
L'instruction n :
DECFSZ f,d décrémente le registre f et place le résultat dans W (si d=0) ou dans f (si d=1) mais saute à l'instruction n+2 si le résultat est nul , s'il n'est pas nul elle exécute normalement l'instruction n+1 .

```
n       DECFSZ f,d
n+1     Instruction suivante
n+2     instruction n+2
```

Saut si résultat 0

En choisissant correctement les instructions n+1 et n+2 il est possible de réaliser un saut conditionnel classique. Des exemples seront développés plus loin ;
Le saut inconditionnel s'appelle tout simplement GOTO adresse

Les instructions du 16C84 peuvent être classées en 3 groupes :
1° Les instructions mettant en oeuvre un registre interne f . Le code machine a dans ce cas la structure générale suivante (Mot de 14 bits) Figure ci contre :

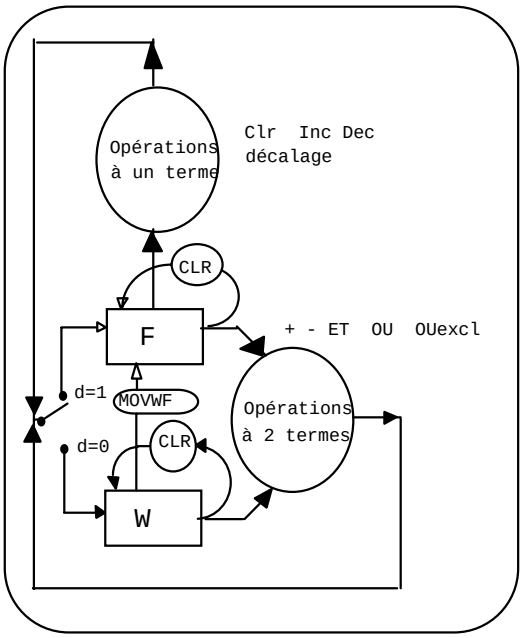
Code opération	Direction d	Adresse du registre f
6 bits	1bit	7 bits

2° Les instructions permettant l'accès aux bits isolés

Code opération	bits bbb	Adresse du registre f
4 bits	3 bits	7 bits

3° Les instructions de contrôle ou faisant intervenir W et un litéral k

Code opération	Literal k
6 bits	8 bits



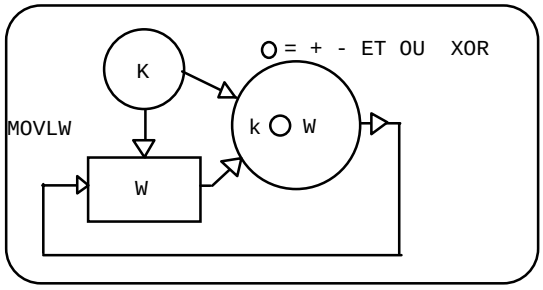
Chaque instruction initialise certains bits (FLAG) dans le registre d'état , par exemple le bit Z ou la retenue C .

Toutes ces instructions , à quelques exceptions près , s'exécutent en une phase machine c'est à dire 4 périodes du quartz .

Les tableaux ci dessous donnent des informations détaillées sur chacune de ces 35 instructions .

INSTRUCTIONS sur les REGISTRES f (Octets)					
Mnémonic	Description	Opération	Code machine 14 bits MSB	Bits d'état affectés	Notes
ADDWF f,d	Add W et f	W+f→d	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF f,d	AND W et f	W ∩f →d	00 0101 dfff ffff	Z	1,2
CLRF f	Clear f	0 → f	00 0001 1fff ffff	Z	2
CLRW -	Clear W	0 → W	00 0001 0xxx xxxx	Z	
COMF f,d	Complément f	/f → d	00 1001 dfff ffff	Z	1 , 2
DECF f,d	Décrément f	f -1 →d	00 0011 dfff ffff	Z	1 2
DECFSZ f,d	Décrément f Saut si 0	f-1 →d saut si 0	00 1011 dfff ffff		1,2, 3
INCF f,d	Incrément f	f+1 →d	00 1010 dfff ffff	Z	1,2
INCFSZ f,d	Incrément f Saut si 0	f+1 →d saut si 0	00 1111 dfff ffff		1,2, 3
IORWF f,d	OR W et f	W ∪ f → d	00 0100 dfff ffff	Z	1,2
MOVF f,d	Move f	f → d	00 1000 dfff ffff	Z	1,2
MOVWF f	Move W to f	W → f	00 0000 1fff ffff		
NOP	No opérations		00 0000 0xx0 0000		
RLF f,d	Rotate left through carry	Rotation vers la gauche du contenu de f → d	00 1101 dfff ffff	C	1,2
RRF f,d	Rotate right through carry	Rotation vers la droite du contenu de f → d	00 1100 dfff ffff	C	1,2
SUBWF f,d	Subtract W from f	f-W →d	00 0010 dfff ffff	C DC Z	1,2
SWAPF f,d	Swap halves f	permutation tétrades	00 1110 dfff ffff		1,2
XORWF f,d	Ou excl W,f	W ⊕ f→d	00 0110 dfff ffff	Z	1,2

INSTRUCTIONS avec LITERAL et de CONTROLE					
Mnémonic		Opération	Code machine 14 bits MSB	Bits d'état affectès	Notes
ADDLW k	Add literal to W	W+k→W	11 111x kkkkkkkk	C DC Z	
ANDLW k	AND literal to W	W∩k → W	11 1001 kkkkkkkk	Z	
CALL k	Call subroutine	PC+1→Pile k→PC(10-0) PCLATH(4-3) →PC(12-11)	10 0kkk kkkkkkkk		3
CLRWDT	Clear Watch Dog	RAZ Chien de garde	00 00000110 0100	/TO /PD	
GOTO k	Go to adresse	Saut inconditionnel	10 1kkk kkkkkkkk		3
IORLW k	W or k	W ∪k →W	11 1000 kkkkkkkk	Z	
MOVLW k	Move literal to W	k →W	11 00xx kkkkkkkk		
RETFIE -	Return from inter	Retour d'interruption	00 0000 0000 1001		3
RETLW k	Return with literal in W	k →W PILE →PC	11 01xx kkkk kkkk		3
RETURN	Return from subroutine	Retour SP	00 0000 0000 1000		3
SLEEP	Go into standby mode	Passage en mode sommeil	00 0000 0110 0011	/TO /PD	
SUBLW k	Subtract W from literal	k - W →W	11 110x kkkk kkkk	C CD Z	
XORLW k	Ou exclusif	k ⊕ W →W	11 1010 kkkk kkkk	Z	



INSTRUCTIONS de manipulation de BITS					
Mnémonic		Opération	Code machine 14 bits MSB	Bits d'état affectès	Notes
BCF f,b	Bit clear f	0 →bit b de f	01 00bb bfff ffff		1 2
BSF f,b	Bit set f	1 →bit b de f	01 01bb bfff ffff		1,2
BTFSC	Bit Test Skip if Clear	Test bit b de f Saut si zéro	01 10bb bfff ffff		3
BTFSS	Bit test Skip if 1	Test bit b de f Saut si 1	01 11bb bfff ffff		3

Notes:

1 Quand la sortie est modifiée par elle même , par exemple MOWF PORTB,1 la valeur utilisée est l'état réel de l'accès correspondant . Si par exemple l'état du latch associé est 1 mais que la broche soit forcée à 0 par un court circuit extérieur la valeur affectée par l'instruction est zéro.

2 Si l'instruction est exécutée sur le compteur du TIMER (RTCC ou TMR0) le pré compteur est remis à zéro si il est utilisé.

3 Si un saut est effectué, cette instruction nécessite deux cycles machine, le second est exécuté comme un NOP

LES OUTILS DE DEVELOPPEMENT

Pour s'initier à l'utilisation d'un microcontroleur il faut s'astreindre à écrire de nombreux programmes en s'inspirant par exemple des textes donnés plus loin. Mais ce n'est profitable que si l'on dispose des outils nécessaires à l'écriture et au test de ces programmes. Avant de passer aux exemples concrets nous consacrerons donc un chapitre à la description de ces outils.

La société MICROCHIP commercialise tous les produits nécessaires matériels (Assembleur, Simulateur) et matériels (Emulateurs, programmeurs) . Les outils logiciels de base sont disponibles sur le CDROM de la société et vendus pour quelques centaines de francs chez les fournisseurs , par exemple SELECTRONIC à Lille Comme nous le verrons le choix du 16C84 permet de n'utiliser qu'un programmateur très simple piloté par un logiciel du domaine public.

La société PARALLAX s'est spécialisée dans les outils de développement pour les microcontroleurs PIC , elle diffuse des produits originaux distribués par SELECTRONICs . Nous ne décrirons ici que les outils MICROCHIP .

L'ASSEMBLEUR

Le fichier source doit être écrit grâce à un éditeur de texte quelconque, par exemple le logiciel EDIT du DOS ou PE (Personnal Editeur ou Professional Editeur) d'IBM ou encore PE de Parallax . On doit trouver sur le WEB de nombreux éditeurs équivalents en shareware.

Le fichier texte XXX.ASM est ensuite assemblé par **MPALC** , l'assembleur de MICROCHIP. Ce logiciel est de structure tout à fait classique:

Pour l'appeler le format est : **MPALC nomdefichier [options**]
(*Dans ce qui suit comme c'est l'usage les parties entre [ ] sont optionnelles).*

Le nom de fichier est celui du fichier texte préalablement édité. Il doit comporter le suffixe .ASM .Lorsque l'assemblage est effectué le fichier objet obtenu doit être écrit dans un format compatible avec le programmateur et le simulateur. Quatre formats sont possibles

INHX16 qui organise le résultat en mots de 16 bits
INHX8S
INHX8M qui organisent le résultat en mots de 8 bits ,nous utiliserons de format .
et PICICE adapté à l'émulateur de MICROCHIP.

L'assembleur doit également savoir quel est le circuit de la famille qui est utilisé, ces informations peuvent lui être fournies à l'appel du programme ,par exemple :

MPALC A:essai1/p16C84/fINHX8M *(attention pas de blancs entre le nom du fichier et le /p)*

Mais il est plus simple de les indiquer au début du programme source , qui commencerait ainsi par :

```
TITLE ' Nom de Fichier '
LISt P=16C84,F=INHS8M        (ne pas oublier la virgule )
```

LE FORMAT DU FICHIER SOURCE

La ligne d'assembleur possède 4 champs :

- Le label , il peut avoir au plus 32 caractères ,doit commencer par une lettre ou _ sans blanc
- La commande qui est le mnémonique d'une instruction une directive d'assemblage ou une MACRO .
- L'opérande , si plusieurs sont nécessaires ils sont séparés par ,
- Les commentaires derrière le signe ;

Il n'y a pas de colonne prédéfinie dans la ligne , le logiciel reconnaît automatiquement le premier mot de la ligne comme un label ou une commande , par exemple :

```
START MOVWF,4       ; ligne 1
label   commande    commentaires
```

Les constantes (Literal) peuvent être présentées

En décimal	45 ou .45
En hexadécimal	0xF3
En octal	Q'173' ou O'777'
En binaire	B'00011101'
Un caractère A'Char' par exemple A'x'	pour le caractère x

,

LES DIRECTIVES DU LANGAGE

Ce sont des commandes qui apparaissent dans le texte source mais ne sont pas directement transcrites en mots codes.

Il y a 4 classes de directives :

1° Les directives concernant les données

DATA crée des données numériques ou alphabétiques (14 bits)

DATA <expression >[,<expression >]

DATA x07F3 place la constante 7F3 dans la ROM

Les expressions peuvent faire appel aux opérateurs arithmétiques ou logiques

+ - * /	les opérations élémentaires
>> et <<	décalages à droite et gauche
==	égalité logique
!=	Non égalité logique
<= >=	Plus petit ou plus grand que
!	non exemple !(x==y)
~	complément
|	ou inclusif exemple x|y
&	AND inclusif
|| &&	OU et ET logiques
^	OU exclusif
$	Valeur en cours du compteur programme exemple GOTO $

ZERO <nombre> met à zéro un nombre indiqué de cases de la mémoire programme

Nom EQU valeur donne à la désignation nom la valeur indiquée

```
expl                PORTA   EQU 5
```

SET Commande équivalente à la précédente mais sa valeur peut être modifiée en cours de programme On peut avoir par exemple successivement :

```
Debut       SET  6
Debut       SET Debut+4
```

INCLUDE '' fichier'' permet d'inclure un fichier source dans un autre . Par exemple on peut donner des noms à toute une série de bornes d'un circuit par des commandes EQU ,stocker cette liste dans un fichier et le réutiliser pour chaque nouveau programme .

2° Les directives de listing

Elles contrôlent le format du fichier .LST fourni par l'assembleur .

LIST < option>[,< option>] définit une série d'options de compilation dont deux ont déjà été citées

C=nombre nombre de colonnes du listing .LST
N=nombre nombre de lignes par page
R=HEX/DEC/OCT base de numération utilisée par défaut
P=16C54/16C55/16C56/.....16C84 choix du boîtier
F=PICICE/INHX16/INHX8S/INHX8M format du fichier .OBJ crée

TITLE ' nom du programme ' spécifie le titre du programme , ce n(est pas le nom du fichier

PAGE Force une nouvelle page dans le listing

3° Les directives de contrôle

ORG Début d'implantation du programme en mémoire

END Fin du programme (obligatoire)

IF ELSE ENDIF définissent une boucle conditionnelle

```
IF < expression> suivi des instructions à exécuter par exemple :
IF PORTA !=0
        MOVF 5,0        ;lecture du port A et transfert dans W
        MOVWF,6         envoi dans le port B
ELSE
        CLRF 6          ; RAZ port B
ENDIF
```

4° Les MACRO

Une MACRO est une séquence d'instructions qui peut être insérée dans un programme par simple appel de son nom .Elle apparaît dans le source comme une instruction.

Définition d'une macro

La définition est divisée en 3 parties , l'entête, le corps et la terminaison

L'entête est formée du nom (LABEL) du mot MACRO et des paramètres passés
Le corps est une suite d'instructions
La MACRO se termine par ENDM

Exemple : la MACRO suivante nommée CFL_JGE compare le contenu du registre FX à la constante CON et saute en SAUT si FX=> CON

```
CFL_JGE     MACRO       FX,CON,SAUT  l'entête
        MOVLW CON & 0xFF                    Ce qui suit MOVWF doit être un
literal d'ou l'opération qui fabrique le literal égal au contenu de CON , cette instruction met à 0 le carry
        SUBWF  FX,1                         Soustraction W-FX
        BTFSC STATUS,CARRY                  Saut si le carry est nul
        GOTO  SAUT                          exécutée si carry=1
ENDM
```

Si **une MACRO doit comporter des sauts internes**, les labels doivent être définis comme locaux sur la ligne qui suit immédiatement celle ou figure le mot MACRO. Le ENDM terminant la macro ne doit pas être précédé d'un label.
Exemple : La MACRO WC transfert le contenu du CARRY sur le fil 2 du port A :

```
WC  MACRO
LOCAL  ZEROW,FINW       ;définition locale des labels
BTFSS   STATUS,CARRY    ;préalablement définis saut si 1
GOTO ZEROW              ;exécutée si 0
BSF PORTA,2
GOTO FINW
```

```
ZEROW   BCF  PORTA,2
FINW
ENDM                    ;seul sur la ligne
```

Le PIC ne dispose pas directement de certaines instructions bien commodes sur le 8031 comme la manipulation de bits. Il est possible d'en disposer en les écrivant sous forme de macros .Par exemple :

Déplacement d'un octet d'une position à une autre :

MOVF1F2 **F1,F2** F1⇒ F2 Attention W est utilisé

```
MOVF1F2     MACRO       F1,F2
MOVF        F2,0
MOVWF       F1
ENDM
```

Transfert du contenu d'un registre dans W ,elle existe dans l'assembleur mais sous une forme moins intuitive .

MOWFW F F ⇒W

```
MOVFW       MACRO       F
 MOVF F,0
RETM
```

Moins évidente , le transfert de deux bits :

MOVB B1,B2 B1 ⇒B2

```
MOVB  B1,B2         MACRO       B1,B2
BTFSS       B1
BCF         B2
BTFSC       B1
BSF         B2
RETM
```

5° Les Pseudo instructions

Ce sont des instructions supplémentaires venant s'ajouter à celles du constructeur du CPU et qui facilitent l'écriture des programmes en effectuant des tâches que les instructions initiales n'ont pas prévu

Ces pseudo instructions sont commodes mais bien sûr non indispensables et de plus spécifiques de l'assembleur MPALC .

LES PSEUDO INSTRUCTIONS DE MPALC

Nom	Syntaxe	Opération	Bits d'ETAT affectès	Notes
Add carry to f	ADDCF f,d	BTFSC 3,0 INCF f,d	Z	Si C=1 F est incrémenté sinon saut à l'instruction suivante
Add Digit Carry to f	ADDDCFC f	BTFSC 3,1 INCF f,d	Z	Comme la précédente mais avec retenue intermédiaire .
BC Branch on Carry to adresse k	BC k	BTFSC 3,0 GOTO k		Si C=1 le saut est exécuté
BDC branch on digit carry to k	BDC k	BTFSC 3.0 Goto k		GOTO si retenue intermédiaire active .
B branch to k	B k	GOTO k		
BNC branch on no carry k	BNC k	BTFSS 3,0 GOTO K		Saut à k si C=0
BNDCK branch on no digit carry to k	BNDC k	BTFSS 3.1 GOTO k		Saut à k si DC=0
BNZ branch on no zéro to adress	BNZ k	BTFSC 3,2 GOTO k		Saut si résultat précédent non nul
Branch on zéro to k	BZ k	BTFSS 3,2 GOTO k		Saut si résultat précédent nul
Clear carry	CLRC	BCF 3,0		
Clear Zero	CLRZ	BCF 3,2		RAZ du bit Zero
Clear digit carry	CLRDC	BCF 3,2		
MOVFW	MOVFW,f	MOVF f,0		(F) $\rightarrow$ W
NEGF Negate file register contents	NEGF f,d	COMG f,1 INCF f,d		Complémente à 2 le contenu de f
Set Carry	SETC	BSF 3,0		
Set digit carry	SETDC	BSF 3,1		
Set Zero	SETZ	BSF 3,2		
Skip on carry	SKPC	BTFSS 3,0		Instruction suivante sautée si C=1
Skip on digit carry	SKPDC	BTFSS 3,1		
Skip on no carry	SKPNC	BTFSC		Saut si C=0
Skip on no digit carry	SKPNDC	BTFSC 3,1		
Skip on no zero	SKPNZ	BTFNC.3,2		
Skil on zéro	SKPZ	BTFSS 3.2		
Substract carry from file register	SUBCF,f,d	BTFSC 3,0		f est décrémenté si C=1
Substract digit carry from f	SUBDCF f,d	BTFSC 3,1 DECF f,d		
Test contents of F	TSTF f	MOVF f,1	Z	Teste le contenu de f et met le bit Z à 1 s'il est nul.

EXEMPLES

Exemple 1 : GENERATEUR DE SIGNAUX CARRES SUR LE PORT B

Le fichier source tapé sous éditeur est le suivant :

```
TITLE ' Générateur de signaux carrès rapides sur port B '
LIST P=16C84,F=INHX8M,R=DEC ;numération décimale par défaut
;
PORTB    EQU 6          ;définition des registres et position bit
TRISB    EQU 6
STATUS   EQU 3
RS0      EQU 5

BSF  STATUS, RS0        ;saut dans la page 1 pour accéder au registre TRISB
CLRF  TRISB             ;Portb en sortie
BFF  STATUS,RS0         ; bit RS0 de STATUS au 0 retour dans la page 1
;
CLRF  PORTB             ;valeur de départ 0
BOUCLE   INCF PORTB,1          ; incrémentation du port B et résultat dans PB
GOTO   BOUCLE
;
END                     ; indication de fin de programme ,nécessaire .
```

Ce programme source est sauvé avec le nom GENESCPB.ASM le suffixe ASM est obligatoire

L'assemblage s'effectue ensuite en appelant :

```
MPALC  GENESCPB.ASM
```

L'assembleur trace 4 ou 5 lignes à l'écran pour indiquer que l'assemblage à pris n secondes et s'est effectué sans incidents . Plusieurs fichiers importants ont été crés : .LST .OBJ et .SYM

Le premier GENESCPB.LST est le fichier d'assemblage dans lequel sont indiquées les erreurs s'il y en a

GENESCPB.LST

```
-----------------------
16c5x/xx Cross-Assembler V4.14 Released  Thu Mar 19 15:29:51 1998  Page 1
 Générateur de signaux carrès rapides sur PORTB

Line   PC    Opcode

0001                LIST P=16C84,F=INHX8M
0002           ;
0003                0006  PORTB      EQU 6
0004                0006  TRISB      EQU 6
0005                0003  STATUS     EQU 3
0006                0003  RS0        EQU 5
0007           ;
0008                0000  1683  BSF          STATUS,RS0
0009                0001  0186  CLRF         TRISB
0010                0002  1283  BCF STATUS,RS0
0011           ;
0012                0003  0186  CLRF         PORTB
0013                0004  0A86  BOUCLE     INCF     PORTB,1
0014                0005  2804  GOTO         BOUCLE
0015           ;
```

16c5x/xx Cross-Assembler V4.14 Released Thu Mar 19 15:29:51 1998 Page 2

Cross-Reference Listing

LABEL	VALUE	DEFN	REFERENCES
BOUCLE	4	13	13 14
PORTB	6	3	3 12 13
RS0	3	6	6 8 10
STATUS	3	5	5 8 10
TRISB	6	4	4 9

Le second est la table des symboles

GENESCPB.sym

```
3BOUCLE 004L00400500D9
3PORTB 006F0000030040025
3RS0 303B0000000020046
3STATUS 003F000000002004D
3TRISB 006F000001001F
```

Le dernier est le fichier objet qui sera utilisé par le programmateur ; Suivant le type de format choisi il présente une forme différente ,avec le format INHX8M les octets sont organisès en octets :

GENESCPB.OBJ

```
:0C0000008315860183118601860A0428FE
:00000001FF
```

On notera au début de la première ligne 0C i,diquant que les bits dans la mémoire programme sont dècrits par 12 (0CH) octets (Soulignès) (Ce sont en réalité des mots de 14 bits) Le dernier octet de la ligne ,ici FE ,est le checksum
Si on avait utilisé le format INHX16 on aurait eu

GENESCPB.OBJ

```
:06000000158301861183018601860A8628040 4
:00000001FF
```

Les bits sont rangès dans un autre ordre ,remarquer les 4 premiers caractéres 1583 ua lieu de 8315, et ils sont groupès en mots de 16 bits (4 caractéres) ,il n'y en a que 6 dans la ligne d'ou le 06 en tête.

Exemple 2 : UTILISATION D'UNE MACRO

Nous reprendrons le même exemple mais en faisant partir le comptage de 80H et non zéro. L'instruction d'nitialisation CLRF PORTB ne convient plus , il faut charger PORTB avec 80H avant de lancer le comptage,malheureusement l'instruction :

MOWLF f,k qui chargerait le litéral k dand le registre f n'existe pas .Bien sûr il est facile d'écrire :

```
MOVLW 80H    (la notation 80H est acceptée , on conseille plutot 0x80 )
MOVWF PORTB
```

mais il est également possible de créer l'instruction manquante sous forme d'une MACRO .Soit

```
MOVLF  MACRO f,k        le nom MOVLF , le mot MACRO et les paramétres f et k
MOVLW k                 k dans W
MOVWF f                 W dans f
ENDM                    fin de MACRO
```

L'appel de la MACRO se fait ensuite simplement par son nom :Le fichier ASM est le suivant :

GENESCMC.ASM

```
TITLE ' Générateur de signaux carrès rapides sur port B'
LIST P=16C84,F=INHX8M
;
PORTB EQU 6
TRISB  EQU 6
STATUS          EQU 3
RS0     EQU 5
;
MOVLFMACRO          f,k        ;définition de la MACRO MOVLW f,k
MOVLW          k
MOVWF          f
ENDM
;
BSF     STATUS,RS0
CLRF    TRISB
BCF     STATUS,RS0
;
MOVLFPORTB,80H      ;appel de la MACRO
BOUCLE          INCF PORTB,1  ;incrément PORTB
GOTO BOUCLE
;
END
```

Et le fichier d'assemblage .LST

```
16c5x/7x Cross-Assembler V4.12 Released  Thu Mar 19 19:22:51 1998  Page 1
 Générateur de signaux carrès rapides sur port B

Line   PC    Opcode

0001            LIST P=16C84,F=INHX8M
0002            ;
0003                 0006  PORTB    EQU 6
0004                 0006  TRISB    EQU 6
0005                 0003  STATUS   EQU 3
0006                 0003  RS0      EQU 3
0007            ;
0008            MOVLF     MACRO          f,k        ;définition de la MACRO MOVLW f,k
0009            MOVLW     k
0010            MOVWF     f
0011            ENDM
0012            ;
0013                 0000  1683  BSF STATUS,RS0
0014                 0001  0186  CLRF          TRISB
0015                 0002  1283  BCF STATUS,RS0
0016            ;
0017            MOVLF     PORTB,80H      ;appel de la MACRO
0018                 0003  3080 mMOVLW    80H
0019                  0004  0086 mMOVWF    PORTB
0020                  0005  0A86  BOUCLE    INCF PORTB,1  ;incrément PORTB
0021                 0006  2805  GOTO BOUCLE
0022            ;
0023                 0000  END

16c5x/7x Cross-Assembler V4.12 Released  Thu Mar 19 19:22:51 1998  Page 2

Cross-Reference Listing
LABEL       VALUE       DEFN       REFERENCES
```

```
BOUCLE      5          20         20     21
MOVLF       7          8          8      17
PORTB       6          3          3      17     19     20
RS0         3          6          6      13     15
STATUS      3          5          5      13     15
TRISB       6          4          4      14
```

Noter aux lignes 18 et 19 la lettre m indiquant que l'instruction provient du développement d'une MACRO .

Exercice 3 ;REALISATION D'UNE BOUCLE DE RETARD .Chenillard lent sur port B

Un retard long est necessaire à chaque fois qu'une lecture par un observateur est souhaitée .Pour réaliser ce retard le programme suivant utilise deux cases dans la RAM interne .

CHENILPB.ASM

```
TITLE ' Chenillard lent sur port B '
LIST P=16C84,F=INHX8M
;
PORTB EQU 6
TRISB EQU 6
STATUS EQU 3
RP0 EQU 5
COMPTE1 EQU 47              ;registre utilisés pour le délai
COMPTE2 EQU 46
;
BSF STATUS,RP0              ;accès à la page 0 et  programation de TRISB
CLRF TRISB
BCF STATUS,RP0
;
CLRF PORTB
INCF PORTB,1  ;             départ avec 1 dans PORTB
NOUVEAU RLF PORTB,1         ;décalage à gauche via le Carry
CALL DELAY                  ;appel du sous programme
GOTO NOUVEAU
;
DELAY                       ;le sous programme
CLRF COMPTE1
CLRF COMPTE2
BCL1 DECFSZ COMPTE2,1       ;décrément et 'skip'  si zéro
GOTO BCL1
DECFSZ COMPTE1,1
GOTO BCL1
RETURN
;
END
```

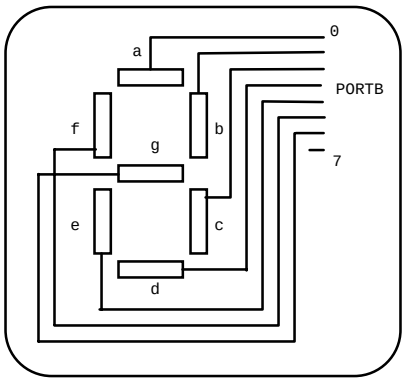
Exercice 4 UTILISATION DE L'INSTRUCTION RETLW

Cette instruction n'a pas d'équivalent direct avec le 8031 ,elle peut être très pratique comme le montre l'exemple ci dessous.

Ce programme examine les 4 bits appliquès sur le port A qui représentent un caractére hexadécimal H (de 0 à F) et envoie sur le port B le mot convenable pour afficher sur un afficheur 7 segments qui y est relié le caractére H . L'afficheur est cablé comme le montre la figure suivante . Les mots code à appliquer sont alors donnès sur le tableau ci joint.

Caract-ére	Code 7seg						
0	7F	**4**	66	**8**	7F	**C**	39
1	06	**5**	6D	**9**	67	**D**	5E
2	5B	**6**	7D	**A**	77	**E**	79
2	4F	**7**	07	**B**	6C	**F**	71

Le programme est alors le suivant :

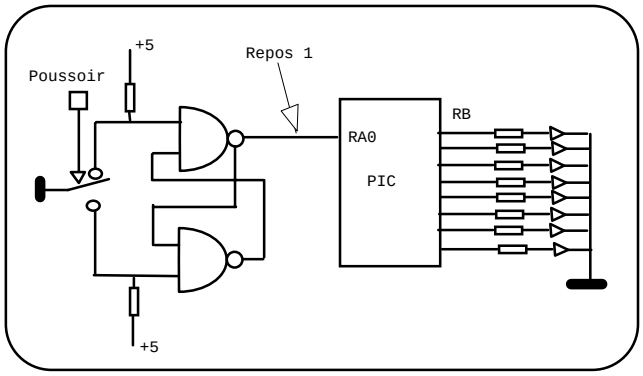



```
Désignation du processeur et du format
Désignation des registres et bits
Port A en entrée
Port B en sortiee

AFFICH      MOVF PORTA,0      ;lecture du port a et char
                              gement dans W
ANDLW       0x0F              .Masquage des 4 bits de
                              plus fort poids
CALL        CONVERT           ;appel du sous programme
                              de conversion
;
CONVERT     ADDWF  PC         ;le contenu de W que l'on vient de lire  sur le port A est ajouté au contenu du compteur programme. L'instruction suivante qui sera exécutée par le CPU est donc,la suivante si le caractére est 0 , la seconde si le caractére est 1 etc...
RETLW       0x7F              ;Le  saut de retour est effectué  mais avec un contenu de W qui est le code 7 segments du caractére , ici 0 .
RETLW       06H               ;même chose mais avec le code 7 segments de 1 etc...
RETLW       5BH
RETLW       4FH
RETLW       66H
RETLW       6DH
RETLW       7DH
RETLW       07H
RETLW       7FH
RETLW       67H
RETLW       77H
RETLW       6CH
RETLW       39H
RETLW       5EH
RETLW       79H
RETLW       71H
END
```

Exercice 5 UTILISATION DU TIMER . Intervalmétre

Au départ les leds reliées au port B sont éteintes. Le programme attend un niveau bas sur le fil RA0 ,dès que ce niveau est détecté le timer est lancé , via le prédiviseur réglé sur /255 , L'état du compteur est noté lorsque le signal RA0 remonte à 1 et sa valeur affichée sur le port B . La valeur affichée est donc la durée du top sur RA0.Ce montage serait intéressant pour étudier les rebondissements de contact sinon il faut pour les éviter utiliser une bascule RS comme sur la figure ci contre.




```
Title 'intervalmetre'
List P=16C84,F=INHX8M
TMR0   EQU 1
COMPTEUR EQU 1              ;le mot OPTION est interdit car c'est le nom d'une instruction du 16C54
STATUS EQU 3
RP0    EQU 5
TRISA  EQU 5
PORTA  EQU 5
```

```
TRISB  EQU 6
PORTB  EQU 6
DEPART EQU 2F
STOP   EQU 2E
;
BSF     STATUS,RP0          ;accès page 1
MOVLW  2                    ;prediviseur par 8 ,oscillateur fosc/4
MOVWF COMPTEUR
CLRF  TRISB                 ;port B en sortie
MOVLW         0xFF
MOVWF TRISA                 ;port A en entrée (inutile il l'est au reset)
BCF     STATUS,RP0          ;retour page 0
;
ATTENT MOVF TMR0,0          ;lecture timer dans W
BTFSC PORTA,0               ;test de la broche RA,0
GOTO ATTENT                 ;executee si non 0
;
MOVWF         DEPART                ;etat du compteur au moment de  l'appui
BOUTON   BTFSS       PORTA,0                ;nouveau test bouton SKIP si 1
GOTO BOUTON                 ;boucle  tant que le bouton n'est pas activé
;
MOVF  TMR0,0                ;nouvelle valeur dans W Noter état du compteur lorsque le bouton est
                            ;relaché
MOVWF         STOP          ;dans stop
MOVF  DEPART,0              ;valeur initiale dans W
SUBWF         STOP,0        ;depart-stop dans W
MOVWF   PORTB               ;affichage de la différence
REPOS   GOTO REPOS          ;boucle sans fin.Pour refaire une mesure il faut relancer le pro
                            ;gramme.
;
END
```

LE SIMULATEUR LOGICIEL MPSIM

MPSIM est un simulateur de logiciel qu'il ne faut pas confondre avec un émlulateur qui est un système matériel simulant en temps réel le comportement du circuit.

Exécutant ,de plusieurs ordres de grandeur plus lentement que le circuit réel , le logiciel à tester il permet de suivre au cours de l'exécution l'état de chacun des registres internes (Ce que permet l'émulateur mais pas le circuit lui même) Il est également possible d'introduire des signaux extérieurs sur les ports A et B à un instant donné ou avant une instruction donnée.

MPSIM joue aussi le rôle d'éditeur, la modification d'une instruction ou l'introduction de nouvelles lignes de programme est possible à tout moment , et une nouvelle exécution peut être relancée. Il est possible de placer dans le programmes des points d'arrêt et de visualiser ou d'imprimer l'évolution du contenu des diverses cases mémoires internes .

Ce logiciel tournant sur n'importe quel PC sous DOS simule le fonctionnement de l'unité centrale de l'un quelconque des boîtiers PIC16CXX. (Dans le cas du 16C84 l'EEPROM interne n'est cependant pas prise en compte avec la version actuelle de MPSIM) Son affichage en mode texte peut être considéré actuellement comme austère mais il est réellement efficace .

L'appel du programme se fait en tapant MPSIM ,l'écran de départ est alors par exemple le suivant :

```
RADIX=X         MPSIM 4.14              16C55           TIME=0.00µ      0
-------------------------------------------------------------------------------------------------------------------------------
W:  00  F1:00  F2:100  F3:00    F4:00   F5:00   F6:00   F7:00


-------------------------------------------------------------------------------------------------------------------------------
% SR X
% ZR
% ZT
% RE
% V  W,X,2
% AD  F1,X,2
% AD  F2,X,2
% AD  F3,X,2
% AD  F4,X,2
% AD  F5,X,2
% AD  F6,X,2
% AD  F7,X,2
rs
Processeur Reset
%  -
```

Cet écran est du au chargement automatique au lancement d'un fichier MPSIM.INI.

La première ligne indique la base numérique choisie par défaut , RADIX=X hexadécimal, la version de MPSIM, le type de processeur , le temps écoulé depuis le début du programme (TIME) et enfin le nombre d'instructions exécutées.

En dessous du trait, une liste de registres internes et leur contenu

Puis une suite d'instructions qui sont celles contenues dans le fichier MPSIM.INI

La première ligne SR X indique que les données numériques seront écrites en hexadécimal par défaut.

ZR remet à zéro tous les registres

ZT remet à zéro l'horloge qui compte le temps (En haut à droite de l'écran)

RE remet à zéro tous les compteurs

V (Visual) W,X,2 commande l'affichage du contenu de W en héxa sur 2 caractères

Les lignes suivantes AD commandent l'affichage du contenu des registres F , en hexadécimal sur 2 caractères (X,2) ces fichiers pourraient être désignés par leur nom si un tel nom a été défini dans le fichier .ASM préalablement chargé.

ATTENTION : *Les adresses dans PCL ne sont définies que sur 7 bits , le huitième étant l'état de RP0 , MPSIM n'exploite pas ce 8ierme bit ; les déclarations de noms de registres ne sont donc valables que dans la page 0 . Ainsi si on a défini par exemple TRISA EQU 5 et PORTA EQU 5 MPSIM considérera que c'est le même registre . Pour accéder aux registres de la page 1 il faut utiliser les noms réservés reconnus par MPSIM :*

Fxx ou xx en héxa de 00 à AF
OPT ou F81 pour le registre d'option (noter que le mot OPTION est réservé)
IOA et IOB ou F85 et F86 pour TRISA et TRISB

Le tiret clignotant terminant la dernière ligne indique que le logiciel attend une entrée . Le détail des commandes est donné dans les pages suivantes .

Nous choisirons pour développer des exemples d'application le programme de chenillard sur le port B décrit dans le chapitre précédent avec cependant une modification pour limiter la durée de la boucle de retard.

CHENILPB.ASM

```
;
TITLE ' Chenillard lent sur port B '
LIST P=16C84,F=INHX8M
;
PORTB EQU 6
TRISB EQU 6                ;voir remarque ci dessus , ce nom ne doit pas êtrte utilisé dans MPSIM
STATUS EQU 3
RP0 EQU 5
COMPTE1 EQU 2F             ;registre utilisés pour le délai
COMPTE2 EQU 2E
;
BSF STATUS,RP0             ;accès à la page 0 et programmation de TRISB
CLRF TRISB
BCF STATUS,RP0
;
CLRF PORTB
INCF PORTB,1     ;                 départ avec 1 dans PORTB
NOUVEAU  RLF PORTB,1   ;décalage à gauche via le Carry
CALL DELAY                 ;appel du sous programme
GOTO NOUVEAU
;
DELAY                      ;le sous programme
MOVLW  5
MOVWF  COMPTE1
MOVLW 10
MOVWF  COMPTE2             ;pour déterminer la durée du retard
BCL1 DECFSZ COMPTE2,1 ;décrément et 'skip' si zéro
GOTO BCL1
DECFSZ COMPTE1,1
GOTO BCL1
RETURN
;
END
```

LES FONCTIONNALITES DE MPSIM

Procédures de chargement et de sauvetage
Visualisation du programme et modification
- Mémoire programme
- Registres et ports
- Timers
- Fonctions d'affichage à l'écran
- Table des symboles
- Table des modifications (patch table)

Exécution et suivi de l'évolution des contenus (Tracing)
Modification du contenu de l'écran
Commandes diverses

Toutes ces commandes peuvent être contenues dans un fichier chargé ou introduites 'à la main ' sur le tiret clignotant en bas de l'écran .

Chargement :

Avant de commencer une simulation le fichier source nom.ASM doit être chargé par la commande :

LO Nomdefichier.ASM ,format

par exemple dans notre cas : LO CHENILPB.ASM INHX8M

Pendant la simulation ce fichier initial peut être modifié, la version modifiée sera sauvée par :

O nomdefichier format

L'écran initial configuré par le chargement automatique de MPSIM.INI n'est sans doute pas celui que vous souhaitez. Il faut alors écrire sous éditeur un fichier .INI , ici CHENILPB.INI puis de le charger en tapant à l'invite de MPSIM , en bas de l'écran sur le tiret clignotant :

GE nomdefichier.INI

soit GE CHENILPB.INI

Pour suivre l'évolution de notre programme nous demanderons l'affichage du contenu de PORTB (en binaire pour suivre le déplacement du 1) du registre d'état STATUS et des deux registre du DELAy COMPTE1 et COMPTE2. .Le fichier CHENILPB.INI peut alors être le suivant .

Pour suivre le déroulement de ce programme il est intéressant de visualiser le contenu du port B , du registre d'état qui contient le Carry en position 0 (Nous l'afficherons en binaire) ainsi que l'état des registres COMPTE1 et 2 pour contrôler le fonctionnement de la boucle de retard.

Notre programme CHENILPB.INI sera par exemple le suivant :

NV		Efface l'écran
P	84	Préciser le type de CPU
SR	X	Hexa par défaut
LO	CHENILPB.ASM INHX8M	Pour charger le programme et indiquer le format ,INHX16 est par défaut
DW	D	Désactivation du WD ,utile pour des boucles de longue durée
V	W,X,2	Pour visualiser W ,ce n'est pas utile ici
AD	PORTB ,B	Le nom a été défini dans .ASM affiché en binaire
AD	STATUS,B	Le registre d'état avecC en position 0
AD	COMPTE1,X,2	
AD	COMPTE2,X,2	Les compteurs
SC	1.25	Ceci définit en microsecondes le temps de cycle , 1,25µS pour le quartz de 3,2Mhz utilisé. Le temps d'exécution sera affiché avec sa valeur réelle.
ZR		Met à 0 le contenu des registres au départ
RE		RAZ du compteur de temps d'exécution et d'instructions
RS		Reset processeur

Il est tapé sous éditeur (sans les commentaires) et sauvé avec son nom CHENILPB.INI

Au lancement MPSIM l'écran initial précédent est d'abord présenté. On tape alors comme indiqué plus haut :

GE CHENILPB.INI

Le programme charge le fichier INI et l'écran devient :

CHENILPB RADIX=X MPSIM 4.11 16C84 TIME:0 µS 0

```
W:00    PORTB:11111111      STATUS:00011000     COMPTE1:00   COMPTE2:00

Listing file loaded
Symbol table loaded
321296 bytes free
%SC    1.25
%DW    D
%SR    X
%V W,X,2
%AD    PORTB,B
%AD    STATUS,B
%AD    COMPTEUR1
%AD    COMPTEUR2
%ZR
%RE
% RS
Reset processeur
321296 bytes free
% -
```

On notera que le reset processeur amène le port B à 1 et le STATUS à 00011000
Le chargement étant effectué la simulation peut commencer.

Exécution et trace

E lance à partir de la position actuelle du compteur programme alors que **G0** lance à partir de 0 . Si aucun point d'arrêt n'a été placé l'exécution continuera jusqu'à ce que l'on presse une touche .

Pour un début il vaut mieux utiliser l'exécution pas à pas **SS**

SS exécute une instruction à la position courante du compteur.

SS nn exécute une instruction à partir de l'adresse nn (ce peut être un label)

Un nouvel appui sur la touche ENTREE ↵ renouvelle la commande précédente qu'il n'est pas nécessaire de réécrire.

Tapons donc SS

il apparaît en bas de l'écran :

0001 0186 CLRF TRISB

L'instruction 0 BSF STATUS,RP0 a été exécutée comme on peut le voir en examinant l'état du bit 5 du STATUS en haut de l'écran et la prochaine instruction est affichée La ligne indique outre le mnémonique , le code machine (mot de 14 bits représenté comme 2 octets) et le numéro de l'instruction.

Une frappe sur ↵ exécute cette instruction et affiche la seconde :

0002 1283 BCF STATUS,RP0

Le compteur de temps TIME= indique le temps écoulé 2.50µS (A 3,2Mhz d'horloge chaque instruction dure 1,25µS) et le nombre d'instructions déjà exécutées 2 .

On peut ainsi continuer aussi longtemps qu'on le jugera utile.. Mais lors de l'exécution d'une boucle un groupe d'instructions peur être traité un grand nombre de fois et la méthode est laborieuse De plus si le programme est long ,l'écran n'ayant qu'une taille limitée il faudra noter les informations au fur et à mesure de leur affichage pour étudier ensuite l'ensemble. Ceci peut être réalisé de façon automatique par la fonction TRACE.

TC 3 par exemple lance l'exécution de 3 instructions à partir de l'état du compteur programme et affiche à l'écran les résultats /

Pour notre exemple , à partir de l'instruction 2

TC 3 affiche :

```
% TC 3
0002  1283  BCF  STATUS,RP0        3.75µ  3  STATUS:1C
0003  0186  CLRF PORTB       5.00µ  4  PORTB:0     F3:IC
0004  0A86  INCF PORTB,1           6.85µ  5  PORTB:1     F3:18
```

Sur chaque ligne sont indiqués N° instruction, mnémonique , code machine, temps et nombre d'instruction écoulées , mais aussi les registres internes qui ont été modifiés Lorsque l'on s'intéresse à un registre particulier on peut le préciser . En faisant précéder les commandes de trace de :

TR nomdefichier

par exemple : TR PORTB Seul le port B est affiché à chaque modification . (Dans la base sélectionnée par SR H -D ou O) TR sans nom de fichier trace tous les fichiers .

Une telle liste sera très utile pour détecter un dysfonctionnement ,elle peut être imprimée ou stockée en fichier par la commande :

TF [Nomdefichier/prn]

qui doit être entrée avant la commande TC précédente .
Par exemple:

```
TF    CHENILPB.TRA
TC    10
```

trace 10 instructions à l'écran et aussi dans un fichier .TRA :

```
0004 0A86 INCF PORTB,1   ; départ avec 1 dans |6.25µ 5 |PORTB:1 F3:18
0005 0D86 NOUVEAU RLF PORTB,1   ;décalage à g |7.50µ 6 |PORTB:2 F3:18
0006 2008 CALL DELAY          ;délai       |10.00µ 7 |[ 7 ]
0008 3005 MOVLW 5                    |11.25µ 8 |W:5
0009 00AF MOVWF COMPTE1  ;pour définir la dur |12.50µ 9 |F2F:5 F3:18
000A 3010 MOVLW 10                   |13.75µ 10 |W:10
000B 00AE MOVWF COMPTE2              |15.00µ 11 |F2E:10 F3:18
000C 0BAE BCL1 DECFSZ COMPTE2,1      ;décré |16.25µ 12 |F2E:F F3:18
000D 280C GOTO BCL1                  |18.75µ 13 |
000C 0BAE BCL1 DECFSZ COMPTE2,1      ;décré |20.00µ 14 |F2E:E F3:18
000D 280C GOTO BCL1                  |22.50µ 15 |
000C 0BAE BCL1 DECFSZ COMPTE2,1      ;décré |23.75µ 16 |F2E:D F3:18
000D 280C GOTO BCL1                  |26.25µ 17 |
000C 0BAE BCL1 DECFSZ COMPTE2,1      ;décré |27.50µ 18 |F2E:C F3:18
000D 280C GOTO BCL1                  |30.00µ 19 |
000C 0BAE BCL1 DECFSZ COMPTE2,1      ;décré |31.25µ 20 |F2E:B F3:18
```

TF Prn aurait directement envoyé ce fichier à l'imprimante
On sort du mode TF en tapant la commande TF seule .

Lorsque l'exécution met en oeuvre un grand nombre d'instructions comme c'est le cas pour une routine de retard , la méthode précédente n'est pas intéressante car elle aboutit à d'énormes fichiers redondants. Il est alors plus intéressants de placer des points d'arrêts et de lancer le programme entre ces points .

Pose de points d'arrêts

Reprenons le fichier précédent et relançons MPSIM . (On peut aussi revenir au début en tapant RE ↵ RS ↵)

Après avoir introduit la commande GE définissons un point d'arrêt par

B DELAY

Un point d'arrêt (Breakpoint) est défini sur le Label DELAY

Lançons alors le programme par :

E↵

L'écran affiche **: 0009 00AF MOVWF COMPTE1**

c'est bien l'instruction qui suit le premier DELAY et le compteur affiche 11.25µS 8 Le programme mettra donc 11,25µS à atteindre cette instruction ,8 ayant déjà été éxécutées. Le port B est alors dans l'état 00000010.

Si on frappe de nouveau ↵ la commande E est de nouveau exécutée et l'écran indique de nouveau : **0009 00AF MOVWF COMPTE1** mais le compteur de temps est passé à 3936.25µS 2100 2100 Instructions sont exécutées pendant la boucle qui dure :
3936,25-11,25=3925 µS

Pour supprimer le poit d'arrêt introduire **BC** (tous les BP sont supprimés , ou BP DELAY pour supprimer ce seul BP)

Au maximum 256 points d'arrêt peuvent être définis .

La commande **DB** affiche les BP actifs
Un BP peut être défini par la valeur d'une variable ,par exemple :

B PORTB=80H
défini un point d'arrêt lorsque le PORTB atteint la valeur 80H .

Dans notre cas :

BP ↵ Supprime les BP
B PORTB= 0x80 ↵ définit le nouveau BP

ATTENTION / Il y a un blanc entre la commande B ,l'adresse PORTB ;le signe= , et enfin entre ce signe = (ce peut être > < >= <= !=) et la valeur numérique De plus en héxa le seul format accepté est 0x80 (et non 80H)

En repartant du début lançons E .

On voit avancer le compteur ainsi que le bit 1 dans le PORTB et l'exécution s'arrête ,sur la même ligne 9 avec un temps d'exécution affiché de 23557,50µS et 12558 instructions .

Lorsque l'exécution s'arrête sur un BP il est possible de la reprendre en tapant E ,elle stoppe alors au BP suivant . La commande **C n** permet de sauter les n BP suivants

Fichiers de stimuli

Il est possible de simuler une action extérieure sur les ports en chargeant un fichier de STIMULI . nomdefichier.STI Ce fichier écrit sous éditeur est chargé par la commande :

ST nomdefichier.STI

A titre d'exemple prenons programme INTERV décrit au chapitre précédent avec pour simplifier la simulation une prédivision par 8 seulement.

```
Title 'intervalmetre'
List P=16C84,F=INHX8M
TMR0   EQU 1
COMPTEUR EQU 1
STATUS EQU 3
RP0    EQU 5
TRISA  EQU 5
PORTA  EQU 5
TRISB  EQU 6
PORTB  EQU 6
DEPART EQU 2F
STOP   EQU 2E
;
BSF  STATUS,RP0       ;accès page 1
MOVLW  2              ;prediviseur par 8 ,oscillateur fosc/4
MOVWF COMPTEUR
CLRF  TRISB   ;port B en sortie
MOVLW         0xFF
MOVWF TRISA           ;port A en entrée
BCF  STATUS,RP0       ;retour page 0
```

```
;
ATTENT MOVF TMR0,0  ;lecture timer dans W
BTFSC          PORTA,0          ;test de la broche RA,0
GOTO ATTENT                     ;executee si non 0
MOVWF          DEPART           ;etat du compteur à l'appui
BOUTON   BTFSS         PORTA,0          ;nouveau test bouton
GOTO BOUTON
MOVF           TMR0,0           ;nouvelle valeur dans W
MOVWF          STOP             ;dans stop
MOVF           DEPART,0         ;valeur initiale dans W
SUBWF          STOP,0           ;depart-stop dans W
MOVWF   PORTB                   ;affichage
REPOS   GOTO REPOS ;boucle sans fin
;
END
```

Pour la simulation nous afficherons à l'écran les deux ports A et B .D'ou par exemple le fichier .INI suivant :

INTERV.INI

```
NV
SC 1                            ;quartz 4mhz  durée de cycle 1µS
P 84
SR X
LO INTERV.ASM  INHX8M
DW D
V W,X,2
AD PORTA,B
AD PORTB,B
AD OPT,B
AD IOA
AD IOB
AD STATUS,B
AD TMR0
AD  DEPART
AD STOP
ZR
RE
RS
```

Après lancement de MPSIM puis GE INTERV.INI la première ligne de l'écran affiche l'état initial des registres :

W:00 PORTA:00011111 PORTB:11111111 OPT:11111111 IOA=1F
IOB:FF STATUS:00011100 TMR0:00

- La frappe de SS lance l'instruction 0 et le bit 5 de STATUS passe à 1 . Lançons l'exécution de 20 instructions par : **TF PRN** ↵
TC 20 ↵

L'imprimante nous délivre :

```
0000 1683 BSF     STATUS,RP0            |1.00µ 1 |STATUS:38
0001 3002 MOVLW  2                     |2.00µ 2 |W:2
0002 0081 MOVWF COMPTEUR                |3.00µ 3 |RTCC:2 F3:38
0003 0186 CLRF  TRISB                  |4.00µ 4 |PORTB:0 F3:3C
0004 30FF MOVLW   0xFF                 |5.00µ 5 |W:FF
0005 0085 MOVWF TRISA                  |6.00µ 6 |PORTA:FF F3:3C
0006 1283 BCF     STATUS,RP0           |7.00µ 7 |STATUS:1C
0007 0801 ATTENT MOVF TMR0,0            |8.00µ 8 |W:0 F3:1C
0008 1805 BTFSC   PORTA,0              |9.00µ 9 |
0009 2807 GOTO ATTENT                  |11.00µ 10 |
.....
----------
.....
0009 2807 GOTO ATTENT                  |31.00µ 25 |
```

```
0007 0801 ATTENT MOVF TMR0,0          |32.00µ 26 |W:3 F3:18
0008 1805 BTFSC   PORTA,0             |33.00µ 27 |
0009 2807 GOTO ATTENT                 |35.00µ 28 |
0007 0801 ATTENT MOVF TMR0,0          |36.00µ 29 |W:4 F3:18
0008 1805 BTFSC   PORTA,0             |37.00µ 30 |
0009 2807 GOTO ATTENT                 |39.00µ 31 |
0007 0801 ATTENT MOVF TMR0,0          |40.00µ 32 |W:4 F3:18
```

On peut observer que W change d'état toutes les 8 µS ce qui correspond bien au taux de division programmé . Evidemment RA0 restant à 1 le programme comme prévu tourne dans la boucle ATTENT .

Pour tester réellement le programme il faut injecter un signal sur RA0 , ce qui est réalisé avec un fichier de stimuli .

Un tel fichier peut par exemple être le suivant :

INTERV.STI

```
STEP        RA0         ! un seul signal extérieur est utilisé
15          0           ! le numéro de l'état doit être en décimal
63          1
```

Ce fichier est tapé sous éditeur et sauvé avec son nom .STI Notez qu'un commentaire s'introduit derrière ! .

Dans le fichier .INI on ajoute alors la ligne : ST INTERV.STI

INTERV.INI

```
.....
LO INTERV.ASM  INHX8M
ST INTERV STI
.....;
```

En lançant MPSIM puis GE INTERV.INI et SS on observe que le bit 0 de PORTA passe à 0 à l'étape 15 alors que la durée d'exécution est de 18µS . A l'étape 16 la valeur initiale du compteur 1 est chargée dans DEPART.

Le bit 0 de PORTA revient à 1 à l'étape 63 alors que le temps écoulé est de 90µS. La valeur finale du compteur 0B est chargée dans STOP à l'étape 67 , et la durée 0B-01=0A affichés sur le port B . C'est bien la durée pendant laquelle le PORTA .0 était bas ;. (90-18)/8=9

Les fonctions d'édition , modifications du programme .

MPSIM permet la modification du programme sans passer par un éditeur .Les deux instructions de base sont:

IA _adresse

Par exemple IA 00 affiche :

```
0  :  BSF  STATUS.RP0
:
```

c'est bien la première instruction du programme .Deux points apparaissent à la ligne suivante, il suffit de taper à la suite la nouvelle instruction qui remplacera l'ancienne

Si on frappe seulement ↵ c'est la ligne suivante qui est éditée.

L'instruction **M Adresse** joue un rôle identique mais affiche le code et non l'instruction symbolique :

M 00 donne :

0 : 1683 :

Il suffit de taper le nouveau code derrière les **:**

Dans les deux cas on sort du mode éditeur par **Q**

Les commandes **Dl adresse1 Adresse2** visualisent à l'écran en symbolique les lignes entre les deux adresses indiquées ,la commande **DM adress1 adress2** fait de même en code machine

Avec **DE adress1 adresse2** on peut supprimer les lignes entre les deux limites indiquées .

Le programme modifié peut être sauvé par **O nomde fichier**

Une commande très intéressante est :

F registre

Elle permet de modifier le contenu d'un registre préalablement défini par son nom .Par exemple modifions le contenu du registre DEPART avant l'exécution de la soustraction .

F DEPART

l'écran affiche :

DEPART : 01 :

Il suffit de taper la nouvelle valeur (dans la base de numération définie par défaut soit ici en hexa ; F0 par exemple.

Pour introduire une instruction entre deux autres on utilisera la commande **IN**

IN adresse,instruction (Ne pas oublier la virgule derrière l'adresse)

Place l'instruction à l'adresse indiquée en décalant les autres. (Elle se place donc avant l'instruction affichée au départ)

Exemple d'utilisation:

Ces instructions d'édition sont cependant limitées et ne sont utilisables que pour de petites modifications, elles n'utilisent pas les noms symboliques comme le montre l'exemple suivant:

Reprenons le programme CHENILPB , nous allons le modifier :

De façon que la rotation des bits se fasse vers la droite et avec un saut de 2 cases à la fois , de plus nous augmenterons le temps de retard pour ralentir le mouvement. .

Il faut d'abord lister le fichier initial CHENILPB.LST pour repérer les adresses à modifier.

A l'adresse 0005 nous avons :

0005 0D86 NOUVEAU RLF PORTB,1

Entrons dans MPSIM et lorsque le fichier .INI est chargé appelons l'éditeur par

IA 5

L'écran affiche :

5 **:** NOUVEAU RLF PORTB,1

:

Il faut taper derrière les **:** la nouvelle instruction :

: NOUVEAU RRF PORTB,1

cette entrée est refusée :

unknow instruction (NOUVEAU)

Le label ne peut pas être modifié , il faut entrer seulement la partie instruction

RRF PORTB,1

On sort alors en entrant Q ↵

Si l'on vient relire la ligne modifiée on trouve :

IA 5

5: NOUVEAU RRF 6

Le label a été conservé mais la notation PORTB n'a pas été utilisée ,la nouvelle instruction est bien exacte mais moins lisible .

Pour obtenir un saut de deux cases il suffit d'ajouter immédiatement derrière une instruction identique .Pour cela nous utilisons la commande IN , mais attention si nous l'exécutons à l'adresse 5 elle se placera en 5 , repoussant la ligne initiale en 6 or nous voulons en réalité qu'elle se place en 6 , derrière le label NOUVEAU qui reste en 5 ,c'est à partir de l'adresse 6 qu'il faut donc agir

Le numéro de l'adresse à utiliser pour IN est donc celle ou se placera l'instruction nouvelle . Soit:

IN 06,RRF PORTB,1

Comme plus haut en relisant la ligne 6 nous trouvons seulement RRF 6 Le label a été re-connu à la lecture mais non utilisé pour créer la nouvelle ligne.

Nous allons maintenant augmenter la durée de la boucle de retard en modifiant l'adresse 9:

IA 9
9 : : MOVLW 5
: **MOVLW 10** ↵
Q↵

La commande 0 permet de sauver un fichier .OBJ compte tenu des modifications apportées.

O Nomdefichier.OBJ [format]

Notez que l'on ne sauve pas le fichier source modifié .ASM , il faut pour cela revenir à l'éditeur.

Après une session de travail , à la sortie par Q , un fichier **MPSIM.JRN** est crée .Ce fichier contient toutes les commandes qui ont été utilisées pendant la session .

LES COMMANDES DE MPSIM

Un résumé de ces commandes est affiché à l'écran par la commande H

AB

Arrête la simulation et sort du logiciel ,identique à Q

AD

AD registre,[base],[nombre de caractères]
Affiche dans la zone réservée de l'écran le contenu d'un registre en cours de simulation dans une base choisie X(Hexa),B(binaire),O(octal) ou D(décimal) La base par défaut est définie par la commande SR
AD FOA . (attention numéro du registre en Hexa)
AD Port1 est accepté si Port1 a été préalablement défini par une instruction EQU dans l'assembleur.

B

B [Adresse] [Opérateur] Poser un point d'arrêt
Adresse est le plus souvent un label dans le fichier assembleur ,opérateur est une condition
BP Boucle Point d'arrêt au label boucle
BP F2 >=5 Arrêt si F2 >= 5

BC

BC [adresse ou registre] Suppression d'un point d'arrêt
BC Boucle suppression du point d'arrêt à boucle , si aucune adresse n'est indiquée tous les PA sont annulés.

C

C [n] continuer jusqu'au point d'arrêt suivant ,continuer jusqu'au n ieme point d'arrêt

DB

DB Afficher tous les points d'arrêt actifs

DE

DE adresse1adresse2 Détruire les codes dans la mémoire programme entre les deux adresses spécifiées Les adresses sont en Hexa
DE 0015 0A10

Dl

Dl Adresse1 Adresse2 Affiche le programme entre les deux adresses spécifiées (dans la base active)

DL

DL symbol Détruit un symbole de la table des symboles
DL Port1 Après cette instruction il faudra nommer le port 1 par son adresse, l'alias n'est plus reconnu

DM

DM adresse1 Adresse2 Affiche le programme comme **Dl** mais en hexa pur

DP Affiche le programme

DR Affiche tous les registres

DS Affiche la table des symboles

DV

DV data Efface un fichier dans la visualisation de MPSIM
DV Port1 Port1 ne figurera plus à l'écran

DW E ou D Valide ou Stoppe le chien de garde

DX Affiche le mode trace

E

E [adresse] Exécuter le programme à partir de l'adresse indiquée ou à partir du dernier point d'arrêt si aucune adresse n'est indiquée

F

F registre Affiche et ou modifie le contenu d'un registre
F 3 Affiche le contenu de 3 sous la forme F3 :20: Pour modifier la valeur taper celle ci derrière les **:**

GE
GE Fichier Lit et exécute un fichier de commandes
GE ESSAIO.INI charge le fichier ESSAIO.INI qui est un fichier texte de commandes MPSIM

GO
GO Exécuter le programme à partir du début

GS

GS symbole,valeur,type[fichier] Génère un symbole avec la valeur indiquée et imprime

H Aide à l'écran

IA

IA Adresse Affiche et permet la modification d'une instruction La nouvelle valeur est tapée derrière les **:**
IA 200 Affiche par exemple : 020 0200 0I45 LABELL CLRF F5;Clear I/O register pour portA :
Pour changer 5 en 6 taper derrière les **:** CLRF 6
Pour sortir taper Q

IN

IN adresse,instruction Insérer une instruction
IN 200,NOP insérer un NOP à l'adresse 200 (Dans la base définie par SR)

IP

IP [temps/pas] injecte un stimulus à un temps ou pas déterminé

LJ

LJ Charge et exécute le fichier journal qui contient tous les ordres utilisés lors de la précédente session du simulateur

LO
LO nom de fichier Charge un fichier Objet et la table des symboles

LS
LS nom de fichier Charge la table des symboles

M

M adresse Affiche et modifie le contenu de la mémoire programme, l'adresse doit être exprimée dans la base courante définie par SR
M010 Le contenu est affiché et suivi de **:** taper la nouvelle valeur derrière ces **:**

NV RAZ de l'écran de visualisation

O

O nom_de_fichier [Format] Ecrit le programme y compris les modifications apportées pendant la session dans le fichier spécifié avec le format indiqué (INHXBM,INHX16,INHXBS,PICICE) ,par défaut INHX16)

P

P [54/55/56/57/71/84] Définition du circuit utilisé

Q
Q Quit

RA

RA restaure tout

RE
RE Remet à zéro le compteur de temps d'éxécution et compteur d'instructions

RP
RP Restaure tous les patches à leur valeur initiale et RAZ la table des patches

SC

SC [longueur de cycle] Affiche et permet la modification du temps de cycle du CPU utilisé par le simulateur pour le calcul du temps . Le temps est en microsecondes

SE

SE data Affiche et permet la modification des zones de données

SF

SF adresse1 , adresse2 , registre Cherche entre les deux adresses indiquées les instructions qui mettent en oeuvre un registre particulier

SI

SI adresse1 ,adresse2,instruction Comme la précédente mais recherche une instruction désignée par son mnémonique

SM

SM adresse1,adresse2,instruction Idem mais l'instruction est définie par son code dans la base active

SR

SR [0/XJD] Précise la base utilisée Octal Hexa ou Décimale]

SS

SS [adresse] Exécution pas à pas à partir de l'adresse indiquée ou de l'état du compteur programme- En pressant simplement ↵ un SS s'exécute automatiquement permettant l'examen pas à pas d'un programme sans qu'il soit besoin de retaper SS à chaque pas.

ST

ST Nom_de_fichier Lecture d'un fichier de stimuli (.STI)
Introduit des valeurs prèdèfinies dans un registre ou une broche du circuit à des pas spécifiés ou des temps spécifiés. Le fichier de stimulis doit comporter une première ligne commençant par STEP suivi de la liste des broches du circuit qui doivent être manipulées, puis en dessous les numéros (en décimal) des étapes et la table des valeurs , les commentaires sont entrés après ! :

```
STEP    RA0    RA1    !   première ligne
8       0      1
16      1      1
24      0      0
```

TA

TA [adresse1,adresse2] Visualise et imprime les instructions comprises entre les adresses indiquées , par défaut la totalité du programme .

TC

TC #instructions "trace" les instructions ou les n dernières instructions

TF

TF [Nom_de_fichier / prn] Ouvre ou ferme un fichier de rapport ("trace") , avec prn c'est l'imprimante qui est sollicitée

TF PRN
TF SAMPLE. TRC

TR

TR registre [, valeur min,valeur max]

TR W 2 7 "trace" les valeurs de W lorsqu'elles sont comprises entre 2 et 7 (dans la base définie par SR).

TY

TY zone_de_donnèe,[X/O/D/B],#digits change le format de l'écran

V

V signal,[Base,nbr_de_digits] crée une nouvelle visualisation de registre

W

W Affiche et permet de modifier (derrière **:**) le contenu de W

W**:**44**:** introduire la nouvelle valeur derrière **:** soit W:44:22

ZM
ZM adress1,adresse2 met à zéro la mémoire programme entre les adresses 1 et 2

ZP RAZ patch table

ZR Tous registres à zéro

ZT Remet à zéro le compteur de temps d'éxécution (et lui seul)

LE PROGRAMMATEUR DE PIC 16C84

Le 16C84 possède une mémoire de programme interne de type EEPROM (EPROM Flash sur le nouveau 16F84 qui tend à remplacer le 16C84) Cette mémoire est programmable par une procédure de transfert série qui utilise 3 bornes du circuit. Pendant la programmation la broche MCLR doit être portée à un potentiel de 12V , les bornes RB6 et RB7 sont utilisées respectivement comme horloge et accès de données pour le transfert série.

A titre documentaire l'algorithme de programmation du 16C84 est joint en annexe (Document MICROCHIP) ,mais un programme est disponible sur INTERNET ou des revues d'électronique , par exemple ELEKTOR.

Ce programme s'appelle PIP02 et nécessite pour fonctionner un driver COM84 ,la procédure d'appel est la suivante (A placer dans un fichier .BAT) :

COM84 COM1
PIP02
COM84 REMOVE

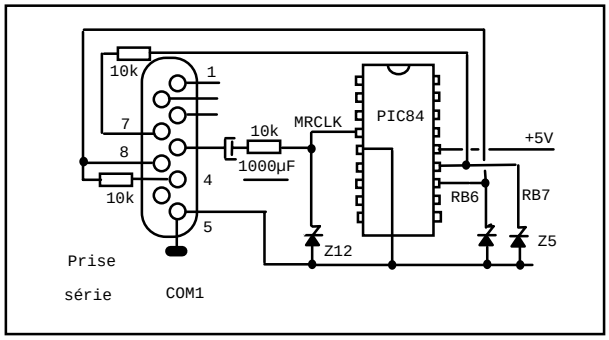
Il utilise le port série du PC pour l'échange.

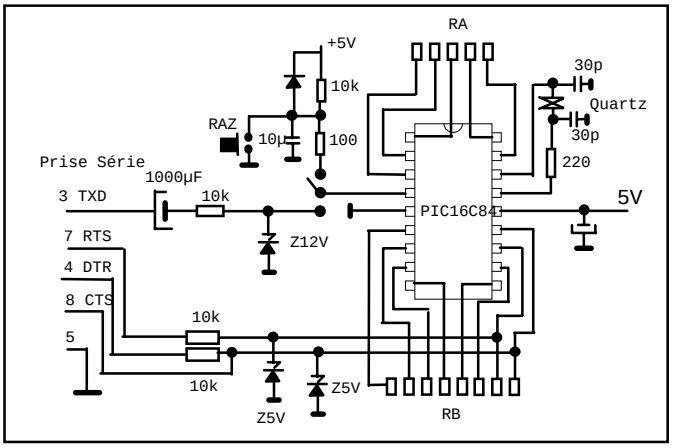
La tension de 12V peut être fournie par une alimentation extérieure, mais elle peut être crée à partir du port série. On sait en effet que la liaison RS232 met en oeuvre deux niveaux +12 et -12 .La position de repos étant -12V . Le saut de 24V -12 à +12 peut être utilisé pour créer du 12V grâce à un condensateur et une diode Zener de limitation. (Attention certains PC ne délivrent pas sur leur prise série les ±12V théoriques mais un niveau bien moindre, avec ce schéma un ±8 ou 9V suffit) .

Le schéma est le suivant :

(Note : la résistance aboutissant à CTS doit être 4,7k et non 10k)

Il est possible de réaliser à partir de ce schéma une plaquette d'essai permettant par simple basculement d'un interrupteur de passer du mode programmation au mode utilisation. Ne pas oublier cependant que le PIC16C84 n'est pas conçu pour être reprogrammé de nombreuses fois, le constructeur annonce 1000 programmations possibles , il est sans doute prudent de ne pas dépasser quelques centaines .





UTILISATION du LOGICIEL

Les fichiers COM84 et PIP02 sont chargés dans la mémoire de l'ordinateur, ainsi que le fichier BAT :

PROG.BAT
COM84 COM1
PIP02
COM84 REMOVE

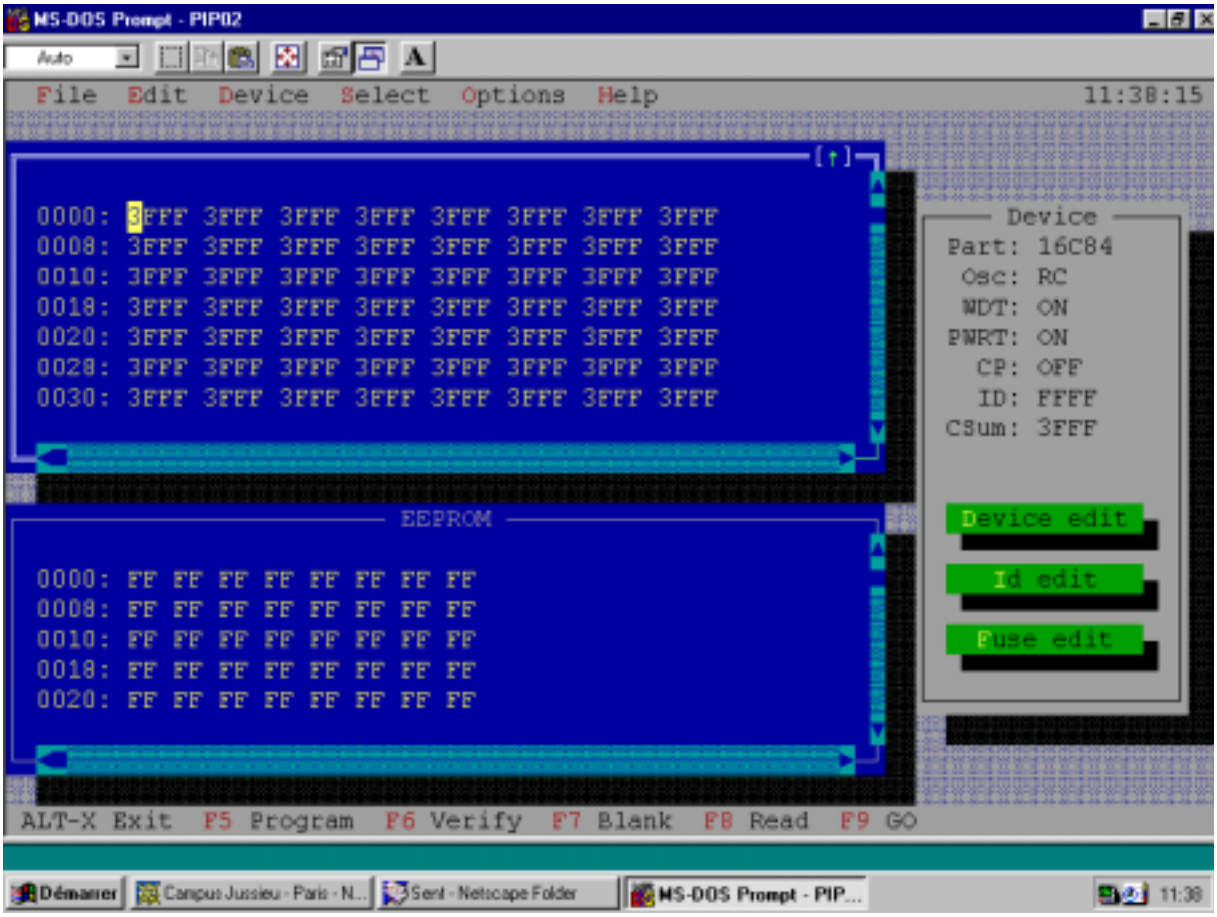
L'appel se fait alors par la commande PROG ↵

L'écran suivant apparaît :

Il comporte deux grandes plages dans lesquelles seront écrits les mots ou octets chargés (ou lus) en mémoire programme et dans l'EEPROM interne.

A la droite de l'écran une fenêtre affiche le type de processeur et l'état des fusibles .

La première ligne contient les boutons de commande classiques E F D S O H



LECTURE DU CONTENU D'UN BOITIER 16C84

Le programmateur est relié au PC via la sortie série COM1 (Prise de souris) ,on peut aussi utiliser COM2 en modifiant le fichier .BAT précédent . Mettre le composant sur le support l'alimenter en 5V et lancer PROG.

La première chose à faire est de définir le composant , pour cela taper Alt S , une petite fenêtre de dialogue s'ouvre :

SELECT
Device
Fuse Word
ID Loc

La case Device est activée (en vert) .Valider par ↵

Une nouvelle fenêtre s'ouvre permettant le choix du boîtier. Sélectionner le boîtier choisi en agissant sur les touches de déplacement ⇑ ou ⇓ du clavier et valider par ↵ (Entrée.) On revient alors à l'écran de départ mais la fenêtre de droite affiche le type de boîtier choisi et les paramètres par défaut.

PART	16C84
Osc	RC
WDT	on
PWRT	on
CP	off
ID	FFFF
CSum	FFFF

Pour lire le contenu de la mémoire programme du composant taper maintenant Alt D

Une nouvelle fenêtre s'ouvre en dessous du bouton Device .Amener avec les touches de déplacement le curseur vert sur Read et valider par ↵

Un fenêtre s'ouvre pendant quelques secondes visualisant le déroulement du transfert puis les deux fenêtres bleues de l'écran principal affichent le contenu de la mémoire programme et de l'EEPROM interne. .La

fenêtre de droite affiche le type de boîtier et l'état de ses fusibles (Type d'oscillateur, validation ou non du Chien de Garde etc....

Si le composant était protégé en lecture par le fusible adapté la lecture est refusée, la seule fonction autorisée est ERASE ,pour effacer le contenu.

PROGRAMMATION D'UN COMPOSANT

Les opérations a effectuer sont les suivantes :

1: Choisir le composant comme plus haut par

Alt Select
16C84
↵

2° Effacer le composant qui contient sans doute un programme antérieur .

Alt D

et dans la fenêtre qui s'ouvre valider ERASE

3° Charger le fichier contenant les mots codes :

Alt F

et dans la fenêtre qui s'ouvre LOAD

Une nouvelle fenêtre se forme et vous propose un fichier .HEX .Entrer le fichier obj qui a été crée par MPALC soit nomdefichier.OBJ et valider par ↵

Les codes machine apparaissent à l'écran .

4°Programmer la zone mémoire :

Alt D

et dans la fenêtre choisir Program et ↵

5° Choisir les fusibles .

Alt S

et dans la fenêtre valider Fuse Word

Une fenêtre s'ouvre , elle comporte deux zones ,pou passer de l'une à l'autre on utilise la touche TAB du clavier ←→ La fenêtre du haut permet le choix du type d'oscillateur. On passe de l'un à l'autre par les touches flèches ⇓ ou ⇑ et l'on sort par TAB . La fenêtre du bas est réservée au Watch Dog et fusibles d'alimentation et sécurité. Le déplacement se fait par TAB et la modification à l'aide de la barre d'espace .

Lorsque la configuration désirée est prête , sortir par ↵

Il ne reste plus qu'à programmer les fusibles par

Alt D

puis dans la fenêtre Prog Fuses

Le boîtier est programmé , on peut relire son contenu par Alt D / READ ou vérifier qu'il est bien identique au contenu du fichier chargé par Alt D / Verify

Le PIC 16F84

Il se distingue du précédent par la nature de la PROM interne qui est du type Flash et non EAPROM. Le soft est exactement le même à une petite différence concernant la programmation des fusibles. Le programmateur précédent convient à condition de disposer de la dernière version de PIP02 .