TD Microcontrôleur, famille PIC

Séances 1 et 2

Lecture écriture d'une mémoire RAM

Dans ce problème, nous étudions la réalisation d'une console permettant de lire et d'écrire « manuellement » dans une mémoire RAM (ces opérations se font normalement sous le contrôle d'un « processeur »), lorsque le calculateur est déconnecté (sorties en haute impédance).

La mémoire RAM que nous étudions est extensible jusqu'à 64 k-mots de **8 bits** (un k-mots vaut 2^{10}=1024 mots). Elle se compose de circuits imprimés ou cartes de 1 k-mots chacune, enfoncées dans les connecteurs d'un jeu de lignes ou MULTIBUS (voir la figure 1).

Le MULTIBUS comprend :

- un bus d'adresses de 16 lignes (A0, A1, …, A15)
- un bus de données de 8 lignes (D0, D1, …, D7)
- une ligne $R/\overline{W}$ (0 = écriture, 1 = lecture)
- des lignes d'alimentations
- éventuellement des lignes d'inhibition du dispositif de sélection (nous n'en ferons pas usage ici)

On désire lire et écrire la mémoire RAM en code hexadécimal à partir d'une console comprenant (voir figure 3) :

- Les touches de codes : en pressant sur l'une de ces touches, on charge le code d'un chiffre hexadécimal 0, 1, 2, .., 9, A, B, C, D, E, F dans la mémoire d'un registre tampon C.
- La touche de sélection d'une adresse : SEL
- La touche INC : cette touche permet d'incrémenter le sélecteur d'adresses et d'obtenir en même temps une lecture de la position mémoire adressée.
- La touche ECR : elle permet d'écrire, à la position mémoire sélectionnée, les deux derniers codes entrés au clavier (moitié droite du registre C), et aussi d'incrémenter l'adresse sélectionnée de manière à faciliter l'écriture de segments de programme.

Deux dispositifs d'affichage fonctionnent en permanence, visualisant d'une part l'adresse et d'autre part le contenu de la mémoire adressée.

I - Etude de la mémoire (figure 2)

La figure 2 donne le schéma d'une demi carte mémoire composée de 4 circuits intégrés de 128 mots de 8 bits chacun (mémoire du type 6810 de Motorola) et de circuits de décodage d'adresse.

Chaque circuit intégré comporte 6 bornes de validation : $CS0$, $\overline{CS1}$, $\overline{CS2}$, $CS3$, $\overline{CS4}$ et $\overline{CS5}$. Le circuit est sélectionné lorsque ses CSi sont à 1 et les $\overline{CSi}$ sont à 0.

Quelles sont les zones mémoires (plages d'adresses : adresse de début et adresse de fin) permettant d'accéder aux circuits 1, 2, 3 et 4 de la demi carte de la figure 2 ?

Même question pour l'autre demi carte (non représentée sur la figure 2).

II - Encodage d'un clavier

On désire concevoir un dispositif encodant 16 touches (0, 1, …9, A, B, …F) sur 4 fils. C'est-à-dire que l'appui sur la touche 0 doit fournir la valeur 0 sur les 4 fils. La touche 1 la valeur 1 et ainsi de suite :

$$\begin{cases} 0 \rightarrow 0000 \\ 1 \rightarrow 0001 \\ \quad \vdots \\ E \rightarrow 1110 \\ F \rightarrow 1111 \end{cases}$$

On supposera que la pression sur une touche ferme un interrupteur simple.

- En analysant cet énoncé, vérifier que le problème est mal posé, le corriger, puis réaliser le système adéquat.

III - Réalisation du registre tampon C (figure 3)

On propose maintenant de réaliser le registre tampon C. Ce registre permettra de charger le bus d'adresses (16 bits) et le bus de données (8 bits.) à partir des valeurs tapées sur le clavier.

Le registre C est formé de bascules type D. Ce registre reçoit un nouveau code du clavier en C0, C1, C2, C3 et décale automatiquement d'un pas les anciens codes (le code situé en C12, C13, C14, C15 est perdu).

- Quelle est la taille (en bits) du registre C ? Combien de bascules sont nécessaires ?
- Dessiner entièrement le registre C. Mettre en évidence le chemin suivi par les données du clavier au sélecteur d'adresses, ainsi que les signaux de contrôle.

IV - Réalisation du sélecteur d'adresses (figure 3)

C'est le circuit qui actionne directement le bus d'adresse mémoire. Il s'initialise avec la commande SEL et reçoit la commande INC.

- Dessiner entièrement ce circuit, en incluant les signaux de contrôle.

V - Réalisation du contrôleur de bus de données (figure 3)

C'est le circuit qui permet d'écrire dans la mémoire. Recevant la commande ECR, il place la mémoire en mode d'écriture puis charge le bus de données avec les valeurs des bits de poids faibles du registre C. Il incrémente ensuite l'adresse mémoire et place la mémoire en mode de lecture.

- Dessiner complètement ce circuit. Quelle caractéristique particulière doit-il comporter ?

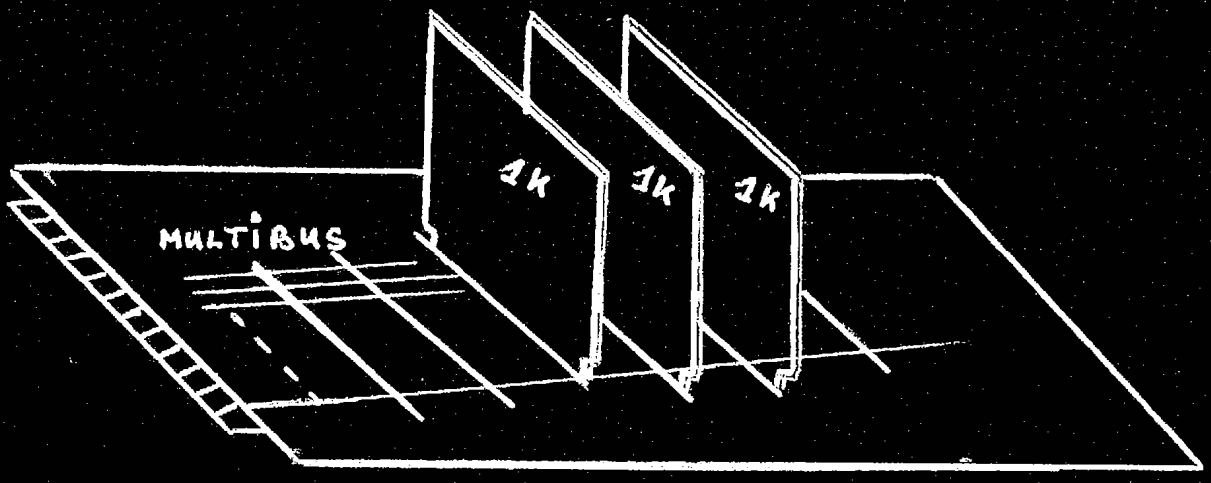


Figure 1 : Les cartes de 1K-mots se placent sur les connecteurs du "MULTIBUS".

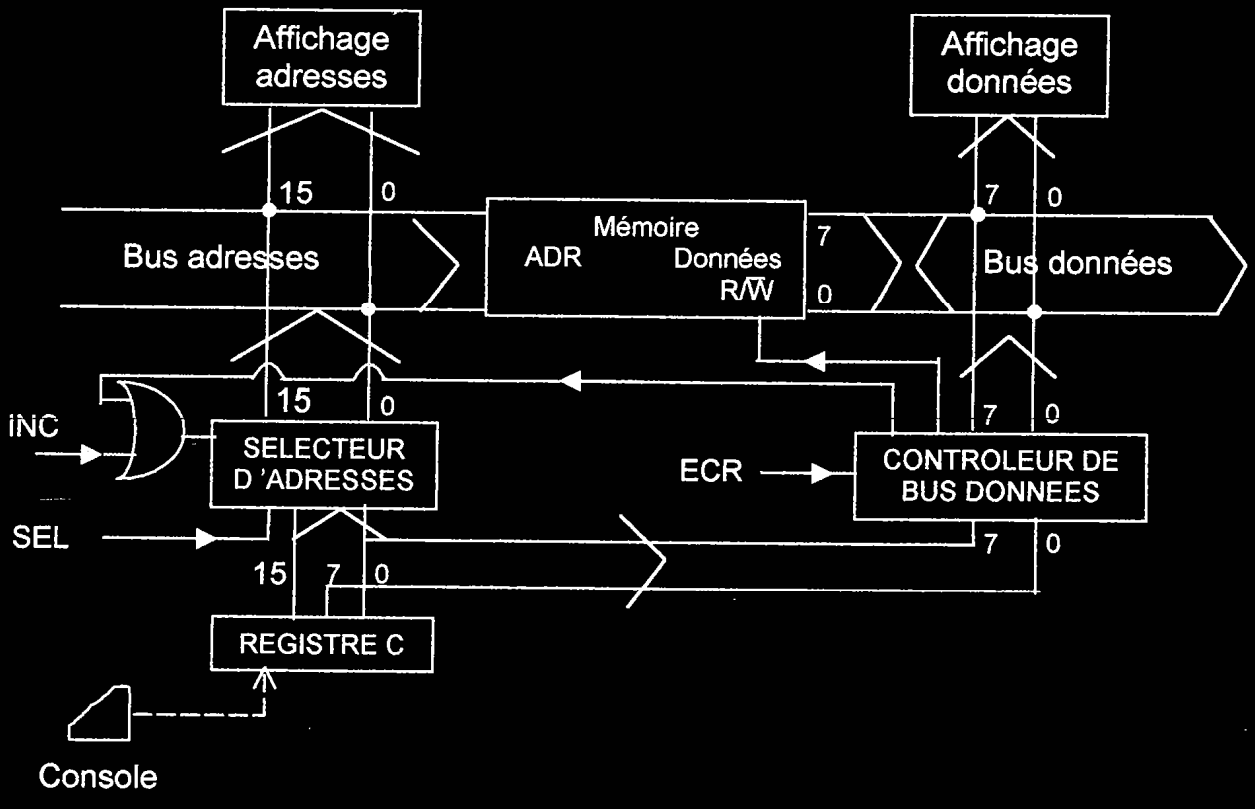
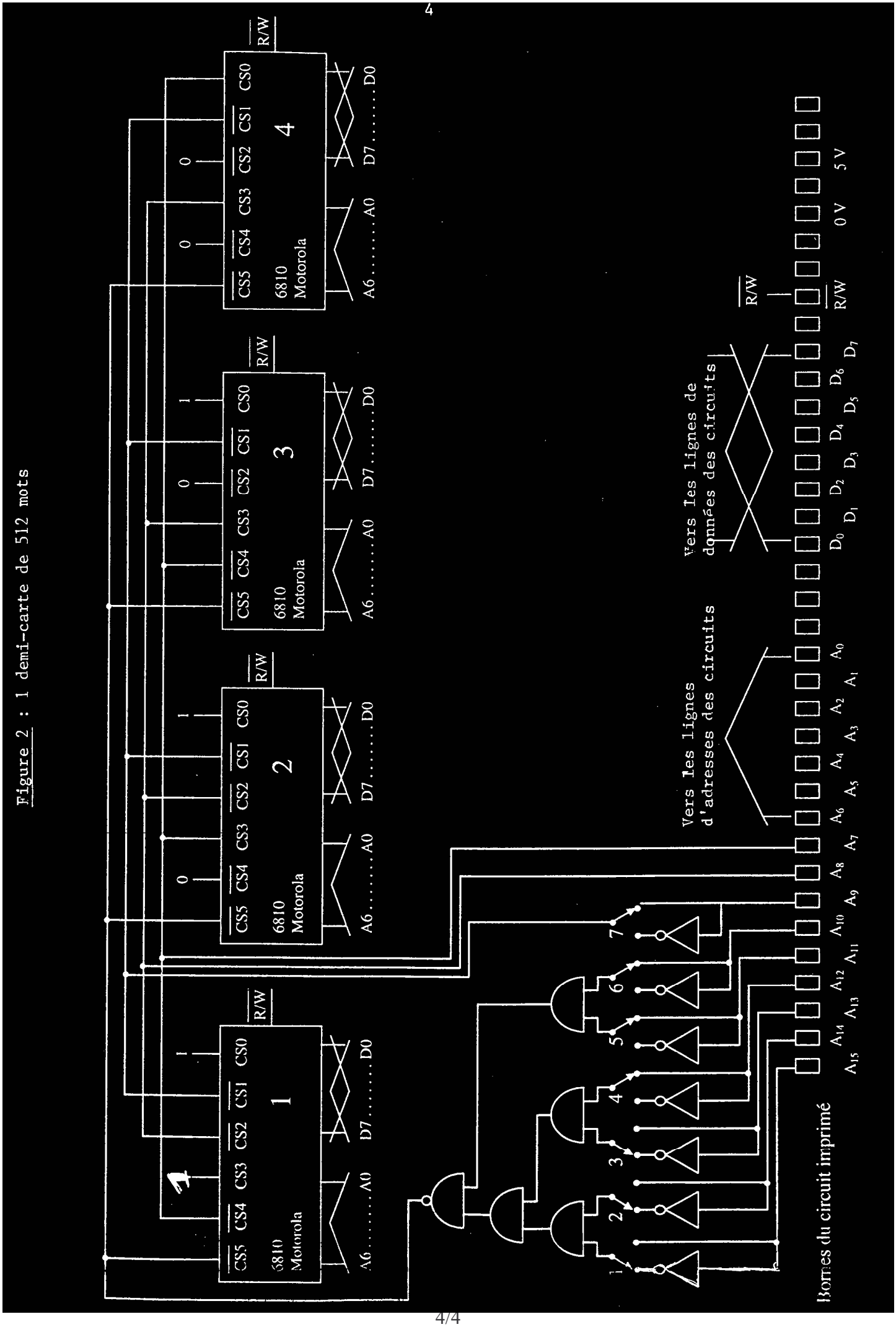


Figure 3 : Bloc diagramme d'ensemble.


4
Figure 2 : 1 demi-carte de 512 mots
6810 Motorola
CS5 CS4 CS3 CS2 CS1 CS0
R/W
A6 ..... A0
D7 ..... D0
1
2
3
4
Vers les lignes d'adresses des circuits
Vers les lignes de données des circuits
Bornes du circuit imprimé
0 V
5 V

TD µC 1, 2
lecture / ecriture d'une RAM

I - 1° demi carte.

bus d'adresses : 16 bits.

Boitier RAM : 128 octets ⇒ $2^7 = 128$

taille du bus d'adresse pour les boitier : 7

Selection des boitiers par les bits de poids fort du bus d'adresse, c-à-d : $A_{15} \rightarrow A_7$.

⇒ Etude de l'activ0. des boitiers grâce aux bits A_{15}-A_7.

vrai pour tous les boitiers → Boitier 1 : $\overline{CS} = \overline{CS5}.\overline{CS4}.\overline{CS3}.\overline{CS2}.\overline{CS1}.\overline{CS0}$

d'après C^{1s} : $\overline{CS} = \overline{CS5}.\overline{CS4}.\overline{CS2}.\overline{CS1}$

d'après cablage : $\overline{CS1} \Rightarrow A9$; $\overline{CS2} \Rightarrow A8$; $\overline{CS4} \Rightarrow A7$

"⇒" correspond à : "est connectée à" = "est vraie pour"

$\overline{CS5} \Rightarrow \overline{((\overline{A_{15}}.\overline{A_{14}}).\overline{A_{13}}.A_{12}) + (A_{11}.A_{10})}$

$\overline{CS5} \Rightarrow A_{15}.\overline{A_{14}}.\overline{A_{13}}.A_{12}.A_{11}.A_{10}$

→ Boitier 1 : $\overline{CS} = \overline{A_{15}}.\overline{A_{14}}.\overline{A_{13}}.A_{12}.A_{11}.A_{10}.\overline{A_9}.\overline{A_8}.\overline{A_7}$

⚠ on veut $\overline{CS_1} = 0$

→ plage d'adresses pour le boitier 1 :

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	1	1	1	0	0	0	X	X	X	X	X	X	X

X : état indifférent 0 ou 1.

valeur d'activ0. du boitier

valeurs indifférentes à l'activ0. du boitier (ce qui tombe bien, puisque il s'agit des lignes d'adresse reliées au bus d'adresse du boitier ☺)

⇒ plage d'adresse : valeur min et max réalisée avec les bits à l'état indifférent.

Boîtier	$\overline{CS}$	Plage
1	$\overline{A_{15}}\ \overline{A_{14}}\ \overline{A_{13}}\ A_{12}\ A_{11}\ A_{10}\ \overline{A_9}\ A_8\ A_7$	1 0 0 0 1 1 1 0 0 0 X X X X X X X X 1 C 0 7 F
2	$\overline{A_{15}}\ \overline{A_{14}}\ \overline{A_{13}}\ A_{12}\ A_{11}\ A_{10}\ \overline{A_9}\ \overline{A_8}\ A_7$	1 0 0 0 1 1 1 0 0 1 X X X X X X X X 1 C 8 F F
3	$\overline{A_{15}}\ \overline{A_{14}}\ \overline{A_{13}}\ A_{12}\ A_{11}\ A_{10}\ \overline{A_9}\ A_8\ \overline{A_7}$	1 0 0 0 1 1 1 0 1 0 X X X X X X X X 1 D 0 7 F
4	$\overline{A_{15}}\ \overline{A_{15}}\ \overline{A_{13}}\ A_{12}\ A_{11}\ A_{10}\ \overline{A_9}\ A_8\ A_7$	1 0 0 0 1 1 1 0 1 1 X X X X X X X X 1 D 8 F F

I-2 les 2 demi-cartes

Boîtier	plage	
1	1C 00 – 1C 7F	1° demi carte.
2	1C 80 – 1C FF	
3	1D 00 – 1D 7F	
4	1D 80 – 1D FF	
5	1E 00 – 1E 7F	2° demi carte.
6	1E 80 – 1E FF	
7	1F 00 – 1F 7F	
8	1F 80 – 1F FF	

$4 \times 128 = 2^9$

$A_{15} \dots A_9 = C^{te}$; $A_9 = 0 = C^{te}$

A_9 doit être = à 1

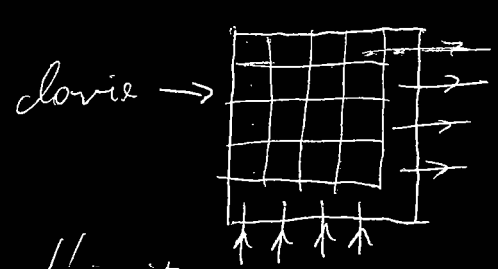
conclure sur : ⇒ Rôle des interrupteurs de gauche?

II Encodage clavier

– Puisqu'on ne dispose que de 4 fils et puisque la pression d'une touche ferme un unique interrupteur simple, il n'est pas possible d'obtenir le codage souhaité qui nécessite quatre interrupteur par touche (au max.)

ex pour la touche F : il faut fermer 4 interrupteurs 1 1 1 1.

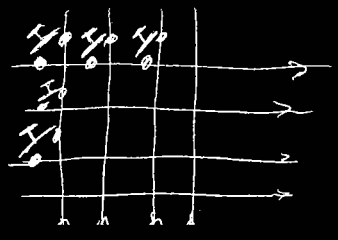
→ il faut utiliser 4 fils supplémentaires et mettre en place une scrutation par balayage.



on active une ligne à la fois :

t_0 : 1 0 0 0
t_1 : 0 1 0 0
t_2 : 0 0 1 0

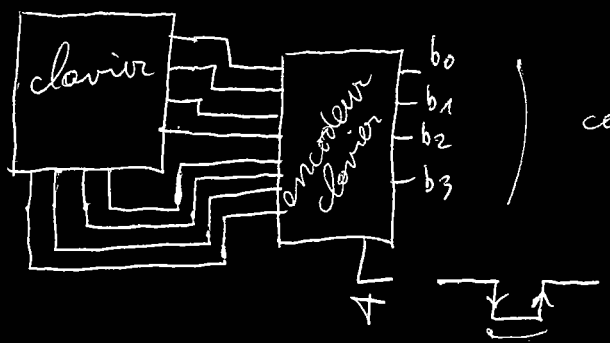
principe du clavier



III Registre tampon C :

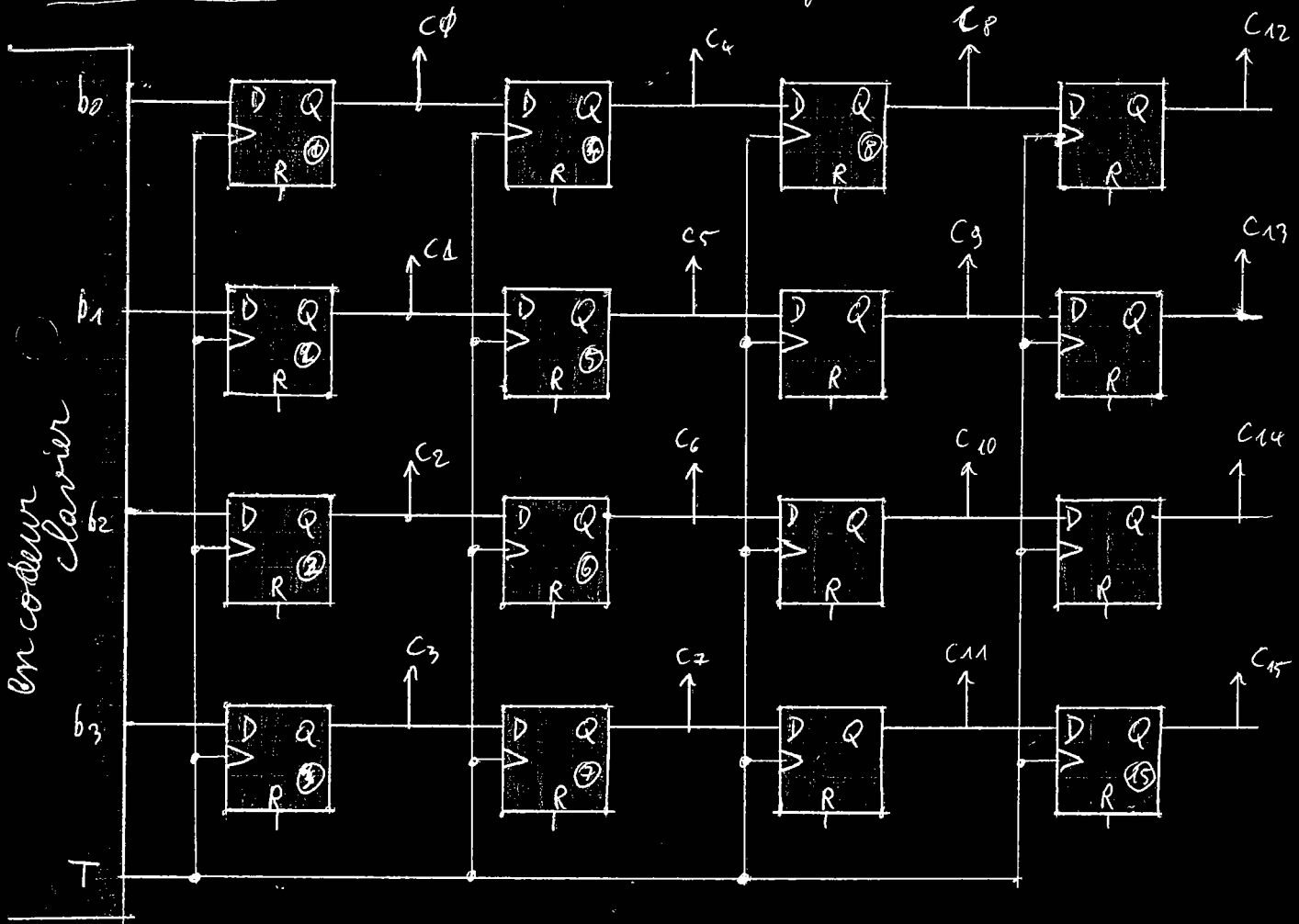
On dispose de 16 bascules D lecture sur front ↓ écriture sur front ↑

L'encodeur clavier donne le code de la touche enfoncé ainsi qu'une information sur la pression de la touche : la sortie T est à 1 quand aucune touche n'est appuyée
la sortie T est à 0 lorsqu'une touche est enfoncée.



0000 0
0001 1
⋮
1110 E
1111 F

Registre C :



→ la lecture sur ↓, puis l'écriture sur ↑ permet (avec le fonctionnement de T) de garantir la propagation des données.
→ sinon utiliser des tempos : D en 1° faire $C_8 \to C_{12}$; $C_9 \to C_{13}$; ... $C_{11} \to C_{15}$

- Les données vont de gauche à droite.
- Le circuit Reset peut être cablé et est commun à toutes les bascules.
- Exemple avec la selection de l'adresse 1 C 3 A

On tape "1", puis "C", puis "3", puis "A"

	bi	C 0 1 2 3	C 4 5 6 7	C 8 9 10 11	C 12 13 14 15	
état initial : (Reset)	0	0	0	0	0	t_0
	0	0	0	0	0	
	0	0	0	0	0	
	0	0	0	0	0	
appuie sur "1" (puis lâché...)	1	1	0	0	0	t_1
	0	0	0	0	0	
	0	0	0	0	0	
	0	0	0	0	0	
appuie sur C	0	0	1	0	0	t_2
	0	0	0	0	0	
	1	1	0	0	0	
	1	1	0	0	0	
3	1	1	0	1	0	t_3
	1	1	0	0	0	
	0	0	1	0	0	
	0	0	1	0	0	
A	0	0	1	0	1	t_4
	1	1	1	0	0	
	0	0	0	1	0	
	1	1	0	1	0	
⋮						⋮

IV Sélecteur d'adresses

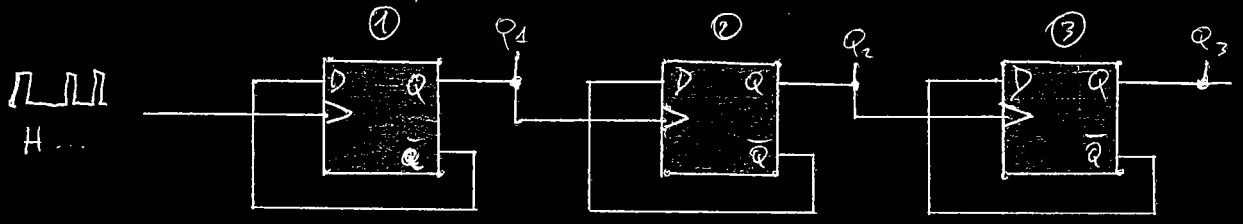
Le selecteur d'adresse possède 3 signaux de controle : un ECR retardé venant du bus de données (réalisant une incrémentaO. de la valeur d'adresse)

INC incrémentant la valeur de l'adresse (ex 1C3A ⇒ 1C3B, le contenu n'est pas modifié...)

SEL permettant de choisir l'adresse à partir du registre tampon C.

> à cause des opéraO. d'incrémentaO., le selecteur d'adresse est un compteur 16 bits dont l'horloge est gérée par ECR et INC, et dont le forcage de valeur (initialisaO. du compteur) est géré par SEL.

Principe du compteur : bascules en cascade et bouclage :

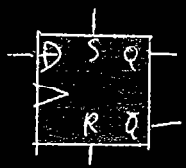


	H	Q_1 Q_2 Q_3	D_1 D_2 D_3	Q_1' Q_2' Q_3' ← si coup d'horloge... état de Q_i à $t+1$
t_0		0 0 0	1 1 1	1 1 1
t_1	⊓	1 0 0	0 1 1	0 1 1
t_2		0 1 0	1 0 1	1 0 1
t_3		1 1 0	0 0 1	0 0 1
t_4		0 0 1	1 1 0	1 1 0
t_5		1 0 1	0 1 0	0 1 0
6		0 1 1	1 0 0	1 0 0
7		1 1 1	0 0 0	0 0 0
8		0 0 0	1 1 1	1 1 1
9		1 0 0	0 1 1	0 1 1
10		0 1 0	1 0 1	1 0 1
11		1 1 0	0 0 1	0 0 1

↑ : écriture
↓ : lecture/écriture

→ Problème du Reset : • soit les bascules ont un Set/Reset de forçage prioritaire.

H



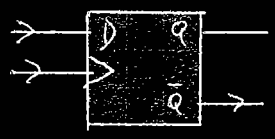
• soit on réalise le Set/Reset avec une logique sur H et D.

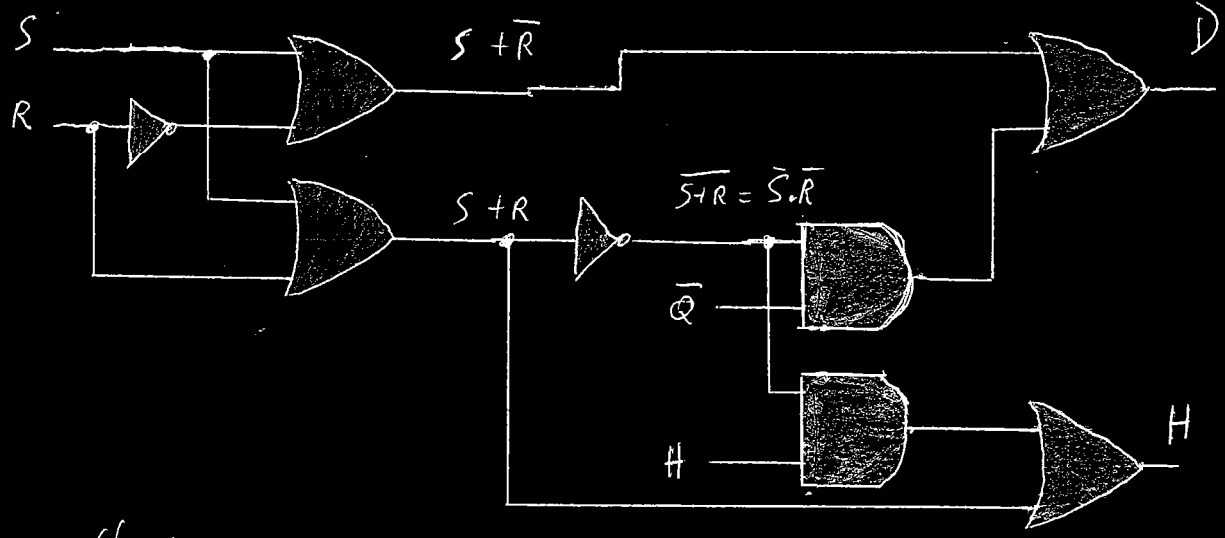
logique sur H et D :

$H = H.(\bar{S}.\bar{R}) + S + R$

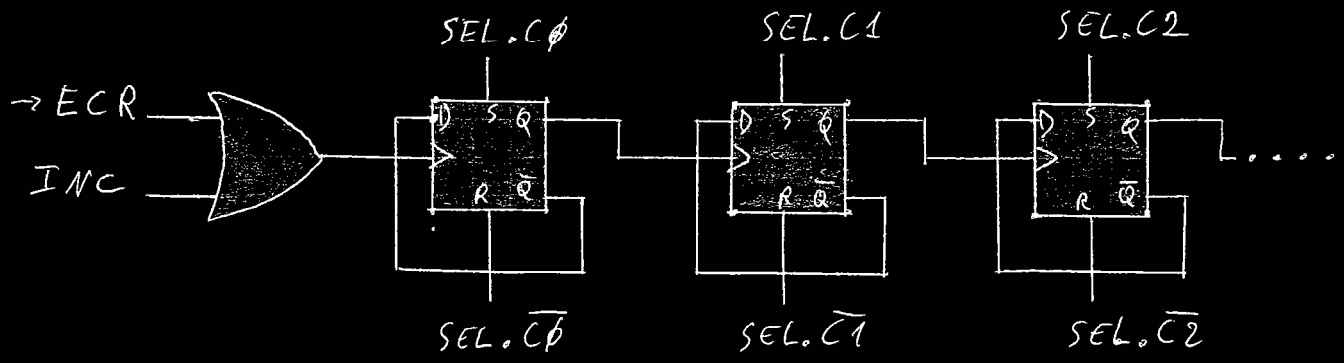
$D = \bar{Q}.(\bar{S}.R) + S + \bar{R}$

H





→ Sélecteur d'adresses avec S/R incorporés...



Rem équations avec logique sur H et D :

$$\begin{cases} D_i = SEL.C_i + \overline{SEL}.Q_i \\ H_0 = SEL + ECR + INC \\ H_{i>1} = Q_{i-1}\,\overline{SEL} + SEL \end{cases}$$

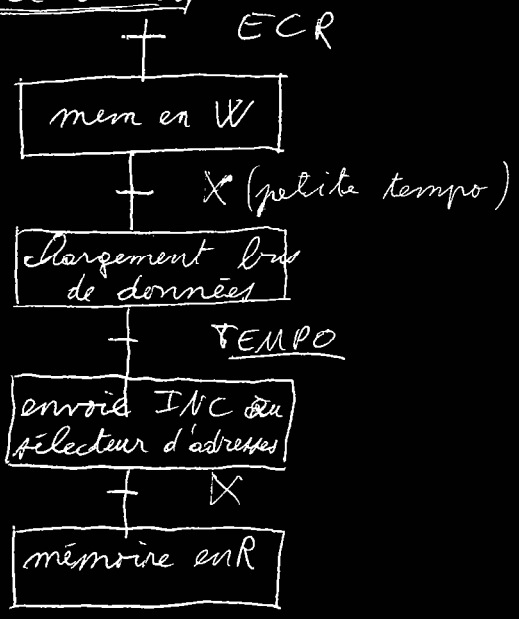
V Contrôleur Bus de données

détails de fonctionnement :

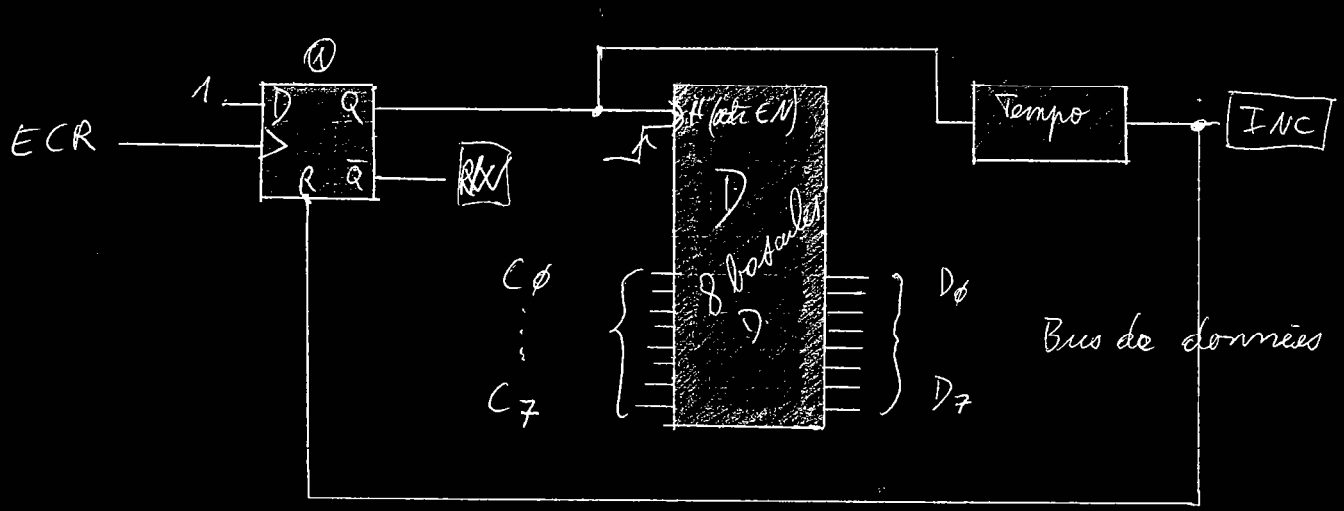
Recevant ECR, il place la mem. en fonct. W et autorise charge le bus de données avec LSB de C

Il incrémente ensuite l'adresse mémoire et place la mémoire en R

séquence d'étapes



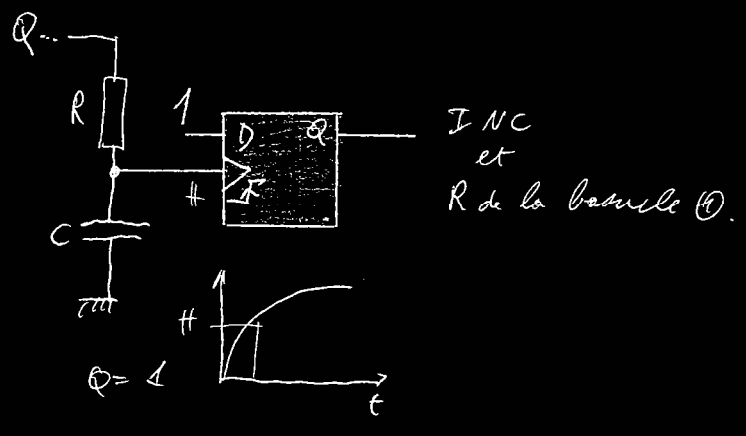
la tempo pourra être réalisée avec :
- R C + 1 compteur
- 1 bascule D et 1 circuit RC

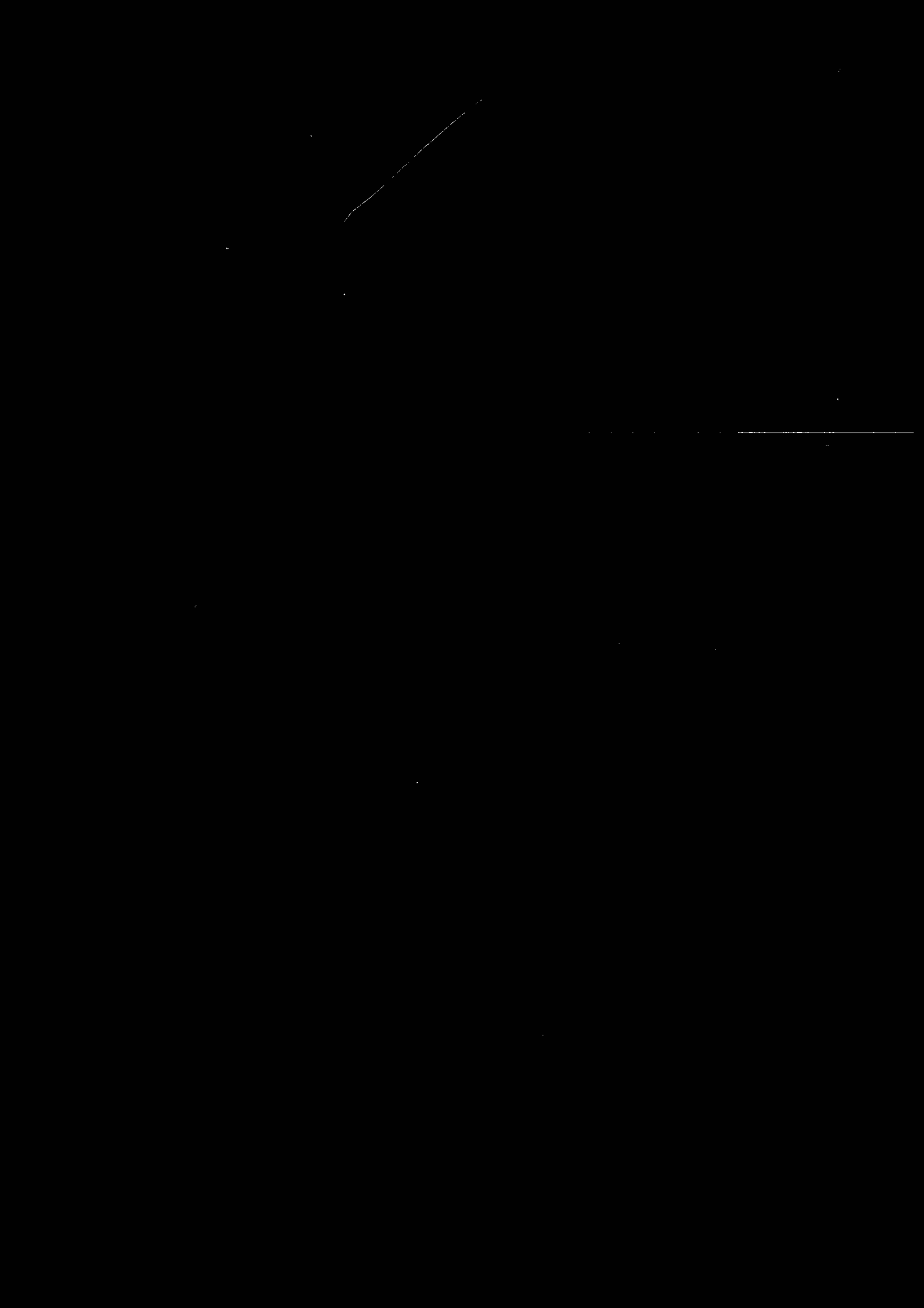


→ R/W̄ : 0 : écriture ; 1 lecture (cf. énoncé)

→ ECR est maintenu grâce à une bascule ① (pour répondre le problème d'un appuie sur ECR plus court que la durée de la séquence [tempo !]).

→ Tempo :





TD Microcontrôleur, famille PIC

Séance 3

Démultiplexage d'adresses

On souhaite commander, avec un microcontrôleur, 16 sorties quelconques associées à des voyants de signalisation. On veut pouvoir modifier l'état de chaque sortie (active : 1, ou non active : 0) sans modifier l'état des autres sorties. On veut aussi pouvoir mettre à zéro toutes les sorties facilement.

Afin de minimiser le nombre de fils nécessaires au pilotage des 16 sorties, on propose d'utiliser deux démultiplexeurs 74HC259 (cf. Figure 1).

La finalité de cet exercice est de réaliser le câblage et la logique entre un microcontrôleur de type PIC et les entrées des 74HC259 de la Figure 1.

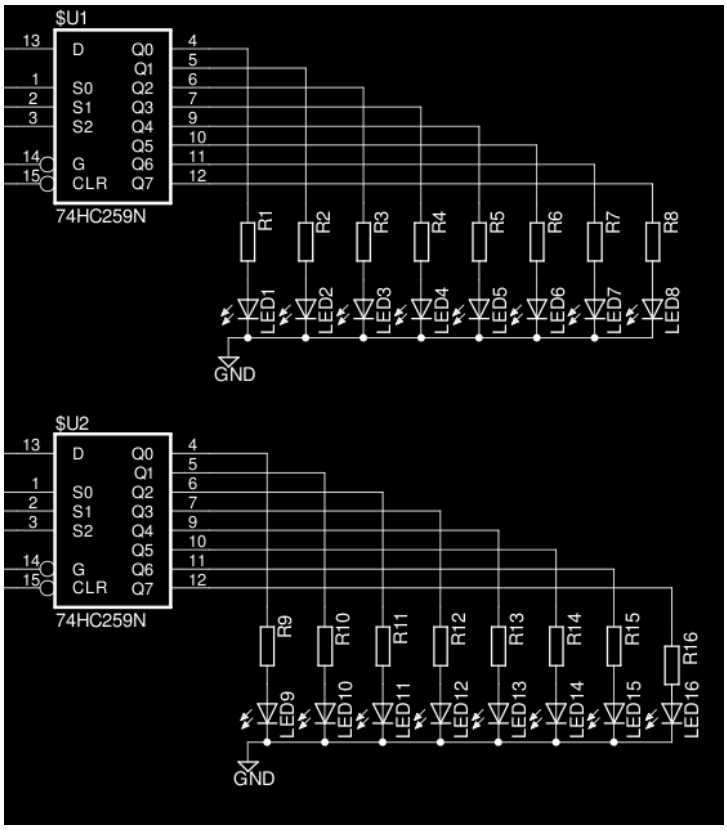


Figure 1 : Solution propose pour le pilotage de 16 sorties. Les LED symbolisent les sorties.

I – Solution sans circuits 74HC259

- Sans les démultiplexeurs 74HC259, combien de fils seraient nécessaires au pilotage des 16 sorties ?

- Cette solution est elle toujours convenable ?

II – Solution retenue : utilisation de circuits 74HC259

Analyser la table de vérité du circuit 74HC259 (cf. Figure 2 et Figure 3).

- Avec un seul circuit, combien de sorties peuvent être pilotées ? Combien de bits (ou fils) sont nécessaires au pilotage de ce circuit ?
- Quels sont les 4 modes de fonctionnement de ce circuit ?
- D'après le cahier des charges, quels sont les modes de fonctionnement du 74HC259 utiles à notre système et quelle(s) fonction(s) leurs sont associées ?
- Pour les modes « *decoder* » et « *adressable latch* », précisez quand la valeur de la donnée **d** est appliquée sur une sortie **Q**.

III – Liaison microcontrôleur ⇔ système de 16 voyants

On se place maintenant coté microcontrôleur. On associe une adresse à chaque voyant (ou sortie). On pilotera l'état d'une sortie en sélectionnant cette sortie grâce à son adresse puis en écrivant son nouvel état sur le bus de données. Enfin, on validera le changement d'état de la sortie.

- Combien de **bits d'adresses** sont nécessaires au pilotage des 16 voyants ?
- De même pour les **bits de données** ?
- Sachant que l'on veut pouvoir mettre toutes les sorties à 0 facilement et qu'il est nécessaire de valider les données pour effectuer le changement d'état d'une sortie, combien de bits sont nécessaires au **contrôle** de ce système ?

IV – Réalisation du câblage

- A quoi doit-on relier les entrées A_0, A_1 et A_2 puis D des 74HC259 ? Prévoir la validation des données.

- Comment peut on piloter 16 sorties ? Réaliser le schéma de câblage du bus d'adresse, ne pas oublier la validation des données…

- Ajouter la logique pour permettre une mise à zéro de toutes les sorties (on disposera d'un bit $\overline{reset}$ provenant du microcontrôleur).

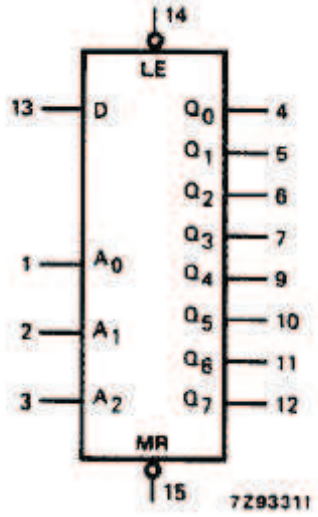


Figure 2 : Schéma fonctionnel du 74HC259.

OPERATING MODES	INPUTS						OUTPUTS							
	$\overline{MR}$	$\overline{LE}$	D	A_0	A_1	A_2	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
master reset	L	H	X	X	X	X	L	L	L	L	L	L	L	L
demultiplex (active HIGH) decoder (when D = H)	L	L	d	L	L	L	Q=d	L	L	L	L	L	L	L
	L	L	d	H	L	L	L	Q=d	L	L	L	L	L	L
	L	L	d	L	H	L	L	L	Q=d	L	L	L	L	L
	L	L	d	H	H	L	L	L	L	Q=d	L	L	L	L
	L	L	d	L	L	H	L	L	L	L	Q=d	L	L	L
	L	L	d	H	L	H	L	L	L	L	L	Q=d	L	L
	L	L	d	L	H	H	L	L	L	L	L	L	Q=d	L
	L	L	d	H	H	H	L	L	L	L	L	L	L	Q=d
store (do nothing)	H	H	X	X	X	X	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
addressable latch	H	L	d	L	L	L	Q=d	q_1	q_2	q_3	q_4	q_5	q_6	q_7
	H	L	d	H	L	L	q_0	Q=d	q_2	q_3	q_4	q_5	q_6	q_7
	H	L	d	L	H	L	q_0	q_1	Q=d	q_3	q_4	q_5	q_6	q_7
	H	L	d	H	H	L	q_0	q_1	q_2	Q=d	q_4	q_5	q_6	q_7
	H	L	d	L	L	H	q_0	q_1	q_2	q_3	Q=d	q_5	q_6	q_7
	H	L	d	H	L	H	q_0	q_1	q_2	q_3	q_4	Q=d	q_6	q_7
	H	L	d	L	H	H	q_0	q_1	q_2	q_3	q_4	q_5	Q=d	q_7
	H	L	d	H	H	H	q_0	q_1	q_2	q_3	q_4	q_5	q_6	Q=d

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care
   d = HIGH or LOW data one set-up time prior to the LOW-to-HIGH $\overline{LE}$ transition
   q = lower case letters indicate the state of the referenced output established during the last cycle in which it was addressed or cleared

Figure 3 : Table de vérité du 74HC259.

TD Microcontrôleur, famille PIC

Séance 4

Première application avec un PIC : faire clignoter une LED

Le but de cette séance de TD est d'écrire le programme assembleur pour le PIC16C84 permettant de faire clignoter une LED. Le schéma de l'application est proposé sur la Figure 1.

Dès que le montage sera mis sous tension, la LED devra clignoter indéfiniment (période de 400 millisecondes).

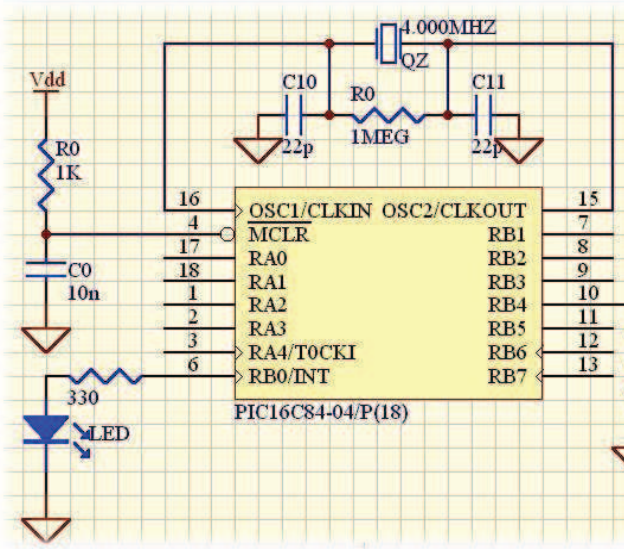


Figure 1 : Schéma de l'application.

I – Organigramme du programme

Dessiner l'organigramme de l'application. Pour la temporisation on fera apparaître une fonction « Tempo ».

II – Fonction Tempo

Le code assembleur suivant est celui d'une fonction de temporisation. Les variables `Tempo_value_B1` et `Tempo_value_B2` sont stockées en RAM aux adresses 0x0020 et 0x0021. Le microcontrôleur est cadencé à 4MHz.

```
1 Tempo_Wms:
2          BCF         STATUS, RP0      ;
3          MOVWF       Tempo_value_B1   ;
4 Tempo_B1:
5          MOVLW       .249             ;
6          MOVWF       Tempo_value_B2   ;
7 Tempo_B2:
8          NOP                          ;
9          DECFSZ      Tempo_value_B2,1;
10         GOTO        Tempo_B2         ;
11         DECFSZ      Tempo_value_B1,1;
12         GOTO        Tempo_B1         ;
13    RETURN                            ;
```

a- Indiquez pour chaque ligne de cette fonction ce qui est réalisé par le microcontrôleur (cf. documents constructeur Figure 3).

b- En déduire l'organigramme de la fonction `Tempo_Wms`.

c- Donner le nombre de cycles correspondant à l'exécution de chaque instruction (cf. documents constructeur Figure 3).

d- En déduire le temps d'exécution de la fonction `Tempo_Wms` en fonction de la valeur contenue dans le registre W (cf. Figure 6).

e- Justifier l'utilisation de l'instruction `NOP` ligne 8 et de la valeur littérale ***249*** ligne 5.

f- Avec un seul appel à cette fonction, quelle est la durée de la temporisation la plus longue que l'on puisse réaliser ?

III – Programme principal

a- Donner les instructions permettant de configurer la broche RB0 pour piloter la LED (cf. documents Figure 2, Figure 4 et Figure 5).

b- Donner les instructions pour allumer et éteindre la LED (cf. Figure 4 et Figure 5).

c- Ecrire en assembleur le programme principal. (Pour obtenir un programme complet, vous pourrez compléter le listing suivant).

```
#include <p16F84.inc>     ; def. des variables du microprocesseur
__CONFIG   _CP_OFF & _WDT_ON & _PWRTE_ON & _RC_OSC; config. du PIC

;***** DEFINITION des VARIABLES
Tempo_value_B1  equ 0x0020
Tempo_value_B2  equ 0x0021

;*************************************************************
      ORG        0x0000     ; vecteur reset
      GOTO       main       ; aller au début du programme

Tempo_Wms:
…
… ; code de la question II
…
      RETURN



main:                       ; programme principal, à vous…

           END                          ; directive 'fin de programme'
```

Annexes :

Extraits de la documentation Microchip PIC16C84

http://ww1.microchip.com/downloads/en/devicedoc/30445c.pdf

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-2 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

FIGURE 4-2: REGISTER FILE MAP

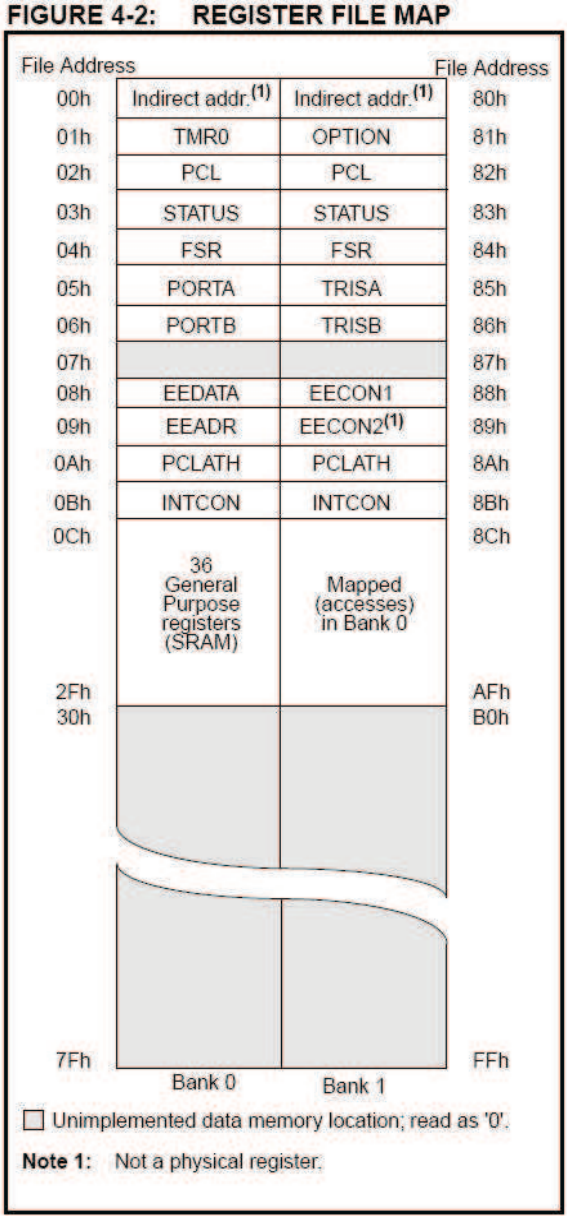


Figure 2: Organisation de la mémoire, PIC 16C84

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

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 3: Jeu d'instructions des PIC 16CXXX

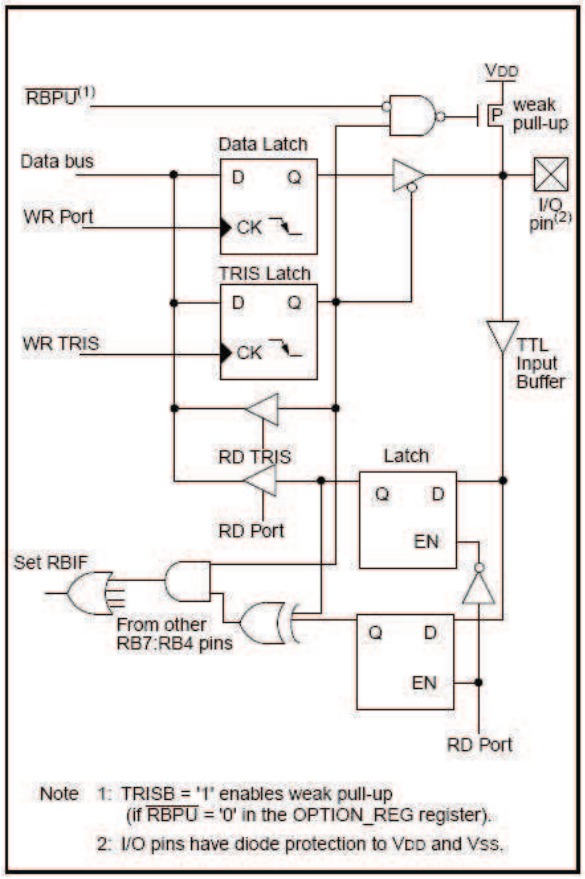
5.2 PORTB and TRISB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. A '1' on any bit in the TRISB register puts the corresponding output driver in a hi-impedance mode. A '0' on any bit in the TRISB register puts the contents of the output latch on the selected pin(s).

Each of the PORTB pins have a weak internal pull-up. A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBPU}}$ (OPTION_REG<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The pins value in input mode are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of the pins are OR'ed together to generate the RB port change interrupt.

FIGURE 5-3: BLOCK DIAGRAM OF PINS RB7:RB4



This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Read (or write) PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set the RBIF bit. Reading PORTB will end the mismatch condition, and allow the RBIF bit to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a key pad and make it possible for wake-up on key-depression (see AN552 in the Embedded Control Handbook).

Note 1: If a change on the I/O pin should occur when a read operation of PORTB is being executed (start of the Q2 cycle), the RBIF interrupt flag bit may not be set.

The interrupt on change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

FIGURE 5-4: BLOCK DIAGRAM OF PINS RB3:RB0

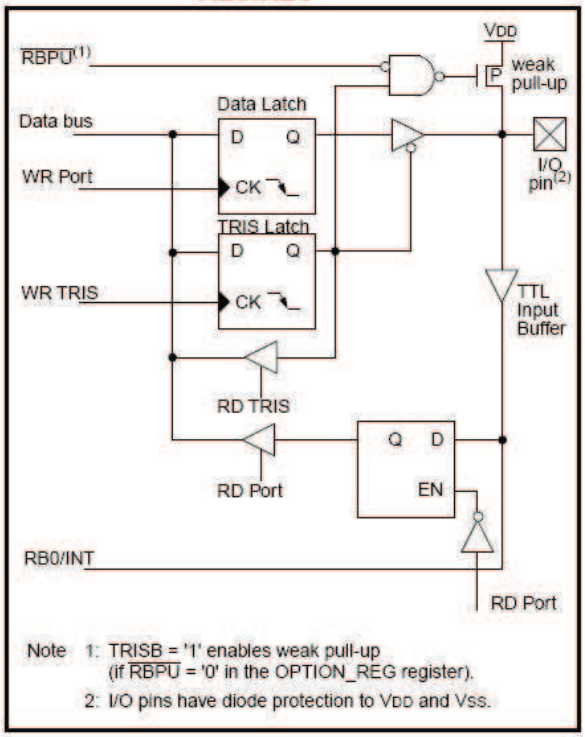


Figure 4 : Fonctionnement du PORTB

```
CLRF    PORTB         ; Initialize PORTB by
                      ; setting output
                      ; data latches
BSF     STATUS, RP0   ; Select Bank 1
MOVLW   0xCF          ; Value used to
                      ; initialize data
                      ; direction
MOVWF   TRISB         ; Set RB<3:0> as inputs
                      ; RB<5:4> as outputs
                      ; RB<7:6> as inputs
```

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

Figure 5 : Exemple d'utilisation et description des registres associés au PORTB.

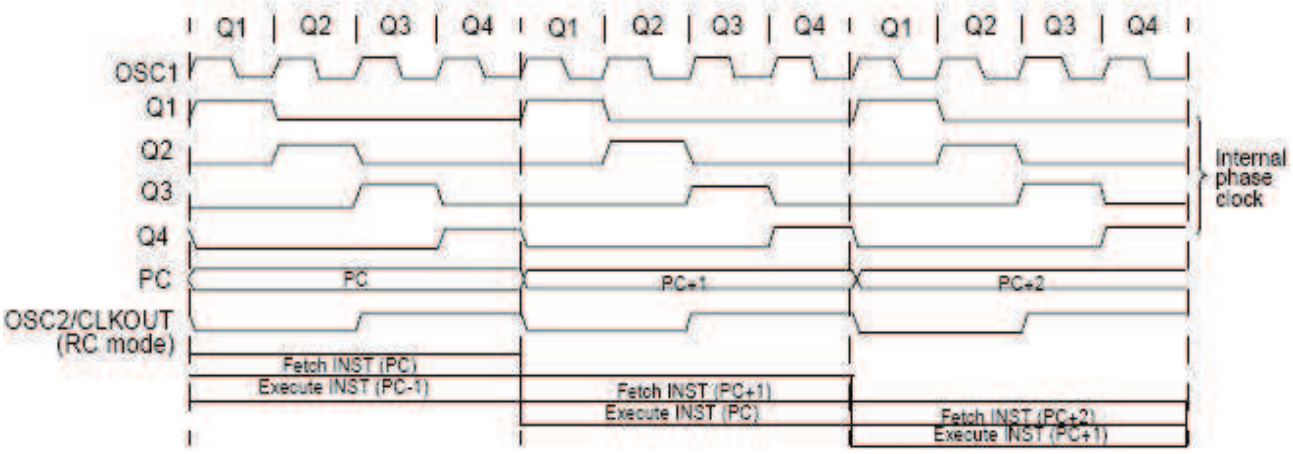


Figure 6 : Cycles horloge et instructions

TD Microcontrôleur, famille PIC

Séances 5 et 6

Scrutation par balayage, application pour le codage des touches d'un clavier

Le but de cette séance de TD est d'écrire en assembleur (famille PIC16XXX) une fonction permettant de scruter un clavier de 16 touches et de donner la valeur de la touche pressée (on parlera de décodage). On supposera qu'une seule touche est appuyée à la fois.

Le schéma interne du clavier est décrit sur la Figure 1.

La fonction de scrutation/décodage sera appelée par le programme principal. Cette fonction devra attendre qu'une touche soit pressée. Elle retournera la valeur décodée (de 0 à F).

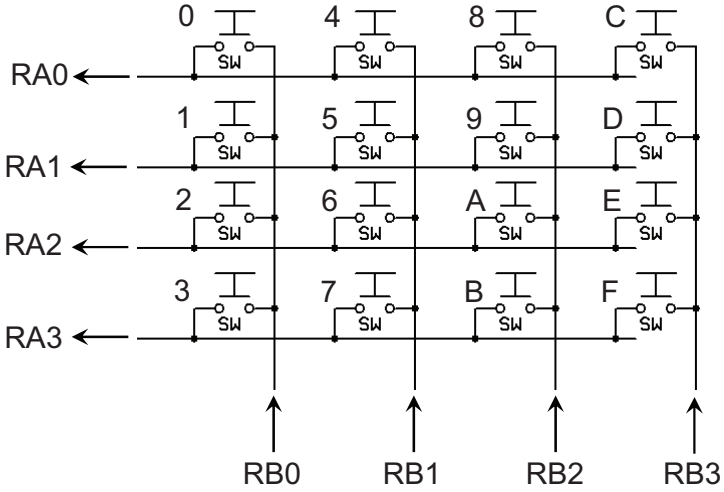


Figure 1 : Schéma fonctionnel du clavier (sans les résistances, les diodes…)

I – Schéma de l'application

- Réaliser le schéma global de l'application (cf. Figure 2). On veillera particulièrement aux résistances de rappels et de protections nécessaires à l'utilisation du clavier.

II – Organigramme

- Décrire le principe de fonctionnement de la scrutation du clavier.
- Donner une méthode permettant de faire le décodage.
- Etablir l'organigramme de la fonction de scrutation/décodage (faire apparaitre ces fonctions).

III – Programmation des fonctions

- Ecrire en assembleur les fonctions de scrutation et de décodage. Le code de la touche (de 0 à F) sera retourné *via* le registre W.

IV – Pour aller plus loin…

- Exemples d'utilisation : minuterie programmable, digicode à 4 chiffres, …
- Programmation en utilisant uniquement le port B (pour la lecture RB4 à RB7), le port A est ainsi libéré.
- Critiquer la solution de programmation mise en place pour la scrutation et proposer une méthode plus adaptée.

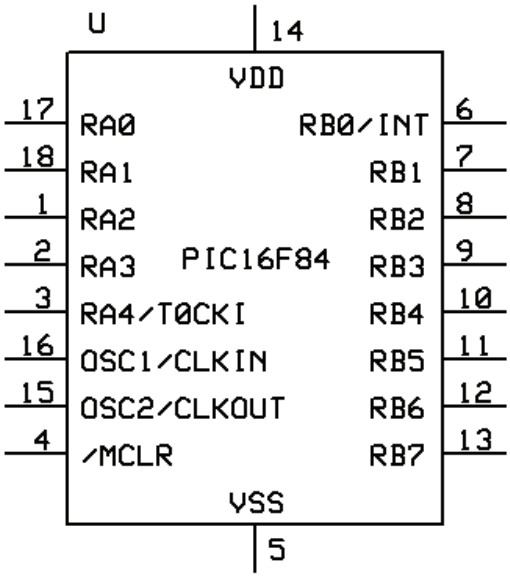


Figure 2: Schéma du PIC 16F84

Annexes :

Extraits de la documentation Microchip PIC16C84

http://ww1.microchip.com/downloads/en/devicedoc/30445c.pdf

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-2 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

FIGURE 4-2: REGISTER FILE MAP

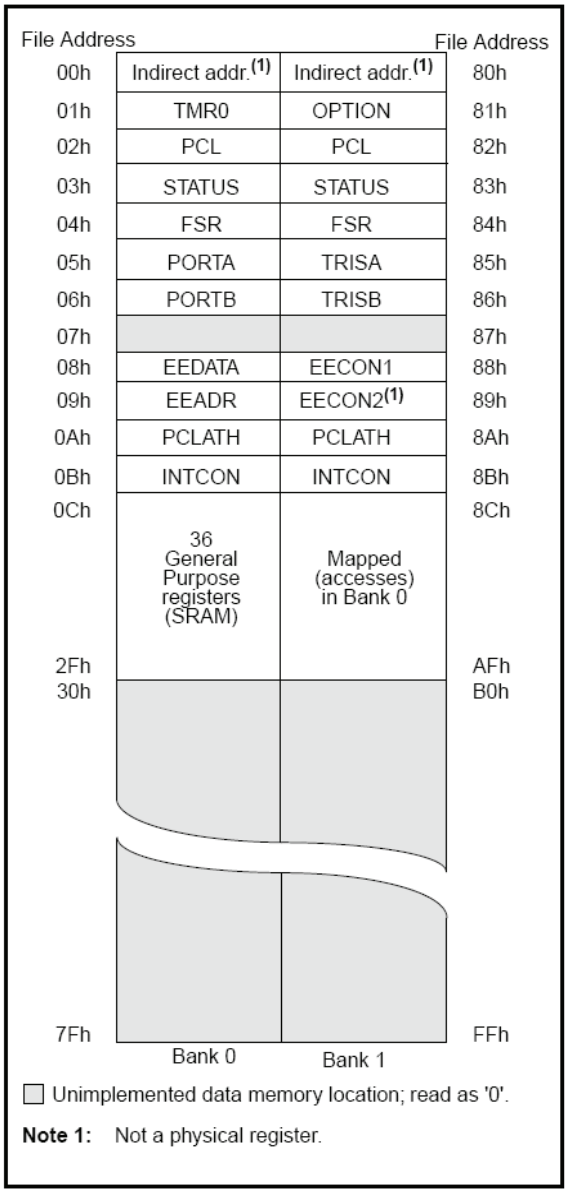


Figure 3: Organisation de la mémoire, PIC 16C84

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

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 4: Jeu d'instructions des PIC 16CXXX

5.0 I/O PORTS

The PIC16C84 has two ports, PORTA and PORTB. Some port pins are multiplexed with an alternate function for other features on the device.

5.1 PORTA and TRISA Registers

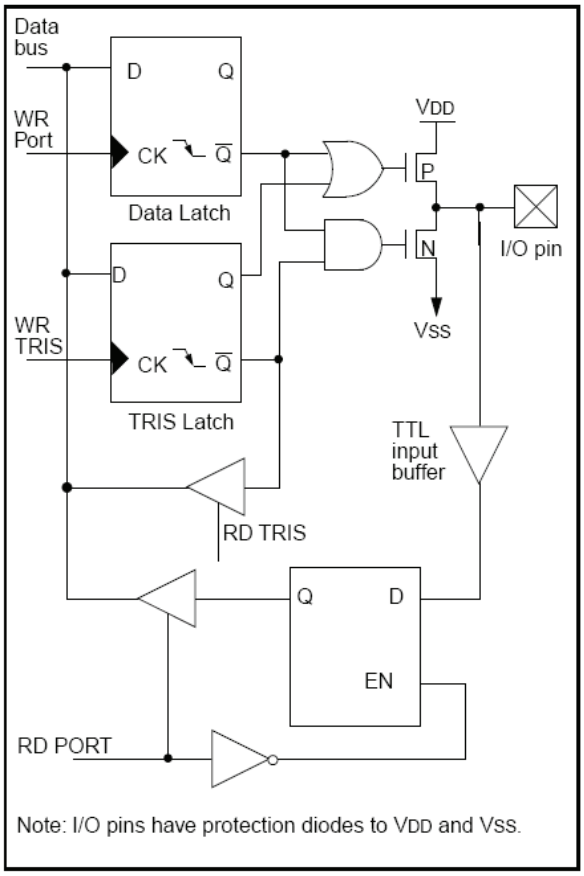
PORTA is a 5-bit wide latch. RA4 is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as output or input.

Setting a TRISA bit (=1) will make the corresponding PORTA pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output, i.e., put the contents of the output latch on the selected pin.

Reading the PORTA register reads the status of the pins whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch.

The RA4 pin is multiplexed with the TMR0 clock input.

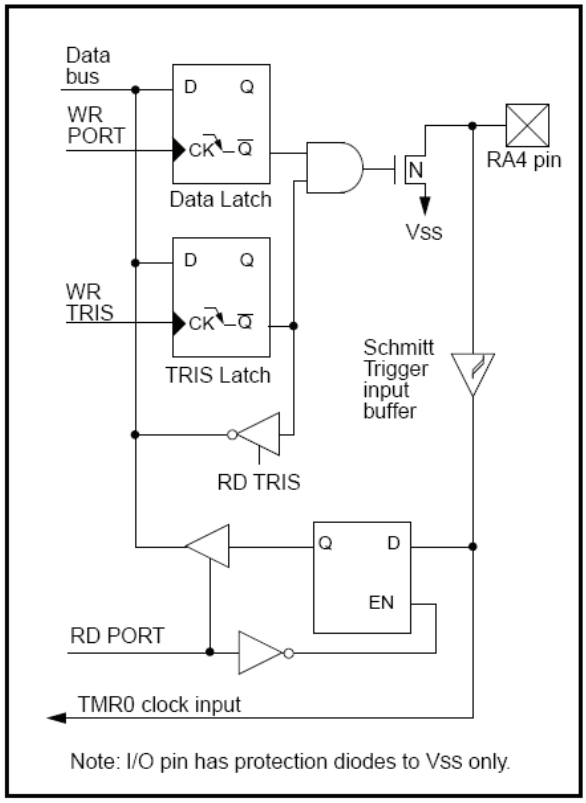
FIGURE 5-1: BLOCK DIAGRAM OF PINS RA3:RA0



EXAMPLE 5-1: INITIALIZING PORTA

```
CLRF    PORTA       ; Initialize PORTA by
                    ; setting output
                    ; data latches
BSF     STATUS, RP0 ; Select Bank 1
MOVLW   0x0F        ; Value used to
                    ; initialize data
                    ; direction
MOVWF   TRISA       ; Set RA<3:0> as inputs
                    ; RA4 as outputs
                    ; TRISA<7:5> are always
                    ; read as '0'.
```

FIGURE 5-2: BLOCK DIAGRAM OF PIN RA4



Note: For crystal oscillator configurations operating below 500 kHz, the device may generate a spurious internal Q-clock when PORTA<0> switches state. This does not occur with an external clock in RC mode. To avoid this, the RA0 pin should be kept static, i.e. in input/output mode, pin RA0 should not be toggled.

Figure 5 : Description du fonctionnement du PORT A

5.2 PORTB and TRISB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. A '1' on any bit in the TRISB register puts the corresponding output driver in a hi-impedance mode. A '0' on any bit in the TRISB register puts the contents of the output latch on the selected pin(s).

Each of the PORTB pins have a weak internal pull-up. A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBPU}}$ (OPTION_REG<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The pins value in input mode are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of the pins are OR'ed together to generate the RB port change interrupt.

FIGURE 5-3: BLOCK DIAGRAM OF PINS RB7:RB4

<

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Read (or write) PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set the RBIF bit. Reading PORTB will end the mismatch condition, and allow the RBIF bit to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a key pad and make it possible for wake-up on key-depression (see AN552 in the Embedded Control Handbook).

Note 1: If a change on the I/O pin should occur when a read operation of PORTB is being executed (start of the Q2 cycle), the RBIF interrupt flag bit may not be set.

The interrupt on change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

Figure 6 : Description du fonctionnement du PORT B

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

Figure 7 : Détails des registres du PIC16C84.

TD Microcontrôleur, famille PIC

Séance 7

Exercices de programmation en assembleur

Il est conseillé de savoir faire ces exercices.

Conseils :

- commencer par faire un organigramme,
- ensuite, traduire rigoureusement et étape par étape votre organigramme
- ne pas hésiter à ajouter des variables temporaires (en RAM) nécessaires au codage assembleur,
- ne chercher pas trop à optimiser votre code et l'utilisation de la mémoire,
- utiliser des noms de variables clairs (valeurs, fonction ou de boucle),
- encore une fois : utiliser des variables (temporaires, paramètres, …) en RAM.

I – Table de retour de fonction

a- Réaliser la fonction de décodage « hexadécimal ➔ 7 segments » (cf. cours).

b- On veut connaître les valeurs approchées de *sin(x)* pour *x* entier et compris entre 0° et 90°. **Proposer** une solution de mise en œuvre (attention aux valeurs réelles…).

II – Adressage indexé

a- Ecrire le programme permettant d'additionner 2 vecteurs de taille *n.* Les vecteurs sont en mémoire RAM aux adresses *v1* et *v2*, ainsi que leur taille *n*. Le vecteur résultat *s* sera aussi stocké en RAM. On suppose que la somme de chaque composante est inférieure à 255. (cf. Figure 4)

b- Les 46 valeurs pour x=0°, 2°, 4°, …, 90° de $\lfloor 255.\sin(x) \rfloor$ ont été écrites dans les 46 premières adresses de l'EEPROM du PIC. Ecrire la fonction retournant la valeur de *sin(x)*, $x \in [0°;90°]$. (cf. Figure 5 et Figure 6).

c- Modifier la fonction précédente afin d'obtenir *sin(x)* pour $x \in [0°;180°]$ (cf. Figure 8).

III – Arithmétique : addition 16 bits

- Faire la fonction ***Add16*** permettant d'additionner deux valeurs *a* et *b* de 16 bits. Le résultat de l'addition *s* fera aussi 16 bits. Chaque variable 16 bits sera composée de 2 octets : la partie basse (octet de poids faible, par exemple *al* pour *a*) et la partie haute (poids fort, *ah*).
- Ecrire un programme utilisant la fonction *Add16*. Les variables *a* et *b* seront à initialiser aux valeurs de votre choix.
- Signaler le dépassement de capacité (cf. Figure 3) en allumant une LED sur PORTB<0>.

IV – Arithmétique : Multiplication 8 bits

- Faire l'organigramme d'une fonction permettant de multiplier deux valeurs 8 bits *a* et *b*. Le résultat sur 16 bits sera écrit dans 2 variables 8 bits *ml* (octet de poids faible) et *mh* (octet de poids fort).
- Ecrire le code assembleur de cette fonction.

Annexes :

Extraits de la documentation Microchip PIC16C84

http://ww1.microchip.com/downloads/en/devicedoc/30445c.pdf

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-2 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

FIGURE 4-2: REGISTER FILE MAP

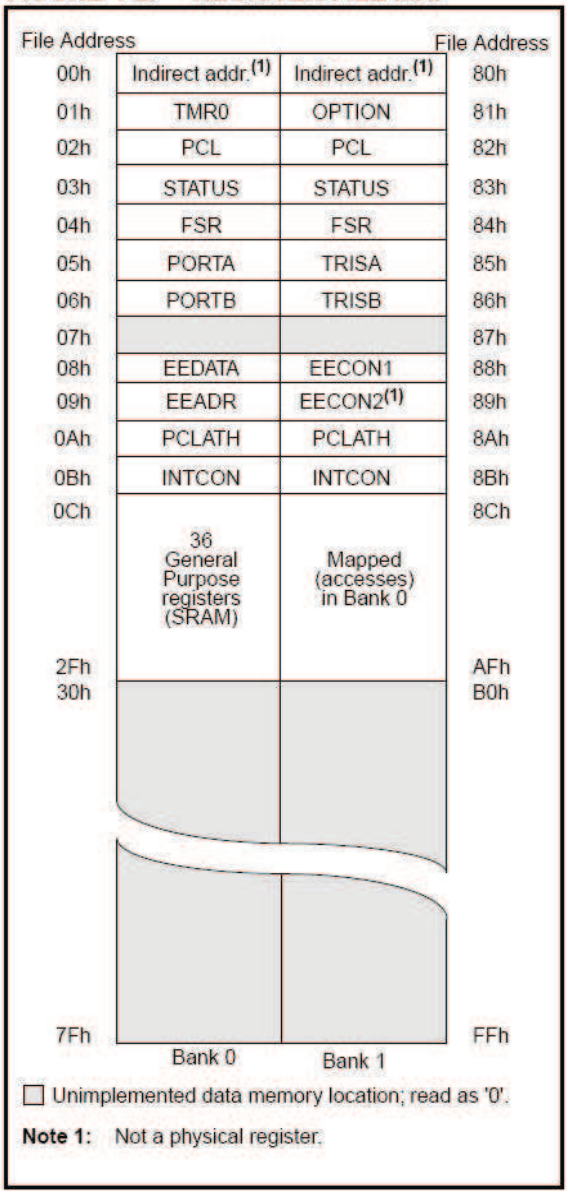


Figure 1: Organisation de la mémoire, PIC 16C84

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

Figure 2 : Détails des registres du PIC16C84.

4.2.2.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper-three bits and set the Z bit. This leaves the STATUS register as `000u uluu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 9-2) because these instructions do not affect any status bit.

Note 1: The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16C84 and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

Note 2: The C and DC bits operate as a $\overline{borrow}$ and digit $\overline{borrow}$ out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

Note 3: When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

FIGURE 4-3: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
0 = Bank 0, 1 (00h - FFh)
1 = Bank 2, 3 (100h - 1FFh)
The IRP bit is not used by the PIC16C8X. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16C8X. RP1 should be maintained clear.

bit 4: **TO**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3: **PD**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions) (For $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Figure 3: Détail du registre STATUS

4.5 Indirect Addressing: INDF and FSR Registers

The INDF register is not a physical register. Addressing INDF actually addresses the register whose address is contained in the FSR register (FSR is a *pointer*). This is indirect addressing.

EXAMPLE 4-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR = 0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected).

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 4-2.

EXAMPLE 4-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

```
        movlw   0x20    ;initialize pointer
        movwf   FSR     ;  to RAM
NEXT    clrf    INDF    ;clear INDF register
        incf    FSR     ;inc pointer
        btfss   FSR,4   ;all done?
        goto    NEXT    ;NO, clear next
CONTINUE
        :               ;YES, continue
```

An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 4-7. However, IRP is not used in the PIC16C84.

FIGURE 4-7: DIRECT/INDIRECT ADDRESSING

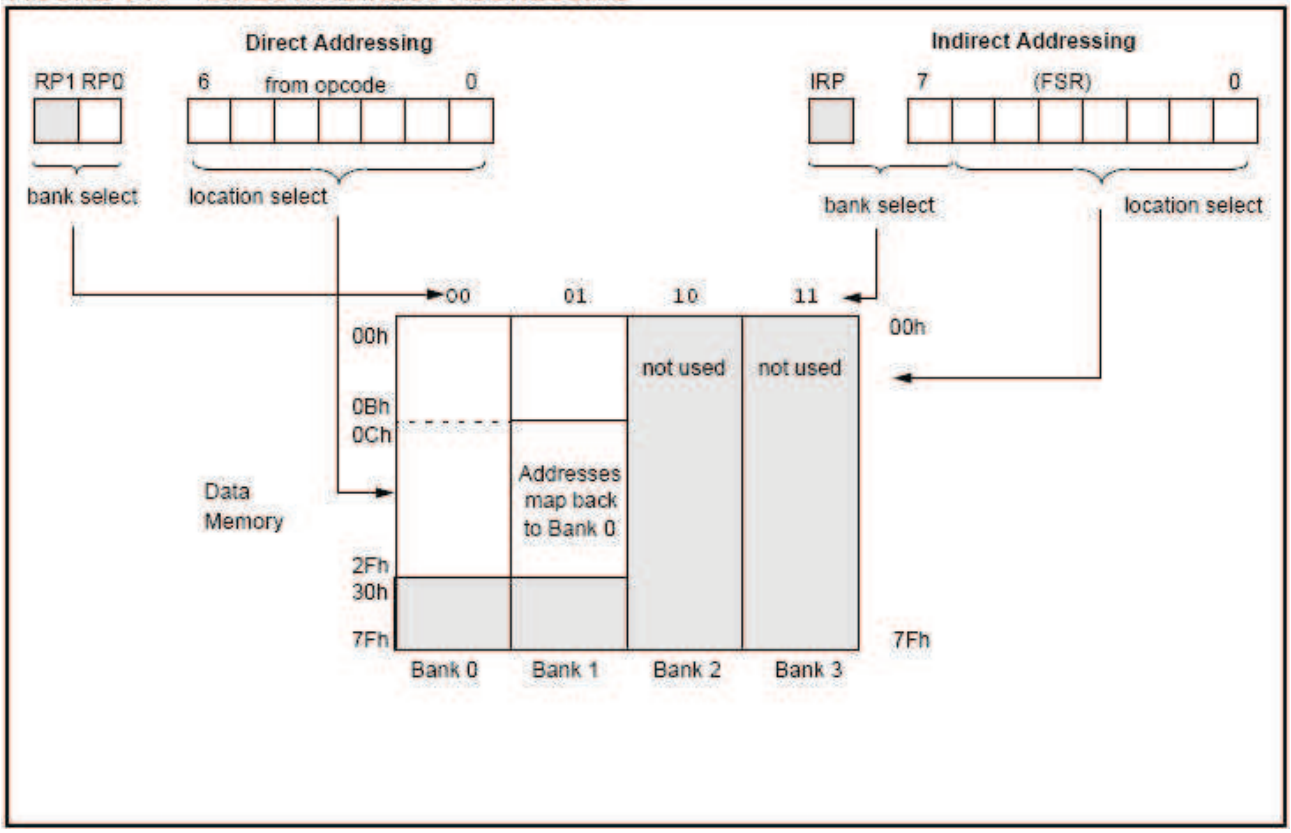


Figure 4 : Description de l'adressage indirect

7.0 DATA EEPROM MEMORY

The EEPROM data memory is readable and writable during normal operation (full VDD range). This memory is not directly mapped in the register file space. Instead it is indirectly addressed through the Special Function Registers. There are four SFRs used to read and write this memory. These registers are:

- EECON1
- EECON2
- EEDATA
- EEADR

EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. PIC16C84 devices have 64 bytes of data EEPROM with an address range from 0h to 3Fh.

The EEPROM data memory allows byte read and write. A byte write automatically erases the location and writes the new data (erase before write). The EEPROM data memory is rated for high erase/write cycles. The write time is controlled by an on-chip timer. The write-time will vary with voltage and temperature as well as from chip to chip. Please refer to AC specifications for exact limits.

When the device is code protected, the CPU may continue to read and write the data EEPROM memory. The device programmer can no longer access this memory.

7.1 EEADR

The EEADR register can address up to a maximum of 256 bytes of data EEPROM. Only the first 64 bytes of data EEPROM are implemented.

The upper two bits are address decoded. This means that these two bits must always be '0' to ensure that the address is in the 64 byte memory space.

FIGURE 7-1: EECON1 REGISTER (ADDRESS 88h)

U	U	U	R/W-0	R/W-x	R/W-0	R/S-0	R/S-x
—	—	—	EEIF	WRERR	WREN	WR	RD
bit7							bit0

R = Readable bit
W = Writable bit
S = Settable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7:5 **Unimplemented:** Read as '0'

bit 4 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{MCLR}$ reset or any WDT reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the data EEPROM

bit 1 **WR**: Write Control bit
1 = initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the data EEPROM is complete

bit 0 **RD**: Read Control bit
1 = Initiates an EEPROM read (read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software).
0 = Does not initiate an EEPROM read

Figure 5: Description des accès à l'EEPROM (1/2)

7.2 EECON1 and EECON2 Registers

EECON1 is the control register with five low order bits physically implemented. The upper-three bits are non-existent and read as '0's.

Control bits RD and WR initiate read and write, respectively. These bits cannot be cleared, only set, in software. They are cleared in hardware at completion of the read or write operation. The inability to clear the WR bit in software prevents the accidental, premature termination of a write operation.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ reset or a WDT time-out reset during normal operation. In these situations, following reset, the user can check the WRERR bit and rewrite the location. The data and address will be unchanged in the EEDATA and EEADR registers.

Interrupt flag bit EEIF is set when write is complete. It must be cleared in software.

EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the Data EEPROM write sequence.

7.3 Reading the EEPROM Data Memory

To read a data memory location, the user must write the address to the EEADR register and then set control bit RD (EECON1<0>). The data is available, in the very next cycle, in the EEDATA register; therefore it can be read in the next instruction. EEDATA will hold this value until another read or until it is written to by the user (during a write operation).

EXAMPLE 7-1: DATA EEPROM READ

```
BCF     STATUS, RP0  ; Bank 0
MOVLW   CONFIG_ADDR  ;
MOVWF   EEADR        ; Address to read
BSF     STATUS, RP0  ; Bank 1
BSF     EECON1, RD   ; EE Read
BCF     STATUS, RP0  ; Bank 0
MOVF    EEDATA, W    ; W = EEDATA
```

7.4 Writing to the EEPROM Data Memory

To write an EEPROM data location, the user must first write the address to the EEADR register and the data to the EEDATA register. Then the user must follow a specific sequence to initiate the write for each byte.

EXAMPLE 7-1: DATA EEPROM WRITE

```
        BSF     STATUS, RP0  ; Bank 1
        BCF     INTCON, GIE  ; Disable INTs.
        BSF     EECON1, WREN ; Enable Write
        MOVLW   55h          ;
Required
Sequence
        MOVWF   EECON2       ; Write 55h
        MOVLW   AAh          ;
        MOVWF   EECON2       ; Write AAh
        BSF     EECON1,WR    ; Set WR bit
                             ;   begin write
        BSF     INTCON, GIE  ; Enable INTs.
```

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. We strongly recommend that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable write. This mechanism prevents accidental writes to data EEPROM due to errant (unexpected) code execution (i.e., lost programs). The user should keep the WREN bit clear at all times, except when updating EEPROM. The WREN bit is not cleared by hardware

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. The WR bit will be inhibited from being set unless the WREN bit is set.

At the completion of the write cycle, the WR bit is cleared in hardware and the EE Write Complete Interrupt Flag bit (EEIF) is set. The user can either enable this interrupt or poll this bit. EEIF must be cleared by software.

Note: The data EEPROM memory E/W cycle time may occasionally exceed the 10 ms specification (typical). To ensure that the write cycle is complete, use the EE interrupt or poll the WR bit (EECON1<1>). Both these events signify the completion of the write cycle.

Figure 6 : Description des accès à l'EEPROM (2/2)

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

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 7: Jeu d'instructions des PIC 16CXXX

SUBLW **Subtract W from Literal**

Syntax: [*label*] SUBLW k

Operands: $0 \le k \le 255$

Operation: k - (W) → (W)

Status Affected: C, DC, Z

Encoding:

11	110x	kkkk	kkkk

Description: The W register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example 1:
```
SUBLW   0x02
```
Before Instruction
W = 1
C = ?
Z = ?

After Instruction
W = 1
C = 1; result is positive
Z = 0

Example 2:
Before Instruction
W = 2
C = ?
Z = ?

After Instruction
W = 0
C = 1; result is zero
Z = 1

Example 3:
Before Instruction
W = 3
C = ?
Z = ?

After Instruction
W = 0xFF
C = 0; result is negative
Z = 0

SUBWF **Subtract W from f**

Syntax: [*label*] SUBWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - (W) → (destination)

Status Affected: C, DC, Z

Encoding:

00	0010	dfff	ffff

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1:
```
SUBWF   REG1,1
```
Before Instruction
REG1 = 3
W = 2
C = ?
Z = ?

After Instruction
REG1 = 1
W = 2
C = 1; result is positive
Z = 0

Example 2:
Before Instruction
REG1 = 2
W = 2
C = ?
Z = ?

After Instruction
REG1 = 0
W = 2
C = 1; result is zero
Z = 1

Example 3:
Before Instruction
REG1 = 1
W = 2
C = ?
Z = ?

After Instruction
REG1 = 0xFF
W = 2
C = 0; result is negative
Z = 0

Figure 8 : Description détaillée des instructions SUBLW et SUBWF

TD Microcontrôleur, famille PIC

Séance 8

Application :

Pilotage d'une sortie parmi 8

On souhaite piloter une sortie parmi 8 à l'aide d'un clavier 16 touches. Seules les 8 premières touches (de 0 à 7) sont utilisées, chaque touche est associée à une sortie. L'appuie sur une touche provoque l'inversion de l'état logique de la sortie correspondante.

A l'initialisation, on souhaite que toutes les sorties soient inactives (état logique bas).

On disposera des fonctions suivantes :

Tempo_Wms : qui permet d'attendre W millisecondes pour un PIC cadencé à 4MHz.

Scrute_Clavier : qui scrute le clavier tant qu'aucune touche n'est enfoncée et retourne la valeur de la touche enfoncée dans W. Le clavier doit être connecté au PORTB : bits 0 à 3 pour l'écriture colonne active, bits 4 à 7 pour la lecture des lignes (détection d'une touche si état haut, état bas si aucune touche n'est enfoncée).

On disposera d'un PIC 16F84.

On utilisera un démultiplexeur pour le pilotage des 8 sorties (type 74HC259, cf. Figure 10 et Figure 11).

Le clavier est un ensemble de 16 boutons poussoir câblés de manière à permettre la scrutation par balayage (Figure 12).

I – Schéma de l'application

a- Expliquer et justifier le choix des composants du schéma de l'application (Figure 1). On s'intéressera notamment aux rôles et aux valeurs des résistances R1 à R8.

II – Organigramme de l'application

a- Faire l'organigramme global de l'application.

b- Détailler le pilotage des sorties (cf. document 74HC259, Figure 11).

III – Programmation

Ecrire le programme en assembleur. On pourra faire appel aux fonctions *Tempo_Wms* et *Scrute_Clavier* (ne pas réécrire ces fonctions).

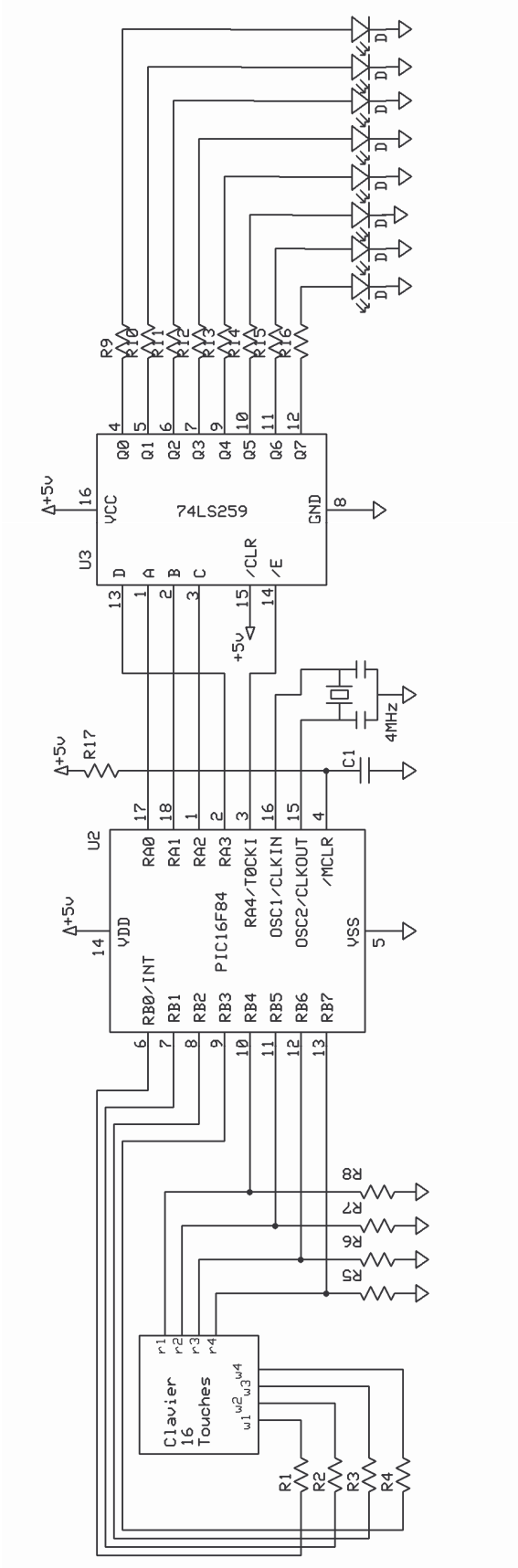


Figure 1 : Schéma Application 8 sorties

Annexe 1:

Extraits de la documentation Microchip PIC16C84

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-2 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

FIGURE 4-2: REGISTER FILE MAP

File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	36 General Purpose registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
2Fh			AFh
30h			B0h
7Fh			FFh

☐ Unimplemented data memory location; read as '0'.

Note 1: Not a physical register.

Figure 2: Organisation de la mémoire, PIC 16C84

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

Figure 3 : Détails des registres du PIC16C84.

4.2.2.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper-three bits and set the Z bit. This leaves the STATUS register as `000u uluu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 9-2) because these instructions do not affect any status bit.

Note 1: The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16C84 and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

Note 2: The C and DC bits operate as a $\overline{borrow}$ and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

Note 3: When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

FIGURE 4-3: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
0 = Bank 0, 1 (00h - FFh)
1 = Bank 2, 3 (100h - 1FFh)
The IRP bit is not used by the PIC16C8X. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16C8X. RP1 should be maintained clear.

bit 4: **TO**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3: **PD**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions) (For $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Figure 4: Détail du registre STATUS

5.0 I/O PORTS

The PIC16C84 has two ports, PORTA and PORTB. Some port pins are multiplexed with an alternate function for other features on the device.

5.1 PORTA and TRISA Registers

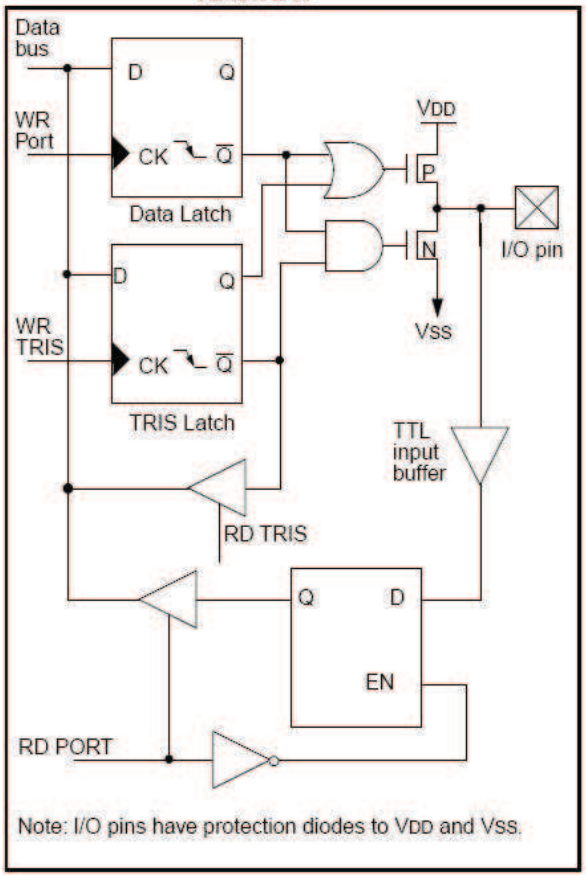
PORTA is a 5-bit wide latch. RA4 is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as output or input.

Setting a TRISA bit (=1) will make the corresponding PORTA pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output, i.e., put the contents of the output latch on the selected pin.

Reading the PORTA register reads the status of the pins whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch.

The RA4 pin is multiplexed with the TMR0 clock input.

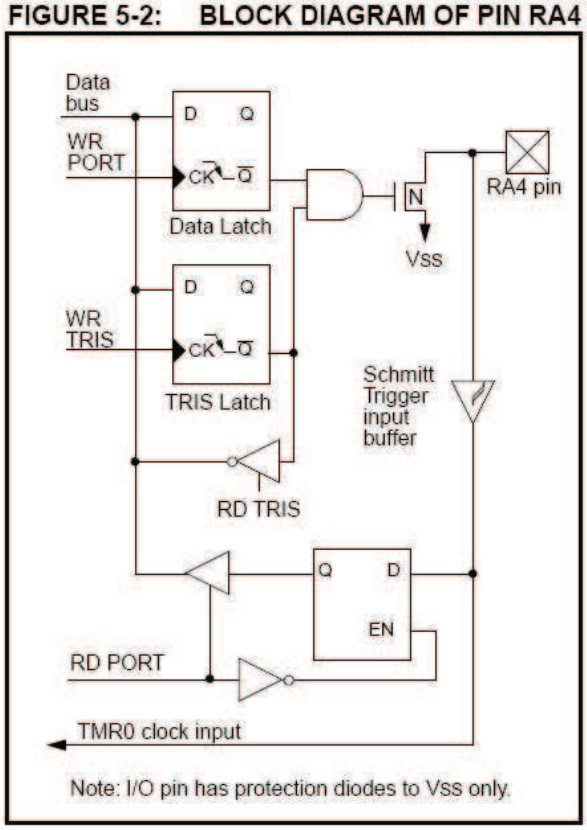
FIGURE 5-1: BLOCK DIAGRAM OF PINS RA3:RA0



EXAMPLE 5-1: INITIALIZING PORTA

```
CLRF    PORTA       ; Initialize PORTA by
                    ; setting output
                    ; data latches
BSF     STATUS, RP0 ; Select Bank 1
MOVLW   0x0F        ; Value used to
                    ; initialize data
                    ; direction
MOVWF   TRISA       ; Set RA<3:0> as inputs
                    ; RA4 as outputs
                    ; TRISA<7:5> are always
                    ; read as '0'.
```

FIGURE 5-2: BLOCK DIAGRAM OF PIN RA4



Note: For crystal oscillator configurations operating below 500 kHz, the device may generate a spurious internal Q-clock when PORTA<0> switches state. This does not occur with an external clock in RC mode. To avoid this, the RA0 pin should be kept static, i.e. in input/output mode, pin RA0 should not be toggled.

Figure 5 : Description du fonctionnement du PORT A

5.2 PORTB and TRISB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. A '1' on any bit in the TRISB register puts the corresponding output driver in a hi-impedance mode. A '0' on any bit in the TRISB register puts the contents of the output latch on the selected pin(s).

Each of the PORTB pins have a weak internal pull-up. A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBPU}}$ (OPTION_REG<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The pins value in input mode are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of the pins are OR'ed together to generate the RB port change interrupt.

FIGURE 5-3: BLOCK DIAGRAM OF PINS RB7:RB4

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Read (or write) PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set the RBIF bit. Reading PORTB will end the mismatch condition, and allow the RBIF bit to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a key pad and make it possible for wake-up on key-depression (see AN552 in the Embedded Control Handbook).

Note 1: If a change on the I/O pin should occur when a read operation of PORTB is being executed (start of the Q2 cycle), the RBIF interrupt flag bit may not be set.

The interrupt on change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

Figure 6 : Description du fonctionnement du PORT B

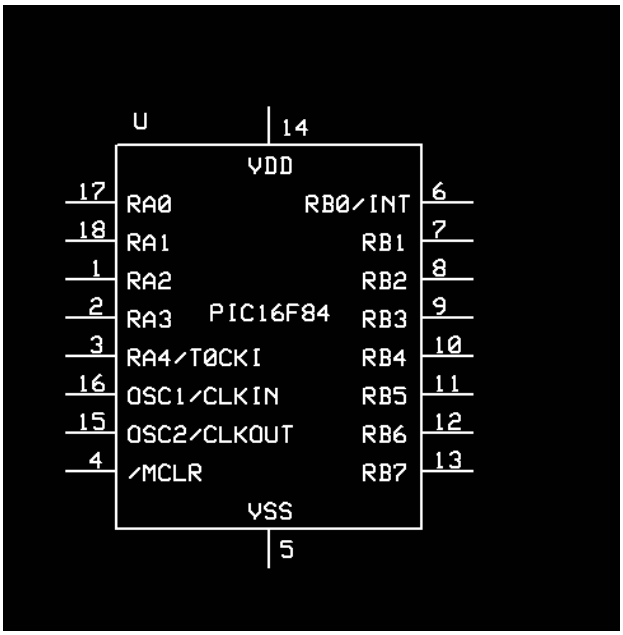


Figure 7: Schéma du PIC 16F84

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

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 8: Jeu d'instructions des PIC 16CXXX

DC Characteristics All Pins Except Power Supply Pins			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C (commercial) -40°C ≤ TA ≤ +85°C (industrial) Operating voltage VDD range as described in DC spec Section 11-1 and Section 11.2.				
Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
		Input Low Voltage					
	VIL	I/O ports					
D030		with TTL buffer	VSS	—	0.8	V	4.5 ≤ VDD ≤ 5.5V
D030A			VSS	—	0.16VDD	V	entire range(4)
D031		with Schmitt Trigger buffer	VSS	—	0.2VDD	V	entire range
D032		MCLR, RA4/T0CKI, OSC1 (RC mode)	Vss	—	0.2VDD	V	
D033		OSC1 (XT, HS and LP modes)(1)	Vss	—	0.3VDD	V	
		Input High Voltage					
	VIH	I/O ports					
D040		with TTL buffer	0.36VDD	—	VDD	V	4.5 ≤ VDD ≤ 5.5V
D040A			0.48VDD				entire range(4)
D041		with Schmitt Trigger buffer	0.45VDD	—	VDD		entire range
D042		MCLR, RA4/T0CKI, OSC1 (RC mode)	0.85VDD	—	VDD	V	
D043		OSC1 (XT, HS and LP modes)(1)	0.7VDD	—	VDD	V	
D070	IPURB	PORTB weak pull-up current	50*	250*	400*	µA	VDD = 5V, VPIN = VSS
		Input Leakage Current(2,3)					
D060	IIL	I/O ports	—	—	±1	µA	Vss ≤ VPIN ≤ VDD, Pin at hi-impedance
D061		MCLR, RA4/T0CKI	—	—	±5	µA	Vss ≤ VPIN ≤ VDD
D063		OSC1/CLKIN	—	—	±5	µA	Vss ≤ VPIN ≤ VDD, XT, HS and LP osc configuration
		Output Low Voltage					
D080	VOL	I/O ports	—	—	0.6	V	IOL = 8.5 mA, VDD = 4.5V
D083		OSC2/CLKOUT (RC osc configuration)	—	—	0.6	V	IOL = 1.6 mA, VDD = 4.5V
		Output High Voltage					
D090	VOH	I/O ports(3)	VDD - 0.7	—	—	V	IOH = -3.0 mA, VDD = 4.5V
D093		OSC2/CLKOUT (RC osc configuration)	VDD - 0.7	—	—	V	IOH = -1.3 mA, VDD = 4.5V

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: In RC oscillator configuration, the OSC1/CLKIN pin is a Schmitt Trigger input. It is not recommended that the PIC16C84 be driven with external clock in RC mode.

2: The leakage current on the MCLR pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

4: The user may use better of the two specs.

Figure 9: Caractéristiques électriques

Annexe 2:

Extraits de la documentation du 74HC259

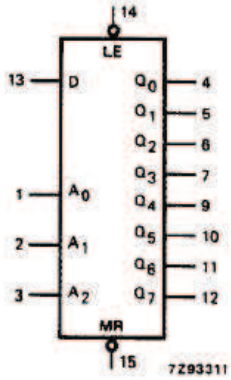


Figure 10 : Schéma fonctionnel du 74HC259

OPERATING MODES	INPUTS						OUTPUTS							
	$\overline{MR}$	$\overline{LE}$	D	A_0	A_1	A_2	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
master reset	L	H	X	X	X	X	L	L	L	L	L	L	L	L
demultiplex (active HIGH) decoder (when D = H)	L	L	d	L	L	L	Q=d	L	L	L	L	L	L	L
	L	L	d	H	L	L	L	Q=d	L	L	L	L	L	L
	L	L	d	L	H	L	L	L	Q=d	L	L	L	L	L
	L	L	d	H	H	L	L	L	L	Q=d	L	L	L	L
	L	L	d	L	L	H	L	L	L	L	Q=d	L	L	L
	L	L	d	H	L	H	L	L	L	L	L	Q=d	L	L
	L	L	d	L	H	H	L	L	L	L	L	L	Q=d	L
	L	L	d	H	H	H	L	L	L	L	L	L	L	Q=d
store (do nothing)	H	H	X	X	X	X	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
addressable latch	H	L	d	L	L	L	Q=d	q_1	q_2	q_3	q_4	q_5	q_6	q_7
	H	L	d	H	L	L	q_0	Q=d	q_2	q_3	q_4	q_5	q_6	q_7
	H	L	d	L	H	L	q_0	q_1	Q=d	q_3	q_4	q_5	q_6	q_7
	H	L	d	H	H	L	q_0	q_1	q_2	Q=d	q_4	q_5	q_6	q_7
	H	L	d	L	L	H	q_0	q_1	q_2	q_3	Q=d	q_5	q_6	q_7
	H	L	d	H	L	H	q_0	q_1	q_2	q_3	q_4	Q=d	q_6	q_7
	H	L	d	L	H	H	q_0	q_1	q_2	q_3	q_4	q_5	Q=d	q_7
	H	L	d	H	H	H	q_0	q_1	q_2	q_3	q_4	q_5	q_6	Q=d

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care
   d = HIGH or LOW data one set-up time prior to the LOW-to-HIGH $\overline{LE}$ transition
   q = lower case letters indicate the state of the referenced output established during the last cycle in which it was addressed or cleared

Figure 11 : Table de vérité du 74HC259

Annexe 3:

Schéma du clavier.

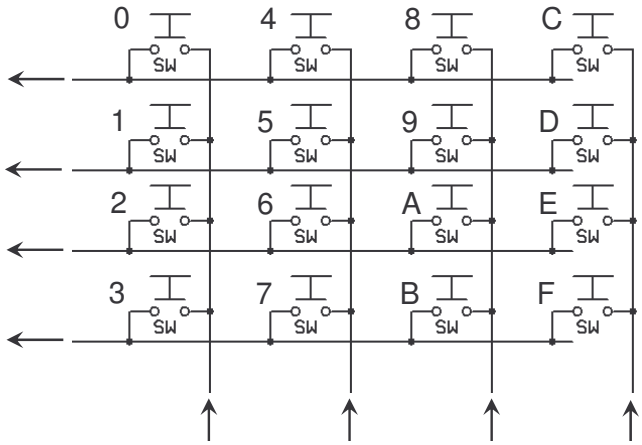


Figure 12 : Schéma fonctionnel du clavier (sans les résistances, les diodes…)