Aix*Marseille université

Faculté des Sciences

Licence 2 SPI

Informatique industrielle

Travaux dirigés

Julien Marot

Polycopiés et examens : vue d'ensemble

Sujet d'examen	Thèmes	Mots-clés présents dans le sujet d'exam	Chapitres correspondants pour chaque thème dans le poly de cours
Sujet 1 : janvier 2010	Généralités	Microcontrôleur, Microprocesseur	Présentation de l'informatique industrielle
	architecture	Bus, UAL, mémoire, ports	Architecture des microcontrôleurs
	Jeu d'instructions		
Sujet 2 : janvier 2012	Interruptions		
	Codage		

Sujet 3 : Mai 2013	Généralités	Critères de performances	Présentation de l'informatique industrielle
		Instruction assembleur	Les instructions
	Architecture		
	Codage		
	Problème		

Aix*Marseille université

Faculté des Sciences

Licence 2 SPI

Informatique industrielle

Travaux dirigés

Introduction au langage assembleur

Le langage assembleur est celui qui sera utilisé en TP pour programmer les cartes PIC. Dans ce TD, on se familiarisera avec ce langage en examinant ses liens avec un langage de programmation de type texte structuré comme Matlab.

1 Objectifs

-Comparer le langage assembleur et le langage Matlab ;
-Comprendre la le lien entre le langage assembleur et la structure du PIC

2 Exercices

2.1 Exercice 1

Voici quelques instructions assembleur. Associez chaque instruction à sa signification.

1) movlw	A) déplacer le contenu du registre W dans un autre registre
2) movwf	B) déplacer le contenu d'un registre dans un autre registre
3) movff	C) déplacer un nombre dans le registre w

Julien Marot

2.2 Exercice 2

Traduisez les lignes de code en langage assembleur ci-dessous en langage Matlab :

movlw (07)h

movwf PORTA

movff PORTA, PORTB

2.3 Exercice 3

Traduisez les lignes de code assembleur ci-dessous en langage Matlab :

```
        movlw (00)h
        movwf PORTA
label   incf PORTA
        movff PORTA, PORTB
        goto label
```

2.4 Exercice 4

Traduisez les lignes de code en langage Matlab ci-dessous en langage assembleur:

```
PORTC=3;
if (PORTA(1)==0)
   PORTB=PORTC;
elseif (PORTA(1)==1)
   PORTD=PORTC;
end
```

Pour cela vous choisirez des instructions assembleur parmi les suivantes :

- movlw, movwf, movff,
- btfsc (bit test file skip if clear),
- goto
- return

Julien Marot

Aix*Marseille université

Faculté des Sciences

Licence 2 SPI

Informatique industrielle

Travaux dirigés

Calcul de la complexité d'une temporisation

L'exercice correspond au programme « clignotant », pages 8 et 9 de l'énoncé de TP. Le TP consiste à implanter une temporisation dépendant de paramètres d'initialisation t10 et t20, pour faire clignoter une LED à une fréquence choisie.

1 Objectifs

-expliquer l'intérêt du « ou exclusif » pour le TP,
-compléter l'algorigramme d'une temporisation
-calculer le nombre d'opérations d'une temporisation, en fonction des paramètres t10 et t20,
-déterminer t10 et t20 pour que la temporisation dure 250 ms.

2 Questions

2.1 Intérêt du « ou exclusif »

Considérez un quartet (ensemble de 4 bits), et effectuez un OU EXCLUSIF (bit à bit) de ce quartet avec le quartet suivant : 0011. Assimilez les bits à des LEDs : 0 pour une LED éteinte, 1 pour une LED allumée. Quel est le résultat de l'application du ou exclusif sur l'état des LEDs ?

2.2 Algorigramme d'une temporisation

La temporisation consiste à décrémenter une variable depuis une valeur de départ jusqu'à 0. La valeur de départ dépend de la durée souhaitée pour la temporisation.

Julien Marot

L’objectif pour le point 2.2 est de tracer l’algorigramme d’une telle temporisation que l’on appellera ‘simple’ car impliquant une seule variable, puis l’algorigramme d’une temporisation que l’on appellera ‘double’ à deux variables.

Complétez l’algorigramme de la figure 1, puis l’algorigramme de la figure 2.
Le premier doit contenir notamment l’affectation et le test d’une variable *t*.
Le deuxième doit contenir notamment l’affectation et le test de deux variables *t1 et t2*.

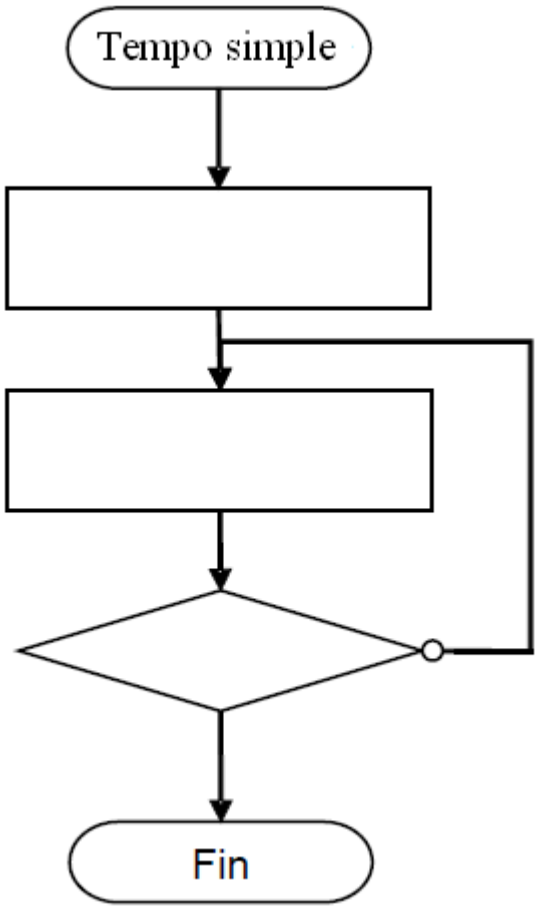

Figure 1

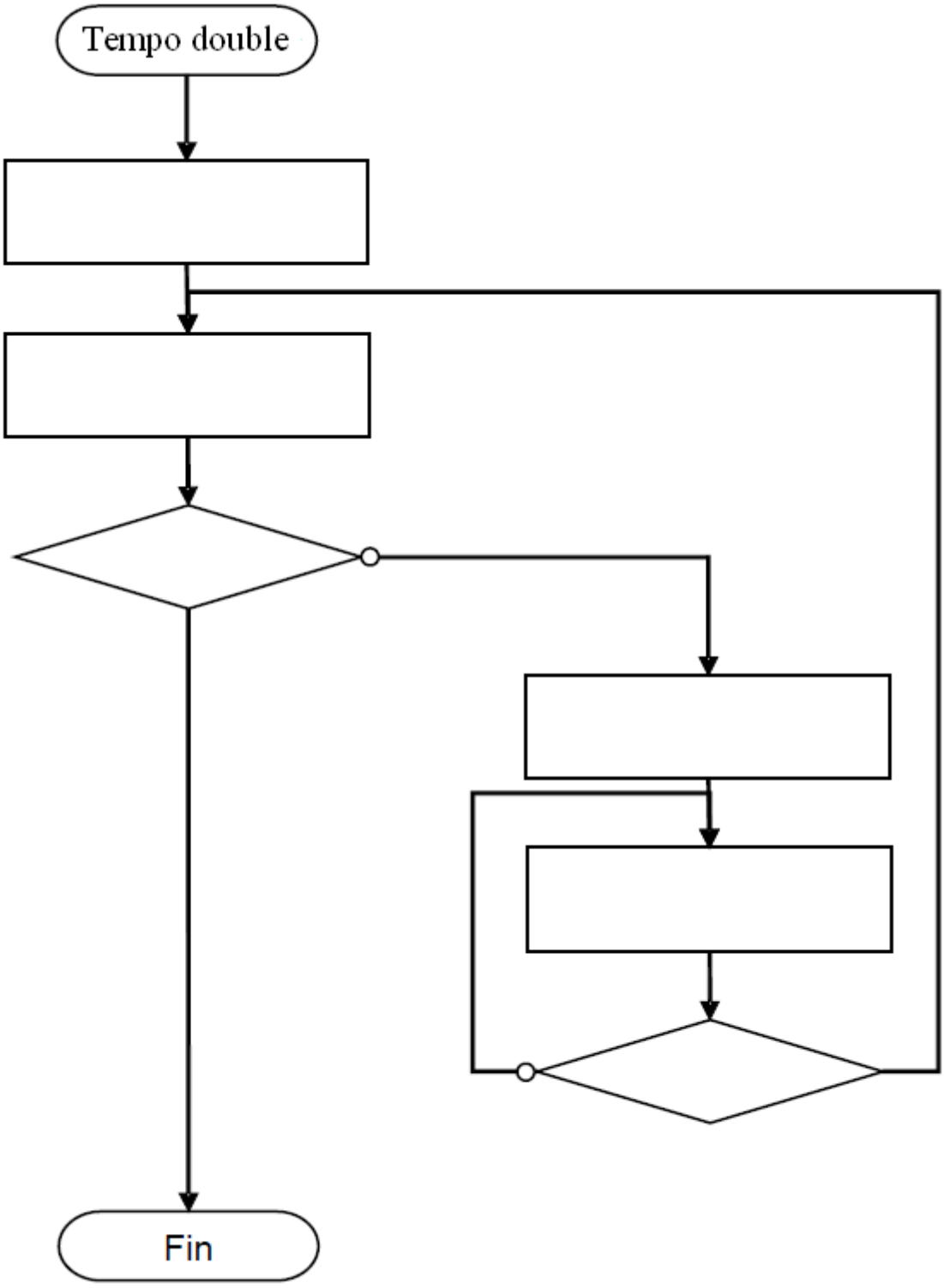

Figure 2

Dans la suite de l'exercice on considérera la temporisation double.

2.3 calcul des valeurs d'initialisatin

On considère la réalisation d'une tempo de 250 ms (~1ms, on néglige le temps dû au programme principal). Choisissez la valeur maximale que l'on puisse coder sur 8 bits (255) pour t20, et **calculez t10**.

Pour calculer t10, vous disposez des données suivantes :

- Avec un quartz à 4 MHz, on a 10^6 cycles par seconde (1 cycle = 4 coups d'horloge).

- Voici un exemple d'instruction sur 1 mot :

  return

- Voici un exemple d'instruction sur deux mots :

  goto label

- Le nombre de cycle(s) que dure une instruction est variable :

<u>Pour chaque instruction :</u>

call : 2 cycles

dcfsnz : si résultat nul : 1 cycle; si résultat non nul : 2 cycles si l'instruction suivante n'est pas une instruction sur deux mots (32 bits), sinon 3 cycles (instruction suivante sur 2 mots).

goto : 2 cycles

return : 2 cycles

movlw : 1 cycle

movwf : 1 cycle

Aix*Marseille université

Faculté des Sciences

Licence 2 SPI

Informatique industrielle

Travaux Dirigés

Les interruptions
de débordement du TIMER0

Julien Marot

1 Objectifs du TD

Une interruption a lieu lorsqu'un événement interrompt un programme dit 'principal'. Nous nous concentrons dans ce TD sur les interruptions qui sont déclenchées lorsqu'un compteur (le 'TIMER0') a fini de compter pendant un temps donné, que l'on appelle la période du TIMER0. L'objectif est de configurer le TIMER0 pour qu'il compte pendant le temps voulu, mais aussi de configurer les interruptions de façon générale, notamment en autorisant les interruptions de débordement du TIMER0 et pas les autres types d'interruption. L'objectif final est de provoquer une interruption toutes le 1 sec. pour changer l'état d'une LED : en résumé, la faire clignoter.

2 Exercice 1 : Introduction au TIMER0

La page 65 du datasheet décrit les registres SFR (special function registers). Repérez quelques registres que vous connaissez déjà, puis certains registres associés au TIMER0. La page 127 du datasheet décrit les registres associés spécifiquement au TIMER0 (voir Fig. 1).

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register, Low Byte								50
TMR0H	Timer0 Register, High Byte								50
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	50
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	52

Legend: Shaded cells are not used by Timer0.

Note 1: PORTA<7:6> and their direction bits are individually configured as port pins based on various primary oscillator modes. When disabled, these bits read as '0'.

FIGURE 1 – Tous les registres associés au TIMER0 (p. 127 datasheet)

En particulier le registre T0CON permet de configurer le TIMER0, et le registre INTCON permet de configurer les interruptions.

2.1 le registre T0CON

Le registre T0CON configure le TIMER0.

D'après la définition des bits du registre T0CON (voir la description Fig. 2), et sachant que par défaut tous les bits du T0CON sont à 0, que faut-il faire pour :

- mettre le TIMER0 sur ON (on dit aussi "lancer" le TIMER0) ;
- choisir l'horloge interne pour compter le nombre de cycles (utilisation "timer" $\neq$ utilisation counter) ;
- configurer le TIMER0 sur 16 bits ;
- choisir 16 comme valeur de prescaler ;
- prendre en compte l'influence du prescaler dans le mode de fonctionnement du TIMER0 ;

Pourquoi n'a-t-on pas besoin de modifier le bit 4 ?

2.2 les registres incrémentés

Parmi les registres donnés sur la Fig. 1, quels sont les deux registres supplémentaires, après T0CON et INTCON, qui sont spécifiques au TIMER0 ?

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON**: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURE 2 – Le registre T0CON : configuration du TIMER0 (p. 125 datasheet)

Quand doit-on utiliser les deux ? Quand peut-on n'en utiliser qu'un seul ? Ces registres sont incrémentés quand le TIMER0 est lancé. Lorsque la valeur max permise par les registres est atteinte, on parle de "débordement" ; la valeur des registres revient alors à 0. On peut se servir des débordements du TIMER0 pour interrompre un programme principal. Cependant l'interruption réaliser doit être bien configurée. Pour cela, et en particulier pour se servir adéquatement du TIMER0 et de ses débordements, le registre INTCON est utilisé.

2.3 le registre INTCON

Le registre INTCON configure les interruptions.

D'après la description du registre INTCON (voir Fig. 3), et en considérant que tous ses bits soient par défaut à 0, comment doit-on configurer ce registre pour se placer dans le contexte suivant :

- autorisation générale des interruptions ;
- autorisation des interruptions de débordement (overflow) du TIMER0 ;

Comment peut-on détecter qu'il y a eu débordement (du registre TMR0) ?

REGISTER 9-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURE 3 – Le registre INTCON : configuration des interruptions (p. 95 datasheet)

3 Exercice 2 : Allumage d'une LED

L'objectif est d'écrire un programme qui fasse inverser l'état d'une LED, à chaque fois que le TIMER0 déborde. Cet exercice est préparatoire au TP 3 "Interruptions et périphériques". On se place dans les conditions de l'exercice 1 (section 2).

Configuration de l'interruption

1) Lorsque le TIMER0 déborde, quelle est la valeur atteinte par TIMER0H,TIMER0L (L signifie Low et H signifie High) ? Cette valeur est appelée valeur "max".

2) L'horloge est un quartz à 4 MHz, et il y a 4 coups d'horloge par cycle. Quelle est la durée d'un cycle ?

3) Le TIMER0 compte de la valeur 0 à la valeur "max" en T secondes, avec $T = T_{cy} * Prescaler * max$, où T_{cy} est la durée d'un cycle, et max la valeur maximale prise par TIMER0H,TIMER0L.
Quelle est la valeur de T ?
Sachant que le TIMLER0 est incrémenté, quelle valeur d'initialisation faut-il donner à TIMER0H,TIMER0L pour avoir $T = 1$ sec. ?

4) Lorsque TIMER0H,TIMER0L atteint la valeur max, un bit de INTCON est modifié, lequel ? Présentez

un intérêt de modifier cette valeur de bit. Pensez par exemple à l'instruction btfsc.

5) Donner deux lignes de codes pour inverser l'état de la LED connectée à la broche 1 du PORTB.

programme principal

6) Faites un algorigramme, avec d'un côté le programme principal, et d'un autre côté le programme d'interruption.
Le programme principal doit contenir : un début ; l'initialisation des vecteurs, la configuration du PORTB, du TIMER0 (T0CON), et des interruptions (INTCON), une boucle, et une fin. A quoi sert la boucle ?
Le programme d'interruption doit contenir : un début, un test avec deux possibilités, la sauvegarde du contexte, la suppression du flag d'interruption, l'inversion de la LED, la restauration du contexte et le retour au programme principal.

7) Complétez le code suivant :

un _ _ correspond à une opérande, trouvez l'opérande qui convient.

```
; Programme Clignotant avec Interruption

LIST P=18F4520
#include <P18F4520.inc>
#include <CONFIG.inc>

;----- Déclaration de variables
CBLOCK 0x00
W_TEMP  : 1 ;Résevations de 3 octects dans la RAM
STATUS_TEMP : 1 ;Pour la sauvegarde du contexte
BSR_TEMP : 1
ENDC

;----- Programme
org h'0000'                         ;Adresse de début du programme sur Reset
goto init

org h'0008'                         ;Adresse de début du programme sur Reset
goto routine_int

init clrf  PORTB                 ;Remise à zéro des bascules D du port B
movlw _ _
movwf _ _                           ;le port B est défini en sortie
movlw _ _
movwf _ _                           ;TIMER0 On, 16bits, Prescaler 16
rcall tmr0_init                ;Init TMR0 pour 1s pile
movlw _ _
movwf _ _                          ;Autorisation Générale des IT et TMR0 IT

boucle _ _                          ;Ne rien faire
goto boucle                     ;Boucle infini

tmr0_init movlw _ _                ;Init du registre TMR0
movwf  TMR0H                    ;pour avoir 1 seconde pile
movlw _ _
movwf  TMR0L
return

;----- Routine d'interruption
routine_int
movwf  W_TEMP                       ;Sauvegarde de W
movff  STATUS, STATUS_TEMP          ;Sauvegarde de STATUS
movff  BSR, BSR_TEMP              ; Sauvegarde de BSR
btfsc  INTCON,2                 ;TMR0IF == 1 _ _
rcall  _ _                          ;traitement de l'IT TMR0IF
movff BSR_TEMP, BSR             ;Restauration de BSR
movff  W_TEMP, WREG             ;Restauration de W
movff STATUS_TEMP, STATUS            ;Restauration de STATUS
retfie                      ;Retour au programme principal

;----- Interruption de débordement du TIMER0
tmr0_overflow
bcf  _ _                            ;Suppression du flag d'interruption
rcall tmr0_init             ;Init TMR0 pour 1s pile
movlw  _ _
xorwf  _ _                      ;Inversion de l'état de la LED l1, RB0
return

END
```

Aix*Marseille université

Faculté des Sciences

Licence 2 SPI

Informatique industrielle

Travaux Dirigés

Les interruptions
Gestion de l'espace mémoire

Julien Marot

Les figures sont utiles aux deux exercices indistinctement.

1 Exercice 1 : taille de l'espace mémoire

Déterminez d'après la figure 1 la taille des mémoires programmes et données.
1) Comparez les résultats obtenus aux information données à la figure 2.
Retrouvez-vous les 32kbytes annoncés ?
2) Idem pour la figure 3. Retrouvez-vous les 3.9 kbytes annoncés ?

2 Exercice 2 : gestion de l'espace mémoire et des interruptions

Notez l'existence de cases mémoire spécifiques dans la figure 2 : RESET, et Interruptions haute et basse.

Le vecteur RESET est un pointeur vers l'adresse mémoire du début du programme principal : il fournit au Program Counter l'adresse de la première instruction du programme.
Le vecteur interruption (basse ou haute) pointe vers l'adresse mémoire du programme à exécuter en cas d'interruption.

En gardant en tête ces définitions, répondez aux questions suivantes :

1. Numérotez les étapes du programme d'interruption dans l'algorigramme de la figure 4 ;
2. Repérez ces étapes dans le code en section 3 ;
3. Quelle est l'unique case mémoire de la figure 2 qui est explicitement dédiée au programme principal ?
4. Quelles sont les deux seules cases mémoire de la figure 2 qui sont explicitement dédiées à un programme d'interruption ?
5. D'après le programme présenté en section 3, quelle est la ligne de code qui est contenue par la case mémoire dont l'adresse est '0000' ? Par celle dont l'adresse est '0008' ?
6. Ces lignes de code permettent de se placer à un emplacement particulier de l'espace mémoire. Déduisez-en la nature du label 'init'. Faites un schéma sur lequel apparaîtront la case mémoire d'adresse '0000', le program counter, et les cases mémoires contenant les deux premières instructions.
7. Quelles sont les cases mémoires de la figure 2 qui sont interdites à l'écriture du programme principal ? Dans quel cas, où le programme principal est mal placé dans la mémoire, une ligne de ce programme serait-elle écrasée et remplacée par `goto routine_int` ?
8. Dans l'exemple donné en section 3, l'instruction 'org ...' est suivie d'une instruction dont l'opcode est `goto`. Quel autre opcode aurait-on pu trouver ?

9. Repérez par une accolade une partie de la mémoire programme (figure 2) qui pourrait contenir le programme principal.
10. Admettons que le label 'init' code l'adresse `0x0180`. D'après le programme en section 3, que contient la case mémoire d'adresse `0x0180` après lecture du vecteur RESET ?
11. Repérez par une autre accolade la partie de la mémoire programme qui peut contenir le programme d'interruption.

3 Programme interruption : extraits

```
org h'0000';Adresse de début du programme sur Reset
goto init

org h'0008';Adresse de début du programme d'interruption
goto routine_int

init clrf  PORTB ;Remise à zéro des bascules D du port B
movlw h'00'
movwf TRISB ;le port B est défini en sortie
movlw h'83'
movwf T0CON ;TIMER0 On, 16bits, Prescaler 16
rcall tmr0_init ;Init TMR0 pour 1s pile
movlw h'A0'
movwf INTCON ;Autorisation Générale des IT et TMR0 IT

boucle nop ;Ne rien faire
goto boucle ;Boucle infini

tmr0_init movlw h'FF';Init du registre TMR0
movwf  TMR0H ;pour avoir 1 seconde pile
movlw h'FD'
movwf  TMR0L
return

;----- Routine d'interruption
routine_int
movwf  W_TEMP  ;Sauvegarde de W
movff  STATUS, STATUS_TEMP  ;Sauvegarde de STATUS
movff  BSR, BSR_TEMP  ; Sauvegarde de BSR
btfsc  INTCON,2  ;TMR0IF == 1 ?
rcall  tmr0_overflow ;traitement de l'IT TMR0IF
movff BSR_TEMP, BSR  ;Restauration de BSR
movff  W_TEMP, WREG  ;Restauration de W
movff STATUS_TEMP, STATUS ;Restauration de STATUS
retfie  ;Retour au programme principal

;----- Interruption de débordement du TIMER0
tmr0_overflow
bcf  INTCON,2  ;Suppression du flag d'interruption
rcall tmr0_init ;Init TMR0 pour 1s pile
movlw  h'01'
xorwf  PORTB  ;Inversion de l'état de la LED l1, RB0
return
```

4 Figures

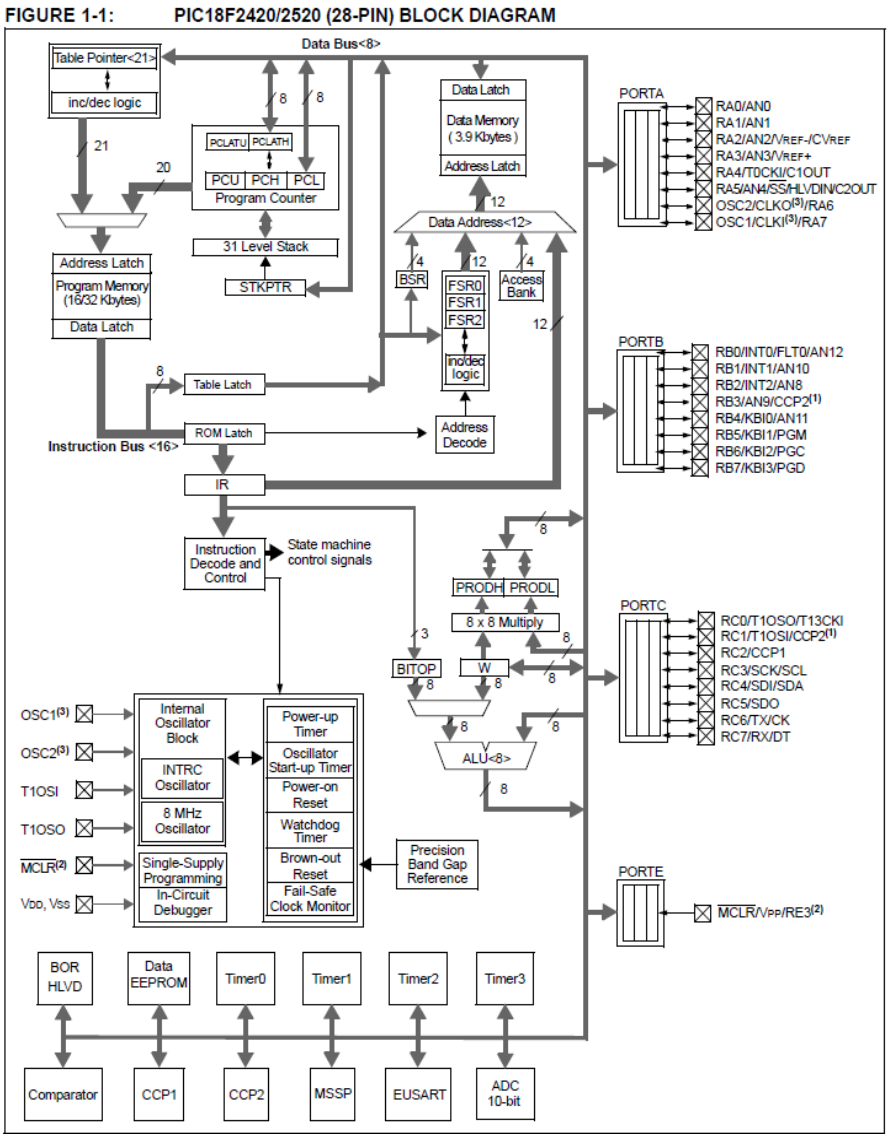

FIGURE 1 – Schéma de principe du PIC

FIGURE 5-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F2420/2520/4420/4520 DEVICES

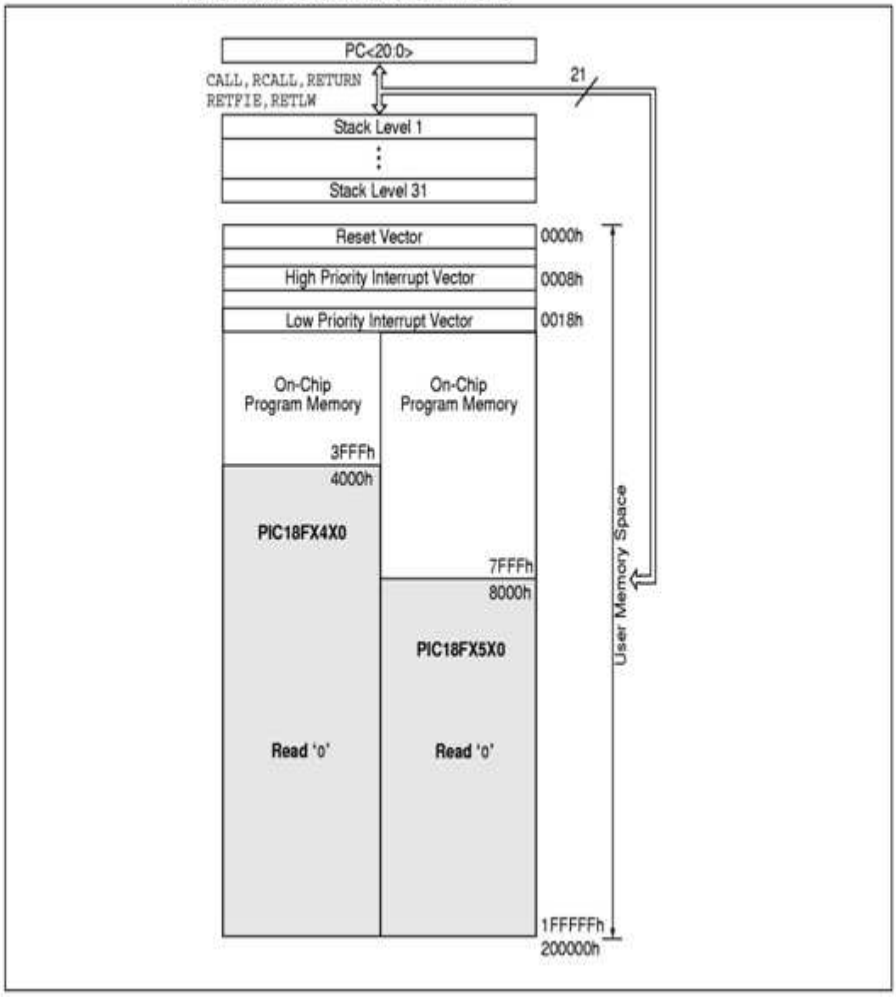

FIGURE 2 – Mémoire programme

FIGURE 5-6: DATA MEMORY MAP FOR PIC18F2520/4520 DEVICES

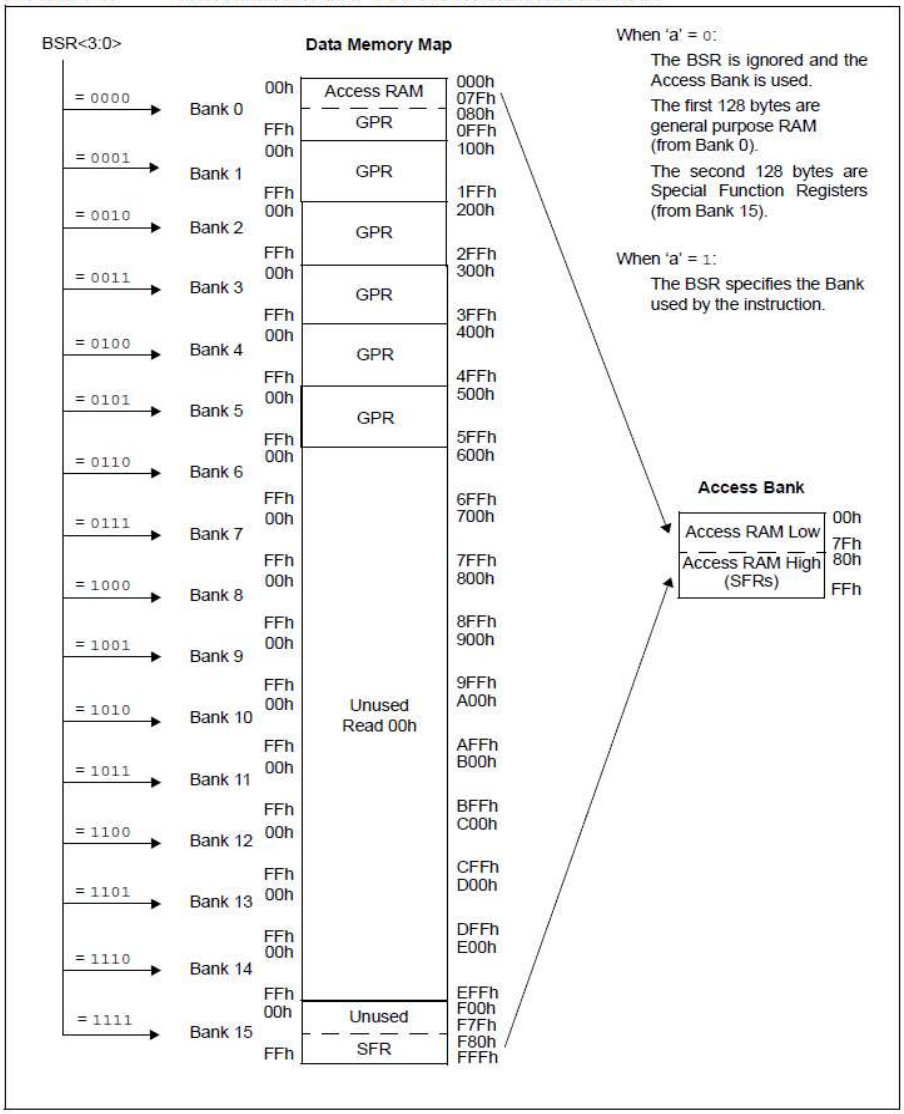

FIGURE 3 – Mémoire données

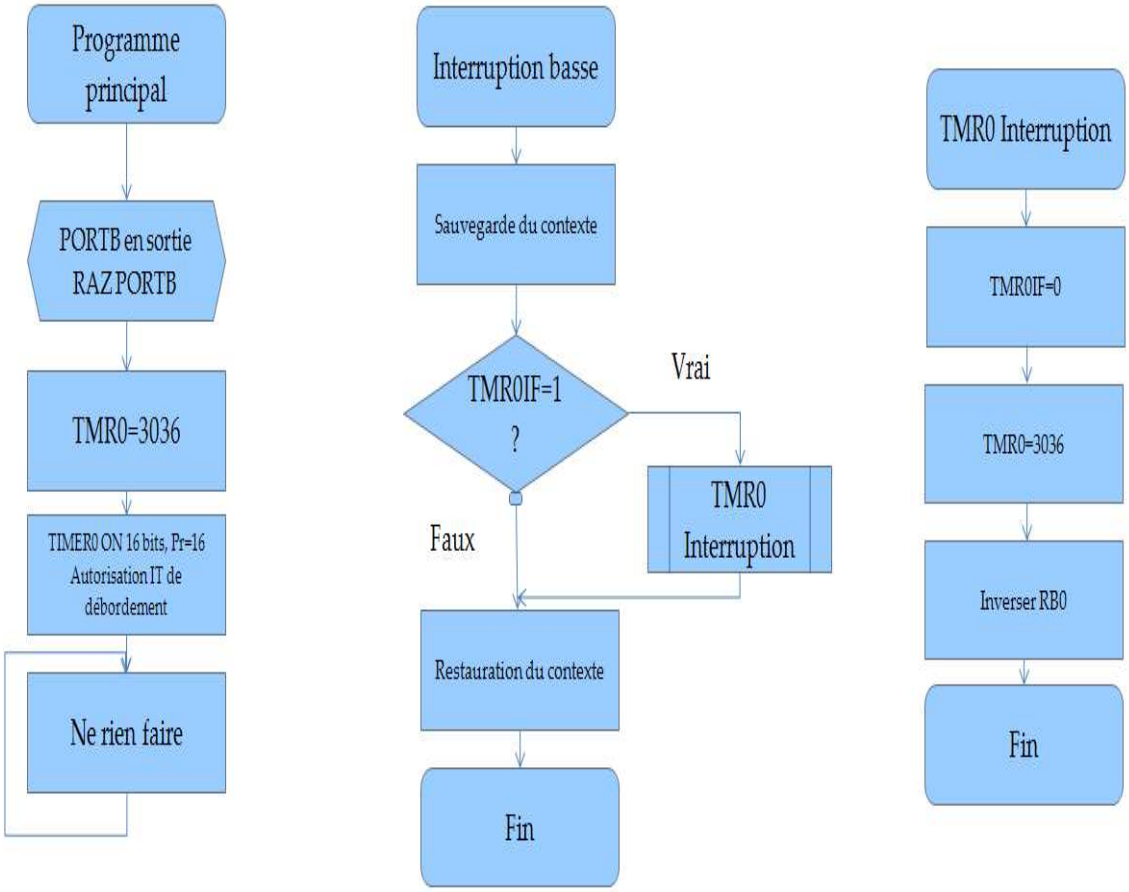

FIGURE 4 – Algorigrammes