II2 TD n°1 : langage C : opérateurs et expressions

Le contenu de ce polycopié peut être trouvé dans un cours wikiversité :
http://fr.wikiversity.org/wiki/Utiliser_les_PIC_16F_et_18F

1. Arithmétique binaire et expressions en C PIC (16F84)

Pour bien comprendre la signification des expressions, il est essentiel d'avoir 2 notions en tête : la priorité et l'associativité. Nous donnons ci-après un tableau des opérateurs par priorité décroissante :

Catégorie d'opérateurs	Opérateurs	Associativité
fonction, tableau, membre de structure, pointeur sur un membre de structure	( ) [ ] . ->	Gauche -> Droite
opérateurs unaires	- ++ -- ! ~ * & sizeof (type)	Droite ->Gauche
multiplication, division, modulo	* / %	Gauche -> Droite
addition, soustraction	+ -	Gauche -> Droite
décalage	<< >>	Gauche -> Droite
opérateurs relationnels	< <= > >=	Gauche -> Droite
opérateurs de comparaison	== !=	Gauche -> Droite
et binaire	&	Gauche -> Droite
ou exclusif binaire	^	Gauche -> Droite
ou binaire	|	Gauche -> Droite
et logique	&&	Gauche -> Droite
ou logique	||	Gauche -> Droite
opérateur conditionnel	? :	Droite -> Gauche
opérateurs d'affectation	= += -= *= /= %= &= ^= |= <<= >>=	Droite -> Gauche
opérateur virgule	,	Gauche -> Droite

Exercice 1
Enlever les parenthèses des expressions suivantes lorsqu'elles peuvent être retirées.
a=(25*12)+b;
if ((a>4) &&(b==18)) { }
((a>=6)&&(b<18))||(c!=18)
c=(a=(b+10));
Évaluer ces expressions pour a=6, b=18 et c=24

Les types du C PIC sont :

```
unsigned char a;//8 bits, 0 to 255
signed char b;  //8 bits, -128 to 127
```

```
unsigned int c; //16 bits, 0 to 65535
signed int d;   //16 bits, -32768 to 32767
long e;         //32 bits, -2147483648 to 2147483647
float f;    //32 bits
```

Exercice 2

b7	*b6*	*b5*	*b4*	*b3*	*b2*	*b1*	*b0*

Si une variable p1 de type signed char (8 bits signés) est déclarée écrire les expressions en C permettant de :
- mettre à 1 le bit b2
- mettre à 1 le bit b3 et b6
- mettre à 0 le bit b0
- mettre à 0 le bit b4 et b5
- inverser le bit b3 (se fait facilement avec un ou exclusif)
- mettre à 1 le bit b2 et à 0 le bit b0
- mettre à 1 les bits b0 et b7 et à 0 les bits b3 et b4

Exercice 3

Soit une variable :

```
char nb;
```

Écrire les expressions permettant de calculer les centaines, les dizaines et les unité de cette variable.

Il existe plusieurs autres méthodes pour positionner les bits
(http://www.microchipc.com/HiTechCFAQ/index.php)
1° méthode pour positionner bit à bit :

```
#define bit_set(var,bitno) ((var) |= 1 << (bitno))
#define bit_clr(var,bitno) ((var) &= ~(1 << (bitno)))
unsigned char x=0b0001;
bit_set(x,3); //now x=0b1001;
bit_clr(x,0); //now x=0b1000;*/
```

2° méthode pour positionner plusieurs bits

```
#define bits_on(var,mask) var |= mask
#define bits_off(var,mask) var &= ~0 ^ mask
unsigned char x=0b1010;
bits_on(x,0b0001);   //now x=0b1011
bits_off(x,0b0011);  //now x=0b1000 */
```

2. Expression booléenne vraie

Dans la suite du cours on appellera expression booléenne (E. B.) une expression qui si elle était affectée à une variable donnerait soit la valeur 0 ou soit la valeur 1. Le C est un langage qui ne permet de fabriquer que des expressions (Expr). C'est très pratique, mais la confusion des deux peut conduire à des erreurs. Donnons un exemple :

expression (Expr)	*expression booléenne (E. B.)*

`char n=10;`
`while(n) {`
`.....;`
`n--; }`
	`char n=10;`
	`while(n!=0) {`
	`.....;`
	`n--; }`

Ces deux constructions sont permises en C car il y a une technique universelle qui permet de transformer une expression en expression booléenne.
- tout ce qui est évalué à 0 est considéré comme faux,
- tout ce qui est évalué différent de 0 est considéré comme vrai.

Si on se rappelle que dans une parenthèse d'un while on a une Expression Booléenne (E. B.). D'un côté n et de l'autre n!=0 sont donc des E. B. La différence entre les deux est que pour n elle peut prendre toutes les valeurs entre -128 et +127 (car n est de type char) alors que n!=0 ne prendra que deux valeurs 0 ou 1. La deuxième sera donc appelée E. B. et la première Expr qui sera transformée comme indiqué.
Le problème du langage C est que l'oubli d'un & ou d'un | conduit en général à une expression souvent en lieu et place d'une expression booléenne ce qui est tout à fait autorisé. Par exemple écrire a & b au lieu de a && b, ou pire encore un "=" au lieu d'un "==".

Exercice 4

Différence entre && et &
Évaluer les expressions :
a&b
a&&b
pour a= 0xF0 et b=0x0F
En déduire les valeurs booléennes correspondantes (si ces expressions étaient utilisées dans un if par exemple).
Construire des expressions booléennes sur les tests suivants
expression vraie si
le bit b6 est à 1
le bit b3 est à 0
le bit b6 est l'inverse du bit b3
le bit b2 est à 1 et le bit b4 est à 0
le bit b2 est à 1 ou le bit b7 est à 0

Les tests d'un bit particulier en C peuvent aussi être réalisés de la manière suivante (http://www.microchipc.com/HiTechCFAQ/index.php)

```
x=0b1000; //decimal 8 or hexadecimal 0x8
if (testbit_on(x,3)) a(); else b(); //function a() gets executed
if (testbit_on(x,0)) a(); else b(); //function b() gets executed
if (!testbit_on(x,0)) b(); //function b() gets executed
#define testbit_on(data,bitno) ((data>>bitno)&0x01)
```

Exercice 5

Quelle opération arithmétique est réalisée par un décalage ? Évaluer pour cela les expressions suivantes (avec a=12 et b=23) :
- a = a >> 1 (ou a >>= 1)
- a = a >> 2 (ou a >>= 2)
- b = b << 1 (ou b <<=1)
- b = b << 2 (ou b <<=2)

Généralisation.
Construire une vraie expression booléenne avec opérateur de décalage, & et ^ qui reprend le test de l'exercice précédent : le bit b6 est l'inverse du bit b3

Exercice 6

On donne le sous-programme suivant (tiré d'un ancien projet inter-semestre) :

```
void conversion(char nb,char result[8]){
  char i;
  for(i=7;i>=0;i--) if ((nb & (1<<i)) == (1<<i)) result[i]=1;
                    else result[i]=0;
}
```

1) Peut-on retirer des parenthèses dans l'expression booléenne du if ?
2) Peut-on écrire `(nb & (1<<i))` au lieu de `((nb & (1<<i)) == (1<<i))` ?
3) Construire l'expression booléenne qui permettrait l'écriture

```
for(i=7;i>=0;i--) if (E.B.?????) result[i]=0;
                     else result[i]=1;
```

en donnant toujours le même le même résultat de conversion.
4) Modifier le programme pour qu'il fasse une conversion d'entier (16 bits) vers le binaire.
5) Est-ce que l'algorithme suivant donne le même résultat de conversion :

```
 for(i=0;i<8;i++) {
     result[i]=nb%(2);
     nb = nb / 2;
  }
```

Exercice 7

Soit le programme suivant :

```
#include <stdio.h>
main() {
  int n=10,p=5,q=10,r;
  r= n == (p = q);
  printf("A : n = %d p = %d q= %d r = %d\n",n,p,q,r);
  n = p = q = 5;
  n += p += q;
  printf("B : n = %d p = %d q= %d\n",n,p,q);
  q = n++<p || p++ != 3;
  printf("C : n = %d p = %d q= %d\n",n,p,q);
  q = ++n == 3 && ++p == 3;
  printf("D : n = %d p = %d q= %d\n",n,p,q);
  return 0;
}
```

Que donnera-t-il comme affichage sur l'écran ?

II2 - TD n°2 Architecture des PIC 16FXXX

1. Architecture générale

1°) Architecture mémoire

La donnée de base de PIC (16FXXX) est l'octet. L'octet comporte 8 bits. Le bit 0 est le bit de poids faible et le bit 7 est le bit de poids fort.

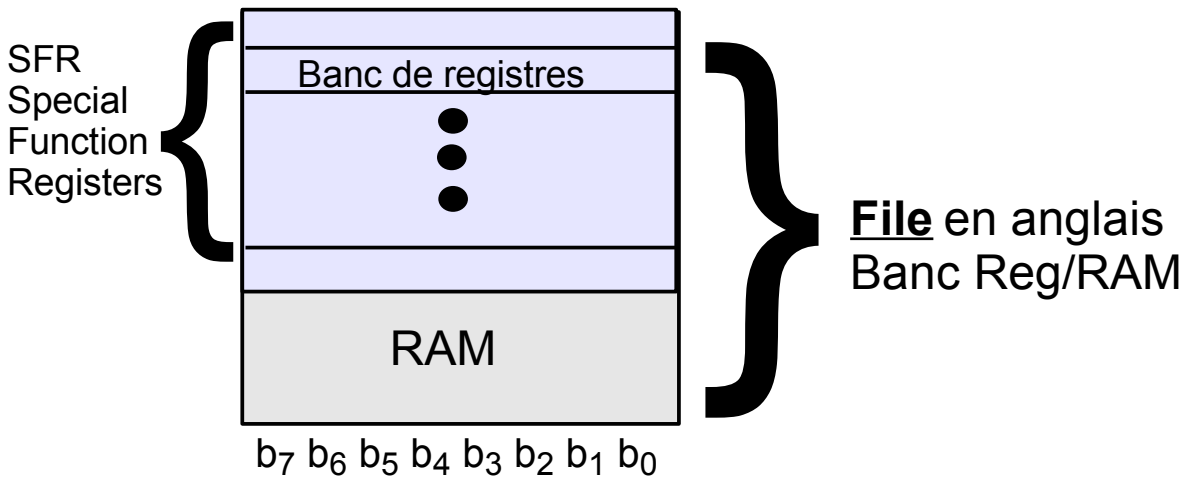
Mémoire programme : la mémoire programme du PIC (16F84) est organisée en mots de 14 bits et elle peut en contenir 1024 (soit 1k). Le bit 0 est aussi le bit de poids faible et le bit 13 est le bit de poids fort. La mémoire mémoire programme comporte donc 1k x 14 bits.

Exercice 1
On rappelle qu'une mémoire est caractérisée par un bus de donnée de largeur n (n bits de D_0 à D_{n-1}) et un bus d'adresse de largeur m (m bits de A_0 à A_{m-1}).
1°) Pour la mémoire programme du 16F84 donner m et n.
2°) La mémoire programme du 16F873 est 4 fois plus grande : donner les nouveaux m et n.
3°) La mémoire programme du 16F887 utilisé en TP est 8 fois plus grande qu'en 1°, donner les nouveaux m et n.

Mémoire EEPROM : le PIC (16F84) dispose de 64 octets de mémoire EEPROM.

Exercice 2
Calculer m et n pour l'EEPROM du 16F84.

Mémoire donnée :
Description simplifiée



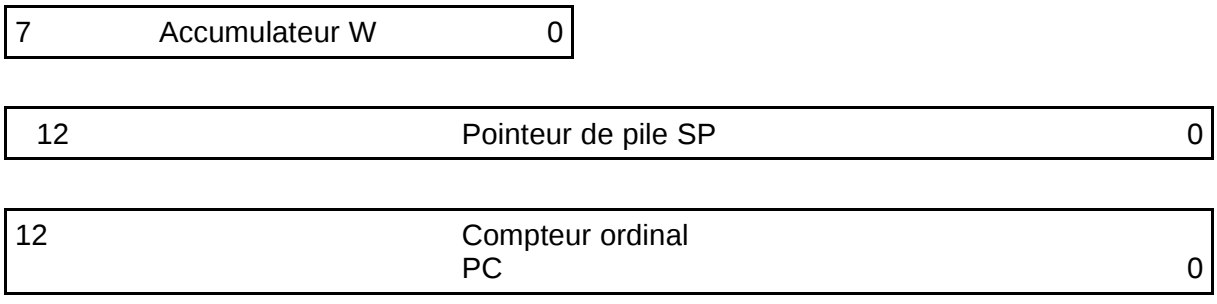
Description détaillée pour le 16F84
Il faut ajouter une division verticale que l'on appelle banque. Le PIC (16F84) possède deux banques (sélectionnées par le bit RP0 du registre **STATUS** subdivisées en deux parties :
- registres de fonctions spéciales (SFR Special Function Registers)
- cases mémoires libres que vous pouvez utiliser à votre guise.

	File Address	Banque 0	Banque 1	File Address	
	00h	Indirect addr.	Indirect addr.	80h	
`bcf status,rp0`	01h	TMR0	OPTION_REG	81h	`bsf status,rp0`
passe en	02h	PCL	PCL	82h	passe en
banque 0	03h	STATUS	STATUS	83h	banque 1
	04h	FSR	FSR	84h	
	05h	PORTA	TRISA	85h	
	06h	PORTB	TRISB	86h	
	07h	---	---	87h	
	08h	EEDATA	EECON1	88h	
	09h	EEADR	EECON2	89h	
	0Ah	PCLATH	PCLATH	8Ah	
	0Bh	INTCON	INTCON	8Bh	
	0Ch	68 cases mémoires SRAM	idem à la banque 0	8Ch	
	...				
	4Fh			CFh	
		inutilisé	inutilisé		
	7Fh			FFh	

La mémoire du 16FXXX est organisée en banques sélectionnées par le(s) bit(s) RP0 (et RP1) du registre **STATUS**. Seul RP0 est utilisé pour le 16F84, soit deux banques.

2°) Le modèle de programmation du PIC (16FXXX)

Il peut être présenté de la manière très simplifiée suivante :



Status

IRP	RP1	RP0	/TO	/PD	Z	DC	C

IRP : sélection Banque en adressage indirect
RP1 : sélection banque
RP0: sélection banque
TO : Timer Overflow
PD : Power Down
Z : zéro
DC : demi-retenue (demi-carry)
C : retenue (Carry)

Le pointeur de pile n'est manipulé directement par aucune instruction. Il ne comporte qu'une profondeur de 8 niveaux.
Le compteur programme est en fait composé de deux registres 8 bits :
- **PCL** (02h/82h) pour les 8 bits de poids faibles (accessible en lecture/écriture)

- **PCLATH** (0Ah/8Ah) pour les 5 bits de poids forts (accessible en lecture/écriture)
Le 16F84 gère seulement 10 bits des 13 bits du PC

Le modèle de programmation complet est plus complexe, en particulier à cause des registres. Il est présenté en annexe à la fin du document.

2. Utiliser les registres du PIC en C

Le registre **STATUS** présenté ci-dessus comporte des bits qui possèdent un nom. Ce n'est pas le seul registre a avoir cette propriété. Mais peut-on en déduire que les bits des registres possèdent tous un nom que le compilateur C connait ? La réponse est non ! Et pour corser le tout il n'y a aucune règle générale. Si vous voulez vous en sortir le seul moyen est de lire les fichiers d'entête correspondant à votre PIC. En général, le nom des registres et des bits est respecté mais il existe des bits qui ont un nom dans la documentation officielle mais pas avec le MikroC.

1°) Exemple de fichier d'en-tête pour MikroC

À proprement parler, pour le compilateur MikroC, les définitions ne sont pas dans un fichier avec une extension (.h) mais dans des fichiers qui se trouvent dans un répertoire defs et qui ont l'extension (.c). On en présente un extrait maintenant :

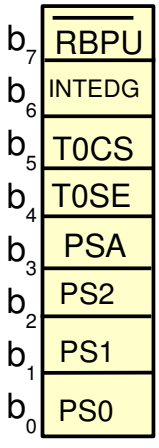
```
//***** MikroC fichier P18F84A.c ******
const unsigned short
  W      = 0x0000,
  F      = 0x0001,
  IRP      = 0x0007,
  RP1      = 0x0006,
  RP0      = 0x0005,
  NOT_TO       = 0x0004,
  NOT_PD       = 0x0003,
  Z      = 0x0002,
  DC       = 0x0001,
  C      = 0x0000,
....
unsigned short register volatile
  STATUS       absolute 0x0003;
....
unsigned short register
  OPTION_REG       absolute 0x0081,
....
```

Cela permet de connaître le nom des bits du registre **STATUS** et de confirmer le nom du registre **OPTION** (comme OPTION_REG). Rappelez-vous que le langage C est sensible à la casse des caractères.

2°) Les trois possibilités pour atteindre un bit particulier

Voici à travers un exemple comment accéder à un bit particulier pour le registre **OPTION** :

OPTION



Exemple en MikroC

```
void main( void) {
....
//*** Toujours *****
  OPTION_REG.F3 =1;
//**** Si on connaît le nom
  OPTION_REG.PSA =1;

...

}
```

Fichier d'entête

```
....
//********** bits de OPTION
NOT_RBPU       = 0x0007,
INTEDG       = 0x0006,
T0CS       = 0x0005,
T0SE       = 0x0004,
PSA      = 0x0003,
PS2      = 0x0002,
PS1      = 0x0001,
PS0      = 0x0000,
...
```

Exercice 3

Pour vérifier la bonne compréhension de ce qui vient d'être dit, nous donnons à droite le fichier d'entête des bits correspondant au registre **INTCON**.

1°) Dessiner le registre correspondant

2°) En utilisant les masques du TD précédent, écrire les instructions C permettant :

- mise à 1 de GIE
- mise à 0 de INTE.
- tester si T0IE est à 1
- tester si T0IF est à 0

3°) Refaire le même travail en utilisant les noms des bits directement.

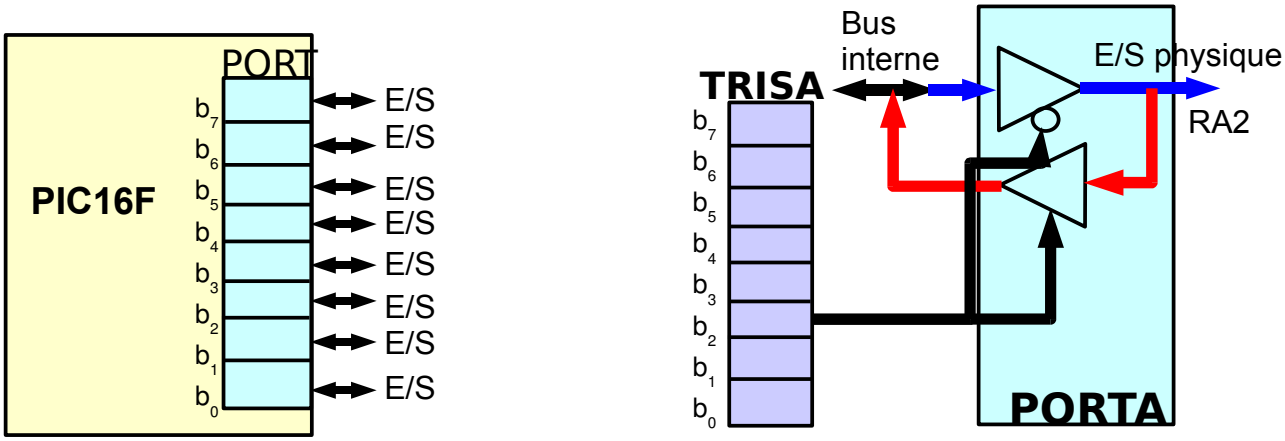
Fichier d'entête

```
//********* bits de INTCON
GIE      = 0x0007,
EEIE       = 0x0006,
T0IE       = 0x0005,
INTE       = 0x0004,
RBIE       = 0x0003,
T0IF       = 0x0002,
INTF       = 0x0001,
RBIF       = 0x0000,
```

II2 - TD n°3 Des LEDs et des boutons poussoirs sur des PORTs

1. Le PORT pour entrer et sortir des informations

1°) Le registre TRISA

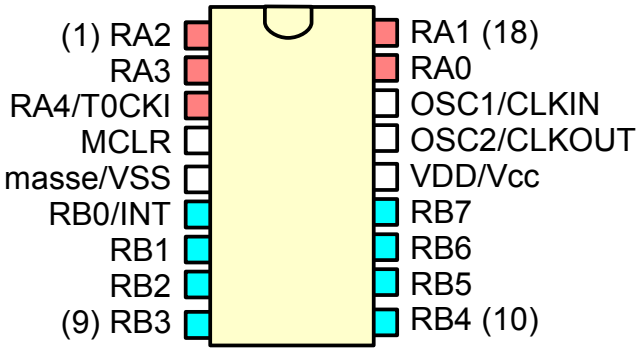


Ce registre est d'un fonctionnement très simple et est lié au fonctionnement du **PORTA**.

Chaque bit de **TRISA** positionné à 1 configure la broche correspondante en entrée. Chaque bit à 0 configure la pin en sortie

Au reset du PIC®, toutes les pins sont mises en entrée, afin de ne pas envoyer des signaux non désirés sur les pins. Les bits de **TRISA** seront donc mis à 1 lors de chaque reset.
Notez également que, comme il n'y a que 5 pins utilisées sur le **PORTA**, seuls 5 bits (b0 ... b4) seront utilisés sur TRISA. Les bits de TRISA sont désignés par TRISA.F0 ... TRISA.F4

Brochage du PIC 16F84



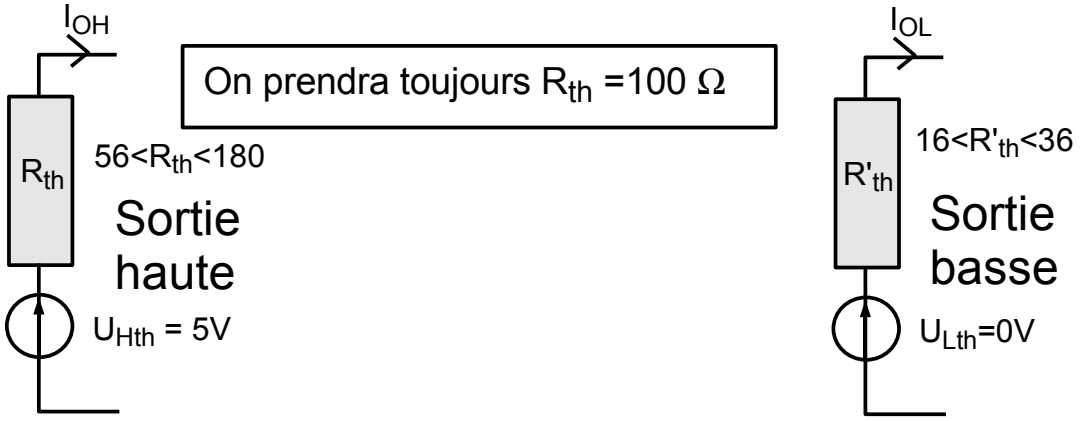
2°) Les registres PORTB et TRISB

Ces registres fonctionnent exactement de la même manière que **PORTA** et **TRISA**, mais concernent bien entendu les 8 pins RB. Tous les bits sont donc utilisés dans ce cas.
Voyons maintenant les particularités du **PORTB**. Les entrées du **PORTB** peuvent être connectées à une résistance de rappel au +5V de manière interne, cette sélection s'effectuant par le bit 7 du registre **OPTION** (effacement du bit7 RBPU pour valider les résistances de rappel au +5V).
Les bits de **TRISB** sont désignés par TRISB.F0 ... TRISB.F7

2. Le PORT et sa modélisation électrique

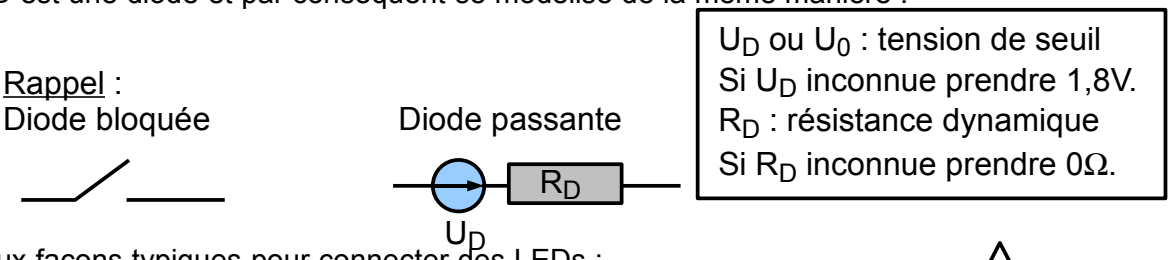
Le modèle électrique est très simple : on le modélise comme d'habitude à l'aide de Thevenin.

Il est raisonnable de prendre un Imax de 10 mA pour 16F84 et 15 mA pour 16F877

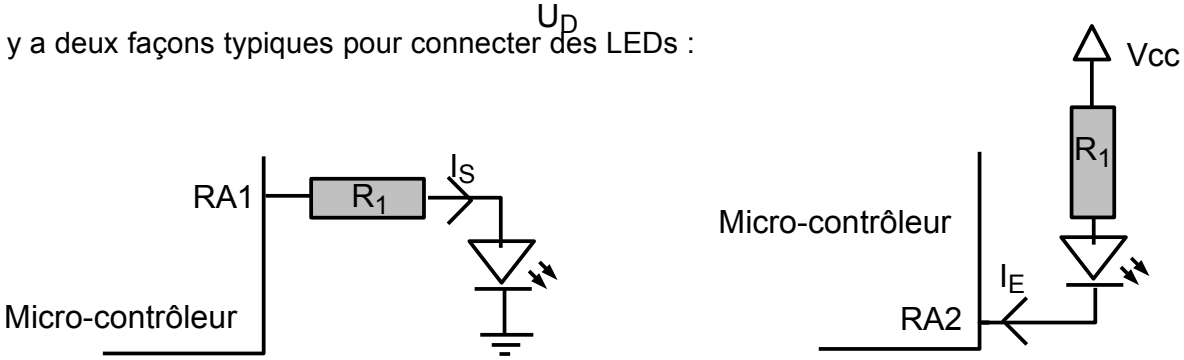


3. Connecter et dimensionner des LEDs

Une LED est une diode et par conséquent se modélise de la même manière :



Il y a deux façons typiques pour connecter des LEDs :



Micro-contrôleur comme source de courant	Micro-contrôleur comme puits de courant

Exercice 1

Nous allons essayer de dimentionner R_1 dans les montages ci-dessus.

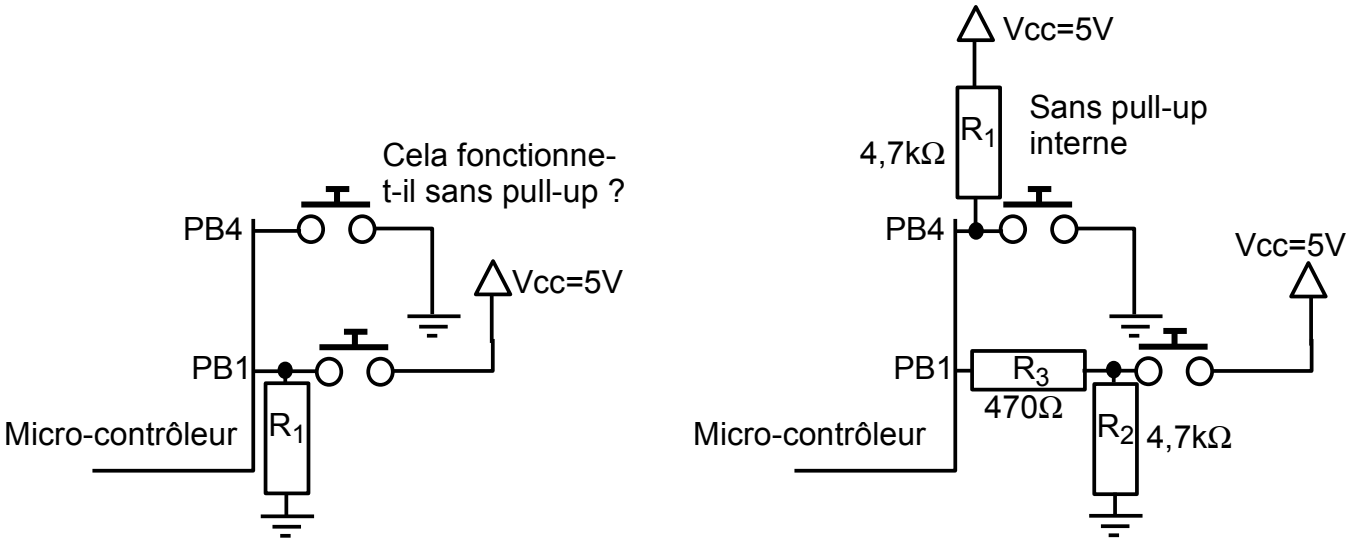
1°) Une led rouge (Kingbright λ = 627nm 15mcd à 10mA V_D = 1,95V) est montée comme à gauche sur le bit b_0 du **PORTB** d'un 16F84. A l'aide du modèle de Thévenin du PORT, on vous demande d'évaluer la résistance R_1 à mettre pour avoir un courant ne dépassant pas 5mA.

2°) Une led verte (Kingbright λ=565nm 12mcd à 10mA V_D = 2,09V) est montée comme sur le schéma de gauche (sur le bit b_2 du **PORTB**). Calculer R_1 sans prendre l'approximation de Rth pour avoir un courant qui ne dépasse pas les 4mA.

3°) On désire commander deux led rouge en série. Faire le schéma et dimensionner correctement la résistance.

4. Connecter des boutons poussoirs

Il est naturellement possible de connecter des boutons poussoirs à un PORT et de demander au micro-contrôleur de dire si le bouton est appuyé ou relâché. (Sur un robot mobile on peut utiliser ce principe pour détecter des objets)



Si votre PORT ne possède pas de résistance de tirage interne, il faut en mettre une à l'extérieur.

Exercice 2

On connecte deux interrupteurs comme indiqué dans la partie gauche sans utiliser de résistance pull-up interne.
Les deux LEDs de l'exercice 1 sont elles aussi connectées.
1°) On désire écrire un programme C qui ne fait rien si on n'appuie sur aucun bouton poussoir, fait clignoter la LED rouge si l'on appuie sur un bouton, fait clignoter la led verte si on appuie sur l'autre bouton, et les deux LEDs si l'on appuie sur les deux boutons.
1-a) Donner les 4 valeurs possibles de la variable interrupteurs avec l'instruction

```
interrupteurs = PORTB & 0x12;
```

1-b) Compléter alors le morceau de code ci-dessous pour en faire un programme :

```
 // boucle d'attente ATTENTION PB4=1 et PB1=0 au repos
while(interrupteurs.F1==0 &&  interrupteurs.F4==1)
     interrupteurs = PORTB & 0x12;
switch(interrupteurs) {
  case 0x12 : PORTB = PORTB^0x01;break;
  case 0x00: PORTB = PORTB^0x04;break;
  case 0x02 : PORTB = PORTB^0x05;break; // les deux
}
Delay_ms(1000);
```

2°) Peut-on modifier facilement le programme pour que les deux LEDs fonctionnent à deux fréquences différentes ?

Exercice 3

On désire réaliser un dé comme indiqué sur le schéma en page suivante.
1°) Calculer les sept valeurs prédéfinies pour allumer correctement le dé et les ranger dans un tableau si la première case du tableau éteint tout.
Quel est l'affichage du dé correspondant à :

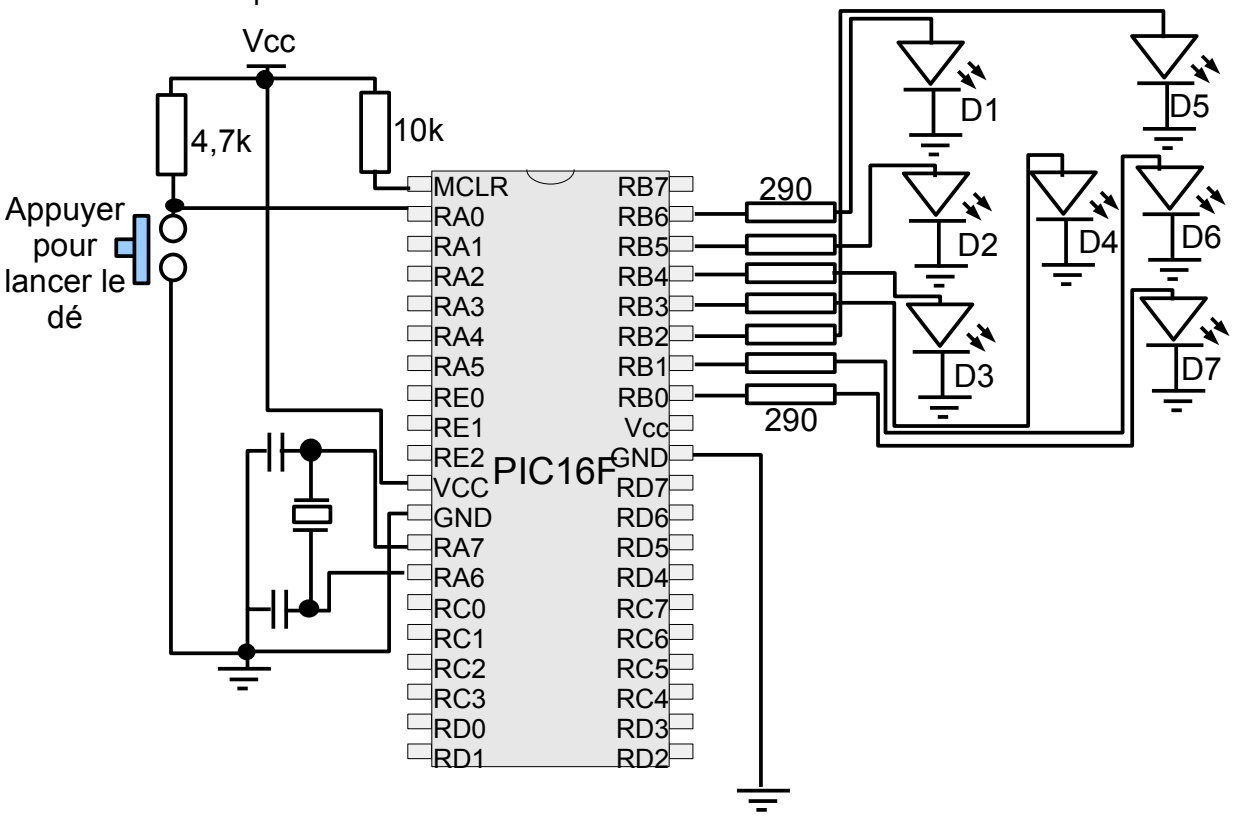
```
unsigned char tableau[7]={0x00,0x08,0x22,0x2A,0x55,0x5D,0x77};
```

2°) Écrire un programme complet qui affiche les 6 valeurs du dé avec un délai d'attente de 3s utilisant la fonction "Delay_ms()" de la librairie MikroC.
3°) On désire maintenant générer de manière aléatoire le score affiché sur le dé. Pour cela on

vous propose d'utiliser la fonction ci-contre.
3-a) Calculer la première valeur de cette fonction si Lim=6.
3-b) Ecrire le programme général qui génère les valeurs du dé de manière aléatoire dès l'appui du bouton poussoir (attente de 0,3s pour éviter les rebonds).

```
unsigned char pseudoAleat(int Lim) {
  unsigned char result;
  static unsigned int Y=1;
  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim)+1);//+1 : eviter 0
  return Result;
}
```

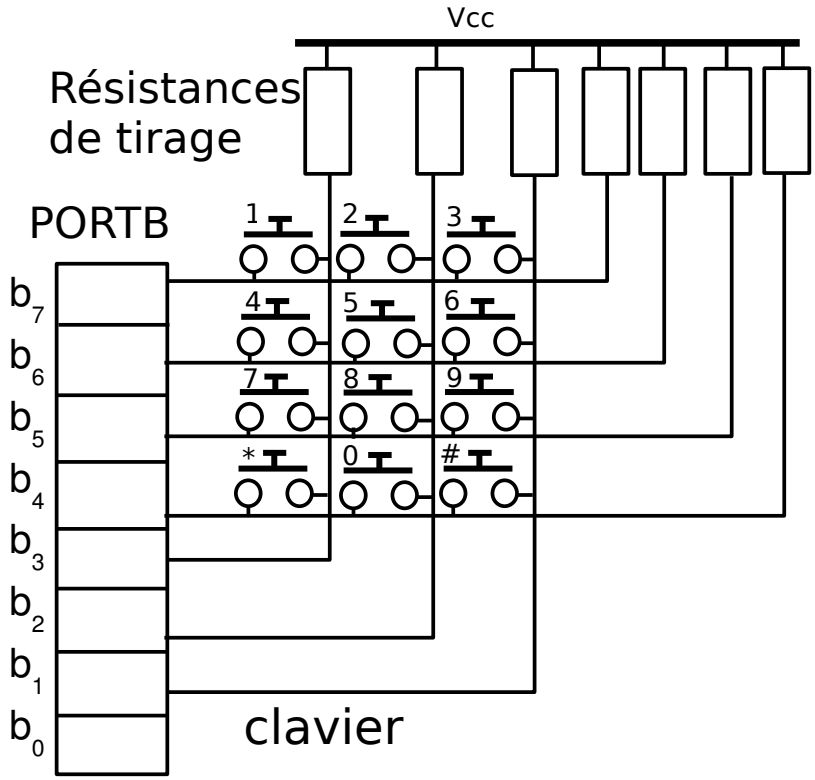
4°) Proposer un schéma pour réaliser deux dés en utilisant un seul PORT de sortie sur 8 bits. Écrire le programme correspondant.
L'astuce consiste à regrouper les leds qui s'allument ensembles (pour un dé) sur un même bit du PORT : il faut ainsi 4 bits par dé.



5. Interfacer un clavier

Sur un PC, le clavier est complètement décodé. C'est à dire que lorsqu'une touche est appuyée, sa position sur le clavier est envoyée sur la liaison PS2. Le fait d'envoyer la position et non le code ASCII permet de gérer les claviers en divers langues.
Pour de petites applications, on utilise un clavier à 12 touches. Il est composé de simples contacts et le décodage est réalisé par le système informatique. Avec seulement 8 touches, un PORT de 8 bits en entrée suffit. Si le clavier possède plus de 8 touches, il faut:
- soit utiliser d'avantage d'entrées,
- soit multiplexer les entrées en deux étapes.

En utilisant 4 fils pour les lignes et 4 fils pour les colonnes, on peut différencier par croisement 16 touches. On utilise donc 8 fils reliés à 8 bits d'un PORT pour 16 touches. Pour nos 12 touches on peut câbler comme indiqué ci-dessus. Il s'agit ensuite de procéder en deux phases, une pour la détection de la colonne et une autre pour la détection de ligne.



Exercice 4

Question 1 : détermination du numéro de colonne

```
#define NOTAKEY 127
// colonne 1 à gauche
char lecture_colonne(){
  char ch;
  TRISB=0x0F; // 0000 1111
  PORTB = PORTB & 0x0F; // B4, B5,B6 et B7  mis à 0
  ch = PORTB & 0x0E; // on ne garde que les bits intéressants B1, B2 et B3
  switch (ch) {
  case 14 : return 0;//aucune touche
  case 6 : return 1;//a gauche
  case 10 : return 2;// au milieu
  case 12 : return 3;//a droite
// si autre cas, deux touches ou autre
  default : return NOTAKEY;
  }
}
```

Commenter en testant les bonnes valeurs du case.

Question 2 : détermination du numéro de ligne

Programmer les directions avec **TRISB** (PB7-PB4 en entrée et PB3-PB1 en sortie).
Quel est le code correspondant : sous-programme `char lecture_ligne()`

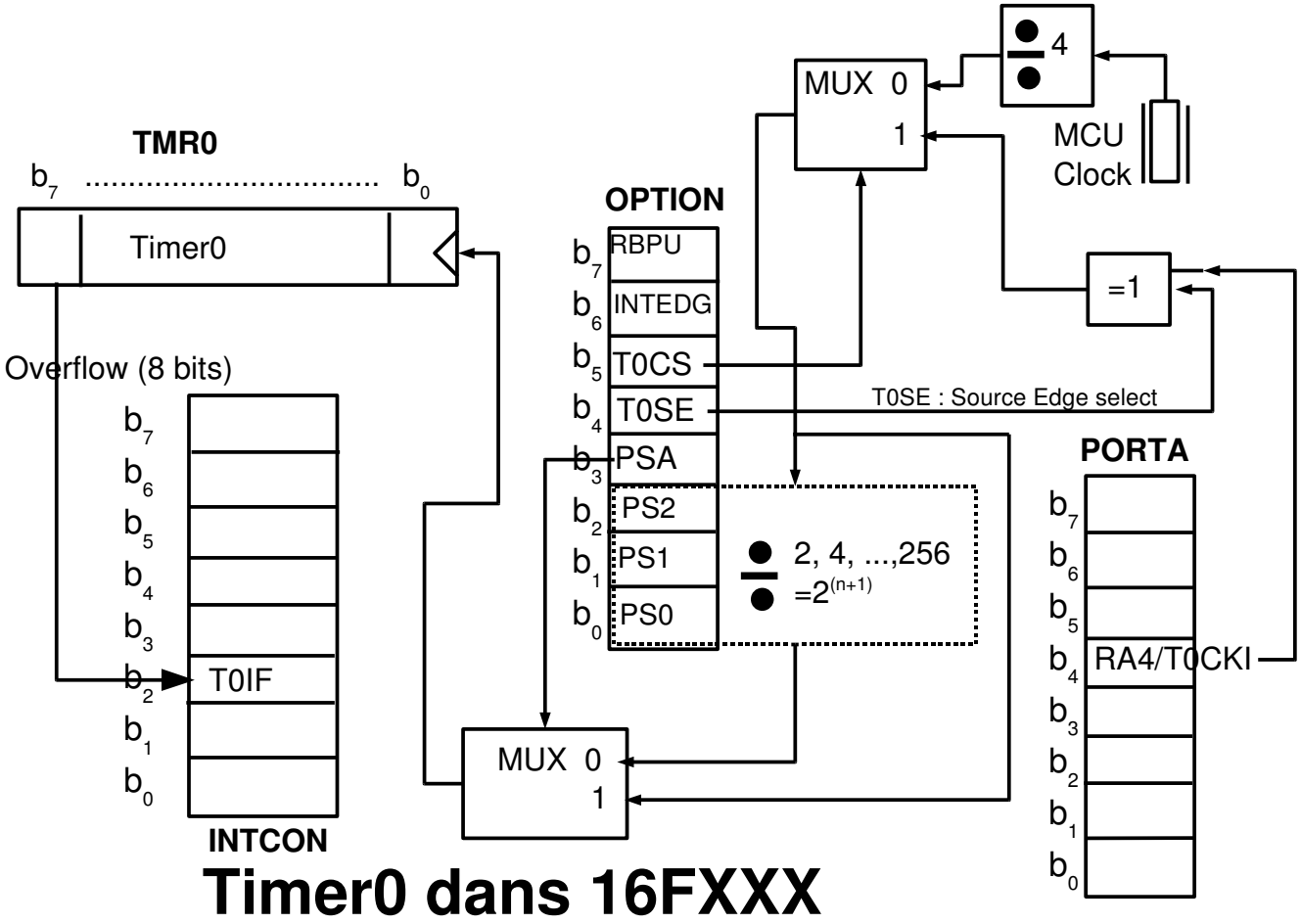
Question 3 : détermination du caractère

A partir des deux informations précédentes transformer le numéro de colonne et le numéro de ligne en caractère correspondant sur le clavier : '1' ou '2' ou ... ou '0' ou '#'
Remarque : les bits RB4-RB7 peuvent servir à déclencher une interruption (RB port change Interrupt). Si l'on veut utiliser cette interruption il faudrait câbler notre clavier autrement. Les interruptions seront abordées plus loin.

II2 - TD 4 Le Timer0 des PIC® 16FXXX

1. Le Timer 0

La documentation du Timer0 du PIC® 16F877 est présentée maintenant sous forme schématique. Notez que le registre TMR0 est un registre huit bits, dans lequel on peut écrire ou lire (en une seule instruction).



Remarque : le registre **OPTION** se note OPTION_REG en MikroC.

2. Les différents modes de fonctionnement

Le timer0 est en fait un compteur. Mais qu'allez-vous compter avec ce timer ? Et bien, vous avez deux possibilités :

- En premier lieu, vous pouvez compter les impulsions reçues sur la pin RA4/T0CKI. Nous dirons dans ce cas que nous sommes en mode compteur
- Vous pouvez aussi décider de compter les cycles d'horloge du PIC® lui-même. Dans ce cas, comme l'horloge est fixe, nous compterons donc en réalité du temps. Donc, nous serons en mode « timer ».

La sélection d'un ou l'autre de ces deux modes de fonctionnement s'effectue par le bit 5 du registre **OPTION** : T0CS pour Tmr0 Clock Source select bit.
T0CS = 1 : Fonctionnement en mode compteur
T0CS = 0 : Fonctionnement en mode timer
Dans le cas où vous décidez de travailler en mode compteur, vous devez aussi préciser lors de quelle transition de niveau le comptage est effectué. Ceci est précisé grâce au bit 4 du registre **T0CON** :
T0SE pour Timer0 Source Edge select bit.
T0SE = 0 : comptage si l'entrée RA4/TOKI passe de 0 à 1 (front montant)

T0SE = 1 : comptage si l'entrée RA4/TOKI passe de 1 à 0 (front descendant)

3. Mesure du temps d'exécution d'un algorithme

L'optimisation d'un algorithme en vitesse (ou en taille) est très importante dans les systèmes embarqués réalisés par des micro-contrôleurs. Une recherche d'algorithmes sur Internet vous donnera des résultats qu'il vous faudra évaluer. Par exemples, le site :
http://www.piclist.com/techref/language/ccpp/convertbase.htm
vous propose un algorithme de division par 10 que voici :

```
unsigned int A;
unsigned int Q; /* the quotient */
        Q = ((A >> 1) + A) >> 1; /* Q = A*0.11 */
        Q = ((Q >> 4) + Q)     ; /* Q = A*0.110011 */
        Q = ((Q >> 8) + Q) >> 3; /* Q = A*0.00011001100110011 */
        /* either Q = A/10 or Q+1 = A/10 for all A < 534,890 */
```

Exercice 1
1°) Sans chercher à comprendre l'algorithme de division, on vous demande de le transformer en une fonction `unsigned int div10(unsigned int A);`
2°) Écrire un programme complet qui mesure le temps d'exécution du sous programme de division par 10, puis modifier le programme pour qu'il puisse comparer avec une division par 10 normale.

4. Le mode de scrutation du flag

Nous devons savoir à ce niveau, que tout débordement du timer0 (passage de 0xFF à 0x00) entraîne le positionnement du flag T0IF, bit b_2 du registre **INTCON**. Vous pouvez donc utiliser ce flag pour déterminer si vous avez eu débordement du timer0, ou, en d'autres termes, si le temps programmé est écoulé. Cette méthode à l'inconvénient de vous faire perdre du temps inutilement dans une boucle d'attente.
Petit exemple :

```
while (INTCON.T0IF==0); //attente active
```

Exercice 2
Le quartz est choisi à 4MHz dans ce problème.
Question 1
On donne le programme suivant concernant le timer 0 :

```
1       void main(void){
2       // initialisation du timer   division par 8
3               OPTION_REG = 0x02; // prescaler 8 , entrée sur quartz
4               TMR0 = 0x00; // clrf  tmr0     ; début du comptage dans 2 cycles
5       // bit RA0 du PORTA en sortie
6               TRISA0=0;
7               while(1) {
8                       INTCON.T0IF = 0;
9                       while (INTCON.T0IF == 0);
10                      RA0 = ~RA0; // on bascule : ~complement bit a bit
11              }
12      }
```

Pouvez-vous donner la fréquence d'oscillation du bit b_0 du **PORTA** avec quelques explications ?

Question 2
Écrire en langage C un programme qui fait la même chose que le programme assembleur ci-dessus : initialise le timer0, efface le flag et attend à l'aide d'une boucle le positionnement de ce dernier mais 100 incrémentations seulement.

Remarque : en fait toute modification de **TMR0** entraîne un arrêt de comptage de la part de celui-ci correspondant à 2 cycles d'instruction multipliés par la valeur du pré diviseur. Autrement dit, un arrêt correspondant toujours à 2 unités **TMR0**. Il faut donc tenir compte de cette perte, et placer « 256-98 » et non « 256-100 » dans le timer.

Question 3

Générer un signal de fréquence 1 KHz. Pour cela :
- calculer la valeur de la pré division
- calculer la valeur de décomptage
- Écrire le programme.

Question 4

Même chose mais il faut que la période diminue progressivement

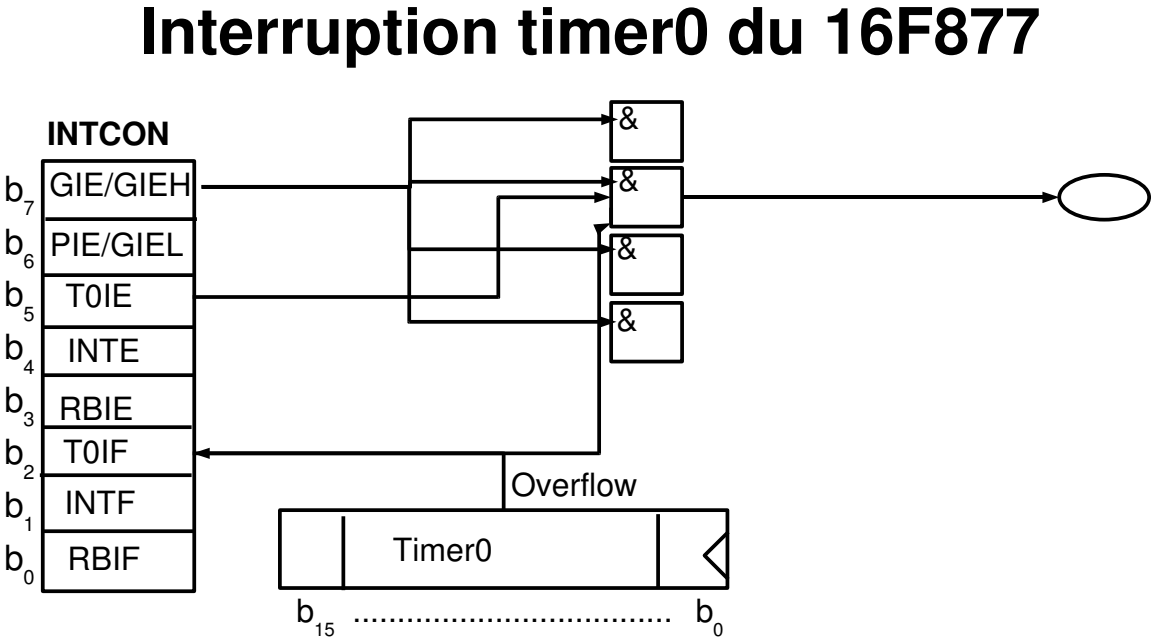
Question 5

Générer un signal de sortie de rapport cyclique 1/4 sur le même principe.
Il y a mieux à faire avec les PICs, utiliser le module CCP.

II2 - TD 5 : Interruption timer0

1. Le timer 0

La documentation du Timer0 du PIC® 16F877 a été présentée dans le TD5. Nous passons donc à la description de son interruption associée.



Ce schéma de principe fonctionne de la manière suivante. Il y aura interruption si on arrive à réaliser un front montant dans l'ellipse, sachant que le matériel ne gère que l'overflow (bit TMR0IF du registre **INTCON**).
Le programme qui gère l'interruption sera en adresse 0x004, mais ce sera transparent en langage C.

2. Les interruptions en C

La gestion des interruptions en C est fortement dépendante des compilateurs, comme le montre le tableau ci-dessous :

MikroC (notre compilateur)	**Hitech C**

`// interruption haute priorite seulement`
`void interrupt() {`
`  // traitement interruption`
`  // positionnement du FLAG`
`}`
 	
`void main() {`
`  // traitement`
`  while(1);`
`}`
	`#include <pic1684.h>`
	`void interrupt it_gene(void);`
	`main(){`
	`  // traitement`
	`  while(1);`
	`}`
	`void interrupt it_gene(void) {`
	`  // traitement interruption`
	`  // positionnement du FLAG`
	
	`}`

Avec le compilateur mikroC le nom de l'interruption est fixé.
L'écriture d'une interruption se fait normalement en trois étapes :
- spécifier que le sous-programme n'est pas un sous-programme normal, mais un sous-programme d'interruption,
- fixer l'adresse du sous-programme d'interruption
- initialiser le mécanisme d'interruption dans le programme principal.

Pour certains compilateurs (MikroC) les deux premières étapes se font simplement en donnant un nom prédéterminé au sous-programme d'interruption.

Exercice 1

Un PIC16F84 est enfoui dans un FPGA. Sa seule particularité est de fonctionner à 50MHz contre 10 (resp. 20 MHz) de fréquence maximale d'horloge pour les PIC 16F84 (resp. 16F84A). Il exécute le programme suivant (écrit avec le compilateur Hitech C) :

```
#include  <pic1684.h>
void interrupt decalage(void);
unsigned char nb;
main(void) {
  TRISA = 0xF9; // 6 entrees, 2 sorties pour A
  TRISB = 0x00; // 8 sorties pour B
  OPTION = 0x07; // prescaler 256 , entree sur quartz
  INTCON = 0xA0;  // autorise l'interruption timer
  PORTB = 0x01; // une seule diode allumee
  TMR0 = 0x00 ;
  nb=0;
  while(1) {
   // on ne fait rien que recopier sur 2 segments la valeur de SW1
      if ((PORTA & 0x01) == 1) PORTA = 0x06;
    }
}

void interrupt decalage(void) {
  nb++;
  //TMR0 = 0x00; //c'est fait car ici par overflow
  if (!(nb % 16))
  PORTB = (PORTB << 1) ;
  if (PORTB == 0x00)   PORTB = 0x01;
  T0IF = 0;  // acquittement interruption
}
```

Remarquez comment est écrit une interruption avec ce compilateur.

1°) Repérer et modifier les lignes de ce programmes pour qu'il fonctionne avec le compilateur MikroC.

2°) Calculer si le chenillard réalisé par ce programme est visible à l'œil humain (fréquence de changement de position des LEDs inférieure à 20 Hz).

3°) Comment peut-on écrire l'instruction "`if (!(nb % 16))`" pour plus d'efficacité.

4°) Quelle est la suite des états (LEDs allumées) réalisée par ce programme.

Les deux questions suivantes sont issues du Devoir Surveillé de Juin 2010.

5°) Le programme suivant est donné comme exemple du compilateur MikroC et tourne dans un PIC 16F84 qui a un quartz de 4 MHz.

```
unsigned cnt;

void interrupt() {
  if (INTCON.T0IF==1) {
    cnt++;                  // increment counter
    INTCON.T0IF = 0;         // clear T0IF
    TMR0   = 96;
  }
}
```

```
void main() {
  OPTION_REG = 0x84;       // Assign prescaler to TMR0
  ANSEL  = 0;              // Configure AN pins as digital
  ANSELH = 0;
  C1ON_bit = 0;            // Disable comparators
  C2ON_bit = 0;
  TRISB = 0;               // PORTB is output
  PORTB = 0xFF;            // Initialize PORTB
  TMR0  = 96;              // Timer0 initial value
  INTCON = 0xA0;           // Enable TMRO interrupt
  cnt = 0;                 // Initialize cnt

  do {
    if (cnt >= 400) {
      PORTB = ~PORTB;      // Toggle PORTB LEDs
      cnt = 0;             // Reset cnt
    }
  } while(1);
}
```

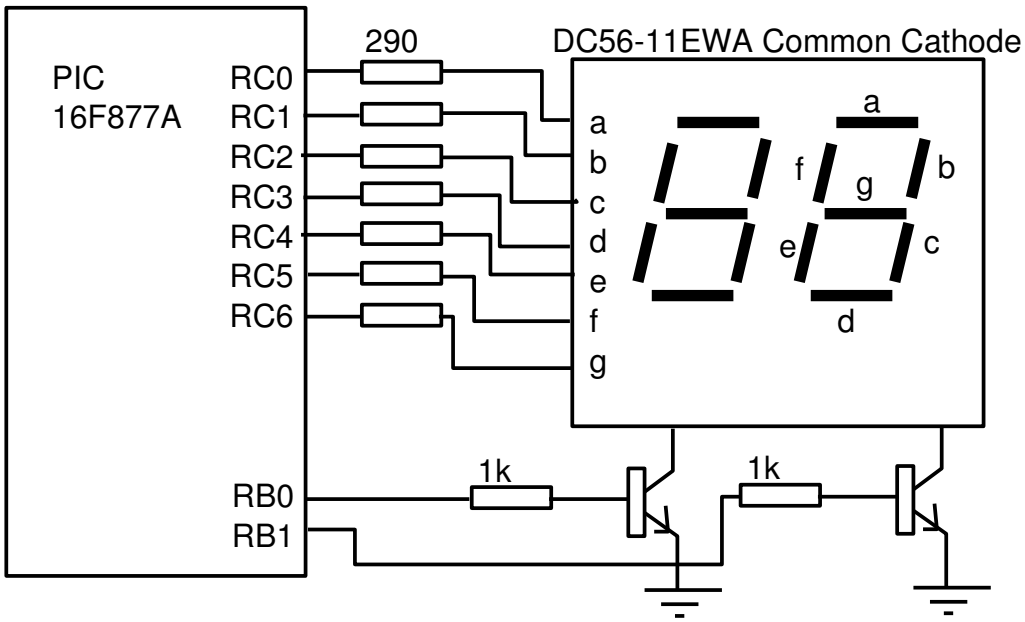
Quelle est la fréquence de clignotement des LEDs reliées au PORTB ?
6°) Modifier le programme principal pour réaliser un chenillard d'une LED se déplaçant vers les poids faibles.

Exercice 2

Une partie matérielle est constituée de deux afficheurs sept segments multiplexés. Les sept segments sont commandés par le **PORTC**, tandis que les commandes d'affichages sont réalisée par les bits b_0 et b_1 du **PORTB**. Un schéma de principe est donné ci-après.
1°) A l'aide de la documentation calculer les valeurs dans un tableau "`unsigned char SEGMENT[] = {0x3F,...};`" pour un affichage des chiffres de 0 à 9.
2°) réaliser une fonction responsable du transcodage :

```
unsigned char Display(unsigned char no) {
    unsigned char Pattern;
    unsigned char SEGMENT[] = {0x3F,....
```

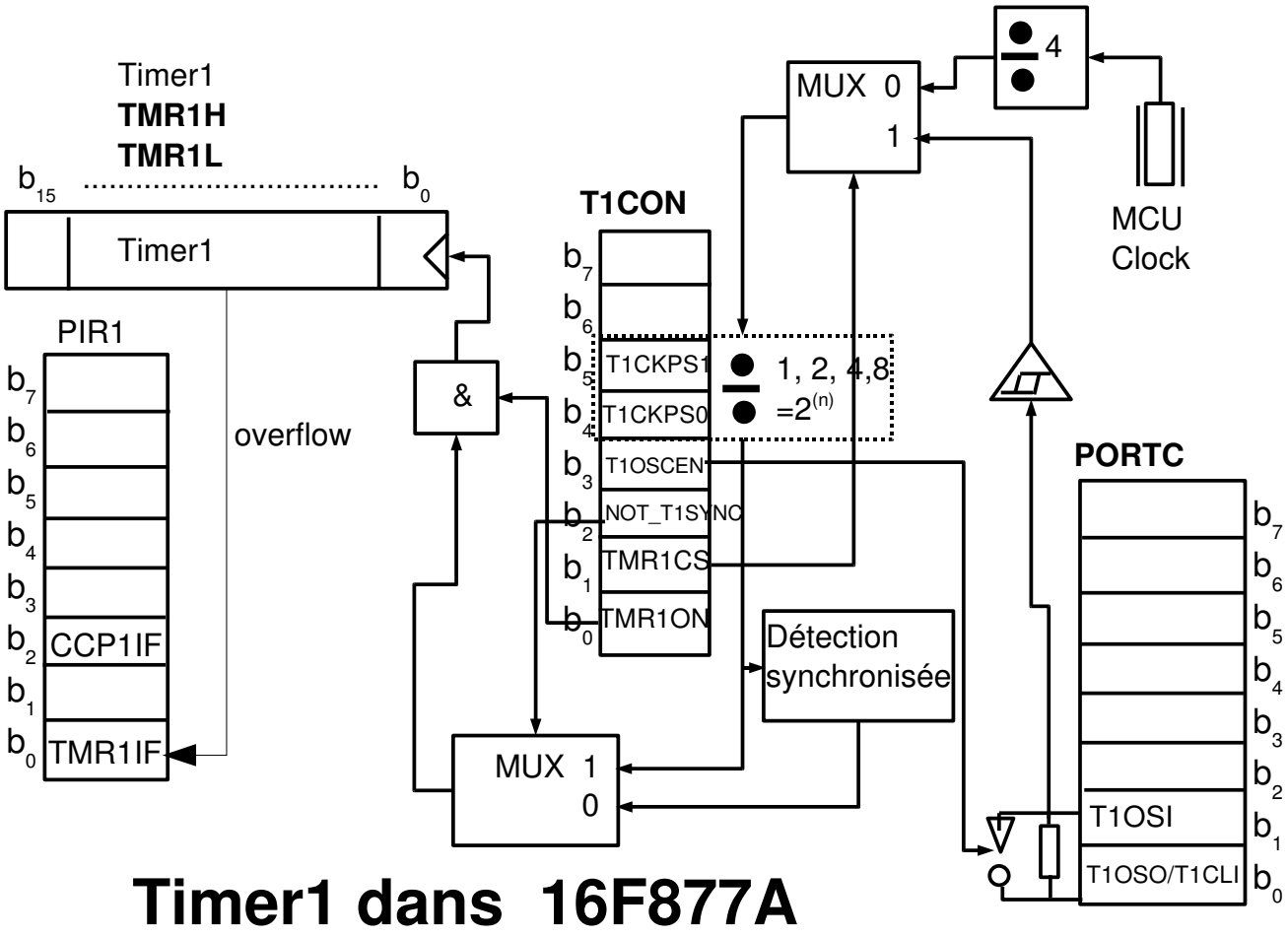
3°) Réaliser le programme main() responsable de l'initialisation de l'interruption qui doit avoir lieu toutes les 10ms (avec un quartz de 4MHz) et qui compte de 00 à 99 toutes les secondes environ (avec un "Delay_ms(1000);")
4°) Réaliser enfin l'interruption qui affichera tantôt les dizaines, tantôt les unités.



TD 6 : Le mode comparaison du module CCP (Capture/Compare/PWM)

L'objectif de ce TD est de générer des signaux de fréquences déterminées. Ce travail est réalisé en général avec le module CCP (Capture/Compare/PWM) et plus exactement son mode comparaison. Nous allons commencer par décrire le fonctionnement du timer 1 du 16F877.

1. Le Timer 1 et ses registres



Timer1 dans 16F877A

Les registres utilisés par le Timer1 sont donc :
- **TMR1H** et **TMR1L**,
- **T1CON**
- éventuellement le **PORTC**, et **PIR1**.

La détection synchronisée doit être obligatoirement utilisée dans les modes compare (de ce TD) et le mode capture du TD suivant.

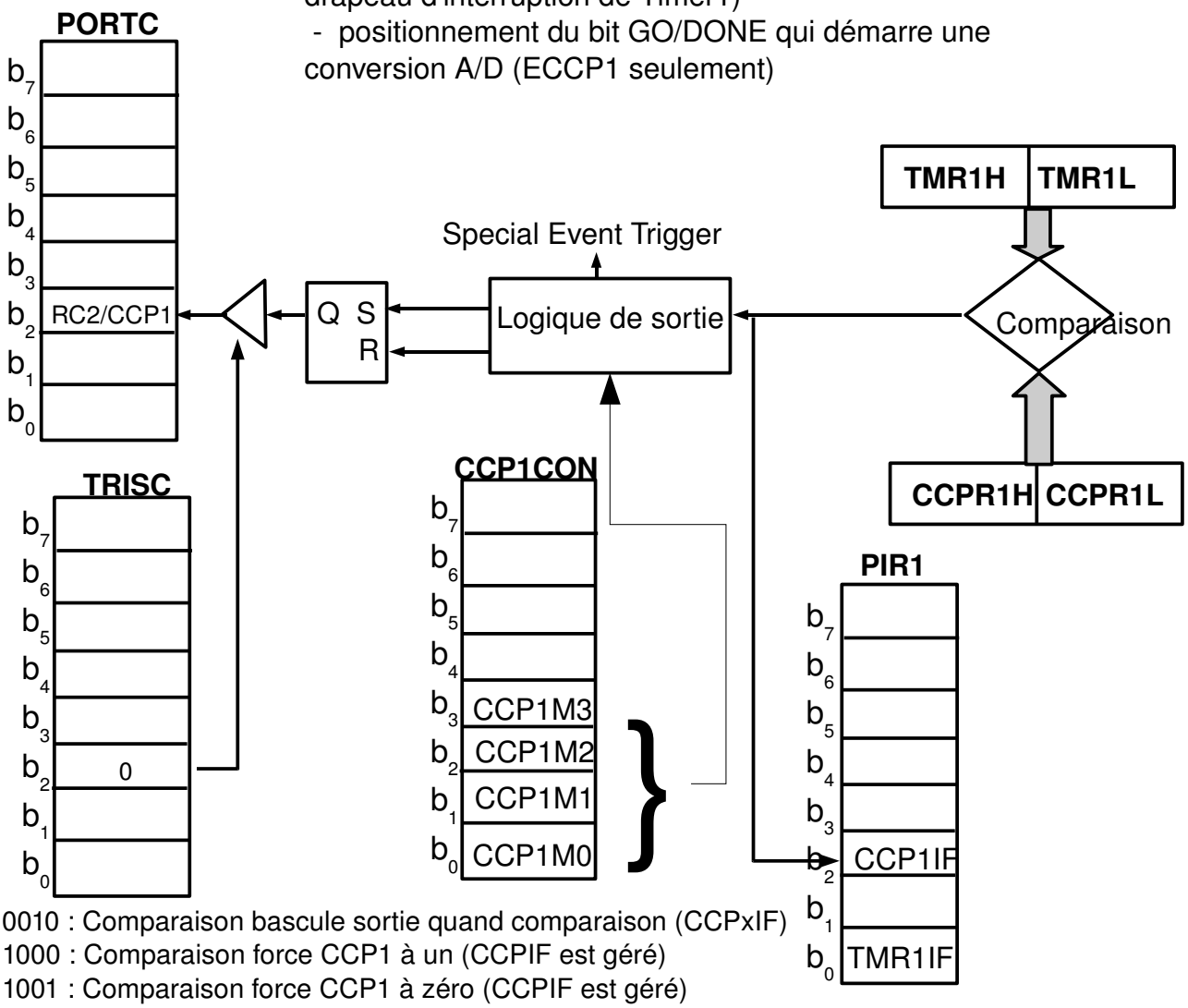
Exercice 1
The PIC® has an oscillator frequency of 4 MHz. What is the slowest Timer 1 interrupt rate it can generate with this internal clock ?

2. Le mode comparaison

Le mode comparaison va nous permettre de générer des signaux de fréquence déterminée.

COMPARAISON PIC 16F877

Special Event Trigger déclenchera:
- un Reset du Timer1 (mais pas un positionnement du drapeau d'interruption de Timer1)
- positionnement du bit GO/DONE qui démarre une conversion A/D (ECCP1 seulement)



0010 : Comparaison bascule sortie quand comparaison (CCPxIF)
1000 : Comparaison force CCP1 à un (CCPIF est géré)
1001 : Comparaison force CCP1 à zéro (CCPIF est géré)
1010 : Comparaison sort rien sur CCP1 mais sur CCPIF
1011 : Comparaison force Special Event Trigger reset sur timer1 (CCP1IF est géré)

Exercice 2 Réalisation d'un signal de 1Hz et son amélioration.

1°) Notre horloge quartz a une fréquence de 20 MHz. Utiliser le résultat de l'exercice 1 pour répondre : par combien doit-on diviser la fréquence de CCP1IF minimale pour obtenir un temps de 500000 µs (0,5 s)?
2°) On choisit une division par 5. Le principe sera donc d'attendre CCP1IF cinq fois sans oublier de le remettre à 0 et d'inverser par logiciel la sortie RC2. Il nous faut donc choisir un mode de fonctionnement logiciel (1010 qui n'agit pas sur RC2). Écrire le programme complet.

Remarque : on pourrait utiliser le flag de débordement du Timer1 (TMR1IF du registre **PIR1**) pour faire la même chose, ce qui n'utiliserait pas le module CCP.

3°) Avec la technique de la question précédente, il nous est absolument impossible de régler exactement la fréquence de sortie. On va essayer de palier à cet inconvénient en utilisant un autre mode qui n'agit pas sur RC2 mais a l'avantage de déclencher une remise à 0 du timer1 quand il y a comparaison : c'est le mode 1011. Calculer la valeur à mettre dans **CCPR1**.
Écrire le programme.

4°) Compléter le programme précédent pour qu'il envoie sur une liaison série (avec un printf) le temps en heure minute seconde.

On verra plus tard qu'une méthode plus précise consiste à utiliser un quartz horloger.

II2 - TD7 Interfacer des éléments de puissance pour faire tourner des moteurs

Quand le courant à sortir dépasse 15 mA (par bit), il faut trouver une solution amplifiée. C'est ce que nous allons examiner dans ce chapitre.

1. Utilisation de transistors BJT

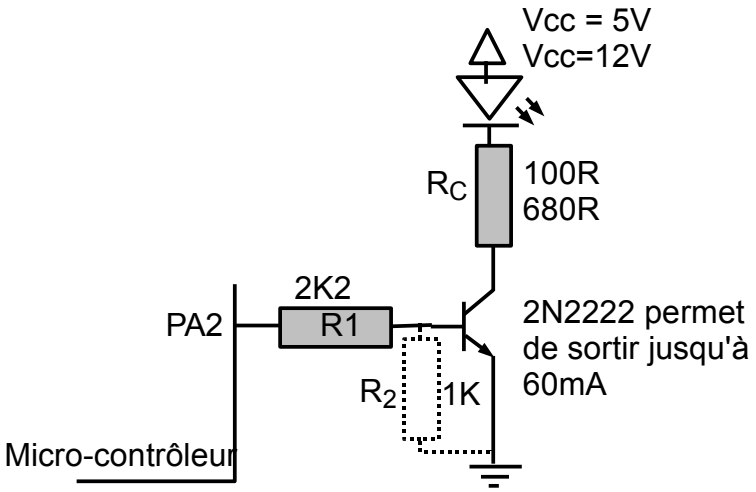
Pour commander des sorties puissantes il faut ajouter des transistors (BJT Bipolar Junction Transistor). Les sorties puissantes peuvent être :
- des relais
- des bobines
- des LEDs
- des moteurs

Exercice 1

On veut commander une LED qui nécessite 16mA. Le transistor choisi a un β de 100.

1°) Calculer I_B ?

2°) En déduire R1 (si l'on prend une marge de 30% ou si l'on prend 1mA par défaut)



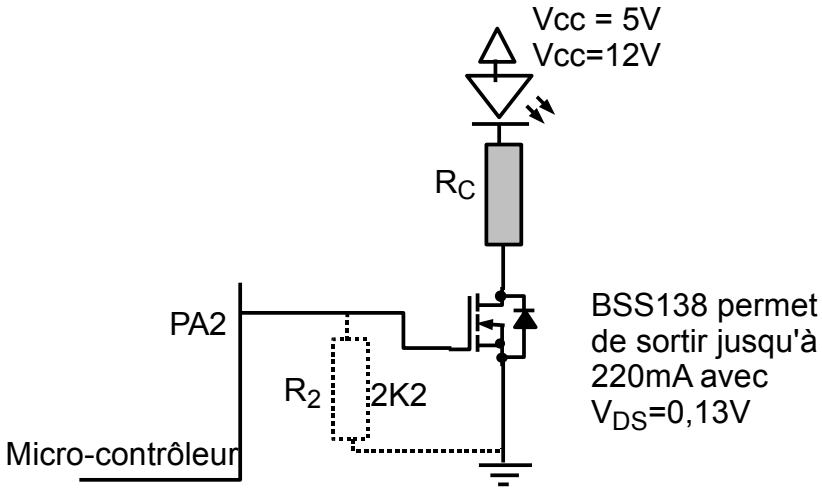
3°) En déduire R_C pour un V_{CEsat}=0,2V (pour une alimentation Vcc=5V) ?

4°) Une résistance R_C de 1/4W suffit-elle ?

2. Interfacer des sorties puissantes avec des FET

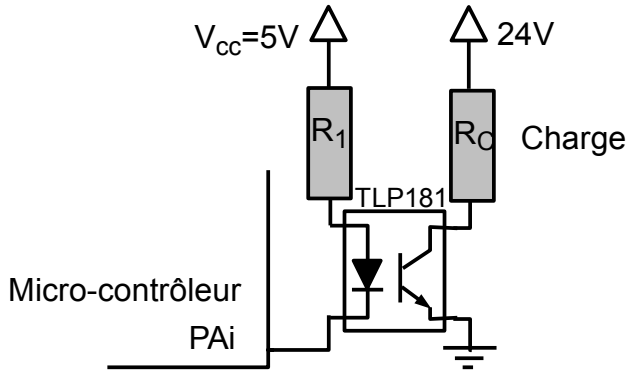
On peut utiliser des transistors à effet de champ à la place des BJT. Nous examinons le cas des MOSFET. Un paramètre important à vérifier est la tension de grille nécessaire à la commutation. En effet pour les micro-contrôleur alimenté en 3,3V cette tension peut s'avérer un peu forte pour des MOSFET de puissance.

Il existe des familles spécialement faites pour être commandées directement par des PORTs. Voir ZVN4206A (600mA) et ZVN4306A (1,1A) de chez Zetex.



3. Encore plus de puissance

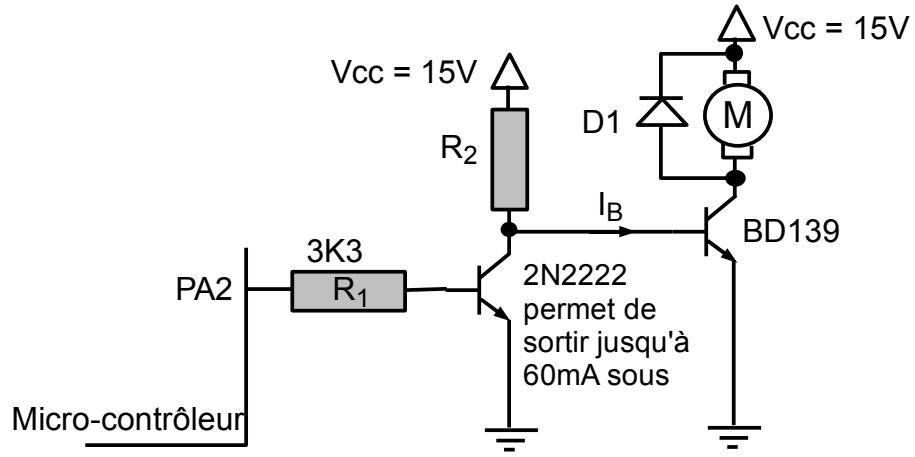
Il faut utiliser des opto-coupleurs. Il permettent aussi une isolation galvanique ce qui est important si la tension d'alimentation est très différente de celle du micro-contrôleur.



TLP181 :
16mA < I_{Dmax} < 20mA avec V_D <1,3V à 10mA
1mA < I_C <10mA
Dimensionner R_1.

4. Commander des bobines moteurs et relais avec BJT

Il existe des transistors de puissance adaptés. Par exemple le transistor BD139 propose un gain β compris entre 25 et 250. On peut l'utiliser avec un courant I_C=1,5 A (3A en pointe). Son faible gain nécessite un courant I_B assez important, allant au-delà des possibilités des ports traditionnels (typiquement 10mA). Par exemple pour commuter 1A, il faut un I_B=40mA dans le pire des cas. Il faut donc un transistor supplémentaire.



Ce montage appelé Darlington existe en boîtier unique : par exemple le BCX38C peut commuter des courants jusqu'à 800mA.

Exercice 2
Le transistor 2N2222 est bloqué, il circule 40mA dans I_B
1°) Si R_2 est choisi à 330 Ω, quelle puissance passe dans R_2 ?
2°) Si le transistor 2N2222 est saturé (V_{CEsat}=0,2V) quelle puissance passe dans R_2 ?
3°) Lorsque le BD139 est saturé, il circule Il circule IC=1A et l'on a VCEsat=0,70V. Quelle puissance est dissipée dans le BD139 ?

Exercice 3 tiré du DS final de Juin 2011 (Construction d'un signal PWM pour un moteur)

Dans tout cet exercice, on dispose d'un quartz générant MCU Clock à 4 MHz. On utilisera le timer 1 documenté au TD précédent.
C'est le Timer1 qui va être responsable de la réalisation de la période de notre signal PWM (MLI Modulation de Largeur d'Impulsion) ou à rapport cyclique variable. Cette période est choisie à 20 ms et on rappelle que notre horloge a une fréquence de 4MHz. Le fonctionnement général du programme est le suivant :
- initialise le timer
- initialiser la comparaison
Boucle infinie
    - attendre une comparaison et mettre la sortie RC2 à 0

- attendre que le timer1 déclenche un overflow et mettre systématiquement la sortie RC2 à 1
fin boucle

1°) Donner la ou les instructions C permettant de réaliser une division par 8. Quelle période a-t-on alors à l'entrée du bloc "Détection synchronisée" ? Quelle est la fréquence F_{DS} correspondante ?
Réponses :

2°) Par combien doit-on diviser la fréquence F_{DS} pour obtenir nos T = 20ms ?
Réponse :

3°) Si la division de la question 2 est réalisée par le timer1, avec valeur doit-on initialiser le timer1 (en décimal et en hexadécimal)
Réponses :

4°) Donner l'instruction C qui sélectionne le mode "détection synchronisée" et le quartz comme entrée ainsi que TRM1ON.
Réponses :

5°) Quelle instruction C attend le débordement du timer1 ?
Réponse :

6°) Voir documentation sur la comparaison au TD précédent.
La comparaison est choisie comme : "1010 compare sort rien sur CCP mais sur CCPIF". Quelle instructions C permettent de choisir ce mode ?
Réponses :

7°) Quelles valeurs faut-il mettre dans CCPR1H et CCPR1L pour réaliser un rapport cyclique de 10% , c'est à dire une attente de 2ms ?
Réponses :

8°) Quelle est l'instruction C qui permet d'attendre la comparaison ?
Réponse :

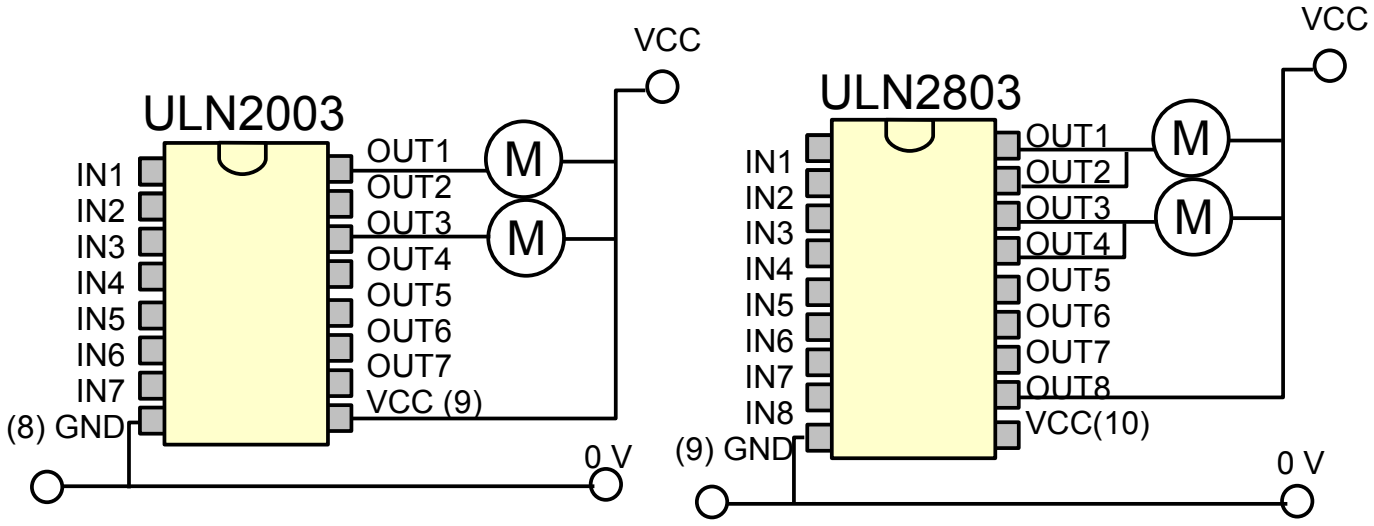
Questions hors DS
9°) Complétez les valeurs manquantes (commentaires // ??) dans le programme ci-dessous qui réalise complètement le PWM (en mode 1010)

```
1         TRISC.F2=0; //sortie
2         // timer1
3         T1CON = T1CON |    ; // ??
4         // comparaison
5         CCP1CON = CCP1CON & 0xFA | 0x0A; // mode 1010 pour la comparaison
6         CCPR1H = 0x      ; // ??
7         CCPR1L = 0x      ; // ??
8         PIR1.CCPIF=0;
9         PIR1.TMR1IF=0;
10        while(1) {
11          while(PIR1.CCPIF==0);
12          PORTC.F2=1;
13          PIR1.CCPIF=0;
14          while(PIR1.TMR1IF==0);
15          // c'est parti pour un tour
16          TMR1H = 0x       // ??
17          TMR1L = 0x       // ??
18          PORTC.F2=1;
19          PIR1.TMR1IF=0;
20        }
```

10°) On utilise le schéma avec un 2N2222 et un BD139 en échangeant PA2 en RC2. Quel courant moyen passe dans le moteur si le dimensionnement des résistances est tel que le maximum de courant est de 1,5 A ?

5. Commander plusieurs moteurs avec un circuit unique

Il existe des circuits capable de commander 7 moteurs comme le ULN2003 qui a à peu près les mêmes caractéristiques que le BCX83C (mais sept fois). Ils sont appelés Darlington driver IC. Le ULN2803 possède 16 broches et peut commander 8 moteurs avec la possibilité d'utiliser deux sorties identiques pour commander un seul moteur.

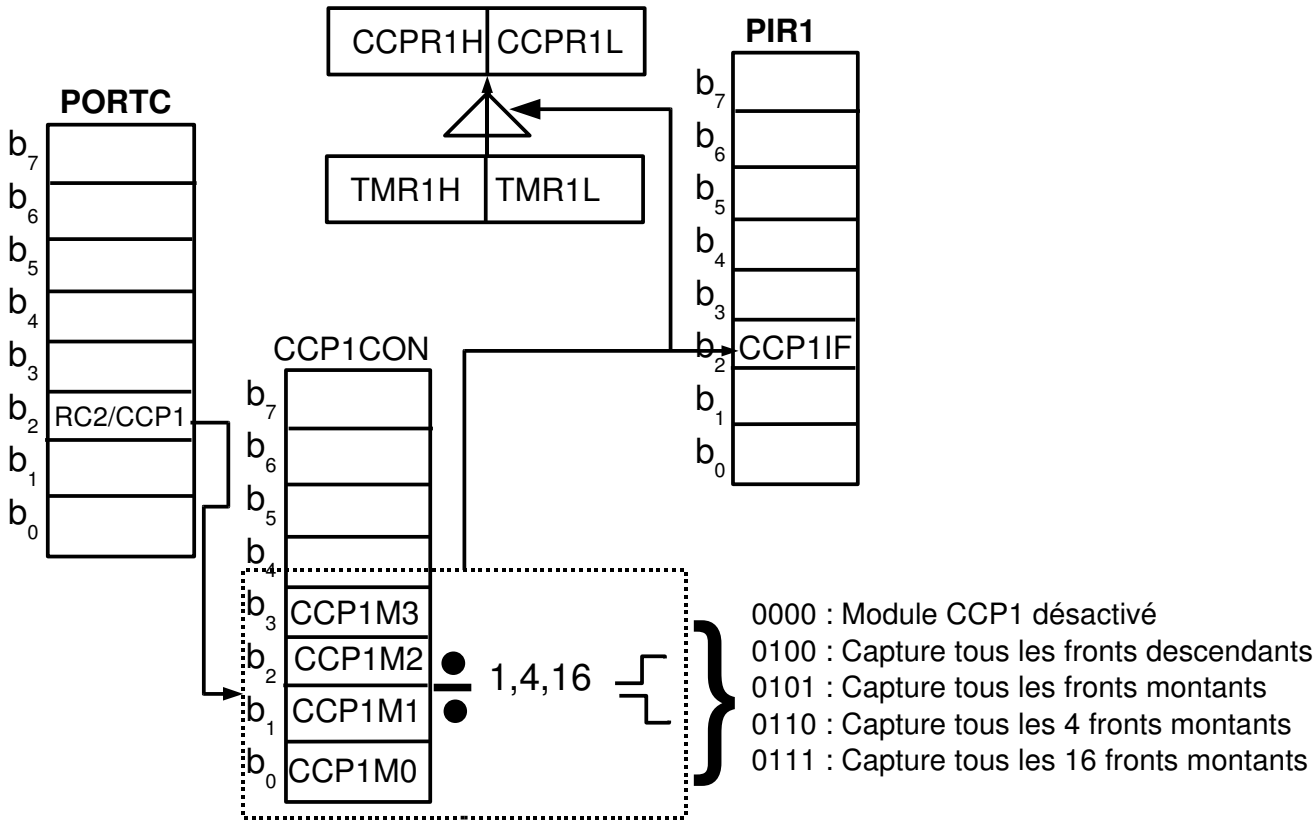


Le circuit ULN2003 est capable de délivrer 500 mA sous 50V.

TD 8 : Le mode capture du module CCP (Capture/Compare/PWM)

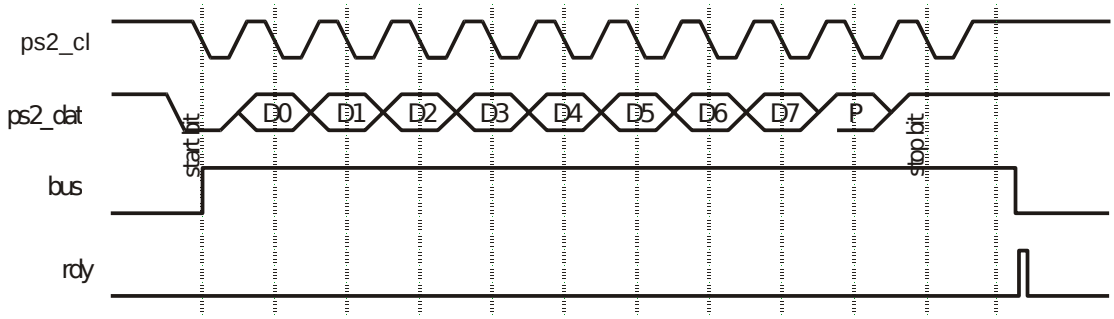
Dans le mode capture, **CPPR1H** et **CPPR1L** capturent la valeur du timer 1 ou du timer 3 quand un événement se produit sur RC2/CPP1 du port C. Un événement est défini comme :
- un front descendant
- un front montant
- tous les 4 fronts montants
- tous les 16 fronts montants

CAPTURE POUR 16F877



Attention le **PORTC** doit être configuré en entrée pour le bit b_2 (avec **TRISC**).

Exercice : Utilisation du mode capture pour lire un clavier PS2
Remarque préliminaire : en toute rigueur, il serait préférable de réaliser cette capture avec une interruption. Les interruptions sont abordées au TD 5.
Présentation du protocole PS2 :
Le protocole PS2 complet est un mélange de protocole synchrone et asynchrone :



où seules les premiers signaux ps2_clk et ps2_data vont nous intéresser. Comme c'est le clavier qui émet les données c'est à lui de fabriquer l'horloge. La fréquence d'horloge est comprise entre 10kHz et 16kHz et on va utiliser la partie capture non pas pour mesurer la fréquence de l'horloge (dans un premier temps en tout cas), mais pour détecter les fronts descendants de l'horloge grâce au bit CCP1IF du registre PIR1.
1°) Écrire le sous-programme "`void initKBD(void)`" destiné à initialiser le fonctionnement du module CCP pour qu'il intercepte les fronts descendants de PS2_clk, si les données ps2_data sont sont sur le bit b_1 du PORTC. Ne pas oublier de mettre les deux bits du PORTC en entrée.
2°) Écrire le programme principal qui lit les informations à chaque fois qu'un front descendant arrive et construit le bit du scan-code après lecture pour le sortir sur le PORTB (positionné en sortie).
3°) Utiliser maintenant complètement le mode capture pour calculer la fréquence moyenne de l'horloge ps2_clk si notre fréquence de quartz est 4 MHz..

Remarque : le compilateur MikroC possède une librairie de lecture du PS/2.

Prototype	void Ps2_Init(unsigned short *port);
Description	Initializes port for work with PS/2 keyboard, with default pin settings. Port pin 0 is Data line, and port pin 1 is Clock line. You need to call either Ps2_Init or Ps2_Config before using other routines of PS/2 library.
Prototype	void Ps2_Config(char *port, char clock, char data);
Description	Initializes port for work with PS/2 keyboard, with custom pin settings. Parameters data and clock specify pins of port for Data line and Clock line, respectively. Data and clock need to be in range 0..7 and cannot point to the same pin. You need to call either Ps2_Init or Ps2_Config before using other routines of PS/2 library.
Prototype	char Ps2_Key_Read(char *value, char *special, char *pressed);
Description	The function retrieves information about key pressed. Parameter value holds the value of the key pressed. For characters, numerals, punctuation marks, and space, value will store the appropriate ASCII value. Routine “recognizes” the function of Shift and Caps Lock, and behaves appropriately. Parameter special is a flag for special function keys (F1, Enter, Esc, etc). If key pressed is one of these, special will be set to 1, otherwise 0. Parameter pressed is set to 1 if the key is pressed, and 0 if released.

Voici un exemple trouvé dans sa documentation :

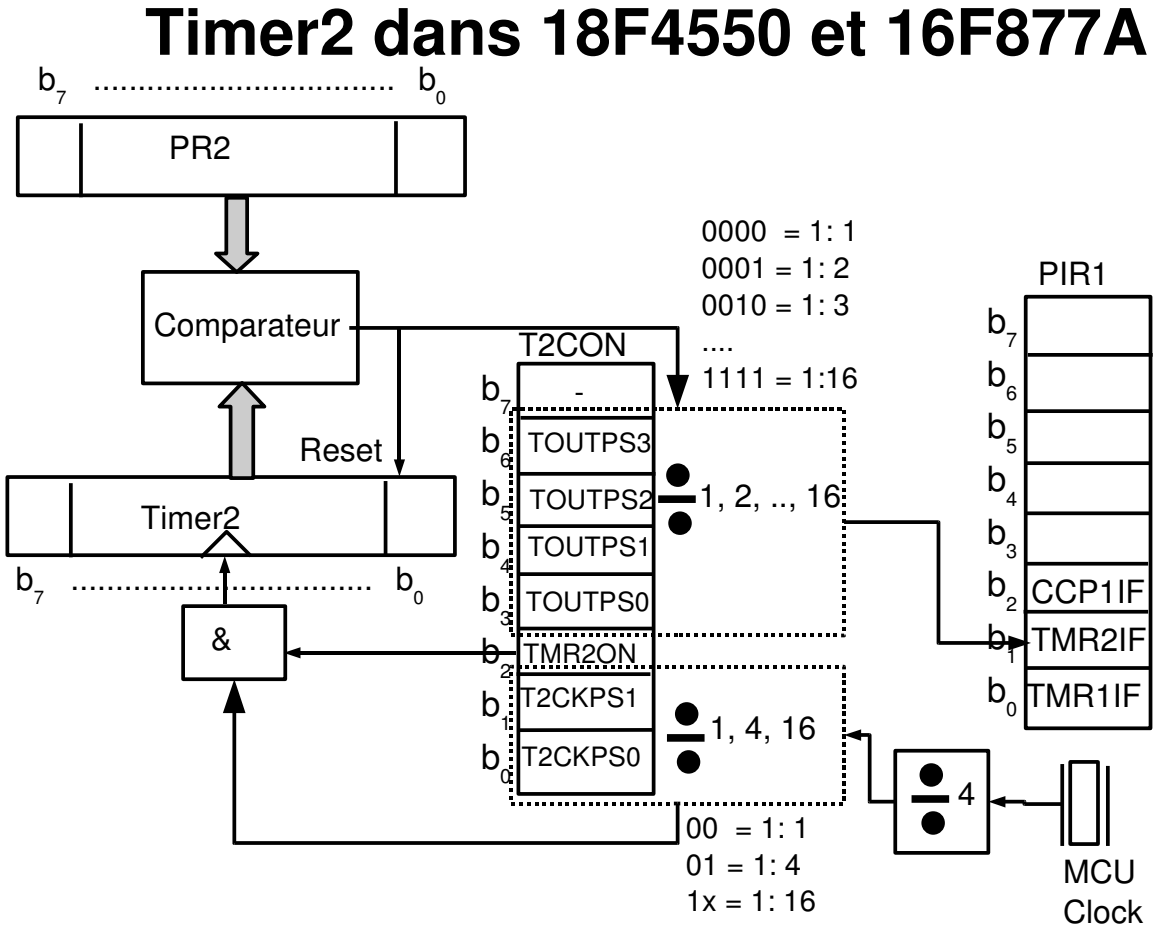
```
unsigned short keydata, special, down;
void main() {
  CMCON = 0x07;     // Disable analog comparators (comment this for PIC18)
  INTCON = 0;       // Disable all interrupts
  Ps2_Init(&PORTA); // Init PS/2 Keyboard on PORTA
  Delay_ms(100);    // Wait for keyboard to finish
  do {
    if (Ps2_Key_Read(&keydata, &special, &down)) {
       if (down && (keydata == 16)) {// Backspace
         // ...do something with a backspace...
       }
       else if (down && (keydata == 13)) {// Enter
         Usart_Write(13);
       }
       else if (down && !special && keydata) {
         Usart_Write(keydata);
       }
    }
    Delay_ms(10);     // debounce
  } while (1);
}//~!
```

TD 9 : Le mode PWM du module CCP (Capture/Compare/PWM)

PWM signifie « Pulse Width Modulation », ce qu'on pourrait traduire par modulation de largeur d'impulsion. Son objectif est de changer une valeur moyenne en jouant sur le rapport cyclique. Puisque ce module utilise essentiellement le timer 2, nous commençons donc à le décrire.

1. Description du timer 2

Le timer 2 est identique sur le PIC® 16F877 et le 18F4550. Il s'agit d'un timer 8 bits dont la description schématique est présentée maintenant.



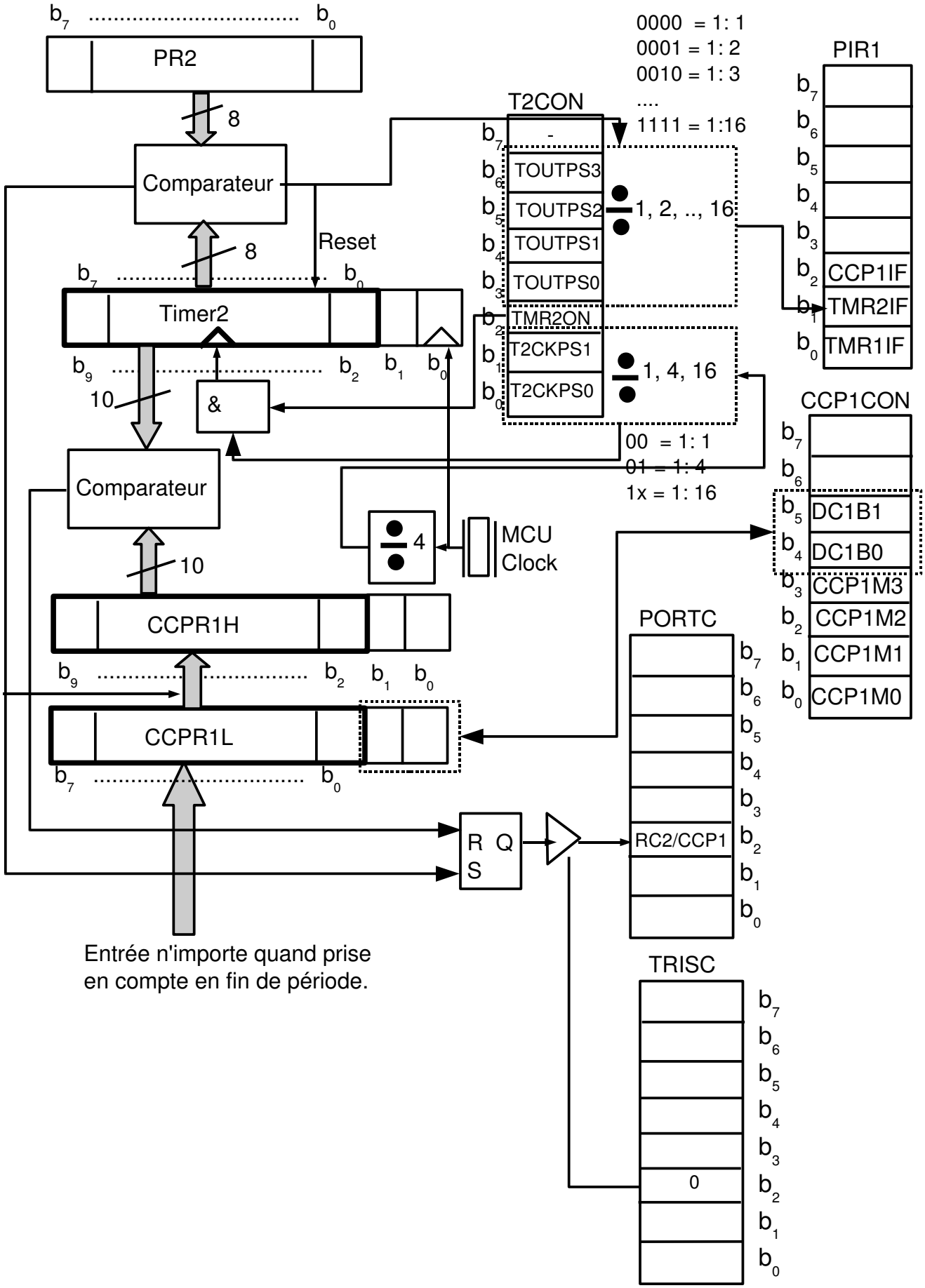
Sa seule entrée est l'entrée horloge du PIC® divisée par 4 suivie d'un pré-diviseur géré par deux bits. Un comparateur donne un signal qui sert de reset au timer2 et d'entrée à un post-diviseur capable de diviser par un nombre entre 1 et 16.
La période T peut être calculée par :
T = (**PR2** + 1) x (Timer2 input clock period) soit :

T = (**PR2**+1) x (T_{osc} x 4 Timer 2 prescale value)

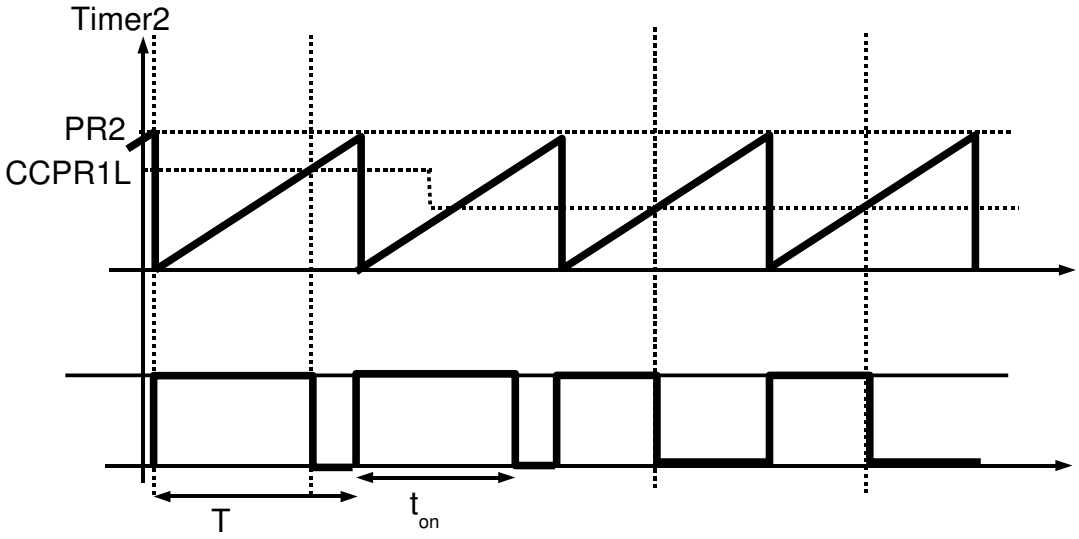
Exercice 1
The PIC® has an oscillator frequency of 4 MHz. What is the slowest Timer 2 interrupt rate it can generate ?
By writing to the **T2CON** register, Timer 2 is switched on, with neither pre- nor postscale. What is the value you have to write to **PR2** to realize a 4,00 kHz frequency ?

Quand on s'intéresse au PWM le choix de la période est réalisé simplement comme dans l'exercice 1. Puisque le tipmer2 est sur 8 bits, la résolution de la période est de 256. Le choix du rapport cyclique est par contre défini par le contenu d'un registre 8 bits **CCPR1L** avec deux bits supplémentaires DC1B1 et DC1B0. Ceci est montré dans la documentation ci-dessous :

PWM dans 18F4550 et 16F877A



Le PWM ne fonctionnera correctement que si $CCPR1L \leqslant PR2$.



La formule permettant de calculer t_{on} est :

t_{on} = (pulse width register) x (T_{osc} x Timer2 prescale value)

Remarque : notez la disparition du terme 4 dans cette formule : ce n'est plus 4 T_{osc} mais bien T_{osc}.

Exercice 2

The Timer 2 and PWM module of a 16F873A are operating in an application with a clock oscillator frequency, F_{OSC}, of 4MHz. The Timer 2 prescaler is initially set to 1:4, the **PR2** register is loaded with a value 240 (decimal), and CCPR1H with a value of 30 (decimal).

1 - a) Which register controls the period of the PWM waveform, and which register controls the 'on' time?

1 - b) For the settings described, what is the resulting PWM period ?

1 - c) For the settings described, what is the approximate resulting 'on' time ?

2°) Ecrire le programme C qui réalise ce fonctionnement.

Remarque : le compilateur MikroC possède une librairie de gestion du PWM. Voici un exemple trouvé dans sa documentation :

```
/*The example changes PWM duty ratio on pin RC2 continually. If LED is
connected
to RC2, you can observe the gradual change of emitted light. */
char i = 0, j = 0;
void main() {
  PORTC = 0xFF;           // PORTC is output
  Pwm_Init(5000);         // Initialize PWM module at 5KHz
  Pwm_Start();            // Start PWM
  while (1) {
    // Slow down, allow us to see the change on LED:
    for (i = 0; i < 20; i++) Delay_us(500);
    j++;
    Pwm_Change_Duty(j);   // Change duty ratio
  }
}
```

Prototype	void Pwm_Change_Duty(char duty_ratio);
Description	Changes PWM duty ratio. Parameter duty_ratio takes values from 0 to 255, where 0 is 0%, 127 is 50%, and 255 is 100% duty ratio. Other specific values for duty ratio can be calculated as (Percent*255)/100.

ANNEXE

Description de la mémoire RAM

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset
Bank 0										
00h	INDF	Utilise le contenu de FSR pour adresser les données (pas un registre)								---- ----
01h	TMR0	Timer temps réel 8 bits								xxxx xxxx
02h	PCL	Compteur programme (8 bits de poids faible)								0000 0000
03h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C	0001 1xxx
04h	FSR	Pointeur indirect de mémoire								-
05h	PORTA	-	-	-	RA4/ TOCKI	RA3	RA2	RA1	RA0	---x xxxx
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	
07h	----	Non implémenté								
08h	EEDATA	Registre de données EEPROM								xxxx xxxx
09h	EEADR	Registre d'adresses EEPROM								xxxx xxxx
0Ah	PCLATH	---	---	---	Compteur programme (poids haut) jusqu'à 5 bits					---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RIBF	0000 000x

Réalisé avec OpenOffice sous Linux