.:: Architecture ::.

semaine n°6
Interruptions et Timers

Le MCS251-Interruptions

• Les interruptions sont un mécanisme qui permet d'interrompre l'exécution courante du micro-contrôleur pour lancer l'exécution d'une procédure pré-définie.

• Les interruptions sont déclenchées par un signal émis par un périphérique :
    • Déclenchement du bouton d'arrêt d'urgence
    • Réception de données sur le port série
    • ...

• Ce mécanisme permet la prise en compte « instantanée » d'évènements asynchrones (dont la fréquence n'est pas déterminée)

Le MCS251-Interruptions

- Dans le MCS251, les interruptions peuvent être issues :
  - De broches externes : INT0 et INT1
    - On connecte ce que l'on veut : boutons, capteurs ...
  - Compteur / Temporisateur : Timer0, Timer1, Timer2
    - Déclenchement périodique d'un traitement
    - Déclenchement d'un traitement après un nombre précis d'évènements
  - Transmissions séries : Receive / Transmit
    - Déclenchement d'un traitement à réception de données
    - Déclenchement d'un traitement après transmission des données en attente (pour envoyer de nouvelles données)

Voir P23

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Déclenchement IT

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:    MOV    R0,#0
5003h:    MOV    R1,#2
5006h :   ADD    R0,R1
5008h:    ADD    R0,R1
```

Routine d'interruption :

```
4003h:    NOP
4004h:    NOP
4005h:    RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Lors de la réception d'une interruption, l'exécution normale du programme s'arrête pour être déroutée vers une routine d'interruption :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
5008h:      ADD     R0,R1
```

Routine d'interruption :

```
4003h:      NOP
4004h:      NOP
4005h:      RETI
```

Le MCS251-Interruptions

• Le déclenchement d'une interruption correspond, en quelque sorte, à l'insertion dynamique d'un CALL dans le programme en cours d'exécution.

• Attention : la routine d'interruption doit sauvegarder TOUS les registres qu'elle modifie.

• Comment le micro-contrôleur sait-il quelle procédure appeler lorsqu'une interruption se déclenche ?

- Il peut utiliser une table de correspondance : un vecteur d'interruption. Cette table contient pour chaque interruption l'adresse où aller, inscrite par le programmeur.
- Il peut sauter par construction à une adresse prédéfinie.

Le MCS251-Interruptions

- C'est cette seconde méthode qui est utilisée pour le MCS251:
  - Le processeur saute automatiquement à l'adresse
    FF:0003h + 8*(le numéro de l'interruption)
  - Soit pour l'interruption 2 (INT1) à l'adresse FF:0013h

- Chaque procédure d'interruption ne dispose donc que de 8 octets... Lorsque la routine sera plus longue on placera dans ces 8 octets un JMP vers un autre lieu de la mémoire.

- C'est le cas de notre système qui par défaut redirige les routines d'interruption vers une zone en mémoire RAM qui elle-même les redirige vers la fonction que vous écrirez.

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:    MOV    R0,#0
5003h:    MOV    R1,#2
5006h :   ADD    R0,R1
```

Déclenchement IT2

Handler standard :

```
FF:0013h:    JMP    00:4013h
```

Handler en RAM :

```
00:4013h:    JMP    00:4100h
```

Routine d'interruption :

```
4100h:    NOP
4101h:    NOP
4102h:    RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:    MOV    R0,#0        Déclenchement IT2
5003h:    MOV    R1,#2
5006h :   ADD    R0,R1
```

Handler standard :

```
FF:0013h:    JMP    00:4013h
```

Handler en RAM :

```
00:4013h:    JMP    00:4100h
```

Routine d'interruption :

```
4100h:    NOP
4101h:    NOP
4102h:    RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:    MOV    R0,#0
5003h:    MOV    R1,#2
5006h :   ADD    R0,R1
```

Handler standard :

```
FF:0013h:    JMP    00:4013h
```

Handler en RAM :

```
00:4013h:    JMP    00:4100h
```

Routine d'interruption :

```
4100h:    NOP
4101h:    NOP
4102h:    RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Ce qui donne :

Programme normal :

```
5000h:      MOV     R0,#0
5003h:      MOV     R1,#2
5006h :     ADD     R0,R1
```

Handler standard :

```
FF:0013h:   JMP     00:4013h
```

Handler en RAM :

```
00:4013h:   JMP     00:4100h
```

Routine d'interruption :

```
4100h:      NOP
4101h:      NOP
4102h:      RETI
```

Le MCS251-Interruptions

- Déclaration en C d'une routine d'interruption :

```
_interrupt(n°interruption) void nomDeLaFonction( void ) {
    /* code de l'interruption */
}
```

exemple :

```
_interrupt(2) void nomDeLaFonction( void ) {
    /* code de l'interruption */
}
```

==> Le compilateur ira automatiquement ajouter à l'adresse 00:4013h l'instruction JMP nomDeLaFonction
==> Le compilateur effectue la sauvegarder de tous les registres.

Le MCS251-Interruptions

- Les interruptions ont aussi des paramètres permettant :
  - De les autoriser ou de les interdire
  - D'influer sur leur priorités des unes par rapport aux autres
    => Une interruption peut être elle-même interrompue par une interruption plus prioritaire. A priorité logicielle égale, c'est la priorité matérielle qui l'emporte.

==> La configuration des interruptions se fait au travers de registres spécifiques : les SFR. Ce sont des adresses mémoires qui correspondent à des registres de configuration du micro-contrôleur ou de ses périphériques.

Le MCS251-Interruptions

- Activer / Bloquer les interruptions :
  - Registres EX0, EX1 => interruptions INT0, INT1
  - Registres ET0, ET1 => interruptions Timer0, Timer1
  - Registre ES => interruptions Port Série
  - Registre EA => toutes les interruptions (vient en complément des autres et non en remplacement)

  - Utilisation en C :

```
ET0 = 1;      // active l'interruption Timer 0
EX0 = 0;      // bloque l'interruption INT0
EA = 1;       // active toutes les interruptions
```

Le MCS251-Interruptions

- Registre de configuration TCON
  - Permet de configurer INT0/1 en mode « front » ou « état »
    - INT0/1 correspond au déclenchement sur un signal externe
    - En mode front, l'interruption ne se déclenche qu'une seule fois lorsque le signal externe est actif.
    - En mode état, l'interruption se déclenche tant que le signal externe est actif.
  - Sur ce même registre nous trouvons des indicateurs IE0/IE1 et TF0/TF1 : ces bits sont positionnés à « 1 » lorsqu'une interruption est détectée et jusqu'à ce qu'elle soit prise en compte.

Le MCS251-Interruptions

- Registre de configuration TCON

B7	B6	B5	B4	B3	B2	B1	B0
TF1		TF0		IE1	IT1	IE0	IT0

- Pour modifier ou lire seulement 1 des bits, il faudra employer les opérations de masquage vues précédemment en cours.

- Exemple : Positionner IT1 à 1 pour demander un fonctionnement sur front :

```
TCON = (TCON | 0x04); // forcer à 1 le bit B2
```

Le MCS251-Interruptions

- Registres IPL0 et IPH0 : gestion des priorités
  - Permet de définit quelles interruptions pourront interrompre d'autres interruptions

  => Par exemple pour rendre un bouton d'arrêt d'urgence plus prioritaire que tout.

  - Les priorités sont définies sur 2 bits donc de 00 (moins prioritaire) à 11 (plus prioritaire)
    - IPL0 est le poids faible de ce nombre
    - IPH0 est le poids fort de ce nombre
    - Le bit de IPx0 correspond à l'interruption configurée

```
=>   IPL0.2 = 1;        // fixe à l'interruption 2 (INT1) la prio 1
     IPH0.2 = 0;        // ...
```

Voir tableau récap page 27

Le MCS251-Interruptions

- Petit QCM de noël ...

  Soit un nombre 8 bits noté b7 b6 b5 b4 b3 b2 b1 b0

  Exemple pour la valeur A8h 1 0 1 0 1 0 0 0

  1-Indiquez l'opérateur et le masque (en hexa) qui permettent

  a – de forcer à 1 les bits b7 b5 b4

  b – permuter les bits b7 b6 b2 b1 b0

  c – forcer à 0 les bits b7 b6 b3 b1

  2 – Sachant que la variable « **inter** » contient l'état des interrupteurs, écrivez en C la ligne de chargement de la variable « **v1** » qui contient la valeur des bits b4 et b3 ; alignée sur les poids faibles.

Le MCS251-Interruptions

- Petit QCM de noël ... correction

1-Indiquez l'opérateur et le masque (en hexa) qui permettent

a – de forcer à 1 les bits b7 b5 b4

OU (ORL) 0xB0

b – permuter les bits b7 b6 b2 b1 b0

XOR (XRL) 0xC7

c – forcer à 0 les bits b7 b6 b3 b1

ET (ANL) 0x3A

– Sachant que la variable « **inter** » contient l'état des interrupteurs, écrivez en C la ligne de chargement de la variable « **v1** » qui contient la valeur des bits b4 et b3 ; alignée sur les poids faibles.

v1 = (inter & 0x18) >> 3; ou v1 = (inter >> 3) & 0x3;

Les Compteurs / Temporisateurs

Les Timers sont des éléments matériels : des périphériques utilisés pour compter des évènements. C'est pour cela que l'on parle de compteurs. Lorsque ces évènements sont du temps, on parle de Temporisateurs.

=> Ils servent à compter de façon précise le temps qui passe pour déclencher périodiquement des traitements. Il permettent de compter des évènements pour déclencher des traitements lorsque, par exemple, un nombre donné de tours de roues ont été effectués, dans une application de compteur de vitesse.

Le MCS251-Timers

Un compteur est en quelque sorte un registre d'un nombre de bits défini qui s'incrémente, automatiquement à la réception d'évènements.

Le compteur / temporisateur indique qu'il a terminé son comptage lorsqu'il déborde, c'est à dire, lorsque son incrémentation le fait repasser à la valeur 0.

Nous pourrons donc choisir le nombre d'évènements à compter en initialisant ce registre à :

sa valeur de débordement – le nombre d'évènements à compter.

Soit pour un compteur de 8 bits devant compter 10 évènements :

Valeur d'initialisation = 256 – 10 = 246

La valeur d'initialisation est placée dans les registres THx et TLx

Le MCS251-Timers

Outre sa valeur d'initialisation les compteurs / temporisateurs utilisent d'autres registres de configuration :

Registre TMOD :	B7	B6	B5	B4	B3	B2	B1	B0
	gate	C/T	M1	M0	gate	C/T	M1	M0
	---------TIMER1--------				--------TIMER0---------			

Ce registre permet de configurer les TIMER1 et 0 :

M0	M1	Permet de choisir le mode de fonctionnement
0	0	Timer sur 13 bits (TLx sur 5 bits + THx sur 8bits) Valeur de débordement : 2^13 (MCS51, peu utilisé)
0	1	Timer sur 16 bits ; débordement 65536
1	0	Timer sur 8 bits avec rechargement automatique Valeur de débordement 256
1	1	Non utilisé dans ce cours.

Le MCS251-Timers

Outre sa valeur d'initialisation les compteurs / temporisateurs utilisent d'autres registres de configuration :

Registre TMOD :	B7	B6	B5	B4	B3	B2	B1	B0
	gate	C/T	M1	M0	gate	C/T	M1	M0
	----------TIMER1--------				-------TIMER0----------			

Ce registre permet de configurer les TIMER1 et 0 :

C/T : Configuration en mode Compteur (1) ou Temporisateur (0)
En mode Temporisateur, le Timer est incrémenté toutes les 12 périodes d'horloge, soit à 8MHz toutes les 1,5 micro-secondes.

En mode Compteur, il compte les évènements extérieurs dont la fréquence maximum est de 24 périodes d'horloge.

Le MCS251-Timers

Outre sa valeur d'initialisation les compteurs / temporisateur utilisent d'autres registres de configuration :

Registre TMOD :	B7	B6	B5	B4	B3	B2	B1	B0
	gate	C/T	M1	M0	gate	C/T	M1	M0
	----------TIMER1--------				-------TIMER0----------			

Ce registre permet de configurer les TIMER1 et 0 :

gate : conditionne le comptage par un signal externe. Nous ne l'utiliserons pas et mettrons toujours 0.

Le MCS251-Timers

Registre TMOD :	B7	B6	B5	B4	B3	B2	B1	B0
	gate	C/T	M1	M0	gate	C/T	M1	M0
	----------TIMER1--------				--------TIMER0----------			

Exemple de configuration du TIMER0 en temporisateur, mode 1 :

```
TMOD = (TMOD & 0xF0) |  0x1;     // sans modifier TIMER1
```

Configuration de TIMER1 en mode compteur, mode 2 :

```
TMOD = (TMOD & 0x0F) | 0x60;    // sans modifier TIMER0
```

Le MCS251-Timers

Registre de contrôle des Timers :

TRx (TR0 ou TR1) : Permet de démarrer (1) / arrêter (0) le timer

TFx (TF0 ou TF1) : Permet lire l'état du Timer ; s'il est à 1 c'est que le Timer à débordé – fini de compter.

Exemple complet : compter 15ms de temporisation avec Timer 0

1. 15ms = 10000 x 1,5us > 2^13 => retenons le mode 1 (2^16)
2. Valeur d'initialisation : 65536 – 10000 = 55536

```
TH0 = ( 55536 >> 8 );
TL0 = ( 55536 & 0xFF );
```

3. Configuration du Timer 0 (Temporisateur mode 1)

```
TMOD = (TMOD & 0xF0) | 0x01;
```

4. Lancement :

```
TF0 = 0;
TR0 = 1;
```

5. Attendre la fin du comptage

```
while ( TF0 == 0 ) ;
TR0 = 0;                    // Arrêt du compteur
```

==> Ici nous effectuons une attente par SCRUTATION ou ATTENTE ACTIVE, le flag de fin est lu tant qu'il n'est pas à 1. Alors seulement on passe à la suite du programme.

==> Il est possible d'utiliser les interruptions à la place.

Le MCS251-Timers

- Timer à rechargement automatique :
  - Dans les autres modes, lorsqu'un timer finit de compter il dépasse sa capacité et le registre prend pour valeur 0. Le timer va continuer à compter des évènements à partir de cette nouvelle valeur. IL DOIT DONC ÊTRE REINITIALISE A CHAQUE FOIS.

  - Pour éviter cette étape et permettre un comptage plus régulier (très utile pour les transmissions de données) le mode à rechargement automatique permet, lors du dépassement de capacité, de remettre dans le registre la valeur initiale stockée dans le registre TH0.

Le MCS251-Timers

- Exemple de comptage de temps en mode 2 (15us) (timer 1)
  15us = 10 * 1,5 us
  TH0 = TL0 = 256 – 10 = 246

```
TH0 = TL0 = 246;
TMOD = (TMOD & 0x0F ) | 0x20
TF1 = 0
TR1 = 1
while(1) {
    while(TF1 == 0);        // Attente de 15us
    TF1=0;
    // action
}
```

Le MCS251-Timers

- Exercice : faire clignoter des LED toutes les secondes

1. Ecrire une fonction, utilisant un Timer, comptant 1 seconde.
2. Ecrire le programme principal.

3. Ecrire le même programme sous interruption.

Le MCS251-Timers

- Exercice : faire clignoter des LED toutes les secondes

```
void sleep(void) {
    int i;
    TMOD = (TMOD & 0xF0) | 0x01;      // Timer mode 1
    for ( i = 0 ; i < 20 ; i++ ) {
        TH0 = ( 65536-33333 ) >> 8;          // 50ms
        TL0 = (unsigned char)( 65536-33333 );
        TF0 = 0;
        TR0 = 1;
        while(TF0==0);
        TF0 = 0;
        TR0 = 0;
    }
}
```

Le MCS251-Timers

- Exercice : faire clignoter des LED toutes les secondes

```
void main(void) {
    while(1) {
        ecrire(P4,0xFF);
        sleep();
        ecrire(P4,0x00);
        sleep();
    }
}
```

Le MCS251-Timers

• Exercice : version avec interruption

```
int pass, val;
void main(void) {
        TMOD=(TMOD&0xF0) | 0x1;            // première init.
        TH0=(65536-33333) >> 8;            // pour lancer la
        TL0=(unsigned char)(65536-33333);  // première itération
        TF0=0;                             // du clignotement
        TR0=1;
        pass = 0;
        val = 0;
        ET0=1;EA=1;                        // activation des int.
        while(1);                          // boucle infinie
}
```

Le MCS251-Timers

- Exercice : version avec interruption

```
_interrupt(1) void clignote(void) {
        TH0=(65536-33333) >> 8;                 // reinit du timer
        TL0=(unsigned char)(65536-33333);       // pour l'iteration
        TF0=0;                                  // suivante
        if ( pass == 20 ) {
            val ^= 0xFF;                    // clignote 1 coup
            ecrire(P4, val);                    // sur 20 = 1seconde
            pass = 0;
        } else pass ++;
}
```