Programmation système : Test et positionnement de bits

Module Info 2

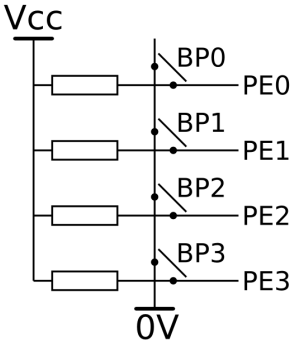
Sebastien.Kramm@univ-rouen.fr

IUT GEII Rouen

2013-2014

Pourquoi tester un bit ?

- Il arrive qu'on souhaite connaître l'état d'un bit **indépendamment** des autres.
- Exemple : clavier 4 touches connecté au port E. On veut connaître l'état d'une touche quel que soit l'état des autres.



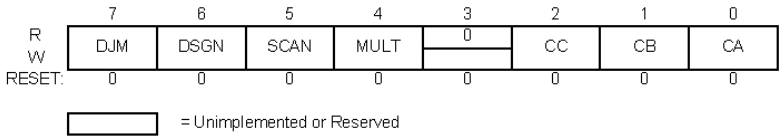
- Si BP0 appuyé et les autres au repos : on lit $0E sur le port :

```
1       ldaa PORTE
2       cmpa #%00001110
3       beq  BP0_APPUYE
4       ...
```

- Si BP0 et BP1 appuyés, on lit $0C ⇒ le prog. ci dessus ne fonctionne plus.

Pourquoi positionner un bit ?

- Allumer une Del sur un port sans modifier les autres.
- Positionner un bit de contrôle dans un registre sans modifier les autres bits.
  Par exemple : *positionner le bit* `MULT` *à 1 dans le registre ATDCTL5.*



ATD Control Register 5 (ATDCTL5)

Principe de base

- Toutes ces manipulations (test et positionnement) sont basées sur les opérateur booléens ET et OU.

- Opérateur OU (*OR*)

```
  0101
+ 0011
-------
  0111
```

- Opérateur ET (*AND*)

```
  0101
. 0011
-------
  0001
```

Sommaire

Forçage à 1

- Pour faire passer un bit à 1 : opérateur OU.
- Exemple : on veut faire passer le bit 5 d'un octet à 1.
  ⇒ On utilise un **masque** valant %00100000 (=32)

```
    0000 1111          1111 0000
 +  0010 0000       + 0010 0000
---------------    ---------------
    0010 1111          1111 0000
```


Forçage à 0

- Pour faire passer un bit à 0 : opérateur ET.
- Exemple : on veut faire passer le bit 5 d'un octet à 0.
  ⇒ On utilise un **masque** valant %11011111

```
    0000 1111          1111 0000
 .  1101 1111       . 1101 1111
---------------    ---------------
    0000 1111          1101 0000
```

- Remarque : on fournit un masque avec des '0' là ou on veut forcer le bit.

Implémentation en assembleur

- On peut utiliser les instructions `and` et `or`
- Par exemple :

```
1    ldaa  adresse
2    anda  #%11110111 ;forçage bit 3 à 0
3    oraa  #%10000000 ;forçage bit 7 à 1
4    staa  adresse
```

- Question : le contenu de "`adresse`" vaut $68. Quelle sera sa valeur après l'exécution des lignes ci-dessus : `________`
- Pas très commode :
  - utilise un des 2 accumulateurs,
  - le masque n'est pas le même pour mettre à 0 et pour mettre à 1.

  ⇒ On préfère utiliser les instructions dédiées `bset` et `bclr`.

Instructions dédiées

- Le processeur 9s12 dispose de 2 instructions de manipulation de bits :
  - `bset` : positionnement de bits à 1 (*Bits Set*)
  - `bclr` : positionnement de bits à 0 (*Bits Clear*)
- Avantages :
  - pas d'utilisation d'accumulateur,
  - masque identique pour forcer à 0 et à 1.
- Exemples

```
1   bset  adresse,%100 ; forcage bit 2 à 1
2   bclr  adresse,%100 ; forcage bit 2 à 0
```


Utilisation de symboles

- Afin d'améliorer la lisibilité, il est préférable de prédéfinir des masques (une seule fois, dans un fichier séparé).

```
1   BIT0 equ %00000001 ( = 1 = $01 )
2   BIT1 equ %00000010 ( = 2 = $02 )
3   BIT2 equ %00000100 ( = 4 = $04 )
4     ...
5   BIT7 equ %10000000 ( = 128 = $80 )
```

- On peut alors écrire :
  - pour forcer le bit 2 de 'adresse' à **1** : `bset adresse,BIT2`
  - pour forcer le bit 2 de 'adresse' à **0** : `bclr adresse,BIT2`
- On pourra même combiner des masques :

`bclr adresse,BIT2+BIT3` ↔ `bclr adresse,$0c`

$04+$08=$0C (0000.1100)

En langage C

- En C , pas d'instructions de traitement de bit : on doit utiliser les opérateurs binaires ET et OU.
- Exemples :
  - forçage à 1 du bit 3 du port A : `PORTA = PORTA | 0x08;` (0000 1000 = $08)
  - forçage à 0 du bit 2 du port B : `PORTB = PORTB & 0xfb;` (1111 1011 = $fb)
- Inconvénient : le masque n'est pas le même suivant qu'on force à 0 ou à 1...

Utilisation de symboles en C

- Afin d'améliorer la lisibilité, on utilisera des masques prédéfinis :

```
1   #define BIT0 0x01
2   #define BIT1 0x02
3   #define BIT2 0x04
4     ...
5   #define BIT7 0x80
```

- On peut alors écrire :
  - Forçage à 1 : `a = a | BIT3;`
  - Forçage à 0 : `a = a & ~ BIT3;`
- Rappel : l'opérateur unaire "complément à 1" ("barre") s'écrit ~
  Exemple :
  `char a = 0xFA; char b = ~a;` ⇒ b vaudra 5

Sommaire

Test de bits en assembleur

- Le processeur dispose de 2 instructions dédiées
  - `brset` : *Branch if bits Set*
    ⇒ Se branche si les bits indiqués à l'adresse spécifiée sont à **1**.
  - `brclr` : *Branch if bits Clear*
    ⇒ Se branche si les bits indiqués à l'adresse spécifiée sont à **0**.
- Il faut spécifier 3 champs :
  1. A quelle adresse se situe la valeur à tester ?
  2. Quel est le bit testé (masque) ?
  3. A quel adresse doit-on se brancher ?
- Exemples :
  `brset $1234,%100,LABAS`
  Ou mieux :
  `brset ADRESSE,BIT2,LABAS`

Test de bits en C

- On utilise **toujours** l'opérateur **ET**, on réalise un **masquage**.
- Principe : "cacher" les bits qu'on ne veut pas voir
- Exemple : test du bit 5 de la variable 'a' de type 'char' :
  ⇒ On utilise le masque 0010 0000.

```
   0000 1111 (valeur)        1111 0000
 .  0010 0000 (masque)     .  0010 0000
--------------            --------------
   0000 0000                 0010 0000
      ⇒= 0                      ⇒≠ 0
```


En pratique

- Pour exécuter un bloc d'instructions si le bit 5 d'une variable est à '0' :

```
1 if( (a & BIT5) == 0)
2     { bloc; }
```

- Pour exécuter un bloc d'instructions si le bit 5 d'une variable est à '1' :

```
1 if( (a & BIT5) != 0)
2     { bloc; }
```

- Attention aux parenthèses !
  `( (x & y) == 0 )` est différent de `( x & y == 0 )`
- Le principe sera le même pour les autres structures de test, par exemple :
  "Attendre que le bit 2 du PORTB passe à 0"

```
1 while( (PORTB&BIT2) != 0 ) // tant que...
2   ;                        // faire : rien
```

Evaluation des expressions en C

- En C, un test sera
  - VRAI si l'expression est différente de zéro
  - FAUX si l'expression est égale à zéro
- Exemple : if(a) sera equivalent à if(a !=0)
  - VRAI si a $\neq 0$
  - FAUX si a $= 0$

Langage C : Attention

- Ne pas confondre & et &&
- Exemple avec le ET :

```
if( a & b )
```
⇒
```
if( ( a & b ) != 0 )
```

```
if( a && b )
```
⇒
```
if( a != 0 && b != 0 )
```
⇒
```
if( a != 0 )if ( b != 0 )
```

- Exemple numérique : a=1, b=2

- l'expression `( a && b )` est **vraie** :
  a=1 (vrai), b=2 (vrai)
  ⇒ vrai & vrai = vrai

- l'expression `( a & b )` est **fausse** :

```
   01
.  10
-------
   00 => faux
```


Sommaire

Inversion de bits

- Il arrive qu'on souhaite inverser l'état d'un bit (clignotement d'une del)
- Première approche : le tester :
  si le bit est à 0, on le met à 1, sinon à on le met à 0
- En assembleur :

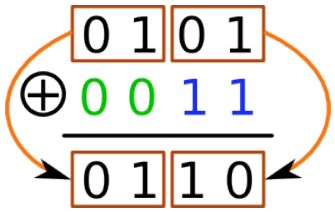
```
1         brclr ADRESSE,BITX,A_ZERO
2         bclr  ADRESSE,BITX
3         bra   SUITE
4  A_ZERO bset  ADRESSE,BITX
5  SUITE    ...
```

- En C :

```
1  if( (var & BITX) == 0 )
2     var = var | BITX;
3  else
4     var = var & ~ BITX;
```

Inversion de bits : OU Exclusif

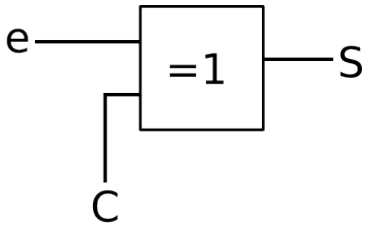
- Table de vérité :



- Propriété (x est un bit)

$$x \oplus 0 = x$$
$$x \oplus 1 = \overline{x}$$

- Utilisation en inverseur commandé



C=0 => S=e
C=1 => S=$\overline{e}$

(commande)

Implémentation OU Exclusif

- Pour inverser un (des) bits, on utilise un masque :

$$\begin{array}{r} 0\,0\,0\,0\;\;1\,1\,1\,1 \\ \oplus 0\,0\,1\,0\;\;0\,0\,0\,0 \\ \hline 0\,0\,1\,0\;\;1\,1\,1\,1 \end{array} \qquad \begin{array}{r} 1\,1\,1\,1\;\;0\,0\,0\,0 \\ \oplus 0\,0\,1\,0\;\;0\,0\,0\,0 \\ \hline 1\,1\,0\,1\;\;0\,0\,0\,0 \end{array}$$

- Implémentation

- En assembleur 9s12

```
1    ldaa ADRESSE
2    eora #BIT5
3    staa ADRESSE
```

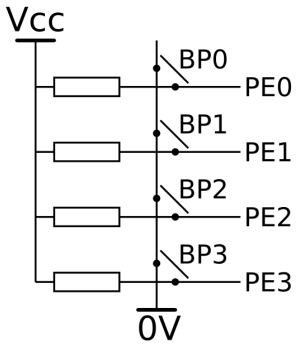
- En C

```
1    a = a ^ BIT5;
```


Exemple d'application : clavier

- En assembleur 9s12

```
1  DEBUT brclr PORTE,BIT0,BP0
2        brclr PORTE,BIT1,BP1
3        bra   DEBUT
4
5  BP0   ; on traite l appui du BP 0
6         ...
7        bra   DEBUT
8
9  BP1   ; on traite l appui du BP 1
10        ...
11       bra   DEBUT
```



Exemple d'application : clavier

- En C :

```
1  int main()
2  {
3      while(1)
4      {
5          if( ( PORTE & BIT0 ) == 0 )
6              traitement_BP0();
7          if( ( PORTE & BIT1 ) == 0 )
8              traitement_BP1();
9      }
10      ...
11 }
```

- `traitement_BP0()` et `traitement_BP1()` sont les fonctions de traitement, appelés en cas d'appui sur les BP.