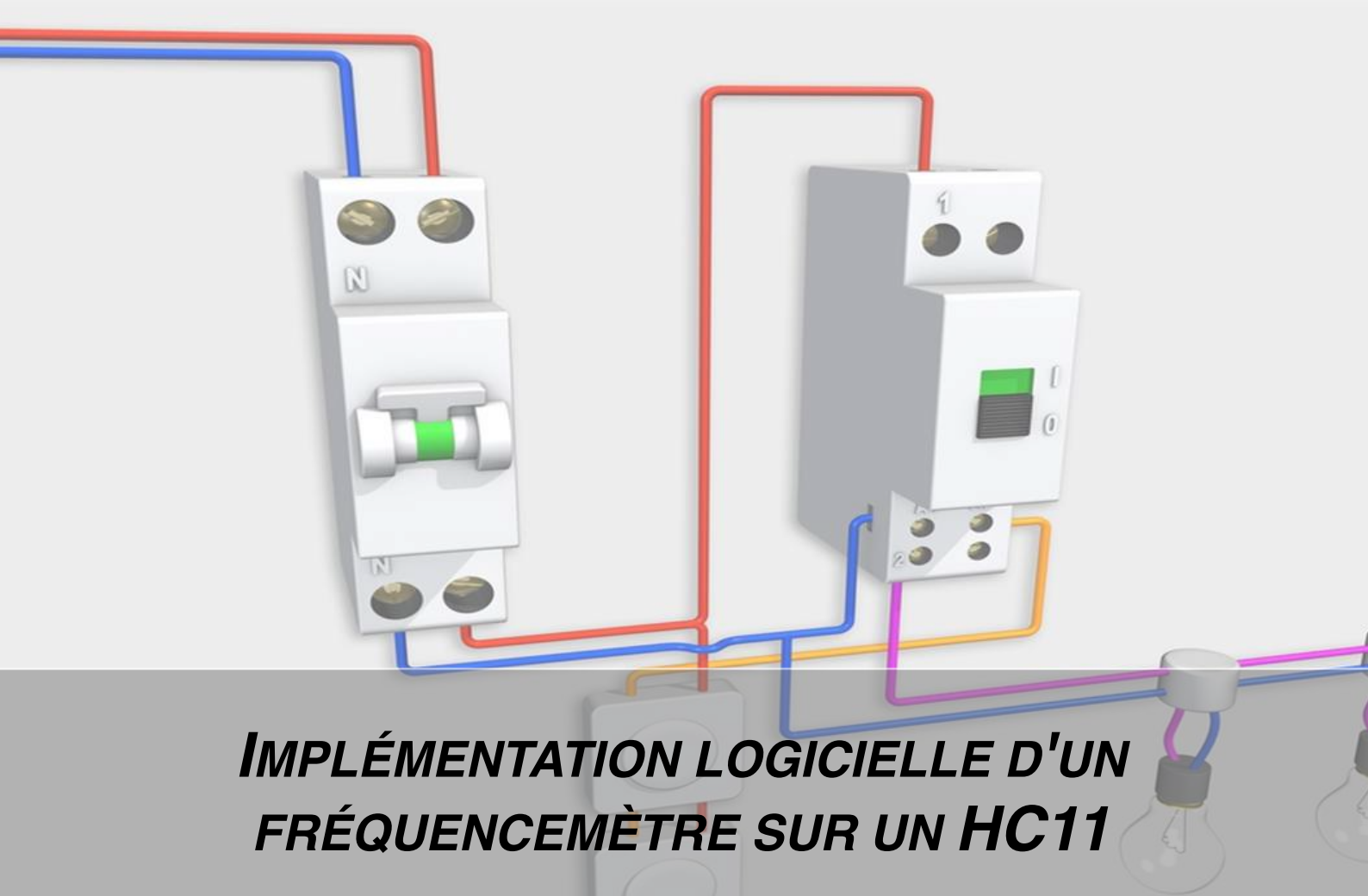




# ***IMPLÉMENTATION LOGICIELLE D'UN FRÉQUENCEMÈTRE SUR UN HC11***

***PROJET - SYSTÈMES NUMÉRIQUES***

***3 SÉANCES***

***BOUAZIZ AXEL  
CLAIRVILLE STEPHEN***

***L2 - GROUPE B***

## Sommaire :

---

|                                                                                |                  |
|--------------------------------------------------------------------------------|------------------|
| Introduction                                                                   | 3                |
| <hr/>                                                                          |                  |
| <b><i>Les différentes méthodes de programmation de mesure de fréquence</i></b> | <b><i>4</i></b>  |
| <hr/>                                                                          |                  |
| <b><i>I/ PREMIÈRE MÉTHODE : PAR SCRUTATION, POLLING</i></b>                    | <b><i>4</i></b>  |
| <hr/>                                                                          |                  |
| 1.1. QU'EST CE QUE LE POLLING ?                                                | 4                |
| <hr/>                                                                          |                  |
| 1.2. BUT DES MANIPULATIONS                                                     | 4                |
| <hr/>                                                                          |                  |
| 1.3. APPLICATION DE LA MÉTHODE ET MESURES                                      | 4                |
| <hr/>                                                                          |                  |
| 1.3.1. Les cycles                                                              |                  |
| 1.3.2. Durée d'un processus                                                    |                  |
| 1.3.3. Réalisation de la mesure                                                |                  |
| <hr/>                                                                          |                  |
| <b><i>II/ DEUXIÈME MÉTHODE : PAR INTERRUPTION</i></b>                          | <b><i>9</i></b>  |
| <hr/>                                                                          |                  |
| 2.1. QU'EST CE QUE LA MÉTHODE DITE "PAR INTERRUPTION" ?                        | 9                |
| <hr/>                                                                          |                  |
| 2.2. APPLICATION DE LA MÉTHODE ET MESURES                                      | 9                |
| <hr/>                                                                          |                  |
| 2.2.1. Interruption du programme                                               |                  |
| 2.2.2. Mesure d'une période                                                    |                  |
| <hr/>                                                                          |                  |
| <b><i>Méthode d'affichage sur une maquette 68HC11 en assembleur</i></b>        | <b><i>14</i></b> |
| <hr/>                                                                          |                  |
| <b><i>I/ AFFICHAGE DE CARACTÈRES</i></b>                                       | <b><i>14</i></b> |
| <hr/>                                                                          |                  |
| <b><i>II/ CONVERSION D'UN ENTIER SUR 1 OU 2 OCTETS</i></b>                     | <b><i>16</i></b> |
| <hr/>                                                                          |                  |
| 2.1. CONVERSION D'UN ENTIER SUR 1 OCTET                                        | 16               |
| <hr/>                                                                          |                  |
| 2.2. CONVERSION D'UN ENTIER SUR 2 OCTETS                                       | 18               |
| <hr/>                                                                          |                  |
| <b><i>Communication par liaison série asynchrone</i></b>                       | <b><i>20</i></b> |
| <hr/>                                                                          |                  |
| <b><i>Réalisation d'un fréquencemètre avec affichage LCD</i></b>               | <b><i>24</i></b> |
| <hr/>                                                                          |                  |
| Conclusion                                                                     | 24               |
| <hr/>                                                                          |                  |

## ***Introduction :***

---

Nous voici confrontés à notre premier projet de Systèmes Numériques. Nous allons nous porter sur l'implémentation logicielle d'un fréquencemètre sur un HC11. Un HC11 est un compteur 16 bits, que l'on nommera dans le projet TCNT, qui permettra de dater l'arrivée des fronts de fréquence.

Pour cela, nous devons dans un premier temps comparer les différentes méthodes de programmation d'une mesure de fréquence. Nous implémenterons ensuite l'affichage sur un écran LCD d'une maquette 68HC11 en assembleur. On apprendra aussi à communiquer avec le port série pour finalement réaliser un fréquencemètre avec un affichage LCD.

Pour se faire, nous utiliserons le logiciel AxIDE, que nous apprenons peu à peu à maîtriser, au cours des précédentes séances de TP qui se sont avérées très formatrices.

### I/ PREMIÈRE MÉTHODE : PAR SCRUTATION, LE POLLING

#### 1.1. QU'EST CE QUE LE POLLING ?

En Anglais, **Polling** signifie "sondage". Dans notre cas, le principe est simple, on scrute en permanence l'arrivée du premier front du signal. On attend un second front de la même nature, et une fois ce front arrivé, on en déduit la période.

#### 1.2. BUT DES MANIPULATIONS

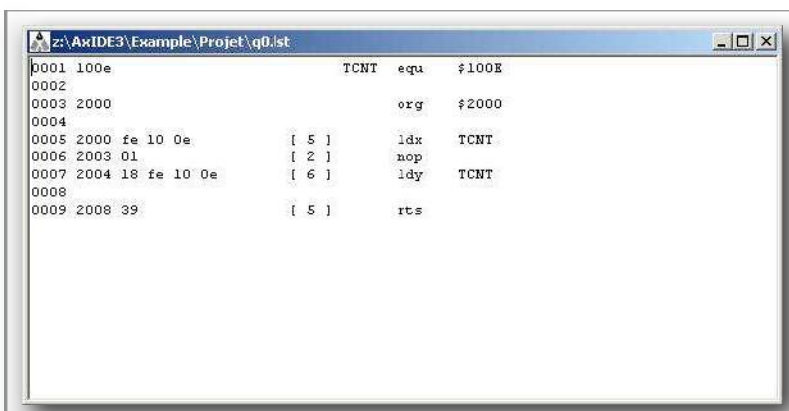
Avec la méthode dite de Polling, nous allons devoir compter les fronts à l'aide du TCNT, qui est le compteur 16 bits HC11. On va donc chercher à savoir si les cycles de l'horloge du CPU et du TCNT sont synchronisés. Ensuite, si c'est le cas, nous pourrions définir la durée d'un processus et bien entendu en déduire sa période.

#### 1.3. APPLICATION DE LA MÉTHODE ET MESURES

##### 1.3.1. Les cycles

On commence donc par s'intéresser aux cycles. La documentation nous permet de savoir que la **durée d'un cycle est de 0,5 microsecondes** et que la fréquence du TCNT est définie en fonction du CPU. A l'appel du TCNT on choisit si l'on divise la fréquence par 2, 4, 8, ou plus... en modifiant les valeurs de PR1 et PR0 dans le registre.

Dans notre cas, on va simplement utiliser la division par défaut, soit la division par 1 : le **cycle du TCNT est donc égal au cycle du CPU**.



Dans notre cas, on fait une mesure du nombre de cycles entre 2 chargements. On fait commencer le programme à l'adresse 2000 et on fait deux "load" de la valeur du TCNT.

Le compteur va donc continuer à s'incrémenter entre la mémorisation du registre x et y. En faisant la différence, on a donc le nombre d'incrémentations qui se sont écoulées.

On utilise l'option de comptage (entre crochets, nous avons le nombre de cycles pour chaque instruction). Entre la fin des deux chargements, il y a **8 cycles** (2 pour le "nop" et 6 pour le "ldy"), soit environ 4 microsecondes.

Vérifions à présent ce résultat en comparant les valeurs présentes dans les registres X et Y, en réalisation simplement la différence  $Y - X$ , on calcule ainsi le nombre de cycles s'écoulant entre les 2 chargements. Pour cela, on va utiliser la fonction trace "t" à partir de l'adresse 2000.

On obtient alors :

$X = BBA5$

$Y = BBAD$

$Y - X$  est bien égal à 8 donc le fonctionnement est vérifié !

### 1.3.2. Durée d'un processus

On sait à présent mesurer la durée d'une instruction sachant que le cycle du TCNT est égal au cycle du CPU et que la durée d'un cycle est égale à 0,5 microseconde. On va donc à présent s'intéresser à la durée d'un processus.

On note que le 68HC11 ne prévoit pas de soustraction pour les registres X et Y. On va donc avoir besoin de réaliser un programme qui va venir une première fois stocker la valeur du TCNT dans les deux premiers octets de l'adresse 2000 et qui va dans un second temps charger de nouveau la valeur de TCNT afin de la comparer à la valeur précédemment stockée. Le résultat sera alors lui stocké dans les deux octets suivants à l'adresse 2000.

```

0001 100e          TCNT equ $100E
0002
0003 2000          org $2000
0004 2000          initial rmb 2
0005 2002          delta rmb 2
0006
0007 2050          org $2050
0008
0009 2050 fc 10 0e  [ 5 ] ldd TCNT
0010 2053 fd 20 00  [ 5 ] std initial
0011
0012              Processus;
0013
0014 2056 fc 10 0e  [ 5 ] ldd TCNT
0015 2059 b3 20 00  [ 6 ] subd initial
0016 205c fd 20 02  [ 5 ] std delta
0017
0018 205f 39        [ 5 ] rts
  
```

On voit clairement qu'entre les deux opérations de chargement, il y a **10 cycles**.

On va regarder si cette valeur est bien présente dans le résultat de la soustraction, qui, on le rappelle, est stocké sur 2 octets aux adresses 2002 et 2003.

```

P-2050 Y-2156 X-214E A-00 B-0A C-D0 S-0041
>md 2000

2000 B7 E6 00 0A 18 FE 10 0E 39 F8 53 FB 3E 37 9B A7      9 S >7
2010 1D 37 F9 EF F9 FF FF AD E5 E6 F5 FB 7F FB ED AB      7
2020 9A 9E 57 ED D6 9F 53 61 F9 FE FF E7 FF C1 67 FF      W Sa g
  
```

On trouve alors 000A, qui représente 10 en hexadécimal. On obtient bien le résultat souhaité.

(On remarque par la même occasion qu'il y a une erreur de 5 cycles. Explication : le comptage a lieu entre les deux appels de TCNT, et par conséquent le temps de l'instruction "std initial" est en trop, soit une erreur de 5 cycles !).

### 1.3.3. Réalisation de la mesure

Nous passons à présent à la réalisation de la mesure. Pour cela, il est essentiel de réaliser un travail préliminaire qui consiste à rechercher les **registres acteurs pour les entrées PA0, PA1 et PA2**.

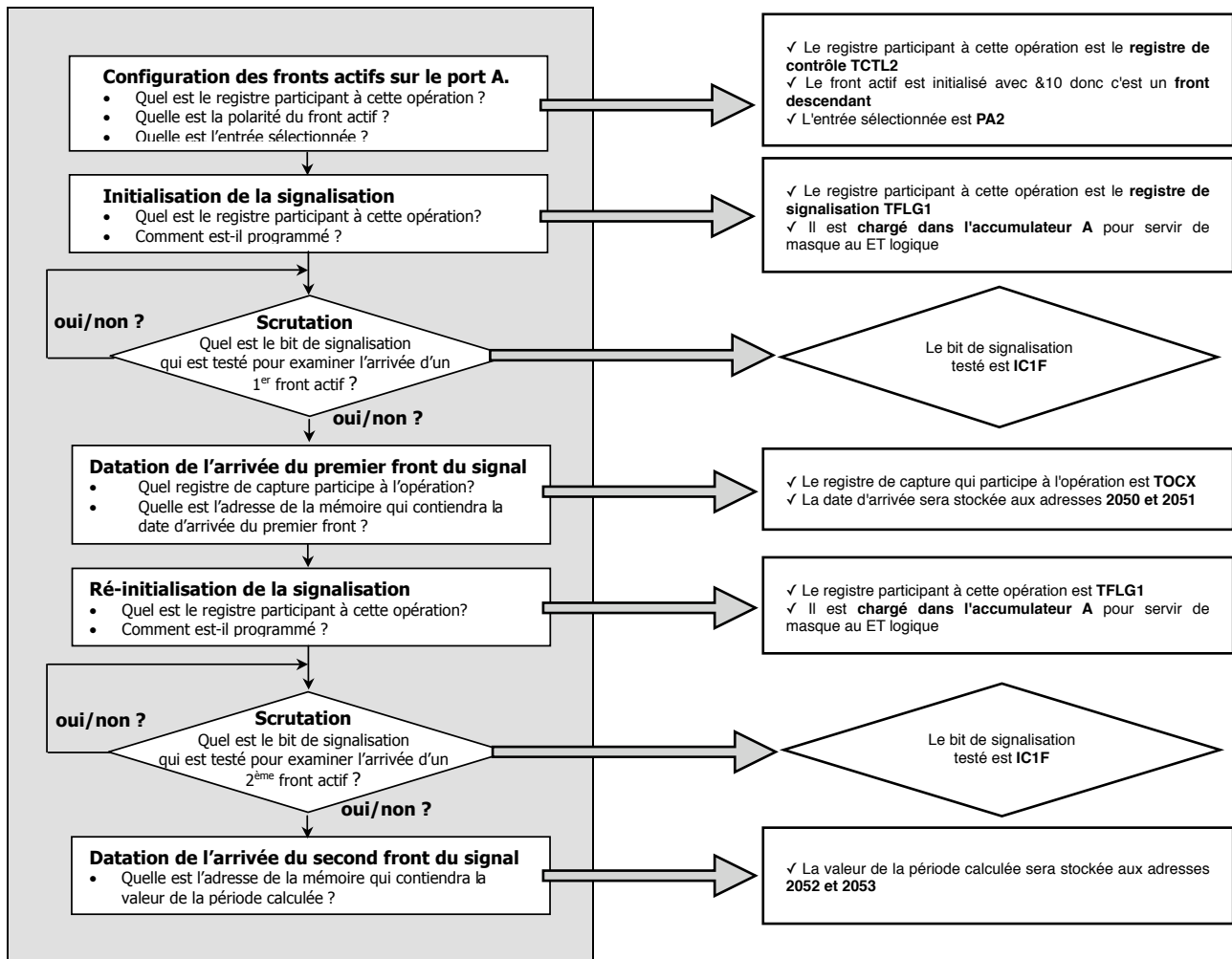
On obtient :

- registres de PA2 : TFLG avec le bit IC1F, TIC1, TMSK1 avec le bit IC1I
- registres de PA1 : TFLG avec le bit IC2F, TIC2, TMSK1 avec le bit IC2I
- registres de PA0 : TFLG avec le bit IC3F, TIC3, TMSK1 avec le bit IC3I

Une fois que les registres ont été sélectionnés, on utilise le **programme** suivant (les lignes à compléter et les commentaires du programmes sont ci-dessous) :

|               |             |               |                                                                                                   |
|---------------|-------------|---------------|---------------------------------------------------------------------------------------------------|
| <b>TIC1</b>   | <b>equ</b>  | <b>\$1010</b> | <i>on assigne l'adresse 1010 au label TIC1</i>                                                    |
| <b>TFLG1</b>  | <b>equ</b>  | <b>\$1023</b> | <i>on assigne l'adresse 1023 au label TFLG1</i>                                                   |
| <b>TCTL2</b>  | <b>equ</b>  | <b>\$1021</b> | <i>on assigne l'adresse 1021 au label TCTL2</i>                                                   |
|               | <b>org</b>  | <b>\$2050</b> | <i>adresse de repère des déclarations</i>                                                         |
| <b>front</b>  | <b>rmb</b>  | <b>2</b>      | <i>on réserve 2 octets assignés au label front</i>                                                |
| <b>period</b> | <b>rmb</b>  | <b>2</b>      | <i>on réserve 2 octets assignés au label period</i>                                               |
| <b>ED1ba</b>  | <b>fcb</b>  | <b>\$10</b>   | <i>on définit la constante 10 au label ED1ba</i>                                                  |
|               | <b>org</b>  | <b>\$2100</b> | <i>adresse de repère du début du programme</i>                                                    |
|               | <b>ldab</b> | <b>ED1ba</b>  | <i>on range le contenu de ED1ba dans l'accumulateur b</i>                                         |
|               | <b>stab</b> | <b>TCTL2</b>  | <i>on range le contenu de l'accumulateur b dans TCTL2</i>                                         |
|               | <b>ldaa</b> | <b>TFLG1</b>  | <i>on range le contenu de TFLG1 dans l'accumulateur a</i>                                         |
|               | <b>anda</b> | <b>#\$04</b>  | <i>on fait un ET logique avec la constante 4 et on stocke le résultat dans a</i>                  |
|               | <b>staa</b> | <b>TFLG1</b>  | <i>on range le contenu de l'accumulateur a dans TFLG1</i>                                         |
| <b>lire1</b>  | <b>ldaa</b> | <b>TFLG1</b>  | <i>on déclare le label "lire1" et on range le contenu de TFLG1 dans l'accumulateur a</i>          |
|               | <b>anda</b> | <b>#\$04</b>  | <i>on fait un ET logique avec la constante 4 et on stocke le résultat dans a</i>                  |
|               | <b>beq</b>  | <b>lire1</b>  | <i>on reste dans la boucle tant que c'est égal à 0</i>                                            |
| <b>suite1</b> | <b>ldd</b>  | <b>TIC1</b>   | <i>on déclare le label "suite1" et on range la valeur de l'horloge TIC1 dans l'accumulateur d</i> |
|               | <b>std</b>  | <b>front</b>  | <i>on range le contenu de d dans front</i>                                                        |
|               | <b>ldaa</b> | <b>TFLG1</b>  | <i>on range le contenu de TFLG1 dans l'accumulateur a</i>                                         |
|               | <b>anda</b> | <b>#\$04</b>  | <i>on fait un ET logique avec la constante 4 et on stocke le résultat dans a</i>                  |
|               | <b>staa</b> | <b>TFLG1</b>  | <i>on range le contenu de l'accumulateur a dans TFLG1</i>                                         |
| <b>lire2</b>  | <b>ldaa</b> | <b>TFLG1</b>  | <i>on déclare le label "lire2" et on range le contenu de TFLG1 dans l'accumulateur a</i>          |
|               | <b>anda</b> | <b>#\$04</b>  | <i>on fait un ET logique avec la constante 4 et on stocke le résultat dans a</i>                  |
|               | <b>beq</b>  | <b>lire2</b>  | <i>on reste dans la boucle tant que c'est égal à 0</i>                                            |
| <b>suite2</b> | <b>ldd</b>  | <b>TIC1</b>   | <i>on déclare le label "suite2" et on range la valeur de l'horloge TIC1 dans l'accumulateur d</i> |
|               | <b>subd</b> | <b>front</b>  | <i>on fait la soustraction de d avec la valeur rangée dans front</i>                              |
|               | <b>std</b>  | <b>period</b> | <i>on range le résultat dans period</i>                                                           |
|               | <b>rts</b>  |               | <i>fin du programme</i>                                                                           |

Maintenant le programme terminé et commenté, remplissons l'**organigramme** en répondant aux questions.



On assemble à présent le programme :

```

0001 1010          TCTL equ  #1010
0002 1023          TFLG1 equ #1023
0003 1021          TCTL2 equ #1021
0004
0005 2050          org  #2050
0006 2050          front rmb 2
0007 2052          periodrmb 2
0008 2054 10      EDiba fcb #10
0009 2100          org  #2100
0010 2100 f6 10 54 [ 4 ] ldab EDiba
0011 2103 f7 10 21 [ 4 ] stab TCTL2
0012 2106 b6 10 23 [ 4 ] ldab TFLG1
0013 2109 84 04    [ 2 ] anda #04
0014 210b b7 10 23 [ 4 ] staa TFLG1
0015
0016 210e b6 10 23 [ 4 ] lire1 ldab TFLG1
0017 2111 84 04    [ 2 ] anda #04
0018 2113 27 f9    [ 3 ] beq  lire1
0019
0020 2115 fc 10 10 [ 5 ] suite2l1dd TCTL
0021 2118 fd 20 50 [ 5 ] std  front
0022
0023 211b b6 10 23 [ 4 ] ldab TFLG1
0024 211e 84 04    [ 2 ] anda #04
0025 2120 b7 10 23 [ 4 ] staa TFLG1
0026
0027 2123 b6 10 23 [ 4 ] lire2 ldab TFLG1
0028 2126 84 04    [ 2 ] anda #04
0029 2128 27 f9    [ 3 ] beq  lire2
0030
0031 212a fc 10 10 [ 5 ] suite2l1dd TCTL
0032 212d b3 20 50 [ 6 ] subd  front
0033
0034 2130 fd 20 52 [ 5 ] std  period
0035
0036 2133 39       [ 5 ] rts
  
```

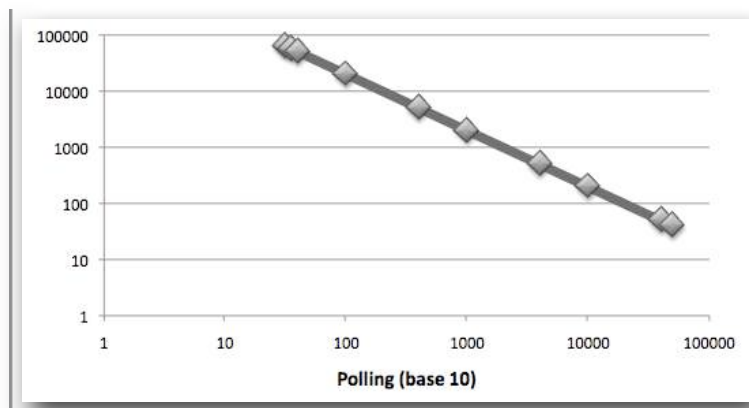
On compte **29 cycles** entre les deux fins de chaque chargement du registre D si la boucle est exécutée une seule fois.

Le temps minimum est donc de 29 cycles, on passe forcément au moins une fois dans chaque boucle.

On teste alors la méthode avec différentes fréquences en cherchant les limites, pour des fréquences allant de 50 Hz à 50 kHz (on a rajouté des fréquences en dessous de 40 Hz pour justifier une seconde fois la limite en BF).

| f (Hz) | Polling (hexadécimal) | Polling (base 10) |
|--------|-----------------------|-------------------|
| 31,35  | F927                  | 63783             |
| 35     | DEA0                  | 56992             |
| 40     | C347                  | 49991             |
| 100    | 4E18                  | 19992             |
| 400    | 1387                  | 4999              |
| 1k     | 07CE                  | 1988              |
| 4k     | 01F4                  | 500               |
| 10k    | 00C8                  | 200               |
| 40k    | 0032                  | 50                |
| 50k    | 0028                  | 40                |

On reporte les mesures effectuées dans un **repère log-log**. La courbe obtenue est la suivante :



La courbe obtenue est une droite décroissante. En effet, la période étant inversement proportionnelle à la fréquence, quand la fréquence augmente, la valeur de la période diminue.

On fait une vérification rapide avec la première mesure :  
 $63783 \text{ cycles} \times \text{durée d'un cycle} = 0,0318 \text{ s}$ , soit une fréquence de 31 Hz.  
 Le fonctionnement du fréquencemètre est donc ainsi vérifié.

La **limite en HF est de 70 kHz** car le nombre minimum de cycles est de 29, c'est la durée minimum de traitement.

La **limite en BF est 31 Hz** car on dépasse le taux, on compte trop de fois et du coup on remet la valeur du registre D à 0.

## II/ DEUXIÈME MÉTHODE : PAR INTERRUPTION

---

### 2.1. QU'EST CE QUE LA MÉTHODE DITE "PAR INTERRUPTION" ?

---

Le principe de la méthode est de créer une notion "d'événementiel". Le programme est interrompu lors de l'arrivée d'un premier front et il déduit automatiquement la période à l'arrivée du second front.

### 2.2. APPLICATION DE LA MÉTHODE ET MESURES

---

Pour réaliser nos mesures, nous allons utiliser les registres TIC1, TFLG1, TMSK1 et TCTL2, soit au final l'entrée PA2. On va dans un premier temps commencer par apprendre à utiliser l'interruption pour ensuite appliquer la méthode

#### **2.2.1. Interruption du programme**

Notre programme tourne dans la **boucle infinie** suivante :

```
boucle  ldd    #$0000 ;
        std    $2000
        bra    boucle ;
```

Dans cette boucle, on réitère le chargement du contenu de l'adresse 0.

Cependant, on crée un événement qui va venir interrompre cette boucle :

```
boucle  sei          ;

        lds    #$0200

        ldab   ED1ba
        stab   TCTL2

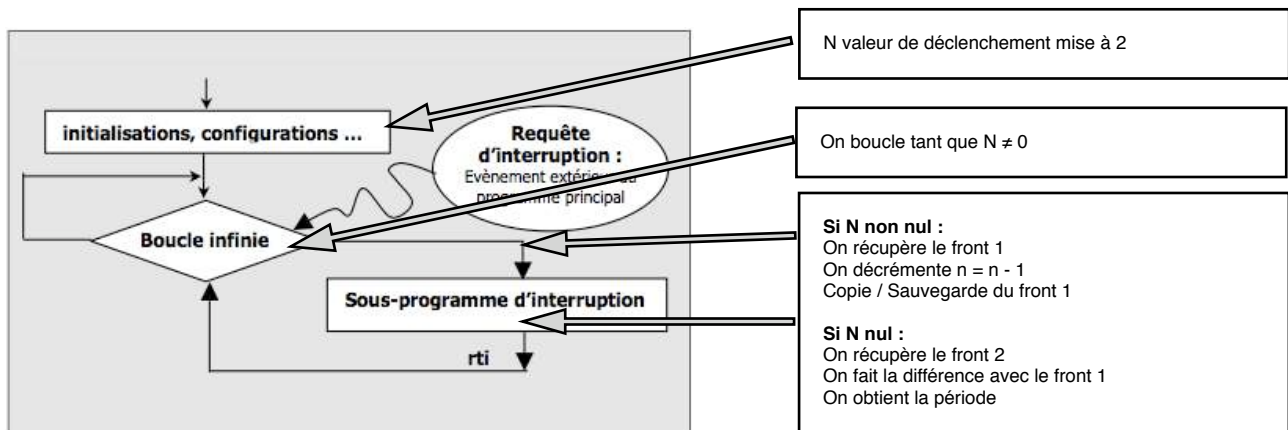
        ldaa   TFLG1
        anda   #$04
        staa   TFLG1

        ldaa   #$04  ;
        staa   TMSK1 ;
        cli          ;
```

Le code ci-dessous est l'**événement d'interruption**. Les fonctions "sei" et "cli" sont les fonctions d'activation et de désactivation de l'option d'interruption. Elles modifient la valeur du bit 1 dans le registre CCR.

Au moment où il y aura un signal sur l'entrée PA2, il va y avoir une écriture à l'adresse \$00E8, le programme va alors quitter la boucle infinie et avec la fonction "jump" on va exécuter la partie "issprog".

Maintenant le programme terminé, détaillons l'**organigramme** du programme.



L'organigramme correspond aux lignes de code données dans l'énoncé découpées de la façon suivante :

#### **Boucle infinie :**

```
boucle  ldd    #$0000 ;
        std    $2000
        bra    boucle ;
```

**Requête d'interruption:** le FLAG passe à 1, on sort de la boucle infinie et on se rend à une adresse prédéfinie

```
boucle  org    $00E8 ;
        jmp    issprog ;
```

**Sous programme d'interruption:** la requête renvoie à une autre partie du programme

```
issprog ldd    #$ACDC ;
        std    $2000 ;
```

On va relever l'état de la mémoire a l'adresse \$2000 **avant l'exécution** :

```

BUFFALO 3.4 (ext) - Bit User Fast Friendly Aid to Logical Operation
>MD 2000

2000 00 00 FE FF 7F 67 FD FE ED FF DD 1F F9 3F EC FD      g      ?
2010 FF 3B F7 DD F9 CF 5F FF 9B FD 6B 7F 7F E4 DD C3      ;      k
2020 CD FB 5E 78 A9 F7 DB BE F7 DE 7B FC F2 77 D9 FD      ^x      w
2030 BF FF EE E9 EC BF 93 D6 EA FF F2 3D ED 3A 3B EB      = :
2040 94 32 17 E4 67 E7 D4 3E 56 F7 D6 BF F1 ED F7 EF      2 g >V
2050 E5 8B A8 D1 10 46 10 00 7D BE FF 7E B9 BE FE AF      F
2060 E5 FE B3 F7 BD FF DD EA FF F4 63 CF FD CA FF FB
2070 CF 7F DF E7 DD BE 41 2D 2F EF 11 66 CF D7 FF DB      A-/ f
2080 67 98 EF FB FE BF 7D 94 DD EF FF E1 FF 6D FE 7F g      m

```

Et on fait la même chose pour l'état de la mémoire à l'adresse \$00E8 :

```
>MD 00E8

00E0 E3 71 7E E3 71 7E E3 71 7E 21 20 7E E3 71 7E E3  q q q !  q
00F0 71 7E E3 71 7E FA 7A 7E E3 71 7E E3 71 7E E3 71  q q z q q q
0100 FF FF FF FD FF FF FF 7F FD FF FF FF FF E7 FF FF
0110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FD FF
0120 FF FF FF FF FF FD FB EF FF FF FF FF FF FF FF FF
0130 FF FE FF FF FF FE FF FF FF FF FF FF FF FF FF FF
0140 FF BF FF FF FF FF FF FF FF FF FF FF FF FF FE FF
0150 FF FF FF FF FF FF FF FF FF FF FF FF FF FA FF FF
0160 FF FF FF FF FF FF FF FF FF FF FF FF FF 7F FF FF
>
```

On va à présent relever l'état de la mémoire à l'adresse \$2000 **après l'exécution** :

```
BUFFALO 3.4 (ext) - Bit User Fast Friendly Aid to Logical Operation
>md 2000

2000 AC DC FE FF 7F 67 FD FE ED FF DD 1F F9 3F EC FD      g      ?
2010 FF 3B F7 DD F9 CF 5F FF 9B FD 6B 7F 7F E4 DD C3    ;      k
2020 CD FB 5E 78 A9 F7 DB BE F7 DE 7B FC F2 77 D9 FD      ^x      w
2030 BF FF EE E9 EC BF 93 D6 EA FF F2 3D ED 3A 3B EB      = ;
2040 94 32 17 E4 67 E7 D4 3E 56 F7 D6 BF F1 ED F7 EF    2  g  >V
2050 E5 8B A8 D1 10 46 10 00 7D BE FF 7E B9 BE FE AF      F
2060 E5 FE B3 F7 BD FF DD EA FF F4 63 CF FD CA FF FB      c
2070 CF 7F DF E7 DD BE 41 2D 2F EF 11 66 CF D7 FF DB      A-/  f
2080 67 98 EF FB FE BF 7D 94 DD EF FF E1 FF 6D FE 7F g      m
```

Et la même chose pour l'état de la mémoire à l'adresse \$00E8 :

```
>md 00E8

00E0 E3 71 7E E3 71 7E E3 71 7E 21 20 7E E3 71 7E E3  q q q !  q
00F0 71 7E E3 71 7E FA 7A 7E E3 71 7E E3 71 7E E3 71  q q z q q q
0100 FF FF FF FD FF FF FF 7F FD FF FF FF FF E7 FF FF
0110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FD FF
0120 FF FF FF FF FF FD FB EF FF FF FF FF FF FF FF FF
0130 FF FE FF FF FF FE FF FF FF FF FF FF FF FF FF FF
0140 FF BF FF FF FF FF FF FF FF FF FF FF FF FF FE FF
0150 FF FF FF FF FF FF FF FF FF FF FF FF FF FA FF FF
0160 FF FF FF FF FF FF FF FF FF FF FF FF FF 7F FF FF
>
```

On lance une première fois le programme sans brancher l'alimentation sur l'entrée PA2, on obtient la valeur 0 aux adresses 2000/2001, il n'y a donc pas d'interruption. Par contre si on applique une tension, la valeur change et on obtient ACDC, on a donc réussi à faire une interruption ! La valeur contenue à l'adresse \$00E8 n'est pas modifiée. Elle vaut 7E et est suivie par 21 dans les deux cas.

### 2.2.2. Mesure d'une période

Le principe du programme est le suivant : le programme commence par des initialisation et des configurations, il rentre ensuite dans une boucle infinie qui ne s'arrête que s'il y a une interruption. Le programme exécute alors un sous-programme, puis de nouveau on retourne dans la boucle infinie.

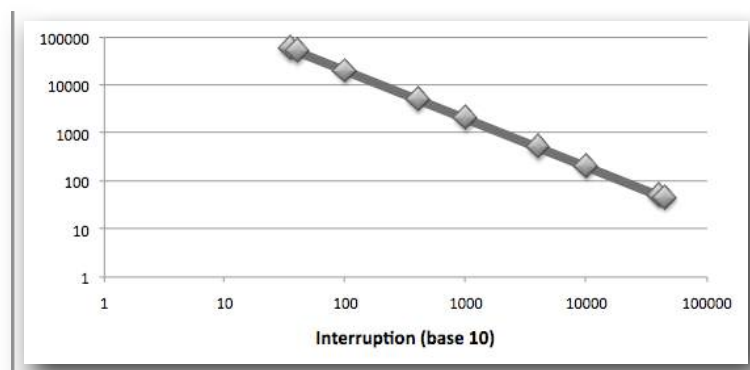
On obtient le **programme** suivant :

|                |             |                |                                                                                      |
|----------------|-------------|----------------|--------------------------------------------------------------------------------------|
| <b>TIC1</b>    | <b>equ</b>  | <b>\$1010</b>  | <i>on assigne l'adresse 1010 au label TIC1</i>                                       |
| <b>TFLG1</b>   | <b>equ</b>  | <b>\$1023</b>  | <i>on assigne l'adresse 1023 au label TFLG1</i>                                      |
| <b>TCTL2</b>   | <b>equ</b>  | <b>\$1021</b>  | <i>on assigne l'adresse 1021 au label TCTL2</i>                                      |
| <b>TMSK1</b>   | <b>equ</b>  | <b>\$1022</b>  | <i>on assigne l'adresse 1022 au label TMSK1</i>                                      |
|                | <b>org</b>  | <b>\$2050</b>  | <i>adresse de repère pour l'initialisation des réservations</i>                      |
| <b>front1</b>  | <b>rmb</b>  | <b>2</b>       | <i>on réserve 2 octets aux adresses 2050/2051 avec le label front1</i>               |
| <b>front2</b>  | <b>rmb</b>  | <b>2</b>       | <i>on réserve les 2 octets suivants sous le label front2</i>                         |
| <b>period</b>  | <b>rmb</b>  | <b>2</b>       | <i>on réserve les 2 octets suivants sous le label period</i>                         |
| <b>ED1ba</b>   | <b>fcb</b>  | <b>\$10</b>    | <i>on stocke la constante 10 à l'adresse de ED1ba</i>                                |
| <b>N</b>       | <b>rmb</b>  | <b>1</b>       | <i>on réserve 1 octet au label N</i>                                                 |
|                | <b>org</b>  | <b>\$00E8</b>  | <i>on définit l'adresse de demande d'interruption</i>                                |
|                | <b>jmp</b>  | <b>issprog</b> | <i>on se place sur le programme issprog</i>                                          |
|                | <b>sei</b>  |                | <i>on met le bit 1 de CCR à 1 pour déclarer l'interruption</i>                       |
|                | <b>lds</b>  | <b>#\$0200</b> | <i>on charge le contenu se trouvant à l'adresse 0200</i>                             |
|                | <b>ldab</b> | <b>ED1ba</b>   | <i>on charge le contenu de ED1ba dans b</i>                                          |
|                | <b>stab</b> | <b>TCTL2</b>   | <i>on copie la valeur de b dans TCTL2</i>                                            |
|                | <b>ldaa</b> | <b>TFLG1</b>   | <i>on charge le contenu de TCTL2 dans a</i>                                          |
|                | <b>anda</b> | <b>#\$04</b>   | <i>on fait un ET logique avec a et la constante 4</i>                                |
|                | <b>staa</b> | <b>TFLG1</b>   | <i>on copie la valeur de a dans TFLG1</i>                                            |
|                | <b>ldaa</b> | <b>#\$04</b>   | <i>on charge la constante 4 dans a</i>                                               |
|                | <b>staa</b> | <b>TMSK1</b>   | <i>on copie la valeur de a dans TMSK1</i>                                            |
|                | <b>cli</b>  |                | <i>on met le bit 1 de CCR à 0 ; c'est la fin de la déclaration de l'interruption</i> |
|                | <b>ldab</b> | <b>#2</b>      | <i>on charge la constante 2 dans b</i>                                               |
|                | <b>stab</b> | <b>N</b>       | <i>on stocke dans la valeur de l'accumulateur b dans N</i>                           |
| <b>boucle</b>  | <b>tst</b>  | <b>N</b>       | <i>on compare la valeur saisie de N avec 0</i>                                       |
|                | <b>bne</b>  | <b>boucle</b>  | <i>tant que le résultat est différent de 0, on boucle</i>                            |
|                | <b>clr</b>  | <b>TMSK1</b>   | <i>on clear TMSK1</i>                                                                |
|                | <b>ldd</b>  | <b>TIC1</b>    | <i>on charge le contenu de TIC1 dans l'accumulateur d</i>                            |
|                | <b>std</b>  | <b>front2</b>  | <i>on copie la valeur de d dans front2</i>                                           |
|                | <b>subd</b> | <b>front1</b>  | <i>on soustrait à la valeur de front1 dans d</i>                                     |
|                | <b>std</b>  | <b>period</b>  | <i>on copie la valeur de d dans period</i>                                           |
|                | <b>rts</b>  |                | <i>fin du programme</i>                                                              |
| <b>issprog</b> | <b>ldaa</b> | <b>TFLG1</b>   | <i>on charge la valeur de TFLG1 dans a</i>                                           |
|                | <b>anda</b> | <b>#\$04</b>   | <i>on fait un ET logique avec a et la constante 4</i>                                |
|                | <b>staa</b> | <b>TFLG1</b>   | <i>on copie le contenu de TFLG1 dans l'accumulateur a</i>                            |
|                | <b>dec</b>  | <b>N</b>       | <i>on décrémente N</i>                                                               |
|                | <b>beq</b>  | <b>sortie</b>  | <i>si N = 0, on sort</i>                                                             |
|                | <b>ldx</b>  | <b>TIC1</b>    | <i>on charge la valeur de TIC1 dans x</i>                                            |
|                | <b>stx</b>  | <b>front1</b>  | <i>on copie la valeur de x à l'adresse de front1</i>                                 |
| <b>sortie</b>  | <b>rti</b>  |                | <i>on stoppe l'interruption</i>                                                      |

On teste alors le programme avec différentes fréquences en cherchant les limites, pour des fréquences allant de 50 Hz à 50 kHz (on a rajouté une fréquence en dessous de 40 Hz pour justifier une seconde fois la limite en BF).

| f (Hz) | Interruption (hexadécimal) | Interruption (base 10) |
|--------|----------------------------|------------------------|
| 35     | DE75                       | 56949                  |
| 40     | C33F                       | 49983                  |
| 100    | EDC7                       | 19911                  |
| 400    | 1382                       | 4994                   |
| 1k     | 07D0                       | 2000                   |
| 4k     | 01F4                       | 500                    |
| 10k    | 00C8                       | 200                    |
| 40k    | 0032                       | 50                     |
| 45k    | 002D                       | 45                     |

On reporte les mesures effectuées dans un **repère log-log**. La courbe obtenue est la suivante :



On retrouve une courbe ayant la même allure que précédemment, on peut donc en faire la même déduction. En réalisant les différentes mesures, on constate que la limite est de **35 Hz en BF** et de **45 kHz en HF** et les justifications restent similaires à l'autre méthode.

### I/ AFFICHAGE DE CARACTÈRES

Pour afficher des caractères, on utilise le programme donné. On se rend compte que l'on a deux sous programmes "longue" et "courte" qui servent de **temporisation pour la mise à jour de l'affichage**. On s'intéresse au nombre de cycles que l'on obtient en activant l'option de comptage :

```
0001 b5f0          LCDCMD equ  #b5f0
0002 b5f1          LCDDAT equ  #b5f1
0003
0004 003e          L4xC20 equ  #3c * 4x20
0005 0001          CLRCH equ  #01
0006 000f          DCBON equ  #0f
0007 0006          CINSHOF equ  #06
0008 0014          SCCHR equ  #14
0009 0002          CSRHOM equ  #02
0010
0011 2100          ORG  #2100
0012
0013 2100 86 3c      [ 2 ] init  ldaa  #L4xC20
0014 2102 b7 b5 f0  [ 4 ] staa  LCDCMD
0015 2105 bd 21 30  [ 6 ] jsr   longue
0016
0017 2108 86 01      [ 2 ] ldaa  #CLRCH
0018 210a b7 b5 f0  [ 4 ] staa  LCDCMD
0019 210d bd 21 30  [ 6 ] jsr   longue
0020
0021 2110 86 0f      [ 2 ] ldaa  #DCBON
0022 2112 b7 b5 f0  [ 4 ] staa  LCDCMD
0023 2115 bd 21 37  [ 6 ] jsr   courte
0024
0025 2118 86 06      [ 2 ] ldaa  #CINSHOF
0026 211a b7 b5 f0  [ 4 ] staa  LCDCMD
0027 211d bd 21 37  [ 6 ] jsr   courte
0028
0029 2120 86 14      [ 2 ] ldaa  #SCCHR
0030 2122 b7 b5 f0  [ 4 ] staa  LCDCMD
0031 2125 bd 21 37  [ 6 ] jsr   courte
0032
0033 2128 86 02      [ 2 ] ldaa  #CSRHOM
0034 212a b7 b5 f0  [ 4 ] staa  LCDCMD
0035 212d bd 21 37  [ 6 ] jsr   courte
0036
0037 2130 ce c3 f0  [ 3 ] longuidx #50000
0038 2133 09        [ 3 ] loop_idx
0039 2134 26 fd      [ 3 ] bne   loop_l
0040 2136 39        [ 5 ] rts
0041
0042 2137 ce 03 e8  [ 3 ] courteldx #1000
0043 213a 09        [ 3 ] loop_cdx
0044 213b 26 fd      [ 3 ] bne   loop_c
0045 213d 39        [ 5 ] rts
```

On se rend compte que les deux boucles sont très longues, à savoir, 50000 et 1000. C'est pourquoi il faut modifier leur durée.

Pour afficher octet par octet, on peut restreindre à un passage pour chaque boucle. A l'aide d'une exécution pas à pas, on en déduit **les différentes étapes** :

- ✓ Initialisation de l'écran (LCDCMD)
- ✓ Rafraîchissement (CLRCH)
- ✓ Affichage du curseur (DCBON)
- ✓ Restreindre son déplacement (CINSHOF)
- ✓ Avancer sa position de 1 vers la droite (SCCHR)
- ✓ Déclarer la position actuelle comme initiale (CSRHOM)

Les commandes pour afficher un caractère sont :

|      |        |
|------|--------|
| ldaa | #'A'   |
| staa | LCDDAT |
| jsr  | courte |

Donc, si on veut afficher l'un après l'autre les caractères A, B, C et D, on doit saisir le programme suivant :

| suite | ldaa | #'A'   | on charge dans l'accumulateur a le code ASCII du caractère A          |
|-------|------|--------|-----------------------------------------------------------------------|
|       | staa | LCDDAT | enregistrement du contenu de a dans la variable donnée de l'écran LCD |
|       | jsr  | courte | saut au pointeur d'instructions "courte"                              |
|       | ldaa | #'B'   | ...etc (on répète pour la lettre B)                                   |
|       | staa | LCDDAT |                                                                       |
|       | jsr  | courte |                                                                       |
|       | ldaa | #'C'   | ...etc (on répète pour la lettre C)                                   |
|       | staa | LCDDAT |                                                                       |
|       | jsr  | courte |                                                                       |
|       | ldaa | #'D'   | ...etc (on répète pour la lettre D)                                   |
|       | staa | LCDDAT |                                                                       |
|       | jsr  | courte |                                                                       |

En effet, après l'exécution du programme compilé correctement, on a réussi à obtenir l'affichage de 4 caractères sur notre écran LCD programmable. Pour ce faire, nous avons du utiliser la commande "CALL 2100" pour appeler l'instruction à réaliser. Après réalisation de ce programme, on arrive à obtenir sur notre écran LCD un résultat tel que le montre l'image ci-contre.



On souhaite à présent **afficher une chaîne de caractères**. Pour cela, on se sert du programme donné et on le complète. La directive '**fcc**' permet de créer une variable qui contient des caractères ASCII.

|               |              |                      |                                                                                                               |
|---------------|--------------|----------------------|---------------------------------------------------------------------------------------------------------------|
| <b>texto</b>  | <b>org</b>   | <b>\$2000</b>        | <i>on déclare les variables à l'adresse 2000</i>                                                              |
|               | <b>fcc</b>   | <b>'La periode='</b> | <i>la variable texto réserve l'espace nécessaire pour contenir la chaîne indiquée, à savoir 'La période='</i> |
| <b>fin</b>    | <b>fcbl</b>  | <b>#0</b>            | <i>pointeur "fin" : déclaration d'une constante sur un octet de valeur 0</i>                                  |
|               | <b>ORG</b>   | <b>\$2100</b>        | <i>début du programme à l'adresse 2100</i>                                                                    |
| <b>init</b>   | <b>.....</b> | <b>(cf q11)</b>      | <i>on utilise le code d'initialisation de la question 11</i>                                                  |
| <b>chaîne</b> | <b>ldy</b>   | <b>#texto</b>        | <i>on charge dans le registre y la valeur de texto</i>                                                        |
|               | <b>ldaa</b>  | <b>0,y</b>           | <i>pointeur "chaîne" : on charge dans l'accumulateur a la valeur contenue à y+0</i>                           |
|               | <b>beq</b>   | <b>suite</b>         | <i>on charge dans le registre y la valeur de texto</i>                                                        |
|               | <b>staa</b>  | <b>LCDDAT</b>        | <i>enregistrement du contenu de A dans la variable donnée de l'écran LCD</i>                                  |
|               | <b>jsr</b>   | <b>courte</b>        | <i>saut au pointeur d'instructions "courte"</i>                                                               |
|               | <b>iny</b>   |                      | <i>on incrémente y</i>                                                                                        |
|               | <b>bra</b>   | <b>chaîne</b>        | <i>on retourne à "chaîne" quelle que soit l'instruction</i>                                                   |
| <b>suite</b>  | <b>.....</b> | <b>(cf q12)</b>      | <i>on utilise le code de la suite avec l'affichage de la question 12</i>                                      |

On intègre maintenant ce programme dans celui de la question 2 et **on procède à son exécution**. La photo de notre écran LCD programmé est le résultat de la programmation d'une chaîne de caractères. On peut alors lire 'La période = ABCD '. Ce résultat validé par Mr Provost nous prouve bien que notre programme une fois compilé répond bien aux instructions et qu'il n'y a pas de bug durant le déroulement des instructions.



## II/ CONVERSION D'UN ENTIER SUR 1 OU 2 OCTETS

### 2.1. CONVERSION D'UN ENTIER SUR 1 OCTET

On commence par compléter les lignes manquantes de l'algorithme qui servira à convertir un entier de 1 octet (deux quartets) en deux caractères (quartet haut et quartet bas) à partir du label conv.

|        |       |               |                                                                                                               |
|--------|-------|---------------|---------------------------------------------------------------------------------------------------------------|
|        | org   | \$2000        | <i>on déclare les variables à l'adresse 2000</i>                                                              |
| texto  | fcc   | 'La periode=' | <i>la variable texto réserve l'espace nécessaire pour contenir la chaîne indiquée, à savoir 'La période='</i> |
| fin    | fcbl  | #0            | <i>pointeur "fin" : déclaration d'une constante sur un octet de valeur 0</i>                                  |
|        | ORG   | \$2020        | <i>début du programme à l'adresse 2020</i>                                                                    |
| table  | fcc   | '0'           | <i>réserve du caractère ASCII '0'</i>                                                                         |
|        | fcc   | '1'           | <i>réserve du caractère ASCII '1'</i>                                                                         |
|        | fcc   | '2'           | <i>réserve du caractère ASCII '2'</i>                                                                         |
|        | fcc   | '3'           | <i>réserve du caractère ASCII '3'</i>                                                                         |
|        | fcc   | '4'           | <i>réserve du caractère ASCII '4'</i>                                                                         |
|        | fcc   | '5'           | <i>réserve du caractère ASCII '5'</i>                                                                         |
|        | fcc   | '6'           | <i>réserve du caractère ASCII '6'</i>                                                                         |
|        | fcc   | '7'           | <i>réserve du caractère ASCII '7'</i>                                                                         |
|        | fcc   | '8'           | <i>réserve du caractère ASCII '8'</i>                                                                         |
|        | fcc   | '9'           | <i>réserve du caractère ASCII '9'</i>                                                                         |
|        | fcc   | 'A'           | <i>réserve du caractère ASCII 'A'</i>                                                                         |
|        | fcc   | 'B'           | <i>réserve du caractère ASCII 'B'</i>                                                                         |
|        | fcc   | 'C'           | <i>réserve du caractère ASCII 'C'</i>                                                                         |
|        | fcc   | 'D'           | <i>réserve du caractère ASCII 'D'</i>                                                                         |
|        | fcc   | 'E'           | <i>réserve du caractère ASCII 'E'</i>                                                                         |
|        | fcc   | 'F'           | <i>réserve du caractère ASCII 'F'</i>                                                                         |
| octet  | fcbl  | #\$36         | <i>octet de test</i>                                                                                          |
| q_haut | rmb   | 1             | <i>réserve d'un octet au label q_haut</i>                                                                     |
| q_bas  | rmb   | 1             | <i>réserve d'un octet au label q_bas</i>                                                                      |
|        | ORG   | \$2100        | <i>début du programme à l'adresse 2100</i>                                                                    |
| conv   | ldx   | #table        | <i>label conv : on charge dans le registre x la valeur de table</i>                                           |
|        | ldaa  | octet         | <i>on charge le contenu de octet dans a</i>                                                                   |
|        | anda  | #\$0F         | <i>on fait un ET logique avec a et 00001111 (intérêt : 4 bits de poids faible)</i>                            |
|        | adda  | 0,x           | <i>on ajoute la valeur a dans le registre x</i>                                                               |
|        | staa  | q_bas         | <i>enregistrement du contenu de a dans la variable q_bas</i>                                                  |
|        | ldaa  | octet         | <i>on re-charge le contenu de octet dans a :</i><br><b>b7b6b5b4b3b2b1b0</b>                                   |
|        | asra  |               | <i>décalage à droite : Xb7b6b5b4b3b2b1</i>                                                                    |
|        | asra  |               | <i>décalage à droite : XXb7b6b5b4b3b2</i>                                                                     |
|        | asra  |               | <i>décalage à droite : XXXb7b6b5b4b3</i>                                                                      |
|        | asra  |               | <i>décalage à droite : XXXXb7b6b5b4</i>                                                                       |
|        | anda  | #\$0F         | <i>on fait un ET logique avec a (récupération des 4 bits de poids fort)</i>                                   |
|        | anda  | 0,x           | <i>on cherche la valeur du symbole qui correspond pour LCD</i>                                                |
|        | staa  | q_haut        | <i>on enregistre la 2<sup>ème</sup> valeur, bit de poids fort</i>                                             |
| init   | ..... | (cf q1)       | <i>partie du programme d'initialisation</i>                                                                   |

|                              |             |               |                                                                                     |
|------------------------------|-------------|---------------|-------------------------------------------------------------------------------------|
| <i>// Suite du programme</i> |             |               |                                                                                     |
| <b>chaîne</b>                | <b>ldy</b>  | <b>#texto</b> | <i>on charge dans le registre y la valeur de texto</i>                              |
|                              | <b>ldaa</b> | <b>0,y</b>    | <i>pointeur "chaîne" : on charge dans l'accumulateur a la valeur contenue à y+0</i> |
|                              | <b>beq</b>  | <b>suite</b>  | <i>si la valeur précédente = 0 , on saute à l'instruction "suite"</i>               |
|                              | <b>staa</b> | <b>LCDDAT</b> | <i>enregistrement du contenu de a dans la variable donnée de l'écran LCD</i>        |
|                              | <b>jsr</b>  | <b>courte</b> | <i>saut au pointeur d'instructions "courte"</i>                                     |
| <b>suite</b>                 | <b>iny</b>  |               | <i>on incrémente y</i>                                                              |
|                              | <b>bra</b>  | <b>chaîne</b> | <i>on retourne à "chaîne" quelle que soit l'instruction</i>                         |
|                              | <b>ldaa</b> | <b>q_haut</b> | <i>on charge le contenu de q_haut dans a</i>                                        |
|                              | <b>staa</b> | <b>LCDDAT</b> | <i>enregistrement du contenu de a dans la variable donnée de l'écran LCD</i>        |
|                              | <b>jsr</b>  | <b>courte</b> | <i>saut au pointeur d'instructions "courte"</i>                                     |
|                              | <b>ldaa</b> | <b>q_bas</b>  | <i>on charge le contenu de q_bas dans a</i>                                         |
|                              | <b>staa</b> | <b>LCDDAT</b> | <i>enregistrement du contenu de a dans la variable donnée de l'écran LCD</i>        |
|                              | <b>jsr</b>  | <b>courte</b> | <i>saut au pointeur d'instructions "courte"</i>                                     |
|                              | <b>ldaa</b> | <b>#'.'</b>   | <i>on charge la valeur '.' dans a</i>                                               |
|                              | <b>staa</b> | <b>LCDDAT</b> | <i>enregistrement du contenu de a dans la variable donnée de l'écran LCD</i>        |
|                              | <b>jsr</b>  | <b>courte</b> | <i>saut au pointeur d'instructions "courte"</i>                                     |
|                              | <b>ldaa</b> | <b>#'.'</b>   | <i>on charge la valeur '.' dans a</i>                                               |
|                              | <b>staa</b> | <b>LCDDAT</b> | <i>enregistrement du contenu de a dans la variable donnée de l'écran LCD</i>        |
|                              | <b>jsr</b>  |               | <i>saut au pointeur d'instructions "courte"</i>                                     |

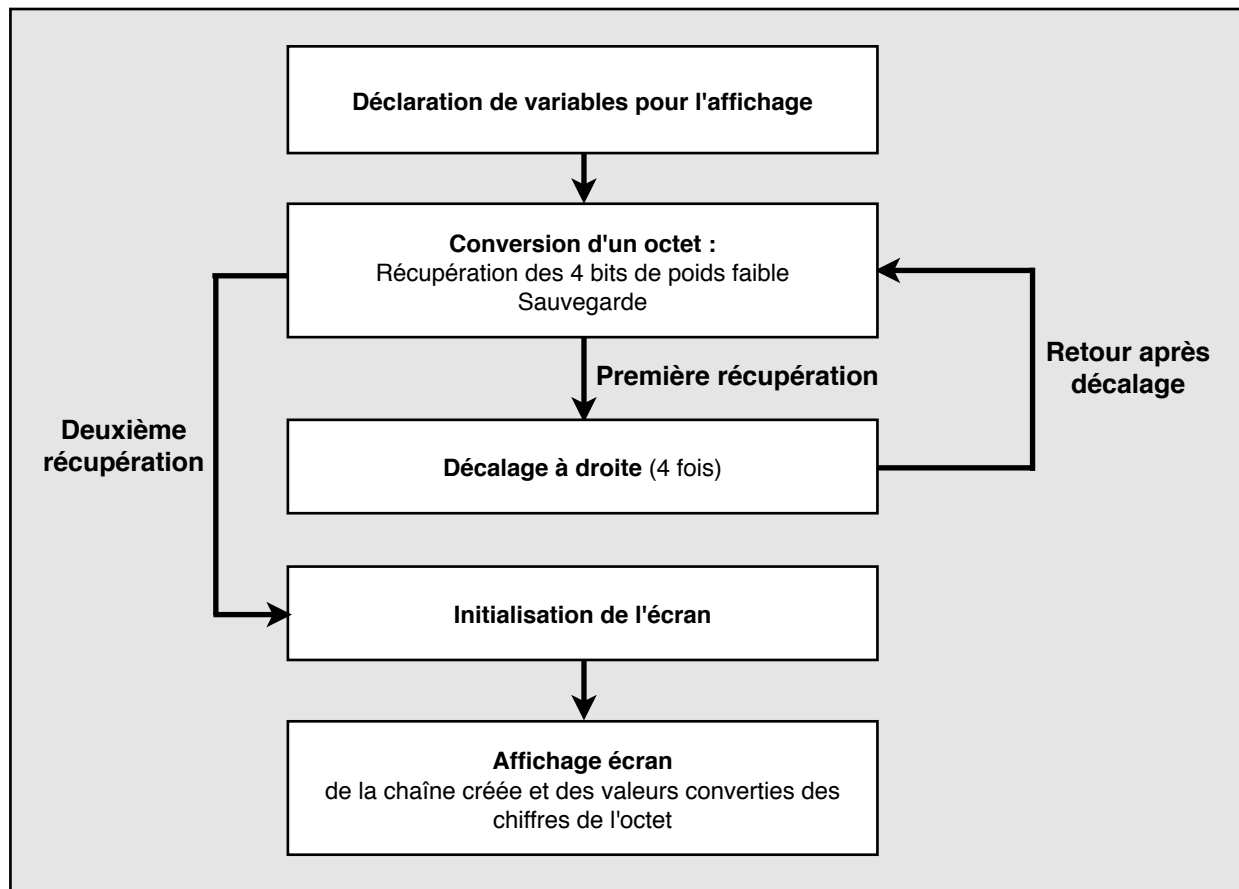
Pour pouvoir afficher sur l'écran LCD un caractère saisi au clavier par un utilisateur quelconque, il est nécessaire de faire une conversion car le code ASCII du caractère tapé au clavier est un entier. Nous rappelons qu'un caractère représente pèse un octet.

Afin de pouvoir afficher des variables d'un programme, il est nécessaire de les stocker dans les accumulateurs. Le problème que nous avons rencontré a lieu lors de l'affichage d'un octet. En effet, pour afficher une valeur écrite en mémoire en hexadécimal, il est nécessaire de séparer cette valeur en deux (les deux parties du nombre). On doit séparer cette valeur en deux car on écrit en hexadécimal, mais les accumulateurs que l'on utilise, eux, sont des octets. Ils ne peuvent donc que stocker un nombre de  $2^8$  bits soit un nombre allant de 00 à FF en hexadécimal.

On cherche donc dans un premier temps à afficher uniquement un seul octet. Comme on l'a dit, cet octet comporte un bit de poids faible et un bit de poids fort (représentés dans nos programmes respectivement par q\_bas et q\_haut). Afin de récupérer ces différentes valeurs q\_bas et q\_haut, nous utilisons l'**organigramme** (ci-dessous), qui reprend le même principe que l'algorithme utilisé.

Dans un premier temps, on a le chargement de la table servant de référence. Ensuite, on a une récupération de la valeur "totale". On effectue par la suite un ET logique entre l'accumulateur et \$0F, puis on ajoute 0x à l'accumulateur, puis enfin on a le stockage du bit de poids faible.

Par la suite, on reprend le même principe pour récupérer le bit de poids fort mais en plus on devra se décaler dans la mémoire de 4 bits. En effet, on souhaite récupérer le bit de poids fort d'un nombre codé sur 8 bits en hexadécimal. Ainsi, on aura réussi à afficher les deux bits de notre hexadécimal.



Nous avons alors réalisé un premier test grâce au programme et nous pouvons lire (visible aussi ci-contre sur la photo) : 'La periode=36'.



## 2.2. CONVERSION D'UN ENTIER SUR 2 OCTETS

On sait que la précision de notre mesure de fréquence atteint 2 octets donc il va falloir à présent faire une conversion sur 2 octets.

Malheureusement, l'affichage d'un nombre sur deux octets n'est pas aussi simple que pour l'affichage d'un octet. Mais il existe une technique que nous avons utilisée qui permet d'afficher les deux octets. L'astuce est en fait de stocker notre valeur non pas dans l'accumulateur a, mais dans l'accumulateur x (qui est la concaténation des accumulateurs a et b). Il nous a alors été possible de reprendre le travail que nous avons fait précédemment pour afficher un octet en affichant consécutivement l'accumulateur a puis l'accumulateur b.

La partie principale du programme à traiter correspond à celle qui se trouve au niveau de la partie "conv", soit celle située au niveau du pointeur. Le programme exécuté traite correctement les deux octets. A l'affichage, on obtient : 'La periode=3669'.



## Communication par liaison série asynchrone

On souhaite à présent réaliser le déclenchement de l'affichage à partir de la saisie d'un caractère dans AxIDE. Afin de tester les méthodes de communication avec le port série, nous avons tenté d'effectuer des saisies pendant une scrutation du programme. Le choix du Polling s'est tout de suite imposé à nous car il est bien plus facile de tester si le programme reçoit bien notre signal.

Voici le programme que nous avons utilisé :

|              |             |               |                                                                                        |
|--------------|-------------|---------------|----------------------------------------------------------------------------------------|
| <b>BAUD</b>  | <b>equ</b>  | <b>\$102b</b> | <i>on assigne l'adresse 102b au label BAUD</i>                                         |
| <b>SCDR</b>  | <b>equ</b>  | <b>\$102f</b> | <i>on assigne l'adresse 102f au label SCDR</i>                                         |
| <b>SCSR</b>  | <b>equ</b>  | <b>\$102e</b> | <i>on assigne l'adresse 102e au label SCSR</i>                                         |
| <b>SCCR1</b> | <b>equ</b>  | <b>\$102c</b> | <i>on assigne l'adresse 102c au label SCCR1</i>                                        |
| <b>SCCR2</b> | <b>equ</b>  | <b>\$102d</b> | <i>on assigne l'adresse 102d au label SCCR2</i>                                        |
|              | <b>org</b>  | <b>\$2200</b> | <i>on définit un repère pour l'adresse de départ du programme</i>                      |
|              | <b>clr</b>  | <b>SCCR1</b>  | <i>on met à 0 le contenu de SCCR1 (clear)</i>                                          |
|              | <b>ldaa</b> | <b>#\$30</b>  | <i>on charge la valeur 30 (en hexadécimale) dans l'accumulateur a</i>                  |
|              | <b>staa</b> | <b>BAUD</b>   | <i>on stocke la valeur de l'accumulateur a à l'adresse de BAUD</i>                     |
|              | <b>ldab</b> | <b>#\$0C</b>  | <i>on charge la valeur 12 (en hexadécimale) dans l'accumulateur b</i>                  |
|              | <b>stab</b> | <b>SCCR2</b>  | <i>on stocke la valeur de l'accumulateur b à l'adresse de SCCR2</i>                    |
|              | <b>clr</b>  | <b>\$2000</b> | <i>on met à 0 ce que l'on a à l'adresse 2000</i>                                       |
| <b>loop</b>  | <b>ldaa</b> | <b>SCSR</b>   | <i>loop : cette ligne est le début d'une boucle</i>                                    |
|              |             |               | <i>on stocke dans l'accumulateur a la valeur de SCSR</i>                               |
|              | <b>anda</b> | <b>#\$20</b>  | <i>on fait un ET entre l'accumulateur a et la valeur 20</i>                            |
|              | <b>beq</b>  | <b>loop</b>   | <i>si le test précédent a mis le flag Z à 1, alors on repart au début de la boucle</i> |
|              | <b>ldaa</b> | <b>SCDR</b>   | <i>on stocke dans l'accumulateur a la valeur de SCDR</i>                               |
|              | <b>staa</b> | <b>\$2000</b> | <i>on stocke cette valeur à l'adresse 2000</i>                                         |
| <b>rts</b>   |             |               | <i>fin du programme</i>                                                                |

Lorsque l'on effectue ce programme, on demande en fait à l'utilisateur de faire une saisie que l'on va stocker à l'adresse 2000.

Les méthodes de Polling et d'interruption présentées sont toutes deux très différentes sauf qu'on relève **un inconvénient majeur dans la méthode de Polling car on ne peut obtenir directement l'affichage du caractère tapé au clavier par l'utilisateur**. Contrairement à la méthode d'interruption, l'affichage est quasi-instantané et il n'est pas nécessaire de réinitialiser la carte électronique.

On a pris un exemple pour illustrer le fonctionnement de ce programme. Dans l'exemple ci-dessous, on appelle le programme situé à l'adresse 2200 et on fait la saisie de la lettre 'a' (son code est 61). Le programme s'arrête alors. On recommence une seconde fois l'opération, et on saisit cette fois-ci la lettre 'b' (son code est 62).

```

P-2200 Y-FFFF X-FFFF A-62 B-0C C-D0 S-0041
>

P-2200 Y-FFFF X-FFFF A-6D B-0C C-D0 S-0041
>call 2200

P-2200 Y-FFFF X-FFFF A-61 B-0C C-D0 S-0041
>md 2000

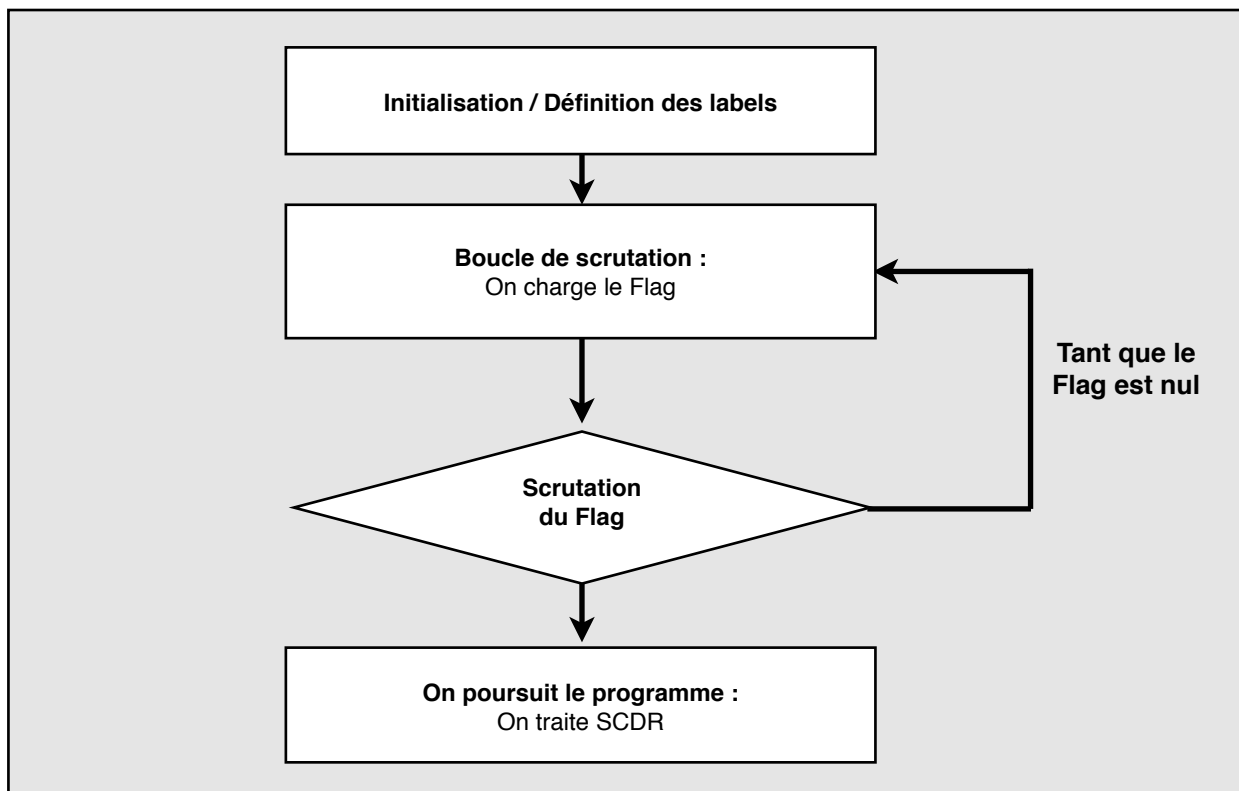
2000 61 CE 3E 1A FB 1C FF 7C 7D 79 80 7F 01 F9 9D 34 a > y 6
2010 EC 9A F5 9A D8 6F 78 47 3C 0E ED 3F DD D3 F8 45 = xG< ? E
2020 C9 AD 9C 9F 3D D1 DF C0 83 71 FF 9F 36 37 93 7F = q 67
2030 AE F3 B3 FF 70 5E F1 0F BA BE E4 F5 BB B6 B1 73 p^ s
2040 D5 4D DF FO 9D F9 6F 39 5C C6 3F 4F 5C FA F6 F1 M o9\ ?0\
2050 3F A3 C5 16 75 5F 6F 53 DF 96 EE E7 EF EA 3E 26 ? uoS >6
2060 FB E7 68 BB F7 FD 39 FO 2E A5 FE D6 DF E7 3C 06 h 9 . <
2070 77 C7 8B DF 5A CF E9 E7 69 DB 16 FE 1D 7B 9F BD w Z i
2080 F7 37 D9 3C C7 C0 5B 6C CB 1A A6 EF DA EF CB F9 7 < [1
>call 2200

P-2200 Y-FFFF X-FFFF A-62 B-0C C-D0 S-0041
>md 2000

2000 62 CE 3E 1A FB 1C FF 7C 7D 79 80 7F 01 F9 9D 34 b > y 6
2010 EC 9A F5 9A D8 6F 78 47 3C 0E ED 3F DD D3 F8 45 = xG< ? E
2020 C9 AD 9C 9F 3D D1 DF C0 83 71 FF 9F 36 37 93 7F = q 67
2030 AE F3 B3 FF 70 5E F1 0F BA BE E4 F5 BB B6 B1 73 p^ s
2040 D5 4D DF FO 9D F9 6F 39 5C C6 3F 4F 5C FA F6 F1 M o9\ ?0\
2050 3F A3 C5 16 75 5F 6F 53 DF 96 EE E7 EF EA 3E 26 ? uoS >6
2060 FB E7 68 BB F7 FD 39 FO 2E A5 FE D6 DF E7 3C 06 h 9 . <
2070 77 C7 8B DF 5A CF E9 E7 69 DB 16 FE 1D 7B 9F BD w Z i
2080 F7 37 D9 3C C7 C0 5B 6C CB 1A A6 EF DA EF CB F9 7 < [1
>

```

Maintenant le programme réalisé, établissons l'**organigramme** du programme.



On s'inspirer du code ci-dessous pour modifier ce programme de manière à **fonctionner en interruption** :

|                |             |                |                                                                                      |
|----------------|-------------|----------------|--------------------------------------------------------------------------------------|
| <b>BAUD</b>    | <b>equ</b>  | <b>\$102b</b>  | <i>on assigne l'adresse 102b au label BAUD</i>                                       |
| <b>SCDR</b>    | <b>equ</b>  | <b>\$102f</b>  | <i>on assigne l'adresse 102f au label SCDR</i>                                       |
| <b>SCSR</b>    | <b>equ</b>  | <b>\$102e</b>  | <i>on assigne l'adresse 102e au label SCSR</i>                                       |
| <b>SCCR1</b>   | <b>equ</b>  | <b>\$102c</b>  | <i>on assigne l'adresse 102c au label SCCR1</i>                                      |
| <b>SCCR2</b>   | <b>equ</b>  | <b>\$102d</b>  | <i>on assigne l'adresse 102d au label SCCR2</i>                                      |
|                | <b>org</b>  | <b>\$00C4</b>  | <i>adresse correspondant à la requête d'interruption</i>                             |
|                | <b>jmp</b>  | <b>issprog</b> | <i>saut au sous programme qui traite l'interruption</i>                              |
|                | <b>org</b>  | <b>\$2200</b>  | <i>on définit un repère pour l'adresse de départ du programme</i>                    |
|                | <b>sei</b>  |                | <i>on met le bit 1 de CCR à 1 pour interdire les interruptions masquées</i>          |
|                | <b>clr</b>  | <b>SCCR1</b>   | <i>on met à 0 le SCCR1</i>                                                           |
|                | <b>clr</b>  | <b>SCSR</b>    | <i>on met à 0 le SCSR</i>                                                            |
|                | <b>ldaa</b> | <b>#30</b>     | <i>on charge la valeur 30 (en hexadécimale) dans l'accumulateur a</i>                |
|                | <b>staa</b> | <b>BAUD</b>    | <i>on stocke la valeur de l'accumulateur a à l'adresse de BAUD</i>                   |
|                | <b>ldaa</b> | <b>#2C</b>     | <i>on charge la valeur 44 (en hexadécimale) dans l'accumulateur a</i>                |
|                | <b>staa</b> | <b>SCCR2</b>   | <i>on stocke la valeur de l'accumulateur a à l'adresse de SCCR2</i>                  |
|                | <b>clr</b>  | <b>\$2000</b>  | <i>on met à 0 ce que l'on a à l'adresse 2000</i>                                     |
|                | <b>cli</b>  |                | <i>on met le bit 1 de CCR à 0 ; c'est la fin de la déclaration de l'interruption</i> |
| <b>loop</b>    | <b>bra</b>  | <b>loop</b>    | <i>loop : cette ligne est le début d'une boucle , boucle infinie</i>                 |
| <b>issprog</b> | <b>clr</b>  | <b>SCSR</b>    | <i>sous programme d'interruption</i>                                                 |
|                | <b>ldab</b> | <b>SCDR</b>    | <i>on stocke dans l'accumulateur b la valeur de SCSR</i>                             |
|                | <b>stab</b> | <b>\$2200</b>  | <i>on stocke cette valeur à l'adresse 2200</i>                                       |
| <b>rti</b>     |             |                | <i>on stoppe l'exécution</i>                                                         |

Pour sortir de la boucle infinie qui est sollicitée dans notre programme, nous avons alors tapé sur la touche 'c' du clavier qui a fait réagir un sous-programme signalant qu'une information est présente. Cette information envoyée est directement logée dans le registre appelé le SCSR et qui renvoie le code ASCII au registre SCDR. Afin de savoir ce qui s'est déroulé, une interrogation de la mémoire est réalisée par nous. L'interrogation mémoire est effectuée par la commande "MD200" du logiciel AxIDE, et on observe d'après la requête la valeur contenue dans l'accumulateur A.

### Pourquoi ?

Tout simplement parce que le programme sauvegarde dans l'accumulateur A et à l'adresse 2000 le contenu de la variable SCDR. Ce résultat que nous montre l'écran LCD est bien correct car en effet, nous avons réussi d'une part à convertir le caractère saisi au clavier, son code ASCII en tant que caractère affiché, mais également nous obtenons son code ASCII. Comme le montre la photo ci-contre, le caractère 'c' a bien pour code ASCII 63 dans la table universelle.

Et le caractère tapé au clavier est bien un 'c'. La réponse logicielle affichée à l'écran est le résultat d'un assemblage d'instructions validées par notre professeur que sont l'initialisation du LCD, la conversion des octets, l'affichage etc... mais surtout d'une exécution fonctionnelle.



## Réalisation d'un fréquencemètre avec affichage LCD

---

Dans le but de réaliser notre fréquencemètre avec affichage LCD, nous devons donc choisir une méthode de mesure de la fréquence. Nous avons le choix comme nous l'avons vu précédemment entre la méthode par scrutation (dite aussi polling) et la méthode par interruption. La méthode que nous avons retenue, est la **méthode par scrutation**, car elle est plus simple à réaliser. De plus, nous n'avons pas besoin d'effectuer une autre tâche durant la mesure de la fréquence, il nous est donc apparu que la solution par scrutation était la meilleure dans notre cas.

Pour réaliser cet affichage, nous avons donc repris l'ensemble des éléments vus précédemment. C'est dire que **nous avons repris les éléments permettant d'effectuer un affichage sur deux octets**. Car les fréquences que l'on mesure sont stockées sur deux octets. De plus, il faut faire attention comme nous l'avons vu, au fait qu'il existe une limite dans les fréquences que l'on peut mesurer grâce à nos méthodes. L'opération nous a ainsi permis d'afficher le nombre de cycles d'une période correspondant à la fréquence que l'on mesure.

-----

### Conclusion :

---

Ce projet nous a donc beaucoup appris sur le langage assembleur (que nous avons eu l'occasion de voir tout au cours du semestre), mais nous a surtout permis d'appliquer de manière concrète un ensemble d'aspects de l'électronique. Nous avons du apprendre à gérer un projet tel qu'il en existe dans le monde professionnel en essayant d'allier notre savoir et les différentes technologies que nous avons à notre disposition.