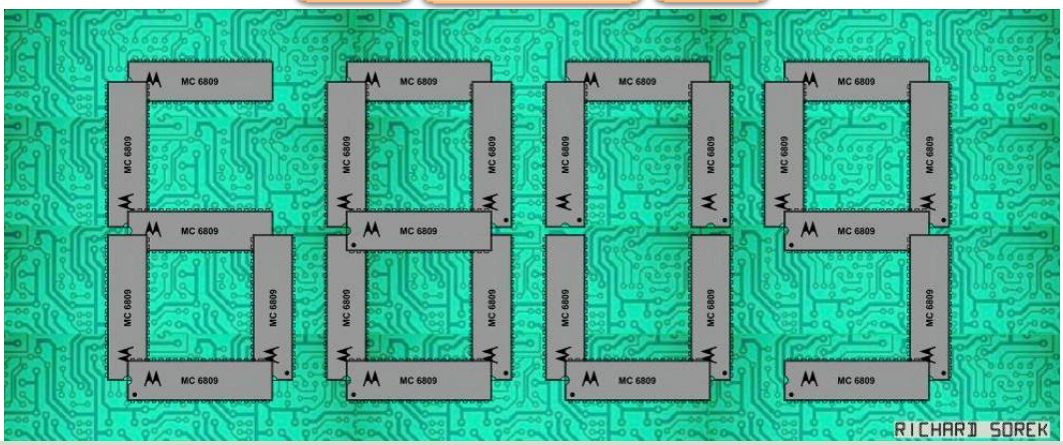


ASSEMBLEUR DU 6809 ET SES PERIPHERIQUES

Préface Sommaire Principal Index

Cet ouvrage de synthèse sur le microprocesseur 6809 et de ses périphériques, est le fruit de plus de trois ans de travail : de mise en page et de création de croquis, durant mes soirées, mes nuits d'insomnies, mes week-ends et mes vacances.

Dans cet ouvrage (sans exception) :

Tous ***les textes*** ont été ***saisis***

Tous ***les tableaux*** ont été ***créés***

Tous ***les croquis*** ont été ***dessinés*** par mes propres soins.

Ceci représente de très nombreuses heures, de très nombreux jours aux services de ce document de travail, ce denier ayant pour seule ambition, d'être un guide de référence pour tous les passionnés du 6809.

Ainsi, j'ai décidé d'offrir gracieusement le fruit de mon travail à toute personne (uniquement pour un usage personnel) qui m'en fera la demande par mail à **keros6809@gmail.com**

En échange, je vous demande d'avoir la grande gentillesse de me rapporter par mail vos corrections éventuelles et/ou vos compléments d'informations, afin de faire vivre ce document. Régulièrement et en fonction des modifications je vous transmettrais, en retour de mail, la dernière version.

Je souhaite remercier chaleureusement **Jacques BRIGAUD** pour les nombreuses corrections.
Ainsi que les modérateurs du forum http://forum.system-cfg.com où vous allez pouvoir y trouver une grande source de renseignements.

Je vous souhaite une bonne lecture et beaucoup de BONHEUR avec le 6809.

Version 4.12 du 30/03/2018
par Richard SOREK

Après avoir recherché de la documentation sur le microprocesseur µp6809, je me suis vite rendu compte que les documents que j'ai eu l'occasion de voir étaient était trop succincts, incomplets et le plus souvent truffés d'erreurs.

C'est pourquoi j'ai créé ce document, il est destiné à la compréhension pragmatique et didactique de l'assembleur du µp6809 et de ses périphériques. Ce travail fut guidé par l'idée d'avoir une documentation précise, détaillée et vivante permettant d'être lue sur des supports électronique moderne (Tablette, Smartphone etc.) et de faire partager ces informations gratuitement.

Le premier document qui a été comparé a mes propres cours datant de mon passage à l'Ecoles des Mines de DOUAI (59500) à été le livre "L'ASSEMBLEUR FACILE DU 6809" de François BERNARD de 1984, trouvé assez facilement sur Internet sur les sites suivant :
http://www.bibliodunet.com/telecharger-livres-gratuits/electricite-electronique-et-electrotechnique/l-assembleur-facile-du-6809.html

http://e-booksland.com/Autres-livres-de-programmation/

Un mot dans le titre de ce livre "……FACILE ……" m'a attiré, ce livre a donc été choisi à tort. Lors de sa lecture, je me suis rapidement rendu compte qu'il comportait abondamment de "BLABLA" inutile et surtout beaucoup d'erreurs, il a donc été fortement condensé et corrigé ("erreurs" ou "coquilles d'impression" ?). Pour information, les 150 pages format A5 du livre ont été résumées en 30 pages format A4 (soit 60 pages format A5, soit plus de la moitié du livre initial).

Puis ces 30 pages, de format A4, ont été agrémentés d'illustrations plus explicatives et pragmatiques, ces dernières sortent de mes cours personnel sur le µp6809 (mon passage en 1983 à l'Ecoles Des Mines de DOUAI). Tous ces illustrations (Schémas, Croquis, Tableaux, Diagrammes, …) on été redessinés par mes soins.

Enfin ce travail de condensation et de résumé terminé, il a été complémenté et comparé aux informations trouvées dans les livres suivants :

- **Ouvrage 01** : L'ASSEMBLEUR FACILE DU 6809" de François BERNARD de 1984 (Lu entièrement, de très nombreuses erreurs)
- **Ouvrage 02** : LE MICROPROCESSEUR 6809 – SES PERIPHERIQUES ET LE PROCESSEUR GRAPHIQUE 9355-66 de Claude DARDANNE de 1991 neuvième édition (quelques erreurs rencontrées dans cet ouvrage)
- **Ouvrage 03** : PROGRAMMATION DU 6809 de Rodnay ZAKS de 1983
- **Ouvrage 04** : MICROPROCESSEURS : DU 6800 AU 6809 MODES D'INTERFACE de Gérard REVELLIN de 1981 (au 01/04/2013 lu partiellement, ultérieurement si nécessaire il restera à implémenter des exemples d'interfaces à des composants électronique)
- **Ouvrage 05** : ETUDES AUTOUR DU 6809 (CONSTRUCTIONS ET LOGICIELS) de Claude VICIDOMINI (2ième édition du hors série de la revue LED)
- **Ouvrage 06** : PROGRAMMATION EN ASSEMBLEUR 6809 de BUI MINH DUC édition EYROLLES de 1983, ce livre est très riche en informations de qualités, mais sa présentation beaucoup trop compacte fait malheureusement de ce livre un ouvrage "imbuvable, monotone et très fastidieux".
- **Ouvrage 07** : LE MICROPROCESSEUR 6809 DE MOTOROLA, document trouvé sur Internet, Circuit d'ordinateur, partie 3 chapitre 4.2.1 les circuits de la famille des microprocesseurs de la série 68XX.
- **Ouvrage 08** : ASSEMBLEUR ET PERIPHERIQUE DES MO5 ET TO7/70 (Guide pratique) de Frédéric Blanc et de François Normand aux éditions du P.S.I. de 1985.

IMPORTANT
A l'inverse des autres livres, la présentation de ce document favorise la présentation sous forme d'indentation, agrémentée de nombreux croquis.

Ce document n'a pas la vocation principale à être imprimer, mais plutôt a être consulté sur les supports modernes, tel que les Ordinateurs, Tablettes ou Smartphones. A ce titre il a été truffé de signets HTML et de liens Hypertexte pour une navigation et une recherche dynamique interne plus aisée que dans un document papier.

Je précise que ce document n'a aucun et n'aura jamais de caractère commercial.

Au vu de l'obsolescence des ouvrages cités ci-dessus (plusieurs dizaines années), mon travail de synthèse et de regroupement des informations est une façon de refaire vivre un des vétérans de la micro-informatique.

Ce document est à l'attention d'une poignée de personnes et **uniquement pour un usage personnel**.

POUR DES MODIFICATIONS OU DES CORRECTIONS DE CE DOCUMENT

Toutes personnes découvrant des erreurs (voir des fautes d'orthographes), peuvent m'envoyer leurs corrections et leurs améliorations par mail. Après vérifications, je me ferai un devoir et un plaisir d'en apporter rapidement la correction et de vous renvoyer une nouvelle version, pour ce faire :

- Effectuer votre correction :
  - Soit en imprimant la page concernée, d'apporter vos remarques de façon manuscrite et de façon lisible et en caractère d'imprimerie puis Scanneriez la page en format PDF
  - Soit en mettant vos annotations en caractère rouge directement dans une des pages en HTML et imprimer dans un fichier au forma PDF
- Envoyer le fichier PDF par mail à l'adresse **keros6809@gmail.com**
- **IMPORTANT :** Ne pas oublier de me préciser votre adresse mail sur les pages modifiées.

A signaler que ce document n'est pas figé, il connaîtra dans l'avenir quelques ajouts et modifications dans un but de clarification, de réduction et/ou de complémentarité des renseignements fournit.

Je souhaite une bonne lecture aux amoureux du

Et de ses périphériques bien évidement.

Je vous invite vivement à me faire remonter toutes : vos remarques, vos corrections, vos améliorations, vos adjonctions d'information, afin que ce document numérique devienne une référence pour les utilisateurs passionnés du µp6809.

Richard SOREK

LISTE DES MISES A JOUR (AJOUTS ET/OU CORRECTIONS)

Sommaire Index

03/04/2013	Modification des explications des instructions PSHS PSHU et PULS PULU suite à des erreurs de l'ouvrage n°02. Mise en forme de la page d'index. Ajout dans le chapitre des interruptions. Ajout dans le chapitre assemblage. Ajout d'un chapitre "Point de Vue Matériel"
14/05/2013	Début de la comparaison avec l'ouvrage n°05 (ETUDES AUTOUR DU 6809 (CONSTRUCTIONS ET LOGICIELS)
04/01/2014	Ajout de textes d'explication concernant le 6809 et le 6821 en fonction de la documentation de la société SGS Thomson (EFCIS) ouvrage n°07. Egalement plusieurs corrections pour les parties µp6809 et 6850.
30/01/2014	Comparaison avec l'ouvrage n°06 (PROGRAMMATION EN ASSEMBLEUR 6809 de BUI MINH DUC édition EYROLLES de 1983).
10/02/2015	Version 3.00 Ajout de quelques précisions et diverses corrections et 1ière phase de stabilisation.
07/03/2016	Modification suivant relecture par Jacques BRIGAUD du forum http://forum.system-cfg.com
22/01/2018	Corrections fautes d'orthographes et mise à jour en fonction de l'ouvrage n°08
26/02/2018	Modification suivant les corrections, précisions et les ajouts de Jacques BRIGAUD du forum http://forum.system-cfg.com
12/03/2018	Un très grand merci à Jacques BRIGAUD pour son aide très précieuse. Pour la relecture complète et les nombreuses corrections et ajouts dans ce document.

SOMMAIRE

Index | Préface | Liens Rapides

INDEX

Sommaire Principal | Préface | Liens Rapides

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

A

B

Index Sommaire Principal Liens Rapides

C

D

E

Index Sommaire Principal Liens Rapides

F

G

H

I

J

K

Index | Sommaire Principal | Liens Rapides

L

M

N

O

P

Index | Sommaire Principal | Liens Rapides

Q

R

S

Index Sommaire Principal Liens Rapides

T

U

V

W

X

Y

Z

Index Sommaire Principal Liens Rapides

Vous trouverez ci-dessous :

Sommaire Principal | Index | Liens Rapides

Sommaire Principal | Index | Liens Rapides

Sommaire Principal | Index | Liens Rapides

Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

INSTRUCTIONS DE CHARGEMENT

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

6821

6850

6840

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

6829

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

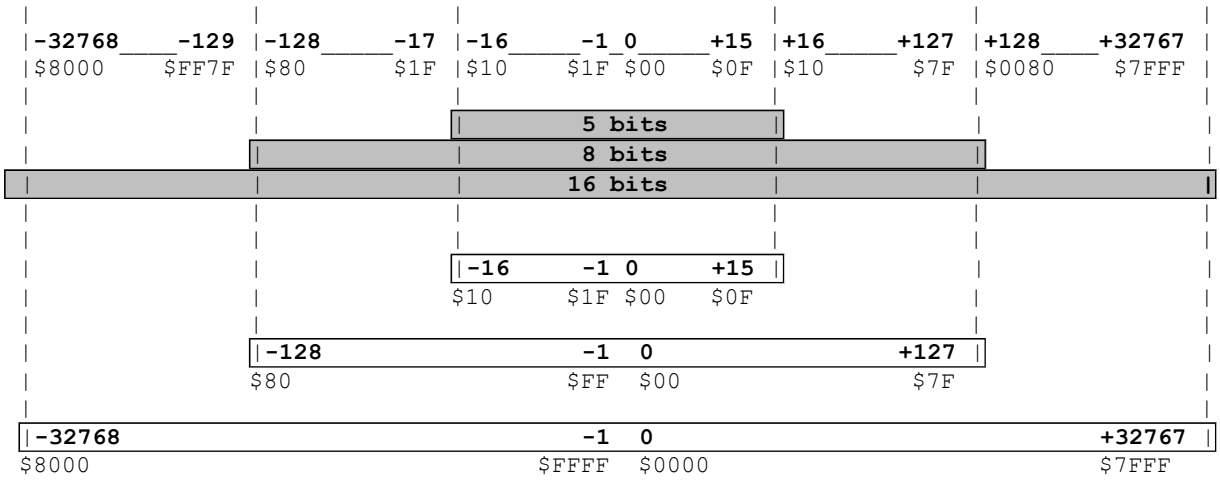
Somm.. Instruc.. | Sommaire Principal | Index | Liens Rapides

ARI : Exemple d'écriture

Préfixe	Nombre	Suffixe	Base	Plage
..... ou &.....	Décimaux	.....T	10	[0 .... à .... 65 535]
@	Octal	.....Q	8	[0 .... à .... 177 777]
%	Binaire	.....B	2	
$	Hexadécimal	.....H	16	[0 .... à .... FFFF]

Sommaire Principal | Index | Liens Rapides

ARI : Nombres Signés sur 5, 8 ou 16 Bits



retour au Sommaire | Liens Rapides | Index

ARI : Notion d'Addition sur les nombres Binaires

Exemple : Addition de $05 et $17

Rappel en binaire 0+0=0 0+1=1 1+0=1 1+1=1

```
   $05        05        0000 0101
 + $17      + 23      + 0001 1100
---------  ------  -------------
 = $1C      = 28      = 0001 1100    = $1C
```

ARI : Notion de Soustraction sur les nombres Binaires

Un microprocesseur ne sait pas faire de soustraction, il ne fait que des additions.
En arithmétique décimal : on additionne au premier l'inverse du second. 13 – 10 ≡ 13 + (-10)
En arithmétique binaire on additionne avec le complément à deux $20 - $15 ≡ $20 + ($-15)

ARI : Représentation des Nombres Négatifs

Considérons l'opération 0 – 1 = -1

```
    0      0000  0000
  - 1      0000  0001
-----      ----------
= - 1      1111  1111    =  $FF   ( -1 sera représenté par $FF )
```

Considérons l'opération -1 – 1 = -2

```
   -1      1111  1111
  -1       0000  0001
-----      ----------
= - 1      1111  1110    =  $FE   ( -2 sera représenté par $FE )
```

ARI : Nombres NON Signés

Les nombres non signés vont de **[0..... à .....+255]** pour le 8 bits. [$00 .....à .....$FF]
Les nombres non signés vont de **[0..... à .....+65535]** pour le 16 bits. [$0000 .....à .....$FFFF]
Le bit 7 (en 8 bits) ou le bit 15 (en 16 bits) n'a pas de rôle dans une représentation non signée.

ARI : Nombres Signés sur 5 Bits

Les nombres signés 8 bits vont de **[-16..... à .....+15]**
Le bit de poids fort (bit 4) sert de signe :

- **Bit 4 = 0** Le nombre est positif. Les chiffres compris entre $00 et $0F
- **Bit 4 = 1** Le nombre est négatif. Les chiffres compris entre $10 et $1F

```
                 { | 0 0000 |     0  $00
                 { | 0 0001 |     1  $01
16 octets ------ { | 0 .... |     .  .
Positifs         { | 0 .... |     .  .
                 { | 0 1111 |    15  $0F
              -----------------------------
                 { | 1 0000 |   -16  $10        -16+32=16     16=$10
                 { | 1 .... |     .  .              .            .
16 octets ------ { | 1 .... |     .  .              .            .
Négatifs         { | 1 .... |     .  .              .            .
                 { | 1 1111 |    -1  $1F         -1+32=31     31=$1F
```


ARI : Nombres Signés sur 8 Bits

Les nombres signés 8 bits vont de **[-128..... à .....+127]**
Le bit de poids fort (bit 7) sert de signe :

- **Bit 7 = 0** Le nombre est positif. Les chiffres compris entre $00 et $7F
- **Bit 7 = 1** Le nombre est négatif. Les chiffres compris entre $80 et $FF

```
                 { | 0000 0000 |     0  $00
                 { | 0000 0001 |     1  $01
128 octets ----- { | 0... .... |     .  .
Positifs         { | 0... .... |     .  .
                 { | 0111 1111 |  +127  $7F
              -----------------------------
                 { | 1000 0000 |  -128  $80        -128+256=128     128=$80
                 { | 1... .... |     .  .              .              .
128 octets ----- { | 1... .... |     .  .              .              .
Négatifs         { | 1... .... |     .  .              .              .
                 { | 1111 1111 |    -1  $FF         -1+256=255     255=$FF
```


ARI : Nombres Signés sur 16 Bits

Les nombres signés 16 bits vont de **[-32 768..... à .....+32 767]**
Le bit de poids fort (bit 15) sert de signe :

- **Bit 15 = 0** Le nombre est positif. Les chiffres compris entre $0000 et $7FFF
- **Bit 15 = 1** Le nombre est négatif. Les chiffres compris entre $8000 et $FFFF

```
                 { | 0000000 00000000 |      0  $0000
                 { | 0000001 00000001 |      1  $0001
32Ko octets ---- { | 0....... ........ |      .  .
Positifs         { | 0....... ........ |      .  .
                 { | 0111111 11111111 | +32767  $7FFF
              --------------------------------------------
                 { | 1000000 00000000 | -32768  $8000   -32768+65536=32767
                 { | 1....... ........ |      .  .                  . 32767=$8000
32Ko octets ---- { | 1....... ........ |      .  .                  .
Négatifs         { | 1....... ........ |      .  .                  .
                 { | 1111111 11111110 |     -2  $FFFE
                 { | 1111111 11111111 |     -1  $FFFF       -1+65536=65535
                                                                65535=$FFFF
```

Exemple en partant d'un nombre en HEXA

```
$FF46 = 65350
        65350 - 65536 = -186      $FF46 = -186
```

ARI : Notion De La Notation En Complément à 2

Respecter les 3 étapes

1°) Mettre le nombre dans un format binaire 8 bits ou 16 bits.
Ex : 17 = `%10001` on va donc l'écrire sous le format `%0001 0001`

2°) Ecrire le complément en inversant chaque bit (un 0 devient un 1 et inversement)
Ex : 17 `%0001 0001` devient `%1110 1110`

3°) On ajoute +1 au nombre

```
  1110 1110
+ 0000 0001
-----------
= 1110 1111
```

L'ordinateur considéra que ce résultat est l'opposé de 17 donc -17

Ce qui donne la représentation binaire -X

Exemple : on recherche la représentation binaire de **-2**

```
2  =  0000  0010

      1111  1101    <-- inversion de tous les bits
            + 1
------------
= 1111  1110     =  $FE  = -2     (pour info -1 sera représenté par $FF)
```

Par la méthode ou notation en complément à 2, il n'est donc possible que de coder des nombres décimaux entre : - 127 et + 127

Rappel : En 8 bits le bit de poids fort (bit 7) de chaque octet est :
b7 = 0 le nombre est POSITIF
b7 = 1 le nombre est NEGATIF

Cette notation en complément à deux n'est pas obligatoire, c'est au programmeur de décider si les nombres qu'il utilise sont compris entre :
0 et 255 (chiffres NON signés)
-127 et +127 (chiffres signés).

ARI : Code BCD

Chaque chiffre compris entre 0 et 9 est codé par un groupement de 4 bits.

Binaire	BCD	
0000	0	
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	.	}
1011	.	}
1100	.	} --------Inutilisé
1101	.	}
1110	.	}
1111	.	}

Exemple : le chiffre décimal **16** sera représenté sous la forme :

```
0001 0110  en BCD
  1    6

au lieu de

0001 0000  en hexadécimal
```

DIV : Quelques renseignements sur le 6809

retour au Sommaire | Liens Rapides | Sommaire Principal | Index

Le 6809 fut introduit vers **1977-1978**, il est le plus puissant microprocesseur 8 bits de l'époque.
Le créateur est la Sté MOTOROLA. Le circuit EF6809 était fabriqué par Thomson-EFCIS. A noter que la société HITACHI a sortie le 6309 une version amélioré.
La puissance dissipée est de 1W max sous 5V

Le 6809 est compatible avec le 6800 sur le plan du code source, même si le 6800 a 78 instructions alors que le 6809 n'en a que 59. Certaines instructions sont remplacées par d'autres qui sont plus générales, et d'autres furent même remplacées par des modes d'adressage.

Les instructions du µp6809 sont un ensemble d'octets de 0 à 255 pour les mnémoniques à simple octet.
Le µp6809 possédant plus que 256 instructions il faudra parfois 2 octets pour coder les instructions.
Ce microprocesseur comporte :

59 instructions de bases différentes,
Combinées aux **9 modes d'adressage**,
On arrive à **1464 instructions différentes**.
Des instructions spécialisées dans le traitement d'information 16 bits

Fabriqué en technologie MOS canal N, il intègre 3 bus indépendants

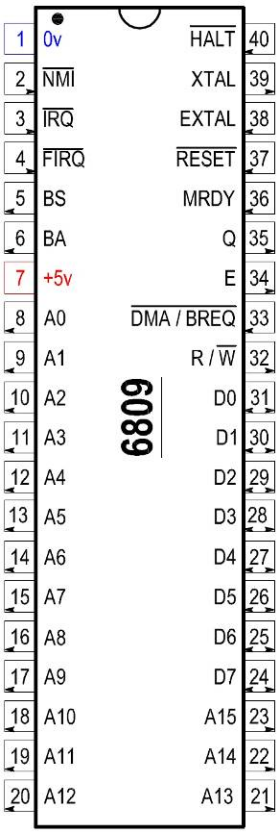
- Le bus des Données sur 8 bits
  (Bidirectionnel à 3 états, chaque broche peut piloter l'équivalent de 8 charges TTL)
- Le bus des Adresses sur 16 bits
  (Unidirectionnel, en sortie à 3 états, chaque broche peut piloter 8 charges TTL)
- Le bus Contrôle
  (Sur 10 bits pour le µp6809, sur 12 bits pour le µp6809 E)

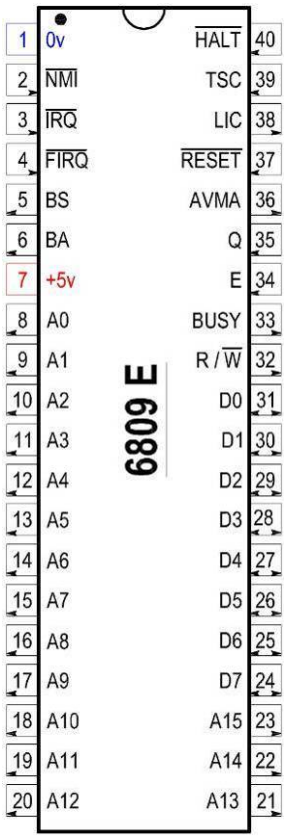
Le 6809 a un générateur de cadence interne qui n'a besoin que d'un cristal externe.
Le 6809E a besoin d'un générateur de cadence externe.
Il y a des variantes où la lettre du milieu indiquait la vitesse d'horloge nominale du processeur.

Version	Fréquence d'entrée	Délivre donc une fréquence de
EF 6809	4 MHz	1 MHz
EF 68 A 09	6 MHz	1,5 MHz
EF 68 B 09	8 MHz	2 MHz

DIV : Brochages du 6809 et du 6809 E

Index | retour au Sommaire | Liens Rapides





DIV : Signification des Diverses Broches

Index | retour au Sommaire | Liens Rapides

Convention d'écriture : FIRQ | ≡ $\overline{\text{FIRQ}}$

Broches	6809	6809E	
1	0v (ou Vss)	0v (ou Vss)	relié à la masse
2	NMI|	NMI|	ligne d'interruption non masquable
3	IRQ|	IRQ|	ligne d'interruption la moins prioritaire
4	FIRQ|	FIRQ|	ligne d'interruption rapide
5	BA	BA	Bus Available Bus Accordé ou bus libre
6	BS	BS	Bus State Etat du Bus
7	Vcc	Vcc	relié au +5v (+/- 5%)
8 à 23	A0 à A15	A0 à A15	Bus d'adresses
24 à 31	D0 à D7	D0 à D7	Bus de données
32	R/W|	R/W|	Read / Write|
33		BUSY	Facilite les applications multiprocesseur
33	DMA| / BREQ|		DMA et rafraîchissement mémoire
34	E	E	Signal horloge du système
35	Q	Q	Signal horloge, en quadrature avec E
36	MRDY		
36		AVMA	
37	RESET|		broche d'initialisation Hard du µp6809 (actif sur un niveau bas)
38	EXTAL		
38		LIC	
39	XTAL		
39		TSC	
40	HALT|		

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

DIV : Fonctionnement des broches BA et BS

DIV : La broche de sortie BA

Broche 6 (Bus Available) Bus accordé ou Bus libre

Indique qu'un signal de commande interne fait passer les bus d'adresses et de données à l'état "Haute impédance".
Le signal de disponibilité du Bus (BA) indique la présence d'un signal de contrôle.
BA = 1 ⇒ A0-A15, D0-D7 et R/W\| dans l'état de haute impédance.

Ce signal est très utile pour les applications possédant un périphérique capable de gérer les bus adresses et données à la place du µp6809 (un DMA par exemple).
Il n'implique pas que le bus soit disponible pendant plus d'un cycle.
Quand BA passe à l'état Bas, il s'écoule un cycle supplémentaire avant que le microprocesseur ne prenne le contrôle des bus.

retour au Sommaire

DIV : La broche de sortie BS

Broche 5 (Bus Status) Etat du Bus

Lorsqu'elle est décodée avec l'état de la sortie BA, représente l'état du µp6809 (validé sur le front montant du signal Q)

Les 4 combinaisons possibles des broches BA et BS permettent de connaître à chaque instant, l'état du µp6809. Ces indications sont validées sur le front montant de la broche Q (pin 35).

Retour Liste Broches | Liens Rapides

BA	BS	Fonctionnement
0	0	Le µp6809 est en fonctionnement **normal**, il gère les bus adresses et données.
0	1	Le µp6809 est en phase de **reconnaissance d'interruptions** pendant deux cycles. Cet état correspond à la recherche matérielle du vecteur interruption : RESET, NMI, SWI, IRQ, FIRQ, SWI2, SWI3 Le µp6809 vient de recevoir une interruption, il recherche le vecteur correspondant. Les 4 lignes d'adresses de poids Faibles indiquent quel est le niveau d'interruption pris en compte et permet une vectorisation par périphériques. Voir "Tableau des Vecteurs d'Interruption" dans le chapitre FEI "Fonctionnement En Interruption".
1	0	**Reconnaissance de synchro externe**, cet état est activé lorsque le µp6809 rencontre l'instruction de synchronisation externe SYNC. Le µp6809 attend cette synchronisation sur une des lignes d'interruption. Les bus sont en haute impédance pendant ce temps.
1	1	**Arrêt ou bus accordé**. Le µp6809 a été arrêté par le signal HALT| Cet état est vrai BA = BS = 1 lorsque le µp6809 est : Dans l'état HALT **Ou** Bus accordé (broche HALT| = 0) Correspond à l'arrêt du µp6809 ou à l'autorisation venant du µp6809 de faire gérer les bus de données et d'adresses par un circuit annexe (ex : DMA). Les bus du 6809 sont en haute impédance, les bus sont disponibles.

DIV : Fonctionnement de la broche R/W | (Read / Write |)

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche 32 (c'est une sortie)

Cette broche indique le sens du transfert des données.
Elle indique à la périphérie si le µp6809 est en lecture ou en écriture de données
Cette broche est une sortie, et est en logique 3 états.

R/W | = 0 (Etat Bas) 0v
Le µp6809 est en écriture sur le bus
Les broches D0 à D7 sont en sorties, Les données Sortent du µp6809

R/W | = 1 (Etat Haut) +5v
Le µp6809 est en lecture sur le bus
Les broches D0 à D7 sont en entrées, Les données Arrivent au µp6809

DIV : Fonctionnement de la broche DMA / BREQ | (uniquement pour un 6809, pas pour le 6809E)

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche 33 (pour un µp6809 c'est une entrée).
(Direct Memory Access / Bus REQuest) Accès direct à la mémoire / Demande de Bus

DMA et rafraîchissement mémoire : L'entrée permet de suspendre l'utilisation des bus par le µp6809, pour faire de l'accès direct ou un rafraîchissement mémoire (mémoires dynamiques).

Lorsque la broche Q est au niveau Haut, le passage à l'état Bas de la broche DMA| / BREAQ| entraîne l'arrêt du programme à la fin de l'instruction en cours.

Les broches BA et BS passent à l'état 1. Indique la prise en compte de la demande faite par DMA| / BREQ| le circuit demandeur aura alors jusqu'à 15 cycle bus avant que le µp6809 ne récupère le bus pour autorafraîchissement.

Lorsque le µp6809 répond BA = BS = 1, ce cycle est un cycle perdu utilisé pour transférer le contrôle au système de DMA.

Les bus de données et adresses ainsi que la broche R/W|, sont à l'état haute impédance, l'horloge interne est bloquée, E et Q continuent à osciller.

Tous les 16 cycles, trois cycles seront nécessaires pour l'autorafraîchissement du µp6809.

Autrement dit : Cette broche permet une utilisation des bus du µp6809 par un circuit extérieur. Les bus sont mis en haute impédance. Exemple d'utilisation de DMA ou du Rafraîchissement mémoire.

Cette entrée offre une méthode de suspension d'exécution et acquisition du bus µp6809 pour une autre utilisation. La transition de la broche DMA| / BREQ| doit se produire pendant le signal Q, un niveau bas sur cette broche arrêtera l'exécution de l'instruction à la fin du cycle en cours.

L'autorafraîchissement nécessite un cycle bus comportant un cycle perdu de début et de fin.

En général le contrôleur DMA fait une demande d'accès au bus en mettant au niveau bas la broche DMA| / BREQ| sur le front montant du signal E.

Les faux accès mémoires doivent être évités pendant tout cycle perdu.

Lorsque la broche BA est remis à 0 (Soit comme résultat de DMA| / BREQ| = 1 ou soit par autorafraîchissement du µp6809), le circuit DMA doit être déconnecté du bus.

Un autre cycle perdu s'écoule avant que le µp6809 ne se voie alloué un accès mémoire pour transférer le contrôle sans litige.

DIV : Fonctionnement de la broche BUSY (pour 6809E)

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche 33 (pour un µp6809E c'est une sortie).
Facilite les applications multiprocesseur (comme la broche AVMA 6809E).

La broche BUSY a le même fonctionnement que la broche AVMA, mais la broche BUSY ne passe à l'état Haut que pendant l'exécution d'une instruction de type lecture, modification, écriture d'une donnée (ASL par exemple).

De cette façon, les zones mémoires ne sont pas adressées simultanément par deux processeurs.

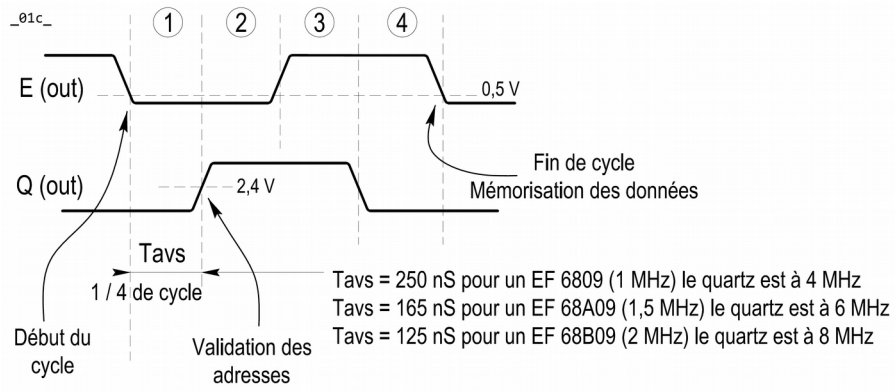
DIV : Fonctionnement des broches E et Q

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

E	Broche 34	Pour un µp6809	(E et Q sont des sorties)
Q	Broche 35	Pour un µp6809E	(E et Q sont des entrées)

Les adresses du µp6809 sont correctes à partir d'un front montant de la broche Q. Les données sont mémorisées sur un front descendant de la broche E.

E (out) : Signal horloge système (synchronisation avec la périphérie), dont la fréquence est celle de base du µp6809. La broche E est identique au signal d'horloge **Φ2** du processeur EF6800
Q (out) : Signal horloge, en quadrature avec la broche E.
Les adresses du µp6809 sont correctes et validées à partir d'un front montant de la broche Q.
Les données sont mémorisées sur un front descendant de la broche E.
La broche Q n'a pas d'équivalence avec le processeur EF6800



DIV : Fonctionnement de la broche MRDY

Broche 36 (pour un µp6809 c'est une entrée) MRDY (Memory Read)
Cette entrée de commande permet l'allongement de l'horloge E pour utiliser des mémoires lentes (temps d'accès aux données augmenté).

L'allongement de E est un multiple de 1/4 de cycle bus, sa valeur maximum est de 10µs. Dans les phases VMA, la broche MRDY n'a pas d'effet sur E.

Si la broche MRDY est à l'état :

- Haut, le signal E est fonctionnement normal.
- Bas, le signal E peut être allongé de multiples (entier de 1/4 de cycles bus) permettant ainsi l'utilisation de mémoires lentes.

L'allongement maximum est de 10 microsecondes. Pendant les accès mémoires non utiles la broche MRDY n'a pas d'effet sur l'allongement du signal E.

Ceci évite le ralentissement de la vitesse du processeur pendant les accès bus non utiles.

DIV : Fonctionnement de la broche AVMA (pour 6809E)

Broche 36 (pour un 6809E c'est une sortie)
Broche AVMA (Advanced Valid Memory Adress) Contrôle des ressources communes en multiprocesseur.

C'est une sorte de validation d'adresse mémoire perfectionnée qui passe à l'état 1 au cours du cycle précédant un accès bus par le µp6809.

Le signal permet un contrôle efficace des ressources communes d'un dispositif multiprocesseur.

Si le µp6809 est en HALT| ou SYNC| ou en calcul interne alors la broche AVMA est au niveau Bas.

Cela permet d'optimiser l'échange de ressources communes dans un dispositif multiprocesseur.

DIV : Fonctionnement des broches XTAL et EXTAL

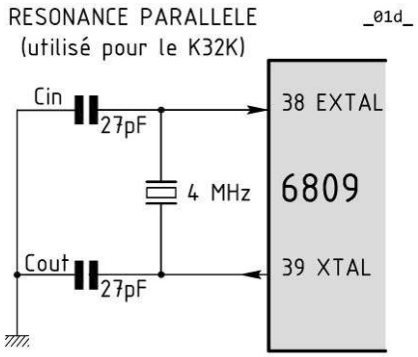
Broche 38 pour EXTAL (pour un 6809 c'est une entrée)
Broche 39 pour XTAL (pour un 6809 c'est une sortie)

Ce sont des entrées horloges, elles sont utilisées pour connecter l'oscillateur interne à quartz.
Le quartz ou la fréquence externe est 4 fois la fréquence du bus.

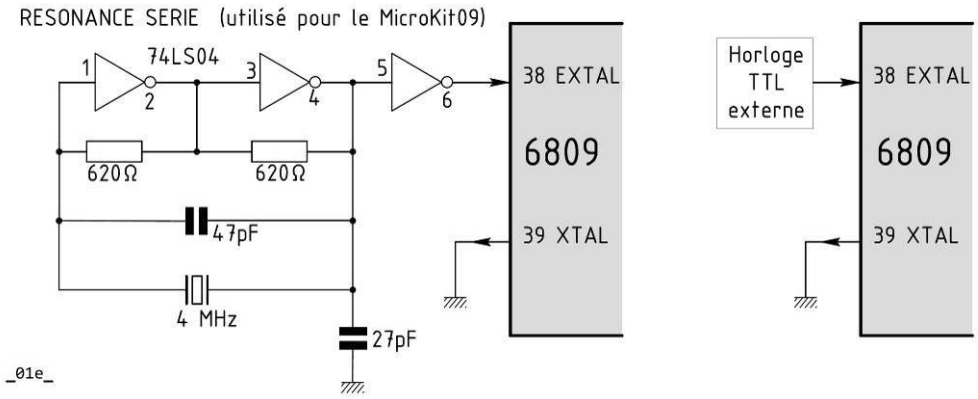
Deux modes de fonctionnement sont possibles :

- L'oscillateur interne est connecté par ces deux broches à un quartz externe et 2 condensateurs (résonance parallèle).

  27 pF dans le cas du kit K32K de DATA RD
  22 pF dans le cas du Microkit09



- Une horloge TTL est connectée aux broches EXTAL, la broche XTAL étant reliée à la masse. Le quartz ou la fréquence externe est quatre fois la fréquence du bus.



DIV : Fonctionnement de la broche LIC (pour 6809E)

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche 38 (pour un µp6809E c'est une sortie)
Broche LIC (Last Instruction Cycle) Dernier cycle d'une instruction.

Cette sortie est à l'état haut pendant le dernier cycle de chacune des instructions exécutées par le µp6809E.

Le cycle qui suit ce signal est donc toujours un cycle de recherche de code opératoire.

La broche LIC est à l'état Haut quand le µp6809 est en attente d'une synchronisation externe (broche SYNC), en phase d'empilage au cours d'une gestion d'interruption, ou dans l'état HALT|

DIV : Fonctionnement de la broche TSC (pour 6809E)

Broche 39 (pour un µp6809E c'est une entrée)
La broche TSC (Three State Control) Contrôle trois états pour 6809E. Contrôle d'état haute impédance des bus.

Cette entrée joue le même rôle que l'entrée DMA| / BREQ| du µp6809 normal.

Quand TSC est à l'état 1, le bus adresse, le bus de donnée et la ligne R/W| sont en haute impédance.

Il est à ce moment possible de faire de l'accès direct ou du rafraîchissement mémoire ou encore de faire la gestion des bus avec un autre µp6809.

DIV : Fonctionnement de la broche HALT |

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche 40 (c'est une entrée)
Broche HALT| (arrêt du µp6809 ou du µp6809E). Cette broche bloque le fonctionnement du µp6809 quand elle est à l'état bas (le 0v).

Cette entrée permet d'interrompre le déroulement d'un programme de façon matérielle (Hardware).
Le µp6809 termine l'instruction en cours, puis positionne les broches BA et BS à 1.

Le déroulement reprend dès que la broche HALT| est à 1 et sans perte d'information.
Les broches IRQ| et FIRQ| sont dévalidées.

Les broches RESET| et NMI| sont valides mais leur traitement ne se fera qu'à la libération du 6809

Un niveau bas sur cette entrée provoque l'arrêt du µp6809 à la fin de l'instruction en cours et il reste à l'arrêt indéfiniment sans perte d'information, le µp6809 reprend la suite du programme dès que la broche HALT | est de nouveau au niveau Haut.

- La broche BS = 1 indiquant que le µp6809 est arrêté ou à l'état bus accordé (bus libre).
- La broche BA = 1 indiquant que les bus sont à l'état haute impédance.

Tant que le µp6809 est à l'arrêt :

- Les demandes d'interruption IRQ| et FIRQ| sont inhibés.
- Les demandes d'accès direct mémoire sont autorisées.
- Les demandes d'interruption prioritaires RESET| et NMI| sont prise en compte mais leur traitement est différé.
- Les horloges fonctionnent normalement.

DIV : Fonctionnement du Bus d'Adresses A0 à A15

retour au Sommaire | Index | Retour Liste Broches | Liens Rapides

(Broches n°8 à n°23)
Lorsque le bus d'adresse n'est utilisé par le µp6809 pour un transfert de données, les broches d'adresse sortent l'adresse $FFFF avec la broche R/W| = 1 et la broche BS = 0.

Les adresses sont validées sur le front montant de la broche Q.

Tous les amplificateurs du bus d'adresses sont mis à l'état haute impédance lorsque la broche de sortie BA est à l'état haut.

DIV : Fonctionnement du Bus de données D0 à D7

(Broche n°31 à n°24)
Bus bidirectionnel, il sert à la transmission de données 8 bits

DIV : Broche d'entrée RESET |

retour au Sommaire | Sommaire | Index | Retour Liste Broches | Liens Rapides

Broche n°37 (c'est une entrée)

Le µp6809 se relance en lecture du vecteur d'initialisation aux adresses $FFFE et $FFFF et dès lors que la condition logique BA = 0 et BS = 1 (reconnaissance d'interruption).

Le µp6809 charge ce vecteur dans le compteur programme PC et se place en exécution.

Un niveau Bas  (pendant un temps supérieur à un cycle du µp6809) sur cette broche entraîne l'initialisation complète du µp6809. Le travail en cours est totalement perdu.

- L'instruction en cours est arrêtée.
- Le registre de page DP est mis à zéro.
- Les interruptions IRQ| et FIRQ| sont masquées.
- L'interruption non masquable NMI| est désarmée.

Cette séquence d'initialisation dure environ une dizaine de cycles processeurs.
L'adresse de départ du programme de traitement du RESET est donnée par le contenu des adresses :

Vecteurs d'Interruption

`{$FFFE} + {$FFFF}` = adresse du programme qui sera lancé après un RESET
LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

L'adresse constituée par ces deux octets mémoire est chargée dans le compteur programme PC puis le µp6809 exécute le programme à partir de cette adresse.

Un simple réseau RC peut-être utilisé pour initialiser l'ensemble du système puisque l'entrée RESET| possède un trigger de Schmidt dont la tension de seuil est supérieure à celle des périphériques 6800. L'entrée RESET| sera donc active plus rapidement sur les périphériques que sur le processeur.

De cette façon, lorsque le µp6809 commence le programme d'initialisation, on est assuré que tous les périphériques ont terminé leur phase de mise sous tension.

Toute mise sous tension du µp6809 commence automatiquement par cette adresse du vecteur RESET en $FFFE et $FFFF.

Il est à préciser ici qu'il y a plusieurs **Assembleur**, les explications qui suivent sont donc d'ordre générale.

Sommaire Principal

Inconvénients du langage machine

Impossibilité de portabilité sur d'autre machine.
Programmes difficiles à lire et à corriger.
Calculs arithmétiques difficiles, la gestion de nombre en virgule flottante est compliquée.
Apprentissage plus important, le langage machine demande beaucoup de rigueur et d'attention.
Structuration difficile d'un programme.

retour au Sommaire

Avantages du langage machine

Vitesse d'exécution maximale
Utilisation plus rationnelle de la taille mémoire, un programme en langage machine occupe moins de place en mémoire.
Possibilité de création de fonctions irréalisables en BASIC.
Utilisation de toutes les ressources de l'ordinateur.
Possibilité de mêler BASIC et langage machine

retour au Sommaire | Index | Liens Rapides

GEN : L'ASSEMBLAGE

GEN : Assemblage : Généralités

Un programme assembleur permet de traduire les mnémoniques d'un programme SOURCE écrit en ASCII en un programme OBJET en HEXA directement exécutable.
L'assembleur accomplit deux tâches principales :

- Il traduit les codes opérations mnémoniques en équivalent binaire.
- Il traduit les symboles utilisés pour les constantes et les adresses en équivalent binaire.

Programme SOURCE

Écrit en mnémonique, listing du programme lisible (n'est pas exécutable directement)

Le programme OBJET

Génère le code OBJET à partir du programme source
Le code OBJET est directement exécutable par le µp6809.

Lors de l'assemblage il se produit deux choses :

- Production d'un listing du programme avec :
  - A droite le programme source inchangé avec en plus une numérotation des lignes.
  - A gauche se trouvent deux colonnes avec l'implantation en mémoire des codes opérations suivi du code machine correspondant au programme.
  - Une détection des erreurs éventuelles.

- Génération d'un code objet destiné à être stocké, avec :
  - Le code machine
  - Son adresse d'implantation en mémoire
  - Son adresse de lancement.

L'assemblage s'effectue généralement en passes (deux lectures) :

- Première passe :
  - Lecture de chaque nom de constante (ou symbole) afin d'établir une table des symboles
  - A chaque étiquette l'assembleur assigne l'adresse qu'occupera le code opération (mnémonique) présent sur la même ligne.
  - Le pointeur d'adresse est fixé à une valeur origine au début du programme
  - A chaque instruction ce pointeur est incrémenté selon la longueur de l'instruction
  - Pendant cette première passe l'assembleur détecte les erreurs de syntaxe.

- Durant la seconde passe :
  - L'assembleur fait l'assemblage proprement dit afin de générer le code objet.
  - Chaque fois que l'assembleur rencontre un nom de variable ou une étiquette, il recherche dans la table et affecte l'adresse ou la donnée stockée lors de la première passe.

GEN : Assemblage : les Erreurs de syntaxe

retour au Sommaire Liens Rapides

Les messages d'erreurs peuvent varier d'un assembleur à l'autre. Très souvent une lettre est affectée à chaque type d'erreur et est placée dans la marge en face de la ligne erronée.

Exemples :	UNDEFINED OPERATION CODE	rencontre d'un faux code mnémonique"
	ILLEGAL FORMAT	l'opérande n'est pas correcte
	…	

GEN : Assemblage : les Erreurs de second niveau

Généralement plus difficiles à corriger.

Exemples :	MISSING LABEL	étiquette absente
	DOUBLE DEFINITION	deux valeurs sont assignées au même nombre
	UNDEFINED NAME	variable utilisée n'a pas été définie

GEN : Assemblage : les Erreurs de conception

Ce sont des erreurs ne faisant pas partie des deux cas si dessus et donc l'assembleur ne détectera pas. Liées à une mauvaise conception du programme ou à la mauvaise traduction de l'organigramme en programme source. Simplicité, clarté, solutions sans astuces seront les recommandations élémentaires.

GEN : Assemblage : les Editeur-assembleurs

retour au Sommaire Liens Rapides

Avant d'être assemblé le programme source doit être introduit au clavier par un Editeur et doit pouvoir être modifié à volonté
Le débogueur (Debugeur en anglais) est un logiciel qui aide le développeur à analyser les bogues (les erreurs) d'un programme, il est un parfois joint à l'Editeur-assembleur.

Le débogueur possède les facilités suivantes :

- Affichage des blocs mémoires pour vérification du code objet généré et des résultats qui seront produit.
- Affichage de tous les registres internes du µp6809.
- Un mode d'exécution pas à pas avec affichage des registres après chaque exécution d'une instruction.

Un CROSS-ASSEMBLEUR : le programme SOURCE est écrit sur une machine (exemple un PC) et le programme OBJET sera utilisé sur un système construit autour d'un microprocesseur.

GEN : Assemblage : les Macro-assembleurs

En plus des fonctions d'un éditeur classique les macro-assembleurs donnent la possibilité de définir des ordres macro. Le sujet étant également très vaste, je vous reporte à l'ouvrage n°06 entre la page II.71 à la page II.97 (L'ouvrage n°06 est décrit dans ma préface en page 2)

retour au Sommaire Index Liens Rapides

GEN : Assembleur Minimal

Appelé aussi micro assembleur ou assembleur résident, parce qu'il est petit et résident en machine.

Le rôle de l'assembleur est de traduire le code machine (la suite d'instructions) en code binaire équivalent et de le ranger dans les postions qui lui sont indiquées.

Il peut être du type :

- Interactif (traduction ligne par ligne)
- À une seule passe (traduction en entier en un seul passage)

Ce qui caractérise un assembleur minimal est l'absence du champ étiquette qui permet d'affecter un nom (d'un certain nombre de caractères) à une adresse ou une donnée.

GEN : Structure d'un listing Assembleur

GEN : Listing Assembleur, Généralité

Certain ASSEMBLEUR évolué distingue les majuscules et les minuscules, très pratique pour le nom des étiquettes.

L'assembleur standard reconnaît la majorité des caractères ASCII (American Standard Code for International Interchange), l'alphabet principal comprend :

- Les majuscules de A à Z (les minuscules sont pour le champ commentaires)
- Les chiffres de 0 à 9
- Le séparateur de champ : . . . . . l'espace sodé $20 en ASCII
- Le retour chariot ou retour de ligne : codé $0D en ASCII
- Les signes principaux : . . . . . + - * / # $ % & @ ' , < > [ ]

Les minuscules et les signes secondaires peuvent être introduit dans un opérande à condition d'être précédé par une apostrophe ‘ dans ce cas le caractère derrière l'apostrophe sera remplacé lors de l'assemblage par le code ASCII en hexa.

‘ **!** Remplacé par $21
‘a Remplacé par $61

L'alphabet du champ commentaire est plus vaste, il contient tous les caractères visualisables (signes secondaires, ainsi que les minuscules. Deux exceptions sont à retenir :

- Les minuscules et les signes secondaires peuvent être introduits dans un opérande à condition d'être précédé par des apostrophes '. dans ce cas ils seront remplacé à l'assemblage par leur code équivalent ASCII (exemples : '! remplacé par $21 !a remplacé par $61)
- La directive FCC, qui manipule les chaînes de caractères, accepte évidemment l'alphabet élargi.

Une instruction écrite en assembleur standard doit comporter au moins 4 composantes appelées champs :

- Champ Etiquette (ou label)
- Champ Instruction (ou mnémonique)
- Champ Opérande
- Champ Commentaire

Le nombre total de colonnes est fonction de l'Assembleur utilisé, il peut pour certain être fixé entre 50 et 120 avec la directive d'assemblage OPT selon les possibilités offertes par les terminaux greffés sur le système.

- A 50 colonnes, tous les commentaires de ligne disparaissent.
- A 80 colonnes, nombre standard, il reste 30 emplacements pour le commentaire si le champ opérande est court.

A la fin du listing, l'assembleur signale le nombre total d'erreurs et d'avertissements relevés durant la phase d'assemblage.

retour au Sommaire Liens Rapides

GEN : Table des références croisées

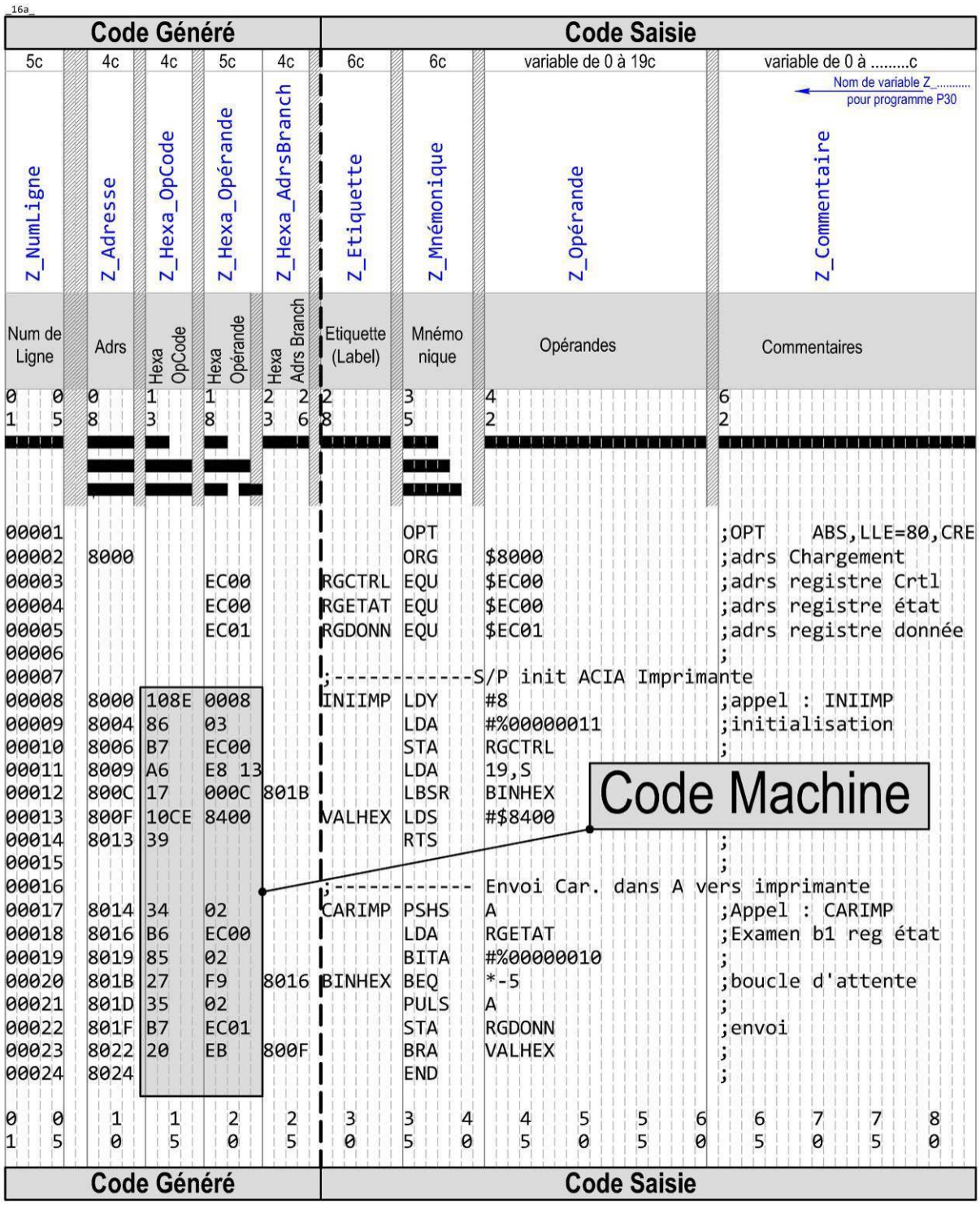
A la fin des listings d'Assemblage on édite une table des références croisées.

Dans cette table on y retrouvera tous les symboles définit pas le programmeur (Etiquette et Nom de constante), le plus souvent classé selon leur ordre alphabétique.
Pour certain Assembleur on trouvera :

- A gauche de ces symboles on trouve les données absolues affectées à leur définition.
- A droite, figurent tous les numéros des lignes leur faisant référence. Ces numéros sont également classés par valeurs croissantes.

Le symbole astérisque ; indique le numéro de la ligne ou le symbole a été défini.
L'utilité d'une telle table dans un gros programme est indéniable.

GEN : Exemple de programme Assemblée (Organisation des Colonnes)



GEN : (Col. 1 à 5) Numéro de Ligne

L'assembleur renumérote toutes les lignes existant dans le listing source.

GEN : (Col. 6) Section Absolue

La lettre A signifie "section Absolue".
Elle n'a de signification que pour les programmes compilés en code translatable puis reconfigurés par l'éditeur de liens.

GEN : (Col. 8 à 11) Adresses Absolues (Z_Adresse)

On y trouve les adresses absolues d'implantation des instructions où se trouvera éventuellement le code binaire exécutable si le programmeur désire charger son programme dans l'espace mémoire pour exécution.
Ces colonnes sont vides pour des lignes de commentaires ou pour des lignes contenant des directives n'invoquant aucun emplacement mémoire du type OPT, END ...

GEN : (Col. 13 à 16) OpCode en Hexa (appelé Op-code ou instruction) (Z_Hexa_OpCode)

Entre dans le code machine exécutable.
Contiennent le code opération ou Op-Code du µp6809, résultat de la traduction des mnémoniques standards, c'est le code généré par l'assembleur.
Certains Op-Codes requièrent 2 octets, ce qui explique les 4 emplacements prévus.

retour au Sommaire

GEN : (Col. 18 à 21 ou 22) Opérande en Hexa (Z_Hexa_Opérande)

Liens Rapides

Entre dans le code machine exécutable.
Contiennent le Code opérande du µp6809 qui peut être de 0, 1 2 ou 3 octets.
Lorsque l'opérande requiert 3 octets, **le post-octet** occupe les colonnes 18 et 19.
Pour les opérandes 16 bits, elles occuperont les colonnes 18,19 et 21,22.

GEN : (Col. 23 à 26) Adresse de Branchement en Hexa (Z_Hexa_AdrsBranch)

Quand il s'agit d'un branchement, les colonnes 23 à 26 fournissent l'adresse absolue de destination, ce qui évite tout calcul de la part de l'opérateur. L'affichage de cette adresse dépend de l'assembleur utilisé.

Exemple la ligne 24, $8011 représente l'adresse sur laquelle about le branchement BEQ *-5.
Ce type de rappel est de grande utilité pour la mise au point "MauP" des programmes.

On note un apparent recouvrement des colonnes avec le code opération précédent. En réalité, une instruction ayant 3 octets dans le code opérande n'est jamais une instruction de branchement, donc l'assembleur n'aura pas à considérer ce cas.

GEN : (Col. 26)

Dans cette colonne apparaît de nouveau la lettre A qui est un rappel de la colonne 6.

retour au Sommaire

GEN : (Col. 28 à 33) Champ étiquette (ou label) (Z_Etiquette)

Liens Rapides

Repère la position des instructions dans le programme.

De 1 à 6 caractères Maxi avec une lettre Majuscule pour le premier caractère ou le caractère " _ " code ASCII 95 ou $5F.
Dans les assembleurs plus récents le nombre de caractères l'étiquette peut aller au dessus de 6.

Utilisé pour les instructions de saut conditionnel ou inconditionnel et pour les sous-programmes.
Mettre des noms de label compréhensifs.

Ce champ accepte :

- Les majuscules A à Z
- Les chiffres 0 à 9
- Le point .
- Le signe $
- Le soulignement _

Eviter de mettre les caractères suivant : **% ; $**

Les symboles réservés pour désigner les registres internes PC, Y, X, S, U, DP, A, B, D, CC et PCR ne peuvent être employé comme étiquette.

L'étiquette ne doit apparaître qu'une seule fois dans tout le programme. Sauf pour les étiquettes définit par la directive SET.

Les symboles des directives d'assemblage sont à éviter comme étiquette pour une raison de clarté.

Dans un format libre, l'étiquette, si elle existe, commence à la première colonne. Si la première colonne contient un espace, l'instruction est considérée comme dépourvue d'étiquette.

Le signe point virgule ; en colonne 28 convertit toute la ligne en commentaire. L'assembleur n'affecte pas de code objet à cette ligne. C'est l'un des moyens utilisés pour créer des lignes blanches.
La colonne 28 est la colonne normalement utilisée pour les étiquettes.

Pour des structures de sélection multiple avec beaucoup de branchements imbriqués, une numérotation peut être envisagée (POT1.1 POT1.2 POT2.1 POT2.2 ....)

Eviter le vocabulaire peu suggestif du type (TOTO, TATA, ESSAI, CHOSE, TRUC, XXX, ...)

Faire attention à des ressemblances formelles du type : O et 0, 1 et l, Z et 2.
Lors d'un chiffrage, une bonne habitude consiste à utiliser deux chiffres (00, 01, 02, ...)

GEN : (Col. 35 à 40) Champ Mnémonique (appelé aussi instructions) (Z_Mnémonique)

retour au Sommaire Liens Rapides

On appel aussi ce champ **CodeOp** (Code Opératoire)
Contient le code mnémonique de l'opération qui peut être :

- Une vraie opération c'est à dire celle qui possède un code binaire (LDA, STA,...).
- Une pseudo opération, encore appelée directive d'assemblage (OPT, ORG, EQU, END,...) qui ne produit en général aucun code objet, mais agit en revanche sur le processus d'assemblage lui-même.
- Un appel de macro-instruction, définit au préalable par le programmeur.

Séparé de la zone Label par un délimiteur le caractère ESPACE

Le champ mnémonique ne comporte que des lettres majuscules de A à Z.

GEN : (Col. 42 à 59) Champ Opérande (Z_Opérande)

Dans les anciens Assembleur à partir de la colonne 42, l'espace restant est partagé entre les champs Opérandes et Commentaires.
Dans le programme EAD (ou P30) que j'ai développé le champ opérande réserve les colonnes 42 à 59 incluse, le champ commentaire commencera à chaque fois en colonne 61

- Complète le champ opération, sert à définir la donnée sur laquelle s'effectue l'instruction.
- L'opérande doit donner à l'assembleur le mode d'adressage.
- Le champ opérande peut contenir tous les caractères de l'alphabet principal.
- Il peut contenir des symboles définis dans le champ étiquette.
- Séparé de la zone Mnémonique par un délimiteur le caractère ESPACE.

Dans ce champ Opérante Opérande on peut trouver :

retour au Sommaire Liens Rapides

GEN : (Champ Opérande) : Des nombres

- Décimaux Préfixe **&**.est facultatif 75 ou &75
- Hexadécimaux Préfixe **$** ou suffixe .H **$**75 ou 75**H**
- Binaire Préfixe **%** ou suffixe .B **%**01000110 ou 01000110**B**
- Octal Préfixe **@** ou suffixe .Q **@**12 ou 12**Q**
- Caractères ASCII : le caractère sera précédé par une apostrophe, ex pour le caractère A : **'A**
- Déplacement par rapport au registre PC : *+5 déplacement de 5 octets, déplacement peut être + ou –

Pour éviter de consommer du temps inutile :

- Le mode avec les suffixes est à éviter.
- Renoncer aux espaces inutiles.
- Faire attention aux plages de valeurs permises.

En respectant dans la mesure du possible certaines conventions :

- Base hexadécimale pour les adresses.
- Code ASCII quand il s'agit des touches d'un clavier, des caractères de commande.
- Base binaires pour le masquage des bits et la configuration des registres des interfaces.

GEN : (Champ Opérande) : Des noms de variable

retour au Sommaire Liens Rapides

Des noms de variable pour définir une adresse ou une valeur numérique.
Ces noms :

- Ont de 6 à 20 caractères maxi.
- Peuvent contenir des caractères A….à….Z, 0….à….9, le point, ou "$"
- Ne pas choisir des noms identiques
  - Aux codes opérations
  - Aux registres internes
  - Aux directives d'assemblages

L'assembleur les traites comme des nombres à conditions qu'une valeur ait été assignée auparavant.

```
VALEUR = $10
```

```
…
ADCA VALEUR    ; addition entre A et la casse mémoire
               ; à l'adresse $0010
```

GEN : (Champ Opérande) : Des noms d'étiquettes

Index | retour au Sommaire | Liens Rapides

6 caractères Maxi avec une lettre pour le premier caractère.

```
JMP   BOUCLE
```

GEN : (Champ Opérande) : Des expressions arithmétiques ou logiques

Certain assembleur autorisent l'écriture d'expressions arithmétique et/ou logiques dans le champ opérande. Quelques exemples :

```
INDEX+OFFSET
INDEX+10
-(ADRFIN-ADRINI)*2
*+5
*-4
```

Ces expressions peuvent aussi utiliser les opérations **+ - * /**

```
ADCA  VALEUR + 1
JMP   BOUCLE - 5
```

Sur le macro-assembleur µp6809, outre les opérations + - * et /, les opérateurs suivant sont aussi valides :

!^ Exponentiation : l'opérande de gauche est élevé à la puissance de l'opérande de droite : 3!^2 = 9 0!2 = 1

!. (AND) ET logique bit à bit entre registre 8 ou 16 bits,
si CAR = $01 CAR**!.**$7F = $01
%0000 0001 %0111 1111 = %0000 0001

!+ (OR) OU inclusif bit à bit entre registre 8 ou 16 bits,
si CAR = $01 CAR**!+**$10 = $11
%0000 0001 %0001 0000 = %0001 0001

!X (XOR) OU exclusif bit à bit entre registre 8 ou 16 bits,
si CAR = $02 CAR**!X**$43 = $41
%0000 0010 %0100 0011 = %0100 0001

!< (Décalage gauche). L'opérande de gauche est décalé vers la gauche d'un nombre de bits spécifié par l'opérande de droite
si CAR = $0A CAR**!<**$01 = $14
%0000 1010 %0000 0001 = %0001 0100

!> (Décalage droite). L'opérande de gauche est décalé vers la droite d'un nombre de bits spécifié par l'opérande de droite
si CAR = $0A CAR**!>**$02 = $02
%0000 1010 %0000 0010 = %0000 0010

!L (Rotation gauche). Permutation circulaire de l'opérande de gauche de bits spécifié par l'opérande de droite
si CAR = $0A CAR**!L**$02 = $28
%0000 1010 %0000 0010 = %0010 1000

!R (Rotation droite). Permutation circulaire de l'opérande de gauche de bits spécifié par l'opérande de droite
si CAR = $0A CAR**!R**$02 = $02 (et non pas $82 % 1000 010)
%0000 1010 %0000 0010 = %1000 0000 0000 0010
L'opération s'effectuant sur 16 bits le résultat n'est pas $8002 mais bien $02

CAR doit être défini par avance, pour se faire on utilise la directive EQU, exemple : `CAR EQU $1A`

GEN : (Col. 62) Champ Commentaire (Z_Commentaire)

Index | retour au Sommaire | Liens Rapides

Le délimiteur est le caractère point virgule **;**
Ce champ est optionnel. Il ne produit aucun code exécutable.

Les commentaires bien souvent négligés, sont très utiles pour documenter le programme et de ce fait le rendre plus lisible. Un programme bien commenté facilite sa maintenance.

On distingue les commentaires :

- De lignes Derrière l'opérande, après un caractère espace ou en colonne 61

- De blocs Sur une ligne entière avec le signe ; en première colonne si le listing n'a pas encore été assemblé ou en colonne 28 (colonne du premier caractère des étiquettes ou Label)

GEN : Conseils et règles à observer pour les commentaires :

- Ils doivent servir à éclairer un programme
- Peuvent servir à expliquer l'utilité d'une instruction ou d'un sous-programme.
- Mettre des commentaires à des endroits clefs. Toutes les lignes ne doivent pas être commentées.
- Utiliser de préférence les minuscules.
- Etre explicite pour les commentaires de blocs, ce commentaire doit être ramené à "l'Adresse Début Tableau".
- Etre très concis pour les commentaires de lignes, supprimer les articles, les ponctuations (si possible) et les liaisons grammaticales.
- Eviter de commenter la nature de l'instruction comme "chargement adresse $8000 dans X" pour l'instruction LDX #$8000. Mais utiliser plutôt un commentaire du style "adrs Début tab. Constantes".
- Définir toutes les abréviations personnelles en début de programme (ex : ATCL pour Attente caractère Clavier).
- Faire attention au format fixe de sortie du listing d'assemblage qui limite le nombre de caractères disponibles sur une ligne.

GEN : Directives d'assemblage

Index retour au Sommaire Liens Rapides

BSZ END EQU FAIL FCB FDB FCC FILL NAM OPT
ORG PAGE REG RMB SET SETDP SPC TTL ZMB

Les directives d'assemblage sont des instructions données au programme assembleur, plutôt que des instructions à traduire en code objet.

On doit placer les directives d'assemblage (appelé aussi pseudo opérations ou les appels de macros) dans le champ Mnémonique.

Il existe 2 catégories de mnémoniques qui n'ont pas de code machine et ne font pas partie de la bibliothèque du µp6809. Elles sont juste destinées à donner des directives à l'assembleur. Ces directives ou pseudo-instructions peuvent différer d'un assembleur à l'autre :

- **Les directives d'assemblage.** (Pseudo opérations). Elles agissent plutôt sur le processus d'assemblage. Tel que par exemple le positionnement du programme et des différents tableaux en mémoire.

- **Les macro-instructions (ou macro).** (Appels de macro). Elles autorisent l'insertion des séquences fixes d'instructions dans le programme principal par des appels très concis. Un assembleur qui autorise la définition des macros reçoit l'appellation de macro-assembleur.

GEN : Directive ORG (Origine)

Directives d'Assemblage Index retour au Sommaire
Liens Rapides

Permet de définir l'origine de départ d'un programme ou d'un sous-programme. Elle permet de définir l'adresse absolue d'implantation du programme. Elle est utile uniquement si on a des contraintes d'implantation de programme (saut absolue par JSR, JMP)

Mais n'est pas obligatoire si le programme est complètement écrit en relatif (BSR, LBSR, BRA ….).

Les instructions suivantes seront assemblées dans les emplacements mémoire situés à partir de la nouvelle adresse contenue dans le compteur de programme.

S'il n'y a pas de directive ORG dans un programme source, le compteur de programme est initialisé à la valeur 0.
L'opérande ne peut pas contenir nom de constante défini plus loin dans le programme.

Cette directive définit donc l'adresse du premier octet du programme objet (programme objet = ensemble du code machine en hexa directement exécutable).

```
ORG    $0400            ; opérande du type constante
```

Après assemblage, le premier octet de la première instruction occupera l'adresse $0400.

```
ORG    RESET            ; opérande du type symbole
ORG    *+100            ; opérande du type expression
ORG    SBR1+LGTAB*8     ; opérande du type expression
```

Etant donné leur rôle, cette directive n'a pas d'étiquette.

La gestion de l'espace mémoire est une tâche importante en assembleur, un programme écrit dans ce langage comporte plusieurs sections ou blocs, dans ces derniers on peut trouver :

- Le programme principal
- Les sous-programmes
- Las adresse des interfaces
- Les adresses des vecteurs d'interruption
- L'adresse de la zone de stockage de paramètres
- L'adresse de la zone des données temporaires
- L'adresse des piles

ORG initialise les débuts des sections et se trouve donc obligatoirement en entête de chaque section.
Aucune étiquette ne doit précéder la directive ORG.

Si un programme est écrit sans la directive ORG, l'assembleur choisit par défaut l'adresse $0000.

GEN : Directive BSZ (Block Storage Zero)

Index | retour au Sommaire | Liens Rapides
RMB | ZMB | Directives d'Assemblage

Voir aussi la directive RMB ou ZMB

Cette directive est plus ou moins identique à RMB (ou ZMB) à la fois dans le principe de réservation des mémoires et le mode d'écriture des formes autorisées.

L'unique différence vient du fait que BSZ initialise à 0 au chargement toutes les mémoires réservées, alors que RMB n'écrit rien en mémoire, RMB ne détruit donc pas les contenus mémoires situés dans les blocs réservés.

BSZ impose à l'assembleur de réserver un bloc d'octets et d'assigner à chacun de ces octets la valeur 0
Le nombre d'octets est donné par l'opérande qui ne doit pas contenir de symbole, l'opérande ne doit pas nul sinon il y aura erreur lors de l'assemblage.

La directive BSZ permet d'initialiser de la mémoire à 0

```
        ORG   $2000          ;
TOTO    BSZ   $10            ; TOTO vaudra $2000
TITI    EQU   *              ; TITI vaudra $2010
                             ; Et en mémoire entre $2000 et $2010, il y aura des 0
```

Le code ci-dessus est équivalent à :

```
        ORG   $2000
TOTO    FCB   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
TITI    EQU   *
```

Ou encore à ça :

```
        ORG   $2000
TOTO    FDB   0,0,0,0,0,0,0,0
TITI    EQU   *
```

GEN : Directive END (Fin)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Définit la fin d'un programme source.
De même qu'il est nécessaire de fournir à l'assembleur une indication de début de programme par ORG, l'assembleur doit connaître l'endroit ou se termine le programme. Cette directive marque la fin logique d'un programme.

Les instructions derrière la directive END ne seront pas assemblées. L'oubli de la directive END à l'édition sera signalé par un avertissement.

END autorise à spécifier l'adresse d'exécution dans son champ opérande par un symbole.

```
DEBUT   LDA    $78      ;
        ADCA   #05      ;
        …               ;
        END    DEBUT    ; indique à ASM que la première instruction
                        ; à exécuter sera à l'adresse DEBUT.
```

```
                    ; Alors que ORG donne l'adresse de
                    ; chargement en mémoire du code objet.
```

Nota : dans le cas des kits K32 de la Sté DATA RD, l'assembleur met en plus un $3F en mémoire, le même OpCode que l'instruction `SWI`.

GEN : Directive EQU (Equate)

Index | retour au Sommaire | Liens Rapides
Directive SET | Directives d'Assemblage

Voir aussi la directive SET

On ne peut pas redéfinir l'étiquette ailleurs dans le programme. L'opérande ne peut pas contenir de nom de constante défini plus loin dans le programme.
Permet d'affecter une valeur 8 ou 16 bits à un nom symbolique. Variables représentant soit :

- Des adresses
- Des données

```
COMPT   EQU   $05           ; donnée 8 bits
ADRDEB  EQU   1000          ; donnée 16 bits
FIN     EQU   DEBUT+$60     ;
```

On placera les directives EQU en début de programme.

A chaque fois que l'assembleur rencontre le nom symbolique, celui-ci est remplacé par la valeur hexadécimale indiquée après le mnémonique EQU.

EQU donne au nom symbolique une valeur qui n'est pas liée au compteur de programme PC.
Les noms définis par EQU ne peuvent pas être redéfinis dans la suite du programme.
Les noms utilisés dans la définition d'autres noms symboliques ou dans le calcul d'une expression doivent être définis au préalable.

La directive EQU accepte également un opérande du type expression avec tous les opérateurs arithmétiques et logiques.

```
VALHEX EQU   $8000               ; VALHEX aura pour valeur $8000
DEBBIN EQU   -$0200              ;
FINATT EQU   NOMVAR+30           ;
FINTIT EQU   NOMVAR-VARFIN       ;
TOUCHA EQU   'A                  ; assigne $0041 au symbole TOUCHA
FONCTA EQU   TOUCHA!+%10000000   ; assigne $00C1 au symbole FONCTA
                                 ;      touche Fonction équiv à "A
```

On peut affecter une adresse à un label, dans ce cas on utilises l'opérateur * qui donne l'adresse courante

```
        ORG   $2000
TOTO    EQU   *      ; TOTO aura pour valeur $2000, car on affecte la
                     ;            valeur courante de l'adresse à TOTO
TITI    EQU   (*-3)  ; TITI aura pour valeurs $1FFD ($2000-3)
```

Remarque : Pour une adresse courante on utilise toujours `EQU *` et pas `SET *`.

GEN : Directive FAIL

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Une directive FAIL incluse dans une ligne quelconque du programme provoque l'affichage sur l'écran ou l'impression sur une imprimante, de l'ensemble de la ligne marqué par FAIL.
Elle est destinée pour repérer le trajet choisi par l'assembleur.

GEN : Directive FILL

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

La directive FILL indique à l'assembleur d'initialiser une zone mémoire avec une valeur constante.
Le premier opérande donne la valeur de la constante et le deuxième opérande donne le nombre d'octets successifs à initialiser.
Le premier opérande doit se trouver dans la gamme de valeurs 0-255.
Les deux opérandes ne peuvent comporter ni de constante (EQU) défini plus loin dans le programme ni de constante non défini.

GEN : Directive OPT (OPTion système)

Index | retour au Sommaire | Liens Rapides
Directives d'Assemblage

La directive OPT sert à fixer le format du fichier de sortie. Elle permet au programmeur de sélectionner ou de contrôler différentes opérations de sorties de l'assembleur.

Les options définies par cette directive sont prioritaires sur celles définies sur la ligne de commande qui a lancé l'assembleur.

Elle est destinée à contrôler le format du listing :

- Nombre de lignes par page.
- Nombre de colonnes par ligne.
- Etc ...

La majorité des options interfèrent sur le mode de fonctionnement de l'assembleur et du système d'exploitation lui-même :

- Création d'un fichier contenant un code objet exécutable.
- Création d'un fichier destiné à l'éditeur de liens.
- Impression de la table des références croisées.
- Etc...

Les options de la directive OPT change d'un constructeur à l'autre, leur nombre et les abréviations peuvent varier.

Le format d'une ligne d'options est :

```
        OPT   ABS,LLE=80,G   Commentaire facultatif
```

Directives d'Assemblage retour au Sommaire Liens Rapides

Quelques principes généraux sont à noter :

- Les options sont séparées par des virgules.
- L'ordre d'apparition des options n'est pas important. A part quelques options comme G, NOG, L, NOL qui peuvent être insérées dans le corps du programme.
- La ligne définissant les options doit se trouver avant la directive ORG.
- Les options peuvent être :
  - Positives, l'assembleur les tient en compte sans que le programmeur ait à les spécifier.
  - Négatives, l'assembleur les supprime initialement.
- Le chiffre (1) signifie que l'option existe par défaut.
- Le chiffre (0) signifie que l'option doit être spécifiée expressément pour être effective.
- Les options négatives sont marquées souvent par le préfixe NO.
- Une option NO.... avec un chiffre (1) est donc une option négative par défaut. Elle équivaut à une option positive avec le chiffre (0).

Pour voir en détail les nombreuses options, je vous reporte à la documentation des macro-assembleurs que vous allez utiliser (exemple le macro-assembleur µp6809 MOTOROLA version 3.05).

La totalité des explications des paramètres ci-dessous est détaillé à la page II.69 de l'ouvrage 06 de la préface de ce document.

`ABS (0)`	Permet de créer un code objet absolu chargeable par le système d'exploitation Motorola.....
`C`	Autorise l'affichage du nombre de cycles dans le listing. Le nombre total de cycles nécessaire à l'exécution de l'instruction sera affiché après le dernier octet de l'instruction, entre crochets rectangulaires.
`LOAD (0)`	Permet de créer un code objet chargeable sur système Exorciser.....
`REL (0)`	Permet d'obtenir un fichier en "code translatable".....
`M (0)`	Autorise dans le cas des options ABS et LOAD (et non REL), un chargement simultané du code objet absolu dans la mémoire centrale pour une éventuelle exécution. .....
`O (1)`	Implique la création d'un fichier objet à l'issue du processus de compilation.....

`NOO (0)`	Supprime toute option conduisant à la création d'un fichier objet de sortie. Elle est donc mutuellement exclusive avec O, ABS, LOAD et REL.
`CRE (0)`	Demande l'impression de la table des références croisées..... cre Autorise l'édition d'une table de référence croisée à la fin du fichier listing. Cette option doit être spécifiée avant le premier symbole dans le fichier source.
`G(0)`	Demande l'impression du code objet généré correspondant au directives FCB, FDB et FCC.....
`NOG (0)`	Demande la suppression de la dernière option G demandé.....
`NOC`	Interdit l'affichage du nombre de cycles nécessaire à l'exécution de l'instruction. C'est une valeur par défaut.
`L (1)` `L`	Permet de rétablir l'impression normale du listing de sortie..... Édite le listing à partir de ce point. Le format du listing est donné en annexe.
`NOL (0)` `NOL`	Supprime l'impression du programme à partir de cette ligne comprise..... N'édite plus de listing à partir de ce point.
`W (1)`	Rétablit l'impression normale des messages d'erreur et d'avertissement annulée précédemment par un NOW.....
`LLE=nnn (1)`	Fixe le nombre de colonne par ligne de listing. La valeur décimale par défaut est 72, les valeurs : mini 50 et maxi 120.
`P=nnn (1)`	Fixe le nombre de ligne imprimée par page de listing. Le nombre par défaut est 58, les valeurs : mini 10 et maxi 255.
`S`	Édite une table des symboles à la fin du fichier listing.
`NOP (0)`	Supprime la pagination. Les directives PAGE, NAM et TTL seront évidemment neutralisées par cette option.

GEN : Directive PAGE

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Cette directive provoque un saut de page dans le fichier listing.
Le mot PAGE ne sera pas reproduit sur le listing.

GEN : Directive NAM

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Donne un nom au programme en dehors du nom donné aux fichiers, le programmeur à la possibilité d'attribuer un autre nom au listing.

Ce nom sera composé d'au maximum 6 caractères ASCII visualisables.

Le nom dans le champ opérande de la directive NAM est répété à chaque début de page du listing, après le numéro de page et le nom du fichier.

GEN : Directive SET

Index | retour au Sommaire | Liens Rapides

(Voir aussi la directive EQU) Directive EQU | Directives d'Assemblage

Set est une assignation temporaire, les étiquettes de cette directive peuvent être définies à nouveau dans un même programme et dans des circonstances variées.
Les noms symboliques définit par SET peuvent être redéfinis dans la suite du programme. La directive SET est très utile pour définir temporairement des noms symboliques ou des étiquettes réutilisables dans la suite du programme.

A l'inverse de SET, une étiquette assignée par EQU conservera sa valeur tout le long d'un programme.

La séquence suivante, écrite avec la directive SET, ne sera pas tolérée si on utilise la syntaxe EQU.

```
ARG     SET   2          ; le symbole ARG prend la valeur 2
```

```
ARG     SET    ARG*2       ; le symbole ARG prend la valeur 4
ARG     SET    ARG*ARG     ; le symbole ARG prend la valeur 16 (4*4)
ARG     SET    ARG+2       ; le symbole ARG prend la valeur 18 (16+2)
```

Permettent d'affecter une valeur à un label

```
TOTO    SET    1           ; TOTO aura pour valeur 1
TATA    EQU    2           ; TATA aura pour valeur 2
TITI    SET    TOTO+TATA   ; TITI aura pour valeur 3
```

Un symbole assigné par SET peut apparaître simultanément dans les champs Etiquette et Opérande d'une même instruction.

Une modification quelconque d'un opérande d'une directive SET peut éventuellement avoir des répercussions sur plusieurs endroits d'un programme.

- La directive **EQU** est liée plus "au Matériel". Une adresse d'interface entrée-sortie, une zone mémoire attribuée au programme utilisateur, l'adresse d'appel au moniteur résident, etc ...

- La directive **SET** est liée à un programme spécifique.

Choisir une étiquette est en réalité une opération délicate, surtout quand on dispose que de 6 caractères. On peut mieux choisir une étiquette quand elle est liée au matériel.

Quand elle se rapporte à des notions abstraites comme un algorithme ou un résultat intermédiaire de calcul le programmeur est souvent embarrassé.
Il faut également tenir compte que multiplier les étiquettes alourdit le programme source et le rend touffu, voire peu compréhensible.

Dans ces circonstances la directive SET apporte une simplification considérable.

GEN : Directive SPC (Space)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Cette directive permet d'insérer une ligne blanche dans le listing après compilation, le plus souvent pour séparer les différents blocs de programme, le programme principal des sous-programmes ou les sous-programmes entre eux, pour améliorer la lisibilité.

Aucune information dans la ligne contenant SPC ne sera imprimée dans le listing.

Si l'assembleur le permet, utiliser de préférence SPC au lieu de mettre un point virgule **;** dans la première colonne à l'édition, car l'astérisque et le numéro de ligne seront inscrits sur le listing.

Le macro-assembleur µp6809 autorise les 3 formes connues de l'opérande pour SPC : la constante, le symbole ou l'expression.

```
SPC           ; équivaut à SPC  1
SPC    $A     ; laisser 10 ligne blanches
SPC    NLB    ; opérande du style symbole
SPC    2*CFLB ; opérande de type expression
```

GEN : Directive TTL (Title)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Donne la possibilité de titrer les différentes partie d'un programme, de sonner un entête différent pour chaque sous-programme ou chaque section.

```
TTL    ; ** Sous Programme d'attente ***
TTL    ---SECTION PARAMETRES---
TTL    Initialisation générale
```

Si NAM admet éventuellement un champ commentaire, la directive TTL exclut tout commentaire

L'opérande peut contenir jusqu'à 45 caractères ASCII visualisables pour un terminal comportant 80 colonnes.

Il est reproduit intégralement au début de chaque page et derrière le nom attribué par la directive NAM.

Chaque page du listing réserve initialement un certain nombre de lignes pour les instructions, y compris les lignes blanches. (58 en standard pour le macro-assembleur MOTOROLA).

Ce nombre peut être néanmoins modifié par une option de la directive OPT.

La fin d'un sous-programme ou d'une section ne correspond pas toujours à la fin d'une page.

L'association des directives SPC et TTL permet de positionner la première instruction d'une section sur un début de page avec un nouvel entête, ce qui évite des coupures aléatoires.

```
NAM  COMFRS        COMmande FRaiSeuse
TTL  ---S/P Vérification Etat Capteurs---
...          ;
...          ; premier corps de programme
...          ;
TTL  ---S/P Lecture des Commandes---
SPC  100     ; le chiffre 100 est factice. N'importe quel
             ; chiffre > au nombre de ligne par page provoque
             ; un positionnement du texte au début de la page
             ; suivante avec un nouvel entête. Il ne faut pas
             ; intervertir l'ordre des deux directives SPC et TTL
             ;
...          ; deuxième corps de programme
```

Ci-dessous, apparaissent les informations contenues dans la première ligne d'une page de listing, avec NAM et TTL combinées.

```
PAGE   006   ASERIMGF.LO:0  COMFRS   Reconnaissance des Paramètres
```

Numéro de page	Nom du fichier	Nom du programme	Entête d'une section particulière

GEN : Directive REG (REGistre)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

D'une importance marginale, cette directive est conçue pour raccourcir l'écriture des instructions d'empilement PSHS, PSHU et de dépilement PULS, PULU.

Cette directive n'existe que sur le macro-assembleur µp6809.
Les noms des registres situés dans le champ opérande sont affectés au symbole du champ étiquette.

En l'absence de REG les instructions opérant sur les piles requièrent des opérandes formés par des successions de noms de registres. Exemples :

```
PSHS  PC,Y,CC          ; deux registres 16 bits et un de 8 bit
PSHS  PC,U,Y,X,DP,B,A  ;
```

A partir de 3 ou 4 registres, l'écriture peut paraître longue, surtout si ces instructions se répètent souvent. Le macro-assembleur autorise alors une définition globale des registres

```
RGT_S  REG   PC,U,Y,X,DP,D,CC   ; tous les registres sauf S
RGT_U  REG   PC,S,Y,X,DP,D,CC   ; tous les registres sauf U
RGABCC REG   A,B,CC             ; on peut aussi écrire  D,CC
RGUYX  REG   U,Y,X
RGUDCC REG   U,D,CC             ; on peut aussi écrire  U,A,B,CC
```

Dans la suite du programme, on peut écrire :

```
PSHS  #RGT_U            ; Attention : le signe # obligatoire
PULU  #RGT_S            ;             devant les symboles
PSHS  #RGABCC!+RGUYX    ; cette écriture est équivalente à
                        ; PSHS   U,Y,X,D,CC
```

Les noms des registres peuvent être spécifiés dans un ordre quelconque, comme pour les instructions portant sur les piles.

L'opérateur **!+** qui est en fait un OU inclusif permet d'associer les registres, il est le seul autorisé.

Plusieurs opérateurs en chaîne sont permis ex : `#RGA!+RGB!+RGCC`

A plus de trois définitions REG, au lieu d'utiliser l'opérateur !+ il est préférable d'utiliser la forme conventionnelle.

Quelques erreurs à ne pas commettre :

```
RGUS   REG   U,S      ; U et S ne doivent pas coexister
RGDB   REG   D,B      ; B est spécifié deux fois (avertissement)
```

GEN : Directive FCB (Form Constant Byte) Réservation d'un Octet)

Créé une constante de 8 bits en mémoire, cet espace mémoire est initialisé à une valeur particulière placée dans le champ opérande.
FCB peut avoir des opérandes multiples séparés par une virgule.
La valeur de chaque opérande est tronquée à 8 bits puis est rangée dans un octet du programme objet.

Les constantes peuvent être une valeur numérique, une constante, un caractère, des autres noms de constantes ou des expressions.

L'opérande peut éventuellement être écrit sous forme d'une formule arithmétique comportant des noms de constantes, dans ce cas elle doit être définie par une directive EQU avant la ligne FCB.

```
     VAL     FCB    $4F      ; place $4F dans la première case
                             ; mémoire et lui assigne VAL
```

La constante $4F est mise en mémoire à l'adresse pointée par le PC courant. Le PC est incrémenté de 1.

```
0100        DONN    FCB    $C,16,'a,,0,$0006  ; six octets, opérande multiple
                                              ;         la , comme séparateur
0106                FCB    45,                ; 2 octets car il a une virgule en
                                              ;         plus derrière l'opérande
0108                FCB    %10011             ; opérande binaire incomplètement
                                              ; rempli
0109                FCB    LGR*8+10           ; Opérande du type expression
                                              ;
            LGR     EQU    $10                ;
```

Les constantes sont dans le programme objet dans le champ Hexa opérande à raison de 4 octets par ligne.

```
0100       0C 10 61 00       DONN    FCB     $C,16,'a,,0,$0006
0104       00 06
0106       2D 00
0108       13
0109       90
```

Au chargement du programme en mémoire, les positions mémoire de $0100 à $0109 seront remplies par les valeurs ci-dessous :

```
(....) représente le contenu de la mémoire

(0100) = $0C  assignation simultanée de la valeur $0100 du compteur
                     PC à l'étiquette DONN
(0101) = $10  conversion du nombre décimal 16 en hexa
(0102) = $61  code hexa du caractère "a"
(0103) = $00  il est permis de ne rien mettre entre deux virgule
                     dans ce cas la valeur est nulle
(0104) = $00
(0105) = $06  c'est un faux opérande 16 bits. L'assembleur analyse
                     entièrement l'opérande sur un mot de 16 bits,
                     teste à posteriori l'octet le plus
                     significatif. S'il est nul, comme c'est le cas,
                     il n'y aura pas troncature, donc message
                     d'erreur.
(0106) = $2D  équivalent hexa du nombre décimal 45
(0107) = $00  la virgule en plus derrière l'opérande 45 ajoute un
                     octet nul supplémentaire (source d'erreur
                     fréquente)
(0108) = $13  équivalent du nombre binaire %10011. Il est préférable
                     d'écrire tous les 8 bits d'un nombre présenté
                     sous la forme binaire
(0109) = $90  équivalent à (TAB * 8) + $10, la valeur de TAB étant
                     $10.
```

Les dépassements entraînant des troncatures à 8 bits seront signalés par un message d'erreur de débordement.

Les formes suivantes sont incorrectes :

```
010A        FCB    LGR*8+$A0  ; après calcul, le nombre obtenu est $0120
                              ; l'octet le plus significatif MSB n'est pas nul
                              ; donc émission d'un message de débordement
                              ;
010B        FCB    $138,256   ; deux dépassement simultanés
```

En général, les valeurs inscrites en mémoires par les directives FCB, FDB et FCC ne sont pas reproduites sur le listing de sortie après compilation à cause de leur encombrement.

Néanmoins, on a la possibilité de les obtenir en spécifiant l'option correspondante dans la directive OPT.

FCB permet de réserver des cases mémoires afin de créer des tableaux de données. On donne une origine au tableau et on utilise la directive FCB, exemple :

```
          ORG    $3000
DEBTAB FCB    0,$AA,25,@377,10
```

Dans la mémoire à l'adresse $3000 on aura :

```
$3000 = $00
$3001 = $AA
$3002 = $19          ; $19 = 25
$3003 = $FF          ; $FF = @377  (octal)
$3004 = $0A          ; $0A = 10
```

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

GEN : Directive FCB et FDB exemples communs

On utilise ces pseudo code, pour, par exemple initialiser des tables. Par exemple

```
       ORG  $2000          ; En mémoire, à l'adresse $2000 on aura
       FCB  1,2,3          ;                    010203000400050006
       FDB  4,5,6          ;
```

On peut mélanger FCB, FDB, et EQU

```
       ORG  $2000          ; En mémoire, à l'adresse $2000 on aura
TOTO   EQU  *              ;                    010203000400050006
       FCB  1,2,3          ;
       FDB  4,5,6          ; et TOTO vaudra $2000
```

Le code juste au-dessus est EXACTEMENT pareil que le suivant :

```
       ORG  $2000
TOTO   FCB  1,2,3
       FDB  4,5,6
```

Le fait de mettre un label en début de ligne, permet d'affecter l'adresse courante au label

GEN : Directive FDB (Form Double constant Byte) (Réservation d'un Double Octet)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

Crée une constante en 16 bits en mémoire. Presque identique à la directive FCB, mais cette fois on est en 16 bits. La directive FDB peut avoir un ou plusieurs opérandes qui sont alors séparés par des virgules.

La valeur codée sur 16 bits de chaque opérande est rangée dans deux octets consécutifs du programme objet. Le rangement commence à l'adresse courante et l'assembleur assigne au nom placé dans le champ étiquette la valeur de l'adresse courante.

Si la valeur des données se situe en dehors des 16 bits donc en dehors de la plage [$0000 , $FFFF], l'assembleur effectue une troncature et émet un message d'erreur de débordement.

La séquence ci-dessous illustre les formes autorisées de cette directive :

```
                    ORG   $0100              ;
0100          ADR   FDB   $2345,36,'a        ; réserv 3 données de 16 bits
0106                FDB   *,,-3456           ; réserv 3 données de 16 bits
010C                FDB   (*-ADR)*2,DEBTAB   ; réserv 2 données de 16 bits
010F          DEBTAB EQU  $F8                ;
```

Les constantes sont restituées sur le listing du programme objet à raison de deux mots de 16 bits par ligne.

```
0100     2345 0024          ADR    FDB   $2345,36,'a   ;
0104     0061                                          ;
0106     0106 0000                 FDB   *,,-3456      ;
010A     F280                                          ; représente -3456
```

Au chargement du programme en mémoire, les positions mémoire de $0100 à $010F seront remplies par les valeurs hexa ci-dessous :

```
(....) représente le contenu de la mémoire

(0100) = $23  assignation simultanée de la valeur $0100 du  compteur PC à
                    l'étiquette ADR
(0101) = $45  octet moins significatif de $2345
-------------------------------------------------------------
```

```
(0102) = $00  FDB place chaques données sur 16 bits, la valeur 36 en
(0103) = $24        décimale est égale à $0024 en 16 bits
-------------------------------------------------------------
(0104) = $00  extension d'une donnée ASCII l'équivalent de
(0105) = $61        "a" en hexa est égal à $0061 en 16 bits
-------------------------------------------------------------
(0106) = $01  le signe * désigne en toute circonstance la valeur du
(0107) = $06        compteur programme au début de l'instruction
-------------------------------------------------------------
(0108) = $00  deux octets nuls remplacent le "rien" entre les
(0109) = $00        deux virgules
-------------------------------------------------------------
(010A) = $F2
(010B) = $80  nombre négatif qui est converti en hexa -3456=$F280
-------------------------------------------------------------
(010C) = $00  le premier astérisque * représente la valeur du compteur
              programme, soit $010C
(010D) = $18        (*-ADR)*2 = ($010C - $0100) * 2 = $18
-------------------------------------------------------------
(010E) = $00  extension d'un symbole à 8 bits
(010F) = $F8        DEBTAB
```

Lors d'une utilisation normale, la directive FCB inscrit les constantes qui seront sollicitées par la suite par des registres 8 bits du type A, B, DP ou CC.

Alors que la directive FDB est destinée à être employée pour le registres 16 bits du type PC, S, U, Y ou X.

Autre exemple : assignons la variable TER à l'adresse $56F7

```
TER     FDB   $56F7
```

En mémoire on aura :

```
ADR   = $56
ADR+1 = $F7
```

La constante $56F7 est mise en mémoire sous la forme de deux octets adjacents à l'adresse pointée par le PC courant. Ce dernier est incrémenté de 2.

L'octet de poids fort MSB du mot de 16 bits est stocké en premier.

En général, les valeurs inscrites en mémoires par les directives FCB, FDB et FCC ne sont pas reproduites sur le listing de sortie après compilation à cause de leur encombrement.

Néanmoins, on a la possibilité de les obtenir en spécifiant l'option correspondante dans la directive OPT.

GEN : Directive FCC (Form Constant Caractères) (Réservation d'un Bloc mémoire)

Directives d'Assemblage | Index | retour au Sommaire | Liens Rapides

FCC permet de stocker en mémoire une chaîne de caractères ASCII.

Le premier octet est stocké à l'adresse courante. Autrement dit le nom placé dans le champ étiquette se voit assigner la valeur de l'adresse du premier octet de la chaîne.

Tout caractère ASCII imprimable codé de $20 à $7F peut se trouver dans la chaîne qui est enfermée entre deux délimiteurs identiques.

Le délimiteur est le premier caractère ASCII imprimable autre que l'espace situé après la directive FCC.
Il n'est donc plus nécessaire de les spécifier par le signe apostrophe '.

A la différence de FCB et FDB, la directive FCC inscrit dans les mémoires le code ASCII des caractères situés dans le champ opérande. $41 pour "A", $31 pour "1", $20 pour " " espace.

FCC est utile pour envoyer une chaîne de caractère sur un terminal, cette chaîne peut être :

- Un message d'erreur
- Un entête de programme
- Une question ou une réponse
- Une simple ligne de texte

Directives d'Assemblage | retour au Sommaire | Liens Rapides

```
0100            MGS115 FCC  "Erreur AA115"   ; délimiteur est le "
010C                   LDA  #45              ;
```

Dans le programme objet (code machine) chaque caractère de la chaîne entre les délimiteurs /..../ est restitué à raison de 4 caractères par ligne

```
0100      45 72 72 65          MGS115 FCC  "Erreur AA115"
0104      75 72 20 41
0108      41 31 31 35
010C      86 2D                       LDA  #45
```

Le slash / est utilisé comme délimiteur de chaîne. Dans l'exemple ci-dessus on réserve 12 caractères ASCII donc 12=$0C, l'adresse qui suit sera donc de $0100 + $000C = $ 010C

La valeur $010C de la ligne qui suit la directive FCC, montre que cette dernière est avide d'emplacement mémoire.

Dans la partie hexa de l'opérande, on place la valeur du premier caractère, se sera ici le caractère E donc la valeur $45.

La valeur $0100 du compteur programme correspond à l'adresse du premier caractère de la chaîne affectée au symbole MSG115.

Si le caractère slash / doit être utilisé, comme dans l'exemple ci-dessous, on peut choisir un autre caractère visualisable qui ne doit pas apparaître dans la chaîne, à l'exception des caractères ESPACE code $20 et SUPPRESSION code $7F. Par exemple on prendra le caractère "<".

```
0200            FCC   <Valeur du rapport a/b : <
0117            FCB   $0D
```

Attention : la tentation qui consiste à fermer la chaîne avec le caractère ">" est grande.

Pour ajouter un caractère extérieur de l'intervalle [$20 , $7F], on doit fermer la chaîne et utiliser la directive FCB ou FDB.

Directives d'Assemblage | retour au Sommaire | Liens Rapides

Quelques erreurs pour directive FCC :

`FCC 5Valeur5`	Délimiteur "5" autorisé mais ne facilitent nullement la compréhension.
`FCC pvaleurp`	Délimiteur "p", même type de fantaisie de mauvais goût.
`FCC 252`	Délimiteur "2", Chaîne formée d'un seul caractère le chiffre 5.
`FCC chaîne`	Délimiteur " ", un ESPACE devant et derrière, mode ambigu signalé par une erreur.
`FCC \MSG216`	Absence de délimiteur de fin de chaîne. Oubli signalé par une erreur de l'opérande.

Autre façon d'utiliser cette directive FCC

Il existe un deuxième format pour l'opérande de la directive FCC. Ce format plus commode pour mette les titres ou les tabulations sur l'écran ou sur vers une imprimante :

```
0100 MSG216     FCC   80,Erreur d'opérande dans une directive
0150
```

- La valeur $0100 du compteur programme en début de chaîne est affectée à l'étiquette MSG216
- Le nombre décimal 80 (uniquement en décimal). Le chiffre est codé sur 8 bits et doit être compris entre 0 et 255.
- Une virgule doit suivre immédiatement le chiffre.
- Si le nombre est > à la longueur effective de la chaîne, comme c'est le cas dans l'exemple ci-dessus, les octets restants jusqu'à la valeur $014F seront remplis avec le code $20 de l'ESPACE.
- Si le nombre est < à la longueur effective de la chaîne, il y aura troncature SANS message d'erreur.

Directives d'Assemblage | retour au Sommaire | Liens Rapides

Autre façon d'utiliser cette directive FCC

On souhaite inscrire un titre "SYNTAXE DE L'ASSEMBLEUR" à partir de la 21ième colonne d'un écran comportant 80 colonnes, puis le souligner par des "-" à la ligne suivante :

```
0100 TITRE      FCC   20,
0114            FCC   60,SYNTAXE DE L'ASSEMBLEUR
0150            FCC   20,
0164            FCC   60,----------------------
01A0
```

Ce format évite l'écriture des caractères ESPACE et joue également le rôle de tabulateur. Il existe évidemment d'autres façons permettant d'écrire les deux lignes sans gaspiller autant d'emplacements pour mettre ces caractères ESPACE.

Quelques erreurs fréquentes

```
FCC    4, ANNEE
```
Troncature à "ANNE", erreur de comptage

```
FCC    $A,20$US
```
Nombre hexa interdit. Le caractère $ sera interprété comme le premier délimiteur de chaîne. La chaîne "inattendue" sera A,20 au lieu de 20$US et cinq Espaces.

```
FCC    6,MSG106
```
Mélange involontaire de formats. Le deuxième format est pris en compte de façon prioritaire car l'assembleur lit et interprète de gauche à droite.

En général, les valeurs inscrites en mémoires par les directives FCB, FDB et FCC ne sont pas reproduites sur le listing de sortie après compilation à cause de leur encombrement.

Néanmoins, on a la possibilité de les obtenir en spécifiant l'option correspondante dans la directive OPT.

GEN : Directive RMB (Reserve Memory Bytes)

(Réservation d'octets en mémoire)

Index | retour au Sommaire | Liens Rapides
BSZ | Directives d'Assemblage

Voir aussi la directive BSZ

Sert à réserver des cases mémoires, soit un nombre d'octets égal à la valeur de l'opérande.
On peut utiliser une étiquette devant cette directive.
RMB provoque dans un programme un saut du compteur PC d'un nombre d'octets égal à la valeur spécifiée dans le champ opérande. Ces octets ne sont pas initialisés à une valeur donnée.

Parmi l'une des plus utilisées, cette directive assigne un ou plusieurs octets en mémoire pour un usage particulier. On utilise cette directive pour réserver une zone de "brouillon".

Les chiffres à gauche représentent les valeurs du compteur programme PC et non les numéros d'ordre. Ces nombres sont en hexadécimal et toujours codé sur 16 bits. Le signe $ n'est jamais utilisé pour les valeurs de PC. Cette convention a pour but d'alléger les écritures.

```
0080         RMB   16             ; réserve 16 octets à partir de l'adrs $0080
                                  ;          L'instruction suivante sera donc
                                  ;          mise 16 octets plus loin
0090         RMB   LGTAB          ; réserve un nombre d'octets égal à la
                                  ;          valeur de LGTAB
1A00 DEBTAB  RMB   LGTAB+$80      ; réserve LGTAB+$80 octets à partir de
                                  ; l'adrs $1A00 et assigne simultanément
                                  ; la valeur $1A00 au symbole DEBTAB
```

Bien qu'il existe apparemment une ressemblance de forme avec la directive EQU, la directive RMB fonctionne tout à fait différemment. Pour RMB ce n'est pas la valeur de l'opérande qui est assigné à l'étiquette, mais une valeur 16 bits du compteur programme PC.

Tous les symboles dans l'opérande doivent être définis avant la rencontre d'une directive RMB. Ceci est dû au mode de fonctionnement de l'assembleur à deux passes.

Les exemples ci-dessous illustrent les possibilités d'emploi de cette directive comme moyen de réservation des mémoires pour le stockage des paramètres et des données temporaires.

```
0100         ORG   $100           ;
0101 IMGA    RMB   1              ; réservation d'un octet et assignation
                                  ; de l'adresse $0100 au symbole
                                  ; IMGA (registre image de A)
0101 IMGX    RMB   2              ; réservation de 2 octets et assignation
                                  ; de $0101 à IMGX
0103 TAB     RMB   $14            ; réservation de 20 octets pour un tableau
                                  ; et repérage de l'adresse $0103 pour le
                                  ; début du tableau par le symbole TAB
```

Plus loin dans le programme, les contenus mémoires aux adresses suivantes seront affectés si l'on rencontre des instructions du type :

```
             STA   IMGA           ; adrs $0100
             STX   IMGX           ; adrs $0101 et $0102
             STB   TAB+5          ; adrs $0108
             STX   TAB+10         ; adrs $010D et $010E
```

Les mémoires à lecture-écriture peuvent donc être réservées pour le stockage des données temporaires.
Ce type de réservation occupe en général la section finale d'un programme.

Autres exemples

```
        ORG    $2000          ;
TOTO    RMB    $10            ; TOTO vaudra $2000
TITI    EQU    *              ; TITI vaudra $2010
```

Et en mémoire entre $2000, et $2010, il y aura ????. On n'est pas capable de le dire.

- Si c'est de la ROM, ça pourra être $00, ou $FF
- Si c'est de la RAM, ça pourra être n'importe quoi (ce qu'il y avait avant en mémoire)

Pour pousser l'exemple encore plus loin : A quoi sert RMB alors ?

La directive RMB peut servir à définir une structure

Si on veut avoir les adresses d'un PIA, et l'adresser de manière indirecte, on peut définir

```
PiaSys            SET  $E7C8      ; PIA System & Printer

; Pia Systeme definition
;---------------------------------------
                  ORG 0
PiaSys.ddra       RMB 1
PiaSys.ora        SET PiaSys.ddra
PiaSys.ddrb       RMB 1
PiaSys.orb        SET PiaSys.ddrb
PiaSys.cra        RMB 1
PiaSys.crb        RMB 1
```

Dans le code 6809, on peut ainsi accéder au PIA de la manière suivante :

```
      LDX   #PiaSys
      LDA   PiaSys.ddra,X
```

Ou encore

```
      LDY   #PiaSys
      STB   PiaSys.cra,Y
```

GEN : Directive SETDP (SET Direct Page)

(désignation de la Page Direct à utiliser)

retour au Sommaire | Directives d'Assemblage | Index | Adressage DIRECT | Liens Rapides

C'est une directive propre au macro-assembleur µp6809 qui indique l'adresse d'une page mémoire (dans le µp6809 pour 64Ko il y a 256 pages de 256 octets) où tout accès en mode étendu doit être converti en mode direct, ceci pour accélérer la vitesse d'exécution (voir "l'adressage Direct" avec page de base).

SETDP indique donc à l'assembleur quelle page de la mémoire utiliser en mode d'adressage en page directe.

Adressage DIRECT

Par défaut, la page 0 est sélectionnée.

`SETDP  $80` $80 indique que l'adressage direct s'effectue de $8000 à $80FF.

Cette directive a pour but d'avertir l'assembleur de l'intention du programmeur.

retour au Sommaire | Directives d'Assemblage | Liens Rapides

Attention,

Cette directive ne met pas une valeur dans le registre DP, cela se fait par une instruction `LDA #$..` Suivie d'une instruction `TFR A,DP`.

La séquence ci-dessous définit une page de base dans l'espace mémoire, à l'exécution, s'étendant entre les adresses $1000 et $10FF.

Toute instruction écrite en mode direct prélève automatiquement la valeur $10 pour remplir l'octet le plus significatif (MSB) de l'adresse.

```
                ORG  $8000      ; adresse de chargement du programme
8000 86 10      LDA  #$10       ; Cette séquence très caractéristique
                                ; permet de charger la valeur $10 dans DP
8002 1F 8B      TFR  A,DP       ;
...                             ;
...                             ;
8100 B7 102A    STA  $102A      ; adrs à l'intérieur de la page de base
8103 F6 113A    LDB  $113A      ; adrs à l'extérieur de la page de base
8106 D7 3A      STB  $3A        ; adrs à l'extérieur de la page de base
```

La séquence ci-dessous "corrigée" sera comparée ligne par ligne à la séquence ci-dessus.

```
                ORG   $8000       ;
8000 86 10      LDA   #$10        ; chargement de DP pour l'exécution
8002 1F 8B      TFR   A,DP        ;
                SETDP  $10        ; pas de code objet généré.
...                               ; chargement de la page DP "fictive"
...                               ;
...                               ;
8100 97 2A      STA   $102A       ; 2 octets au lieu de 3
8103 F6 113A    LDB   $113A       ; code objet inchangé
8106 D7 003A    STB   $3A         ; 3 octets au lieu de 2
```

retour au Sommaire | Adressage DIRECT | Liens Rapides

Ci-dessous : les règles d'emploi de SETDP :

- Elle n'a pas d'étiquette.
- Le nombre de remises à jour de la page fictive est illimité
- L'opérande peut avoir 3 formes connues : Constante, Symbole ou Expression.
- L'expression ne doit pas contenir des symboles définis à posteriori.
- Bien que SETDP implique un chargement immédiat, il ne faut pas mettre le signe #. A s'inscription de `SETDP #$10` conduit à une erreur de syntaxe (par contre le signe # est obligatoire pour la directive REG, mystère !!!)
- Lorsque la valeur d'une expression excède un mot de 8 bits, l'assembleur tronque le mot le plus significatif MSB et prend en compte l'octet le moins significatif LSB `SETDP $1028` chargera $28 dans le registre DP "fictif" avec un avertissement.
- Il faut toujours spécifier dans l'opérande l'adresse comme s'il s'agissait d'un adressage étendu. L'assembleur choisit dans tous les cas le code optimal. Dans l'exemple ci-dessus `STB $3A` est interprété comme `STB $003A` et non pas comme `STB $103A`. Le programmeur n'a qu'à écrire `STB $103A` comme s'il s'agissait d'un adressage étendu; l'assembleur compare l'adresse avec sa page de base fictive et génère le code en mode direct, car $103A se trouve bien à l'intérieur de [$1000 , $10FF].
- On a toujours la possibilité de forcer l'assembleur à prendre le mode d'adressage direct ou étendu en utilisant les signes < ou > devant l'opérande.

Directives d'Assemblage | retour au Sommaire | Adressage DIRECT | Liens Rapides

Toujours en se référant à l'exemple précédant, on peut écrire :

`STA`	`<$102A`	Forçage du mode direct. Inutile ! L'assembleur le fait automatiquement
`LDB`	`<$1113`	Forçage du mode direct pour une adresse située à l'extérieur de la page de base fictive. L'assembleur accepte en émettant un avertissement. On peut d'ailleurs écrire tout simplement `LDB <$3A`
`STA`	`>$102A`	Forçage du mode étendu bien que l'adresse soit à l'intérieur de la page de base. Un code à 3 octets sera généré sans avertissement.
`LDB`	`<$1113`	Forçage du mode étendu pour une adresse à l'extérieur de la page de base. Inutile ! L'assembleur le fait automatiquement.

GEN : Directive ZMB

(Zero Memory Bytes)

BSZ

Directives d'Assemblage | Index | retour au Sommaire | Adressage DIRECT | Liens Rapides

La directive ZMB (ou BSZ) impose à l'assembleur de réserver un bloc d'octets et d'assigner à chacun de ces octets la valeur 0.
Le nombre d'octets est donné par l'opérande, qui ne doit pas contenir de nom de constante (par EQU) défini plus loin dans le programme ou de nom de constante non défini sous peine de provoquer une erreur à l'assemblage.

GEN : Programmes translatables

Un programme est translatable s'il fonctionne quelle que soit l'adresse à laquelle il est implanté.
Il est donc indépendant de sa position mémoire et ne contient pas de référence d'adresse en absolu.

Contrairement au programme relogeable qui dispose d'adresses relatives transformées en adresses absolues après l'édition de liens, le programme translatable n'exige pas de passage par une édition de lien.

Il est implanté en mémoire et exécuté n'importe où.
Dans un programme translatable il faut que les instructions faisant référence à la mémoire soient écrites en relatif, avec des possibilités de saut en avant et en arrière de 32 Ko.

Somm. Branchements

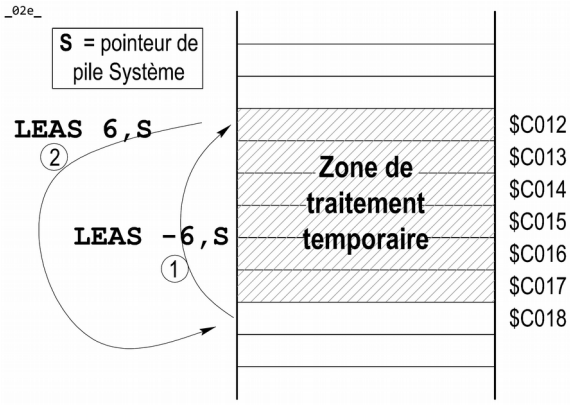
Le programme translatable peut être positionné par rapport au compteur ordinal PC grâce au mode d'adressage INDEXE (Direct ou indirect), dont la base est constituée par le PC lui-même (en absolue) et le déplacement codé sur 8 ou 16 bits.

On peut utiliser des zones de rangement temporaires sur la pile grâce à l'emploi de l'instruction chargement d'adresse effective.

Au début du programme l'instruction `LEAS -6,S` permet de déterminer une zone de travail temporaire sur la pile, cette zone se situe entre 0,S et 5,S.

Cette possibilité est intéressante pour les programmes translatables car on est certain de ne pas travailler dans une zone mémoire déjà utilisé et de ne pas détruire ainsi une partie du programme.

Il faut bien sûr, à la fin de l'utilisation de cette zone de traitement temporaire, réinitialiser le pointeur de pile avec l'instruction `LEAS 6,S` l'inverse de la celle ci-dessus.



Index | retour au Sommaire | Adressage INDEXE relatif au PC | Liens Rapides | Instructions LEA..... S, U, X, Y

Au début le pointeur de pile système S a la valeur $C018, puis S=$C012.
La zone de traitement temporaire se situe donc entre $C012 et $C017
L'instruction chargement d'adresse effective permet d'accéder à des tables placées dans un programme translatable.

L'adressage indexé utilisant le compteur programme PCR comme base permet de charger dans l'index X l'adresse du début de la table.
L'adressage indexé permet ensuite d'accéder aux données de la table.

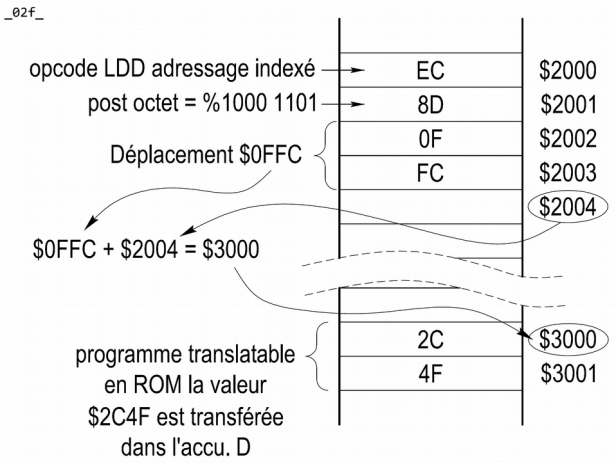
```
LEAX   TABLE,PCR       ; PC + Déplacement → X
```

On peut également avoir accès à des constantes placées dans un programme translatable. Cet accès se fait de manière indépendante de la position en utilisant l'indexation par rapport au compteur programme PC.

```
LDD    ETIQ,PCR        ; valeur à l'adrs ETIQ → D
```

Cette adresse n'est pas connue au moment de l'assemblage du programme translatable.
Il faut intégrer la ROM dans l'application ce qui définit automatiquement l'étiquette ETIQ et qui permet de calculer le déplacement correspondant à l'écart entre la valeur du compteur programme courant et l'adresse ETIQ.

```
LDD    GISSE,PCR     ; intruction LDD placée en $2000
                     ; programme GISSE est placé en $3000
```



Autre Exemple :

Soit un programme résident dans la plage `$8000---$8FFF` (soit 4 Ko) et possédant un tableau de donnée dans la plage `$8000---$807F`
Supposant qu'une instruction implantée à l'adresse **$8200** ait besoin d'une donnée 8 bits à l'adresse **$8000**

Dans un mode étendu on écrirait

```
8200  B6 8000        LDA   $8000
```

Si au lieu de la plage **$8000---$8FFF** on désire charger l'ensemble du programme dans la plage **$2000---$207F** le code à l'adresse $2200 serait toujours égal à `B6 8000  LDA $8000`

Le problème est que le programme ne se trouve plus en $8000, l'instruction `LDA $8000` fait appel à une adresse localisée à l'extérieur du tableau, ce dernier étant maintenant dans la plage **$2000---$207F**

Pour rendre opérationnel le nouveau programme il faudrait réécrire le source en

```
2200 B6 2000          LDA   $2000
```

Le mode indexé avec déplacement relatif au PC permet de résoudre ce problème de translabilité.
Le tableau de données étant toujours au même endroit **$8000---$807F** l'instruction `LDA $8000` est transformé en

```
8200 A6 8D  FDFC      LDA   $8000,PCR
8204
```

`A6` OpCode du mode indexé
`8D` Post byte indiquant qu'il y a mode indexé à déplacement sur 16 bits relatif au PC
`FDFC` représente le code opérande (en 16 bits signé) qui mesure la distance relative entre l'adresse **$8000** et l'adresse de fin d'instruction **$8204**.

C'est un offset 16 bits qui est négatif dans cet exemple.

```
OFFSET =   Adrs inscrite      - Adrs fin d'instruction
$FDFC  =   $8000              - $8204
```

A l'exécution, le processeur effectue le calcul inverse pour retrouver l'adresse de la donnée.

```
Adrs de la donnée =   Adrs fin d'instruction  + OFFSET
$8000             =   $8204                   + $FDFC
```

Lors d'un chargement de l'ensemble du programme à une autre adresse, par exemple entre **$2000---$2FFF** le même code objet se retrouve en **$2200**

```
2200 A6 8D  FDFC      LDA   $2000,PCR
8204
```

Cependant à la lecture des deux octets A6 et 8D, le processeur effectue le calcul suivant pour retrouver l'adresse de la donnée.

```
$2000  =   $2204              + $FDFC
```

Cette fois-ci, la donnée se trouve bien en $2000 et non en $8000 comme dans la situation du mode étendu.

On dit que le programme en entier est rendu translatable, car aucune retouche du code objet n'est nécessaire lors d'un chargement à une adresse différente de l'adresse servant à la phase d'assemblage.

Pour plus de renseignement concernant les programmes translatables, voir l'ouvrage 03 "Programmation du 6809" par Rodnay ZAKS et Willam LABIAK (édition SYBEX). Page 206 et 207, concernant les PIC (Position Independant Code) Code indépendant de la position ou banalisation de l'implantation en mémoire.
Voir également : L'ouvrage n°06 Livre de BUI MINH DUC aux pages IV.23 et III.25.

GEN : Programme réentrant

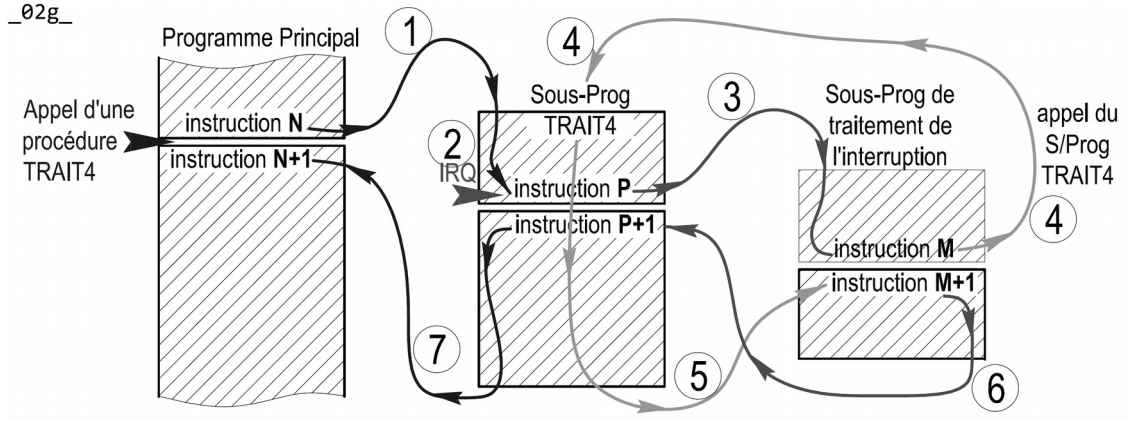
Index retour au Sommaire Liens Rapides

Un programme réentrant est un programme qui peut être utilisé à n'importe quel niveau de priorité.
La notion de réentrance n'est pas liée qu'aux interruptions, c'est aussi lié à la récursivité.

Si une interruption intervient pendant que le µp6809 exécute une tâche dont le niveau de priorité est le plus bas, le traitement est interrompu.

Le programme d'interruption pourra utiliser la même séquence de programme que celle qui était traitée avant l'interruption, si cette séquence est réentrante, c'est-à-dire quelle permet de traiter la demande d'interruption de niveau plus élevé sans pour cela perturber les informations et les résultats de la tâche initiale.

La figure ci-dessous met en évidence le concept de la réentrance.



1. Appel d'un sous-programme appelé par exemple TRAIT4, à partir de l'instruction N.
2. Durant le déroulement du S/Prog TRAIT4 le µp6809 reçoit une interruption IRQ| d'un niveau de priorité plus important que celui du traitement en cours (IRQ| n'est pas masquée). Le µp6809 arrête le traitement du S/Prog TRAIT4 après l'instruction P.
3. Le µp6809 va exécuter le programme de traitement d'interruption IRQ|.
4. Dans le programme de traitement de l'interruption IRQ|, à l'instruction M, il y a un appel au S/Prog TRAIT4.
5. Dès que le S/Prog TRAIT4 est achevé, on retourne au programme de traitement de l'interruption à l'instruction M+1.
6. Dès que le programme de traitement de l'interruption est terminé, on retourne au traitement initial du sous-programme TRAIT4, à l'instruction P+1.
7. Dès que le sous-programme TRAIT4 est terminé, on retourne au programme principal à l'instruction N+1.

Index retour au Sommaire Liens Rapides

Pour que toutes ces opérations se déroulent normalement, il faut :

- D'une part que tous les registres internes du µp6809 soient sauvegardés, ce qui est fait automatiquement sur la pile système.
- D'autre part préserver toutes les données intermédiaires.

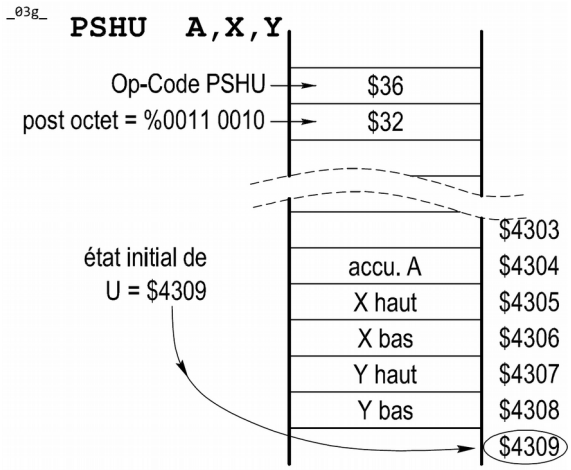
Ces données intermédiaires doivent être sauvegardées dans des endroits différents, ceci en fonction du niveau de priorité; seule l'utilisation d'une ou de plusieurs piles permet cela.

La réentrance est donc possible si le microprocesseur possède plusieurs pointeurs de piles qui permettront de gérer plusieurs zones de sauvegarde affectées aux différents niveaux de priorité.

Dans l'exemple précédent, les données intermédiaires utilisées lors du premier appel que la procédure sont sauvegardées dans la zone 1. Celle du second appelle dans la zone 2.

Le µp6809 possède deux pointeurs de pile S et U dont la gestion est facilitée grâce aux instructions PSHU, PSHS et PULU, PULS qui permettent de ranger ou extraire de la pile n'importe quel registre.
Ces deux pointeurs à son gérés de façon hardware, l'ordre de rangement fixé.

L'instruction `PSHU A,X,Y` avec `U = $4309` occupe l'espace mémoire suivant :



Ces instructions PSHU, PSHS et PULU, PULS permettent donc une manipulation aisée des piles ce qui facilite l'écriture de programmes réentrants.

Il est de plus possible d'utiliser les registres d'index X et Y comme pointeurs de pile gérés par logiciel.

Le mode d'adressage indexé auto-incrémentation/décrémentation par un ou deux permet de gérer des piles logicielles dont les pointeurs sont X ou Y.

Exemple avec X = $2007

L'instruction `STA ,-X` sauvegarde le contenu de A à l'adresse `$2007 - $1`
L'instruction `LDA ,X+` va chercher le contenu de A dans la pile

De la même façon, on peut travailler sur tous les registres du µp6809.

Les instructions suivantes permettent une sauvegarde des registre A, Y et U dans une pile logicielle (avec X = $2007)

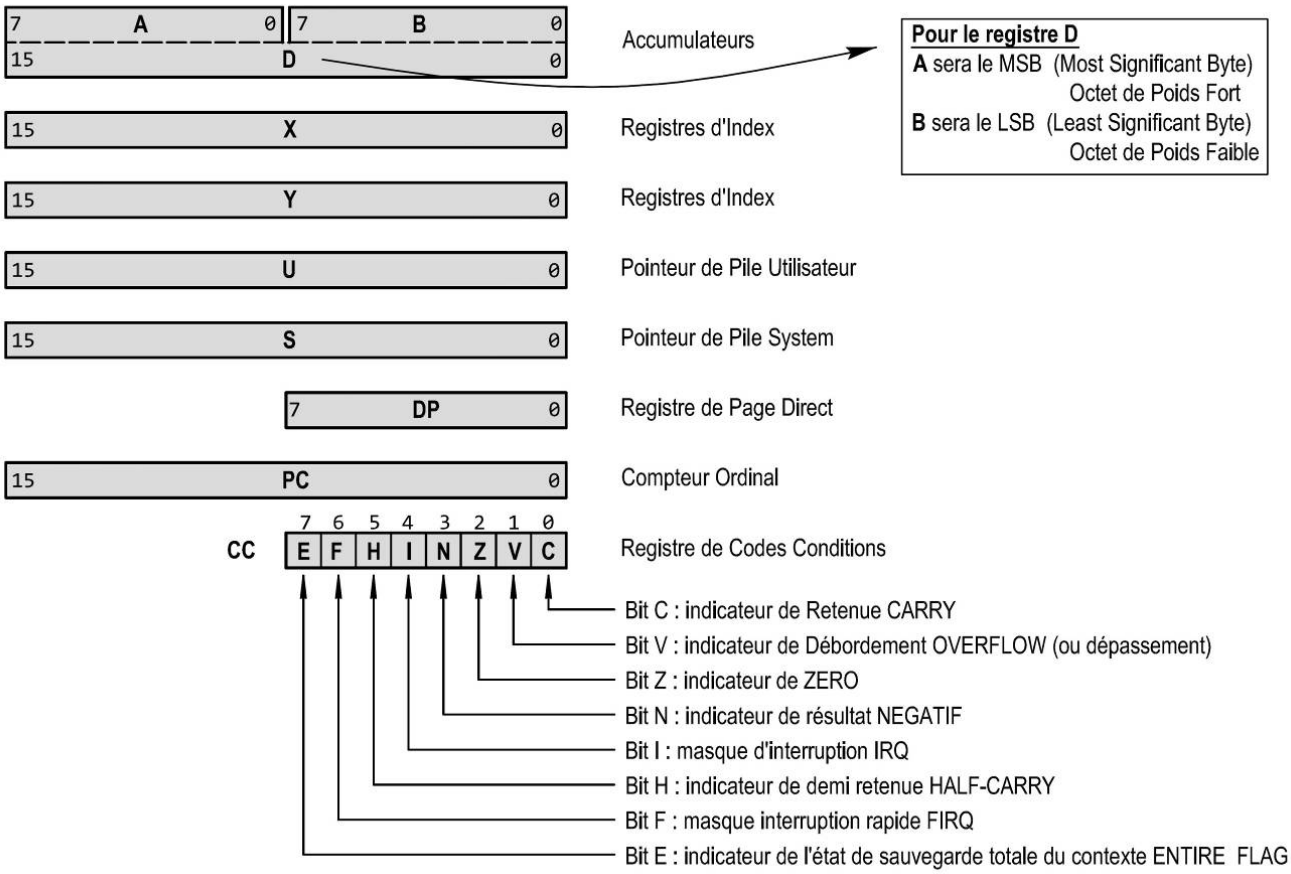
```
STA  ,-X
STY  ,--X
STU  ,--X
```

La restitution des registres s'effectue en sens inverse :

```
LDU  ,X++
LDY  ,X++
LDA  ,X+
```

retour au Sommaire Liens Rapides

_15a_



REG : Les accumulateurs A, B et D

retour au Sommaire | Index | Liens Rapides

On effectuera dans ces deux registres toutes les opérations arithmétiques et logiques. Ils sont pour cela entièrement identiques et mis à part les instructions ABX et DAA, les registres A et B jouent exactement le même rôle sur 8 bits.

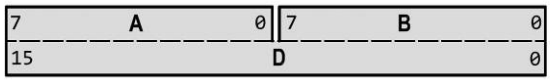
Deux accumulateurs de 8 bits, jouant exactement le même rôle.
Ils sont utilisés pour :

- Les instructions arithmétique et logique.
- Les instructions de comparaisons.
- Les transferts de :
  - mémoire →accumulateur
  - accumulateur →mémoire
  - accumulateur →registre.

A et B peuvent se réunir pour former un accumulateur **D** de 16 bits.

- Le registre A représentant les poids Forts
- Le registre B représentant les poids Faibles

_15c_



REG : Les registres d'index X et Y

retour au Sommaire | Index | Liens Rapides

Ils sont dits registres d'index, car ils permettent d'adresser tout l'espace mémoire avec en plus la capacité d'être pré-décrémenté ou post-incrémenté pour faciliter le traitement de variables en tables.

De préférence, il faut utiliser le registre X à la place d'Y car les instructions comme LDY, STY, et CMPY sont plus longues à exécuter que LDX, STX et CMPX. X et Y sont en 16 bits.

Sont utilisés dans le cadre du mode d'adressage indexé
Ils peuvent être utilisés comme des registres d'usage général :

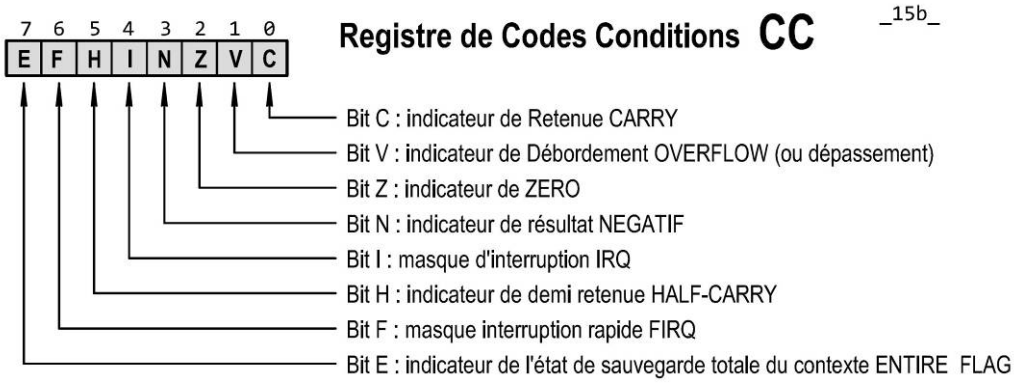
- Pour stocker des résultats intermédiaires

- Comme compteur de boucle.

REG : Le registre de condition CC

retour au Sommaire | Index | Liens Rapides
bit E | bit F | bit H | bit I | bit N | bit Z | bit V | bit C

Ce registre définit à tout instant l'état du µp6809 à la suite de l'exécution d'une instruction. Chaque bit joue un rôle important pour les instructions de saut ou de branchements conditionnels.



Chaque bit du registre CC est mis à 0 ou à 1 :
- Soit lors de l'exécution d'une instruction
- Soit par le programme en forçage direct.

Autrement dit, le registre CC définit l'état du µp6809 après chaque opération arithmétique ou logique. Il aide le µp6809 à prendre des décisions, sous la forme de branchements absolus ou relatifs.

REG : Indicateurs arithmétiques :

_ _ H _ N Z V C C correspond au bit 0

Ils sont positionnés en fonction du résultat des instructions qui manipulent les données.

REG : Indicateurs d'interruption :

E F _ I _ _ _ _ E correspond au bit 7

Ils sont liés au fonctionnement en interruption. Les états de chaque bit peuvent dépendre d'un calcul antérieur.

Certaines instructions particulières peuvent modifier ou lire la valeur de certains bits de CC.
Certaines instructions de branchement ont un déroulement qui dépend de la valeur prise par ces bits.

REG : Bit E (Entire flag) sauvegarde des registres dans la pile bit b7

reg CC | retour au Sommaire | Index | Liens Rapides
bit F | bit H | bit I | bit N | bit Z | bit V | bit C

Indique l'état de la sauvegarde des registres lors d'une interruption. Il indicateur la sauvegarde totale ou partielle d'un contexte. Il est très peu utilisé.

Mis à 0 Une partie seulement des registres est sauvegardée pendant le traitement de l'interruption.
Cas de l'interruption FIRQ ou seul les registre PC et CC sont sauvegardé.

Mis à 1 Tout le contexte est sauvegardé dans la pile système.
Cas de l'interruption IRQ ou tous les registres sont sauvegardé dans la pile S (System).

Ce bit E différencie les deux modes d'interruption FIRQ et IRQ et indique au µp6809 le nombre de registre à dépiler lorsque le µp6809 aura fini d'exécuter le programme d'interruption qui doit se terminer par l'instruction RTI (ReTurn from Interrupt).

REG : Bit F (Fast interrupt mask) Masque d'interruption rapide bit b6

reg CC | retour au Sommaire | Index | Liens Rapides
bit E | bit H | bit I | bit N | bit Z | bit V | bit C

Il conditionne le traitement de la broche FIRQ|.

Il est positionné par le programmeur à l'aide des instructions ANDCC ou ORCC.

Mis à 0 : Lorsqu'il est à 0 ce bit F autorise le traitement des interruptions FIRQ.

Mis à 1 : pour interdire le traitement des instructions FIRQ. Le µp6809 dans ce cas ne tient pas compte des demandes d'interruption arrivant sur cette broche FIRQ|.

Seules les interruptions NMI (Nom Masquable Interrupt) et SWI (SoftWare Interrupt) sont acceptés

Ce bit F est mis à 1 par les interruptions NMI, SWI et FIRQ. Il n'est donc pas affecté par les interruptions IRQ

Si les bits F et I ont été positionnés simultanément à 0, le µp6809 accorde une priorité supérieure à la demande d'interruption rapide FIRQ.

Quand il y a demande d'interruption FIRQ, (à l'inverse de la demande IRQ), seul le registre PC et le registre d'état CC sont sauvegardés dans la pile système S et le bit E (Entire Flag), Etat de sauvegarde, est mis à 0.

REG : Bit H (HALF CARRY) Demi retenue ou retenue intermédiaire bit b5

reg CC | retour au Sommaire | Index | Liens Rapides
bit E | bit F | bit I | bit N | bit Z | bit V | bit C

Il intervient dans les opérations sur les chiffres codés en BCD (Binary Coded Decinal).

Ce bit indique si lors d'une opération, il y a une retenue à faire entre le bit 3 et le bit 4

Mis à 0 : S'il n'y a pas de retenue.

Mis à 1 : S'il y a retenue du bit 3 sur le bit 4.

Voir aussi le code BCD code BCD

En code BCD chaque chiffre compris entre 0 et 9 est codé par un groupement de 4 bits.

Exemple : Le chiffre décimal 16 sera représenté sous la forme :

```
0001 0110 en BCD     au lieu de    0001 0000  en hexadécimal
  1    6                             1    0
```

Exemple : Addition de 9 + 9 = 1 2

```
0000 1001  +  0000 1001  =  0001 0010
```

Cet exemple conduit à un chiffre 12 en notation BCD Ce qui est incorrect

La demi retenue bit H indique alors au processeur qu'il faut ajouter encore 6 à ce résultat pour avoir une représentation correcte de 18 en BCD

```
0001 0010  +  0000 0110  =  0001 1000
                              1    8
```

Est un bit de demi retenue entre les 4 octets de `poids Faibles` et les 4 octets de `poids Forts`.

retour au Sommaire | reg CC | Liens Rapides

Il n'y a pas d'instruction de branchement conditionnel testant ce bit, mais il faut savoir que l'on peut tester ce bit H en utilisant les instructions :

- `TFR CC,A` pour ne pas modifier le registre CC et tester après le registre A
- `ANDCC $20` ce qui transfère la valeur du bit H dans le bit Z.

Fonctionne comme le bit C mais au lieu de détecter le dépassement de capacité du bit 7, le bit H détecte un dépassement de capacité au niveau du bit 3 (utile pour les opérations codées BCD)

Ce bit H est mis à 1 si lors d'une opération une retenue passe du bit 3 au bit 4 dans l'octet concerné par l'opération. Dans ce cas le bit H est appelé retenu du bit 3.

L'instruction d'ajustement décimal DAA utilise ce bit H et le bit C pour corriger le résultat après une addition 8 bits du type ADDA ou ADCA.

Il n'existe pas d'instruction de branchement attribué à ce bit H, qui représente un intérêt mineur.

retour au Sommaire | reg CC | Liens Rapides

Exemples :

```
$02 + $17 = $19      -->  H sera mis à 0
$08 + $19 = $21      -->  H sera mis à 1

    0000   1000     = $08
    0001   1001     = $19
    -----------     -----
    .... 1 0001
```

REG : Bit I (Interrupt mask) Masque d'interruption bit b4

reg CC | retour au Sommaire | Index | Liens Rapides

bit E	bit F	bit H		bit N	bit Z	bit V	bit C

Il est positionné par le programmeur à l'aide des instructions ANDCC ou ORCC.

Il permet d'inhiber les demandes d'interruption de la forme IRQ.

Mis à 0 : Lorsqu'il est à 0 ce bit **I** autorise le traitement des interruptions IRQ.

Mis à 1 : Par le programmeur pour interdire le traitement des instructions IRQ. Le µp6809 dans ce cas ne tient pas compte des demandes d'interruption arrivant sur cette broche IRQ.

Ce bit est positionné à 1 par un RESET.

Quand il y a demande d'interruption IRQ, tous les registres sont sauvegardés dans la pile système S et le bit E (Entire Flag), Etat de sauvegarde, est mis à 1.

REG : Bit N (Négatif) bit b3

reg CC | retour au Sommaire | Index | Liens Rapides

bit E	bit F	bit H	bit I		bit Z	bit V	bit C

Ce bit indique si le résultat d'une opération est négatif.

Toute opération arithmétique ou logique, susceptible de modifier le bit le plus significatif des registres 8 ou 16 bits affecte le bit N.

Ce bit est positionné à la valeur du bit de poids fort du résultat d'une opération.

Mis à 0 : Si le bit 7 (bit de signe) résultat d'une opération est à 0.

Mis à 1 : Si le bit 7 (bit de signe) résultat d'une opération est à 1.

En effet, un monde en complément à 2 est négatif si le bit de poids fort est à 1.

Il indique un résultat négatif, pour toutes les instructions, ce bit N prend la valeur du bit de poids fort de l'opérande ou de l'accumulateur en mouvement.

Il est très utile lors de travaux signés sur les entiers compris entre +127 et -128, il a donc une signification que pour les nombres signés.
On peut se servir du bit N pour déterminer la valeur du bit n°7 de l'octet résultat.

Dans une arithmétique signée

Les chiffres compris entre $0 et $7F (ou $7FFF) sont des chiffres Positifs
Les chiffres compris entre $80 et $FF (ou $8000 ou $FFFF) sont des chiffres Négatifs

Les opérations de chargement LD.... et de stockage ST.... agissement également sur ce bit N. comme pour les opérations arithmétique et logique.

Les instructions CLR, CLRA, CLRB, LSR, LSRA, LSRB mettent le bit N à 0.

Lors d'un dépassement de capacité dû à une opération utilisant le complément à deux, ce bit est incorrect.

Aussi, lors d'une telle opération, le signe est donné par l'opération logique N+V N ou V

REG : Bit Z (ZERO) Indicateur de zéro bit b2

reg CC | retour au Sommaire | Index | Liens Rapides
bit E | bit F | bit H | bit I | bit N | bit V | bit C

Il indique un résultat nul, toutes les instructions positionnent ce bit.

Ce bit est positionné à 1 lorsque le résultat de l'opération précédente est nul.

Attention, ce bit est également positionné par les opérations de chargement LDA, LDB, ... et de stockage STA, STB ....

Mis à 0 Quand le résultat d'une opération quelconque produit un résultat non nul.
si (A ∧ Mém) ≠ 0 instruction BITA
si (B ∧ Mém) ≠ 0 instruction BITB

Mis à 1 Quand le résultat d'une opération quelconque produit un résultat égal à 0 (résultat null).
si (A ∧ Mém) = 0 instruction BITA
si (B ∧ Mém) = 0 instruction BITB

REG : Bit V (OVER FLOW) Dépassement bit b1

reg CC | retour au Sommaire | Index | Liens Rapides
bit E | bit F | bit H | bit I | bit N | bit Z | bit C

Le µp6809 effectue de la même manière l'addition de 2 nombres binaire qu'ils soient signés ou non.
C'est au programmeur d'en décider et de se fixer une convention.

Il indique, lors d'une opération utilisant un complément à deux, s'il y a un dépassement de capacité
Le bit V est le résultat d'un OU EXCLUSIF entre les bits b6 et b7 de l'octet en question.

Le bit V est le bit de débordement en complément à 2.
Il indique un résultat supérieur à ce qu'un octet peut représenter en binaire signé.

Ce bit est significatif uniquement quand le µp6809 manipule des nombres signés.

Seules les opérations comme ADD, ADC, SUB, SBC, NEG et CMP positionnent le bit V à la valeur appropriée.

La multiplication non signée (instruction MUL) et les opérations de décalages à droite n'affectent pas le bit V.

Mis à 0 : (pas de dépassement de capacité)
Dans les opérations de chargements, de stockages et de transfert, dans les opérations logiques, dans les instructions TST, TSTA, TSTB.

Mis à 1 : (il y a dépassement de capacité)
Dans le cas de nombres non signé (de 0 à 255), s'il y a dépassement de capacité.
Les opérations arithmétiques sont seules à mettre le bit V à 1.

Quand le résultat d'une opération arithmétique ne peut être représenté correctement par le contenu des registres.

S'il y a dépassement de capacité (dans le cas ou le µp6809 manipule des nombres signés de -128 à +127). Il est donc positionné si le résultat d'une opération arithmétique en complément à 2 déborde.

Bit V à 1 lorsque :
- Soit si il y a une retenue du bit 6 vers le bit 7 ceci sans retenue du type Carry (bit C)
- Soit il n'y a pas de retenue du bit 6 vers le bit 7, par contre il y a une retenue Carry (bit C)

retour au Sommaire | Index | reg CC | Liens Rapides

Rappel :

Dans le cas de nombre signé (de -128 à +127) le bit de poids fort (bit 7) sert de signe :
- 0 si le nombre est positif
- 1 si le nombre est négatif

Exemple 1 pour le bit V : Addition de $4B et $71

```
  $4B      0100 1011    positif
+ $71    +  0111 0001    positif
------   ------------
 = $BC   =  1011 1100    résultat négatif
```

Le bit V = 1 car il y a retenue du bit 6 vers le bit 7 sans CARRY

Les 2 nombres (dans le cas arithmétique signé) $4B et $71 sont positif mais le résultat est négatif. Il est donc nécessaire d'indiquer que le résultat est erroné bit V = 1.

Exemple 2 pour le bit V : Addition de 2 nombres négatifs $-01 et $-05

```
  $-01        1111 1111   négatif
+ $-05    +   1111 1011   négatif
-------   ---------------
 = $-06   = (1) 1111 1010
```

Dans ce cas le bit V est mis à 0 (retenue du bit 6 vers le bit 7)
Et le bit C à 1

Lorsque V est positionné à 0 le résultat de l'addition est OK
Lorsque V est positionné à 1 le résultat de l'addition est faux

REG : Bit C (CARRY) Retenue bit b0

reg CC | retour au Sommaire | Index | Liens Rapides
bit E | bit F | bit H | bit I | bit N | bit Z | bit V

Il est positionné lors d'une opération arithmétique uniquement (addition, soustraction, multiplication, comparaison, négation, complément à 2, décalage à gauche ou à droite).

Donc les déplacements de données et les opérations logiques n'affecte pas ce bit.

Mis à 0 : (il n'y a pas de retenue)
Par les instructions CLR, CLRA, CLRB.

Mis à 1 : (Il y a une retenue à effectuer)
Quand le résultat d'une opération arithmétique ne peut être représenté correctement par le contenu des registres.
Par les instructions de complémentation à 1, exemple les instructions : COM, COMA, COMB.
Dans le cas d'une addition dont le résultat est supérieur à 255 ($FF) ou lorsque le résultat d'une soustraction (SUB, NEG, CMP, SBC) et positif.

Dans le cas de l'instruction MUL (A multiplié par B résultat sur 16 bits dans D), le bit de **C** est égal au bit b7 du registre D

Cas de l'addition : exemple effectuer la somme de $8A et de $D5

```
    $8A          1000 1010
+   $D5      +   1101 0101
-------      ---------------
 = $15F      = (1) 0101 1111
                   |
                   └──────────mise à 1 du bit C
```

On voit que le résultat est un nombre de 9 bits.
Le bit **C** est positionné à 1 chaque fois qu'il y a une retenue sur le bit de plus fort poids.

REG : Le registre PC (program counter) le compteur ordinal

retour au Sommaire | Index | Liens Rapides

Un programme est exécuté par le microprocesseur de manière séquentielle.

Lorsque le microprocesseur exécute une instruction, il doit connaître l'adresse de la prochaine instruction.

Chaque fois que le microprocesseur va chercher un octet en mémoire, le registre PC (16 bits) est incrémenté de 1 et pointe donc sur l'adresse de la prochaine instruction à exécuter.

Autrement dit, le registre PC détermine l'adresse mémoire à laquelle le µp6809 doit exécuter une instruction. Il sert donc au 6809 pour stocker l'adresse de la prochaine instruction à exécuter. Il est donc modifié à chaque instruction exécutée.

Appelé compteur programme, il est utilisé par le µp6809 pour pointer l'adresse de la mémoire devant être lue et décodé par l'unité centrale à l'étape suivante.

Le registre PC s'incrémente automatiquement à chaque lecture d'un octet a moins qu'une instruction de branchement ou de saut oblige le registre PC à prendre une autre valeur particulière.

Il peut donc être modifié par :

- Une instruction de saut
- Une interruption
- Une instruction de branchement
- Un transfert d'un registre 16 bits dans le registre PC.
- Par un retour de fonction RTS, d'un retour d'interruption RTI ou d'un dépilement **PULS PC** ou **PULU PC**

Certaines instructions du µp6809 considèrent le registre PC comme un registre d'index au même titre que les registres X et Y, à la différence près que l'index varie automatiquement avec l'avancement du programme.

Cette propriété intéressante autorise, entre autre, l'écriture des programmes entièrement translatables dans tout l'espace mémoire.

retour au Sommaire | Index | Liens Rapides

REG : Les registres S et U les pointeurs de PILE

S et U ont un fonctionnement identique. Ils opèrent en mode dernier entré - premier sorti (LIFO Last In First Out)

Comme pour les mémoires vives, les piles servent également à stocker les données ou les adresses à la différence près que la gestion des piles relève d'une procédure très ordonnée.

La rentrée d'une donnée dans la pile, ne détruit pas la donnée précédente et dans le cas du µp6809, des instructions à deux octets peuvent charger l'ensemble des registres du µp6809.

L'organisation des piles du µp6809, mérite une étude approfondie car les erreurs commises dans l'usage des piles sont assez difficiles à détecter car les pointeurs de pile n'ont pas d'adresses fixes.

S et U peuvent à l'occasion servir de enregistre d'index avec la totalité des possibilités de X et de Y.

Le microprocesseur doit connaître à tout instant l'adresse de la prochaine instruction à exécuter.

Le programme principal se déroule jusqu'à l'appel au sous-programme.

Le compteur ordinal registre PC se charge avec l'adresse de début du sous-programme.

Puis il y a exécution du sous-programme jusqu'à la rencontre d'une instruction de retour.

A ce moment le microprocesseur ne sait plus à quelle adresse il doit reprendre le déroulement du programme principal, il faut donc pouvoir mémoriser ces adresses.

Le µp6809 possède deux zones mémoire qui servent de PILE grâce aux registres **S** et **U** :

retour au Sommaire | Index | Liens Rapides

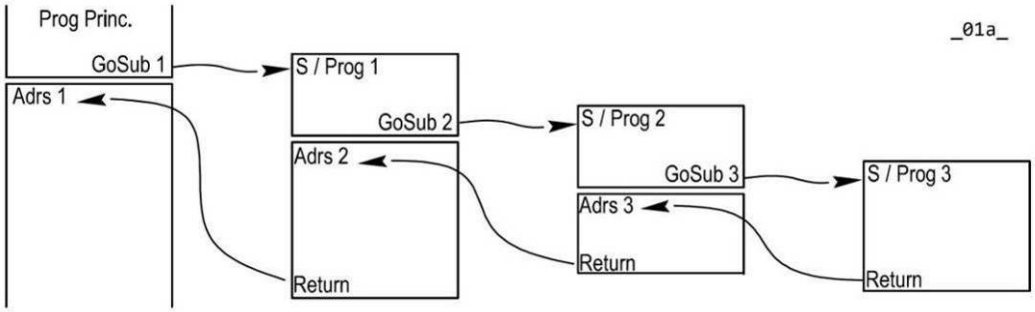
Registre S (16 bits) pour la pile SYSTEME

Il appartient d'office au µp6809, il s'en sert pour mémoriser les états de la machine lors de l'exécution des sous-programmes (saut à un sous-programme) et en cas d'interruption. Il est donc utilisé automatiquement par le µp6809
Sauvegarde des données nécessaires au fonctionnement du µp6809 pour les adresses de retour de sous-programme et le contenu de certains registres internes dans le cas d'interruptions.

Registre U (16 bits) pour la pile UTILISATEUR

Il est entièrement réservé à l'utilisateur.
Le programmeur peut l'utiliser pour transférer ses propres paramètres lors de l'appel des sous-programmes, de la même façon que X et Y.
Il peut être utilisé pour le stockage temporaire de certains résultats et d'aller les retrouver rapidement.

S et U peuvent être installés n'importe où dans l'espace adressable. Leur contenu doit être programmé chaque fois que le µp6809 est mis sous tension, par l'intermédiaire d'instruction spécifique.



La pile système contiendra les octets suivants après l'appel du sous-programme n°3 :
Si le contenu de S était à l'initial `$1F07` par exemple, la répartition des adresse serai la suivante :

retour au Sommaire | Index | Liens Rapides

_01b_

	7 ... 0
$1F01	Adrs S/Pgm 3 H
$1F02	Adrs S/Pgm 3 L
$1F03	Adrs S/Pgm 2 H
$1F04	Adrs S/Pgm 2 L
$1F05	Adrs S/Pgm 1 H
$1F06	Adrs S/Pgm 1 L
$1F07	

Lors du stockage d'une adresse de retour de sous-programme les opérations suivantes sont effectuées :

- Le pointeur de pile est décrémenté
- L'octet de poids faible de l'adresse de retour est chargé
- Le pointeur de pile système est décrémenté
- L'octet de poids fort de l'adresse de retour est chargé

Lors d'une instruction de retour de sous-programme l'opération inverse se produit :

- Le compteur ordinal est chargé par l'octet de poids fort puis par celui de poids faible de l'adresse de retour.
- Le pointeur est incrémenté de 2, l'incrémentation se fait après le chargement (contrairement au cas d'un appel de sous-programme.

Le registre U (utilisateur) fonctionne de la même façon que le registre S.

Les pointeurs de pile S et U pointent sur la dernière donnée stockée dans la pile.

REG : Le registre de page direct DP (Direct Page Register)

retour au Sommaire | Index | Liens Rapides

Format 8 bits, utilisé uniquement dans la mode d'adressage Direct.

Il forme la partie haute de l'adresse à pointer dans le cas d'un adressage direct.

Il est automatiquement remis à 00 par un RESET.

Dans le mode d'adressage direct, une mémoire de 16 bits requiert deux octets en mémoires.

Par contre dans ce même mode d'adressage direct avec la page DP, requiert seulement un octet en mémoire et un cycle machine en moins.

L'unité centrale UC prélève automatiquement le contenu du registre DP pour constituer les 8 bits de poids les plus fort de l'adresse.

Seuls les 8 bits de poids faibles doivent être spécifiés par le programmeur.

_01f_

8 bits de poids FORT		8 bits de poids FAIBLE			16 bits	
7 Valeur du registre **DP** 0		7 Définit par programme 0		=	15 Adresse 16 bits de la mémoire à atteindre 0	

Le mode d'adressage à pour but d'exécuter le code binaire et d'accroître la vitesse d'exécution dans les échanges de donnée avec les périphériques rapides.

retour au Sommaire | Index | Liens Rapides

MA : MODES D'ADRESSAGE DU 6809

Sommaire Principal

Les modes d'adressage complètent le rôle rempli par les instructions et permettent au µp6809 d'accéder à n'importe quelle case mémoire interne ou externe.

MA : Divers types d'adressages.

Index | retour au Sommaire | Liens Rapides

- Implicite ou Inhérent
- Immédiat #
- Direct <
- Etendu > et Etendu indirect [ ]
- Indexé et Indexé indirect [ ]
  - Avec déplacement constant Nul
  - Avec déplacement constant 5 bits
  - Avec déplacement constant 8 bits
  - Avec déplacement constant 16 bits
  - Avec déplacement du contenu d'un accumulateur
  - En auto incrémentation
  - En auto décrémentation
- Relatif au PC Relatif court Relatif long

Ces **9 modes d'adressage** (9 façons de coder les adresses) et les **59 instructions** font du µp6809 le meilleur microprocesseur 8 bits de l'époque, lui procurant 1464 solutions possibles
Le µp6809 sait automatiquement quelle est l'opération à effectuer ainsi que des registres concernés.

MA : Définitions – Données

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

Les informations traitées par le µp6809 se présentent sous la forme de données codées sur 8 ou 16 bits.
A part quelques instructions comme NOP, SYNC qui sont dépourvues de données, les autres instructions implique toujours une transformation ou un mouvement des données.

Dans une instruction de branchement l'adresse de branchement est considérée comme une donnée.
Une donnée est ce que contient un registre ou une mémoire. Elle est encore appelée contenu du registre ou contenu de la mémoire.

MA : Définitions – Adresses

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

A une donnée est affectée une adresse qui indique l'endroit où la donnée se trouve.
Ces cases de rangement (adresses) porte un code sur 16 bits.
L'espace mémoire interne est très restreint, il correspond à l'espace que prend les registres interne du µp6809 c'est-à-dire les registres PC, S, U, Y, X, DP, A, B, D et CC.
L'espace mémoire externe est très étendu, chaque case mémoire porte un numéro codé en hexa.

Voici les points repère à retenir en hexadécimal.

$10	= 16	$1000	= 4096 4 Ko en jargon informatique
$80	= 128	$2000	= 8192 8 Ko en jargon informatique
$FF	= 255 valeur maxi d'un mot de 8 bits	$4000	= 16384
$800	= 2048	$8000	= 32768
		$FFFF	= 65535 valeur maxi d'un mot de 16 bits

MA : Définitions – Conventions d'écriture

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

- L'adresse mémoire externe sera toujours écrite en base Hexadécimale.
- Le contenu d'un registre ou d'une mémoire est écrit avec des parenthèses.
  - (A) est le contenu du registre A.
  - ($F000) est le contenu de la case mémoire ayant pour adresse $F000.
- Pour ranger une donnée de 16 bits le µp6809 mettra :
  Une donnée 16 bits = **MSB** & **LSB**
  - Le mot **MSB** ..... (Most Significant Byte) le plus significatif à l'adresse n.
  - Le mot ..... **LSB** (Least Significant Byte) le moins significatif à l'adresse n+1.

  Exemple si `(X) = $4A3F` Après exécution de `STX $F000` On aura
  `($F000) = $4A`
  `($F001) = $3F` sous un forme plus condensée `($F000)($F001) = $4A3F`
- Dans le cas ou plusieurs adresses sont nécessaires dans une explication, les parenthèses seront remplacées par des < >.

MA : Définitions – Utilité des différents modes d'adressage

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

La principale tâche du µp6809 consiste à traiter les données contenues dans les mémoires.
Les instructions en elle-même accomplissent également cette tâche, mais ces instructions seules ne prennent pas en charge la "situation géographique" de l'octet à traiter.

Les modes d'adressage peuvent, en complément aux instructions :

- Spécifier que la donnée se trouve dans un registre interne du µp6809.
- Spécifier le numéro de la case mémoire à traiter dans l'espace mémoire externe.
- Indiquer la distance relative en octets de l'adresse à traiter par rapport à une adresse de référence.
- Indiquer une adresse intermédiaire ou le µp6809 peut trouver une adresse effective.

Le µp6809 comporte ainsi plusieurs possibilités différentes pour spécifier une adresse de donnée. L'ensemble de ces possibilités constitue les modes d'adressage du µp6809.

Cette méthode évite la multiplication des instructions, au lieu de créer des mnémoniques différents, on a préférer conserver intact le champ opération de l'assembleur et faire varier les modes d'adressages, donc les écritures dans le champ opérande.

Le mnémonique est inchangée, par contre c'est l'op-code qui changera en fonction du mode d'adressage.

MA : Définitions – Notion d'adresse effective EA

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

Quelque soit le mode d'adressage, à l'exécution, le µp6809 détermine toujours l'adresse finale dont le contenu est à modifier. Cette détermination fait intervenir un certain nombre d'adresses intermédiaires.
L'ultime adresse dont le contenu sera traité est désignée comme ADRESSE EFFECTIVE.

MA : Définitions – l'indirection

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

L'espace mémoire du µp6809 est constitué de mots de 8 bits.
Lors du stockage d'une adresse, deux octets adjacents sont nécessaires, on est en présence d'un degré d'indirection.
Ce degré n'est jamais poussé au-delà de 1.

MA : L'adressage INHERENT

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

Appelé aussi adressage implicite.
L'adressage inhérent le code opératoire contient toute l'information nécessaire à l'exécution de l'instruction.
Le code opération comprend implicitement l'adresse de l'opérande qui sera généralement un registre interne du 6809.

N'est pas en réalité un véritable mode d'adressage, car toutes les informations nécessaires à l'exécution de l'instruction se trouvent groupées dans de code opération (op-code).
Utiliser par des instructions qui agissent sur les registres internes et non pas sur la mémoire.
Ces instructions n'ont pas besoin qu'on leur ajoute des opérandes.

MA : Adressage Inhérent Simple

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

Le code opération contient toute l'information nécessaire à l'exécution.

Sur 1 octet

```
ABX,  ASLA, ASLB, ASRA, ASRB, CLRA, CLRB, COMA, COMB, DAA,  DECA, DECB
INCA, INCB, LSLA, LSLB, LSRA, LSRB, MUL,  NEGA, NEGB, NOP,  ROLA, ROLB
RORA, RORB, RTI, RTS,  SEX,  SWI,  SYNC, TSTA, TSTB
```

Exemple : l'instruction `ABX` (X) + (B) → (X) *notion d'écriture (...) veut dire "contenu"*

Avant exécution `(X) = $8034  et  (B) = $15`
Après exécution `(X) = $8049  et  (B) = $15`

Sur 2 octets

```
SWI2, SWI3, TFR, PSHS, PULU, CWAY
```

MA : Adressage Inhérent Paramétré

Index | retour au Sommaire | Mode d'Adressage | Liens Rapides

L'instruction comporte un octet supplémentaire POST-OCTET précise les opérandes intervenant dans l'instruction.

Echange et Transfert de registre . . . . `EXG, TFR`
Instructions d'accès aux piles . . . . . `PSHS, PSHU, PULS, PULU`
Attente d'interruption . . . . . `CWAI`

MA : L'adressage IMMEDIAT

Le code opératoire est directement suivi par un opérande de 1 ou 2 octets.

```
CMPA #$2B06        Comparaison de A avec la valeur hexa 2B06;
ADDB #%1001100     Addition de la valeur binaire 1001100 à B.
```

Le code opératoire est directement suivi par un opérande de 1 ou 2 octets.
On trouvera toujours le signe **#** pour notifier le mode immédiat.
Dans ce mode la donnée 8 ou 16 bits à charger dans un registre se trouve immédiatement dans le champ opérande. Cet opérande est une valeur qui va être : chargé dans ..., comparé à ... ou additionné à un registre du µp6809.
Les instructions comportent deux parties :

- Le code opération codé sur un ou deux octets
- L'opérande sur un ou deux octets, c'est la donnée sur laquelle porte l'instruction.

Exemples :

```
ADDA #$05         ; soit additionner  $05  au contenu de A
CMPS #$12F3       ; soit comparer le contenu de  S  avec la valeur $12F3
CMPX #$4BF6       ; comparaison de X avec la valeur hexa $4BF6
ADDD #%010110     ; addition de la valeur en binaire %010110
```

Ces instructions peuvent atteindre 4 octets comme LDS ou CMPU En mémoire on aura

```
Exemple : 8F00   LDA  #$27      (8F00)=86    (8F01)=27
Exemple : 8C00   LDU  #$2506    (8C00)=CE    (8C01)=25    (8C02)=06
```


MA : L'adressage DIRECT <

Le symbole **<** précise l'adressage direct. Voir également la directive d'assemblage SETDP.
L'adresse 16 bits à atteindre est "scindé" en deux :

- L'octet de poids Fort MSB, est fournit par le contenu du registre **DP**. Cette valeur peut être chargée dans le registre **DP** par l'intermédiaire de l'instruction `TRF A,DP`
- L'octet de poids Faible LSB, est traduit dans l'opérande, qui suit le code opération.

Donc en collaboration avec le registre **DP** (Direct Page register), ce mode permet de charger un accumulateur avec le contenu de n'importe quelle adresse mémoire.
Le registre **DP** permet de spécifier l'une des 256 pages de 256 octets à utiliser (256 x 256 = 64 536 octets).
La page 0 étant les 256 premiers octets des 64 Ko de l'espace total du µp6809.
Après une mise sous tension le registre DP est toujours à zéro.

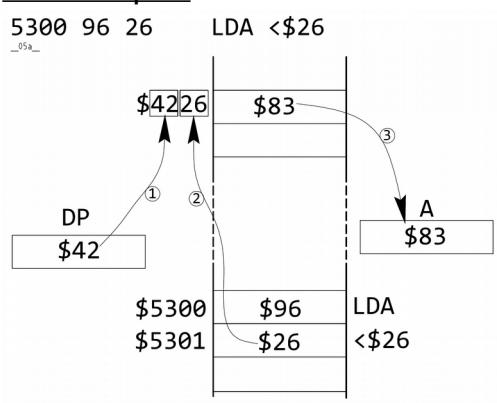
Exemples avec DP = $00

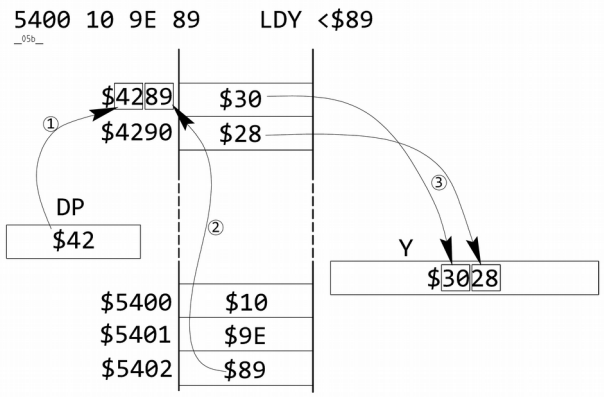
```
LDA <$00    charge A avec le contenu de l'adresse $0000
CMPX <$35   compare X avec le contenu des adresses $0035 et $0036.
```

Le contenu du registre DP doit être préalablement chargé, pour fixer le registre DP il faut utiliser l'instruction TFR pour transférer la valeur de A ou de B dans DP.

```
LDA  #$24     ;
TFR  A,DP     ; initialisation du registre DP
LDA  <$00     ; charge A avec le contenu de l'adresse $2400
CMPX <$56     ; compare X avec le contenu des adresses $2456, $2457
```

Autres exemples :





L'exécution en mode Direct est plus rapide que le mode étendu, du fait de l'économie d'un octet en mémoire et d'un cycle machine.
A l'assemblage, le programmeur doit écrire l'instruction comme en mode étendu, la traduction de l'instruction en code "mode direct" n'est effectuée que si la page de base virtuelle est correctement positionnée par la directive SETDP "Set Direct Page"

MA : L'adressage ETENDU >

C'est un mode qui va concerner le contenu de n'importe quel octet de la mémoire. Ce mode permet donc d'atteindre toute la mémoire mais avec un opérande à 2 octets.

L'adresse de l'opérande suit le code opération. Pour ce mode le symbole d'assemblable est **>**.

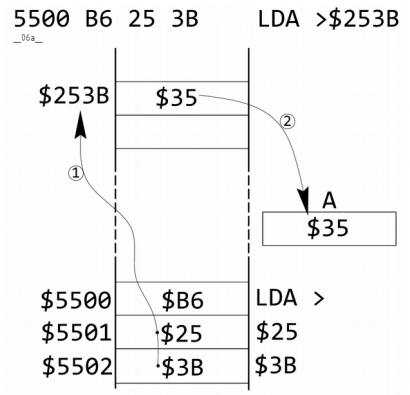
Exemples :

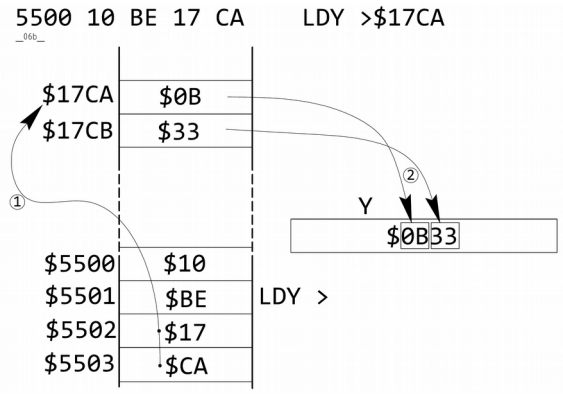
`LDA`	`>$50`	L'accumulateur **A** ne sera pas chargé par le nombre `$50` comme dans l'adressage immédiat. Mais il sera chargé par le contenu de la case mémoire à l'adresse `$0050`.
`LDA`	`>$5100`	Charge le registre A avec le contenu de l'adresse `$5100`.

L'adressage ETENDU est la façon la plus simple de donner une adresse, de donner "en clair" cette adresse à l'aide d'un nombre de 4 chiffres hexa.

L'adressage Etendu spécifie toujours une adresse effective qui ne change pas pendant l'exécution du programme. L'opérande est ici une adresse de 16 bits.

Cette adresse effective peut être écrite dans n'importe quelle base pourvu que sa valeur soit dans la plage **[$0000 , $FFFF]**. Donc grâce à ce mode d'adressage on peut accéder à la totalité de l'espace mémoire du µp6809.





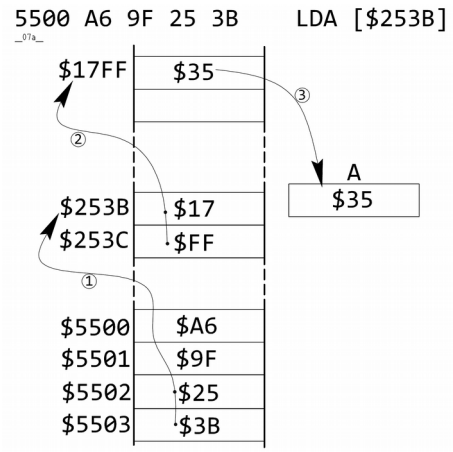
MA : L'adressage ETENDU indirect [ ]

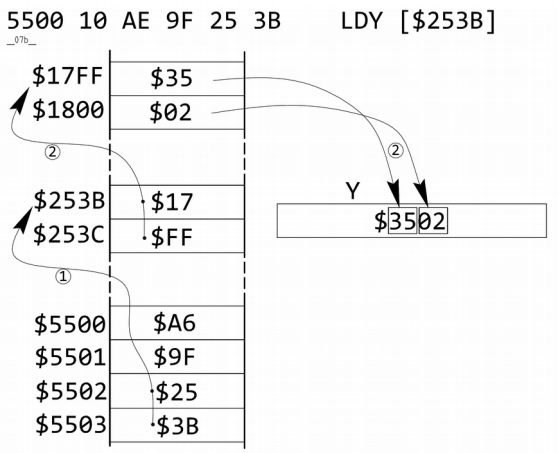
C'est le contenu de l'adresse citée comme opérande qui va indiquer l'adresse mémoire.
Est identique au mode Etendu mais il possède en plus une indirection. On accède à une Adresse Effective en passant par une adresse intermédiaire.
A l'assemblage l'op-code est celui du mode indexé et un post-octet fixe de valeur $9F est inséré entre l'op-code et l'adresse spécifiée dans le champ opérande. La longueur globale de l'instruction atteint alors 4 ou 5 octets.

Supposons que `( $253B ) = $17` `( $253C ) = $FF`

l'instruction `LDA [$253B]` charge A avec le contenu de l'adresse `$17FF`

Cet adressage est très efficace lorsqu'une routine unique doit traiter différentes valeurs. La routine n'a pas besoin d'être doublée ce sont des points de repère qui bougent.
On y trouve aussi une grande utilité dans l'emploi de "tableaux d'adresse" et non des tableaux de données. Ces tableaux d'adresses sont souvent localisés au début ou à la fin d'un programme.





MA : Les adressages INDEXES

Dans ce monde d'adressage, un registre d'index 16 bits (X, Y, U, S ou PC) spécifie une base à laquelle on ajoute un déplacement signé de 0, 5, 8 ou 16 bits.

L'adresse de l'opérande est obtenue par la somme du registre d'index choisi X, Y, U, S ou PC et un déplacement appelé Offset qui suit immédiatement le code opération.

L'offset, s'il existe, indique la distance relative en octets entre l'adresse effective et l'adresse de base.

```
    EA     =    Offset    +     Base
Effective        |              |
Adresse          |              |____Base peut être
                 |                   X, Y, S, U  ou  PC
                 |
                 |_______Déplacement peut être :
                          - Constant (nul, 5, 8 ou 16 bits)
                          - Variable   utilisation de A, B ou D
                          - Auto Incrémenté sur X, Y U ou S
                          - Auto Décrémenté sur X, Y U ou S
                          - Depuis PC en 8 bits
                          - Depuis PC en 16 bits
```

Ce déplacement peut-être défini par une constante faisant partie de l'instruction ou par le contenu d'un des autres registres du µp6809.

Ce mode permet aussi la pré-décrémentation ou la post-incrémentation simple ou double.

Toutes ces possibilités sont précisées par le post octet qui suit le code opératoire.

le mode indexé nécessite un post-octet destiné à renseigner le µp6809 sur le type exact d'adressage à utiliser.

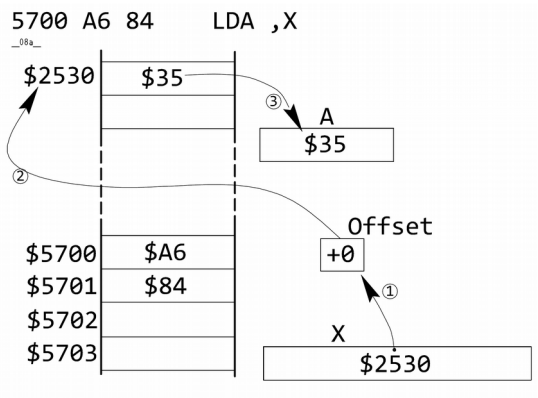
MA : Tableau Regroupant Tous les Types d'Adressage Indexé

	Offset	BASE	INDEXE direct Syntaxe	Binaire	Post Byte	+~	=L9	INDEXE indirect Syntaxe	Binaire	Post Byte	+~	+#
A déplacement constant	Nul	X	,X	1000 **0100**	84	0	0	[,X]	1001 **0100**	94	3	0
		Y	,Y	1010 """"	A4	0	0	[,Y]	1011 """"	B4	3	0
		U	,U	1100 """"	C4	0	0	[,U]	1101 """"	D4	3	0
		S	,S	1110 """"	E4	0	0	[,S]	1111 """"	F4	3	0
	5 bits S = bit de signe vvvv valeur de n	X	n,X	000**S** **vvvv**	00 à 1F	1	0	DKL=0	**Si n est positif** bit5=0 PostByte = n + DKL **Si n est Négatif** bit5=1 PostByte = n +32 + DKL		**+~** Nbre **cycles** ↑ machine supplémentaire **+#** Nombre d'**octets** ↑ supplémentaire après le PostByte	
		Y	n,Y	001**S** **vvvv**	20 à 3F	1	0	DKL=32				
		U	n,U	010**S** **vvvv**	40 à 5F	1	0	DKL=64				
		S	n,S	011**S** **vvvv**	60 à 7F	1	0	DKL=96				
	8 bits	X	n,X	1000 **1000**	88	1	1	[n,X]	1001 **1000**	98	4	1
		Y	n,Y	1010 """"	A8	1	1	[n,Y]	1011 """"	B8	4	1
		U	n,U	1100 """"	C8	1	1	[n,U]	1101 """"	D8	4	1
		S	n,S	1110 """"	E8	1	1	[n,S]	1111 """"	F8	4	1
	16 bits	X	n,X	1000 **1001**	89	4	2	[n,X]	1001 **1001**	99	7	2
		Y	n,Y	1010 """"	A9	4	2	[n,Y]	1011 """"	B9	7	2
		U	n,U	1100 """"	C9	4	2	[n,U]	1101 """"	D9	7	2
		S	n,S	1110 """"	E9	4	2	[n,S]	1111 """"	F9	7	2
Déplacement = accu A, B ou D	Accu A	X	A,X	1000 **0110**	86	1	0	[A,X]	1001 **0110**	96	4	0
		Y	A,Y	1010 """"	A6	1	0	[A,Y]	1011 """"	B6	4	0
		U	A,U	1100 """"	C6	1	0	[A,U]	1101 """"	D6	4	0
		S	A,S	1110 """"	E6	1	0	[A,S]	1111 """"	F6	4	0
	Accu B	X	B,X	1000 **0101**	85	1	0	[B,X]	1001 **0101**	95	4	0
		Y	B,Y	1010 """"	A5	1	0	[B,Y]	1011 """"	B5	4	0
		U	B,U	1100 """"	C5	1	0	[B,U]	1101 """"	D5	4	0
		S	B,S	1110 """"	E5	1	0	[B,S]	1111 """"	F5	4	0
	Accu D	X	D,X	1000 **1011**	8B	4	0	[D,X]	1001 **1011**	9B	7	0
		Y	D,Y	1010 """"	AB	4	0	[D,Y]	1011 """"	BB	7	0
		U	D,U	1100 """"	CB	4	0	[D,U]	1101 """"	DB	7	0
		S	D,S	1110 """"	EB	4	0	[D,S]	1111 """"	FB	7	0
Auto-incrémentation Auto-décrémentation	Auto Incrémenté **+ 1**	X	,X+	1000 **0000**	80	2	0	le + 1 n'est pas possible car en indirect on recherche toujours sur une adresse intermédiaire qui nécessiste 2 octects				
		Y	,Y+	1010 """"	A0	2	0					
		U	,U+	1100 """"	C0	2	0					
		S	,S+	1110 """"	E0	2	0					
	Auto Incrémenté **+ 2**	X	,X++	1000 **0001**	81	3	0	[,X++]	1001 **0001**	91	6	0
		Y	,Y++	1010 """"	A1	3	0	[,Y++]	1011 """"	B1	6	0
		U	,U++	1100 """"	C1	3	0	[,U++]	1101 """"	D1	6	0
		S	,S++	1110 """"	E1	3	0	[,S++]	1111 """"	F1	6	0
	Auto Décrémenté **- 1**	X	,-X	1000 **0010**	82	2	0	le - 1 n'est pas possible car en indirect on recherche toujours sur une adresse intermédiaire qui nécessiste 2 octects				
		Y	,-Y	1010 """"	A2	2	0					
		U	,-U	1100 """"	C2	2	0					
		S	,-S	1110 """"	E2	2	0					
	Auto Décrémenté **- 2**	X	,--X	1000 **0011**	83	3	0	[,--X]	1001 **0011**	93	6	0
		Y	,--Y	1010 """"	A3	3	0	[,--Y]	1011 """"	B3	6	0
		U	,--U	1100 """"	C3	3	0	[,--U]	1101 """"	D3	6	0
		S	,--S	1110 """"	E3	3	0	[,--S]	1111 """"	F3	6	0
Relatif au PC	**depuis PCR 8 bits**		n,PCR	1000 **1100**	8C	1	1	[n,PCR]	1001 **1100**	9C	4	1
				1010 """"	AC	1	1		1011 """"	BC	4	1
				1100 """"	CC	1	1		1101 """"	DC	4	1
				1110 """"	EC	1	1		1111 """"	FC	4	1
	depuis PCR 16 bits		n,PCR	1000 **1101**	8D	5	2	[n,PCR]	1001 **1101**	9D	8	2
				1010 """"	AD	5	2		1011 """"	BD	8	2
				1100 """"	CD	5	2		1101 """"	DD	8	2
				1110 """"	ED	5	2		1111 """"	FD	8	2
	Etendu							[n]	1001 1111	9F	5	2

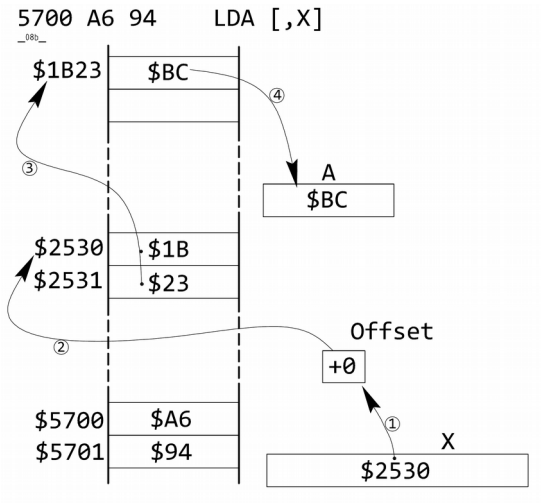
MA : Adressage INDEXE et INDEXE Indirect à déplacement NUL

L'adresse de la donnée correspond au contenu du registre spécifié dans l'opérande.
Le registre d'index contient donc l'adresse effective de l'octet à manipuler.

INDEXE direct



INDEXE indirect



MA : Adressage INDEXE à déplacement 5 bits

Les 5 bits se décomposent `xxxS DDDD` :
`DDDD` de 4 bits de déplacement
`S` d'un bit pour le signe. Ce bit est le 5ième

Pour le registre **X**	de `$00` à `$0F` déplacement Positif	de `$10` à `$1F` déplacement Négatif
Pour le registre **Y**	de `$20` à `$2F` déplacement Positif	de `$30` à `$3F` déplacement Négatif
Pour le registre **U**	de `$40` à `$4F` déplacement Positif	de `$50` à `$5F` déplacement Négatif
Pour le registre **S**	de `$60` à `$6F` déplacement Positif	de `$70` à `$7F` déplacement Négatif

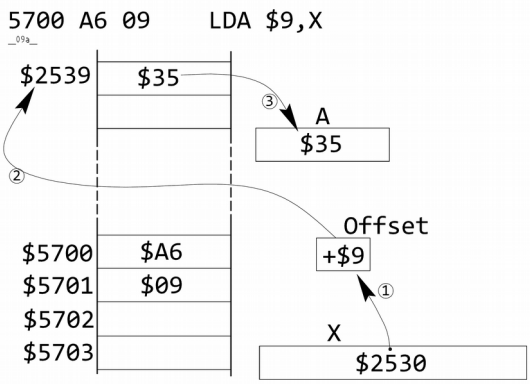
La valeur du déplacement est codée en complément à 2

- bit de signe = 0 déplacement positif
- bit de signe = 1 déplacement négatif

Le déplacement donc est compris entre -16 et +15

L'adresse de la donnée correspond au contenu du registre spécifié dans l'opérande additionné au déplacement compris dans l'opérande.

INDEXE direct



INDEXE indirect

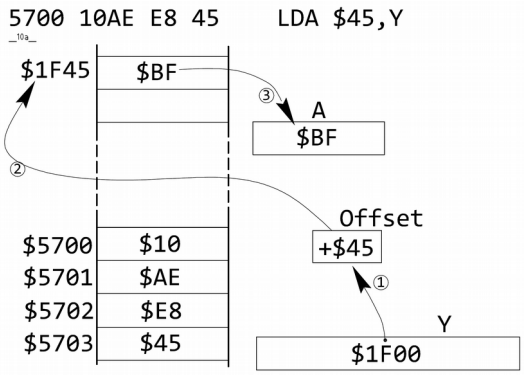
Le mode indexé Indirect n'est pas utilisé dans le déplacement à 5 bits

MA : Adressage INDEXE et INDEXE indirect à déplacement 8 bits

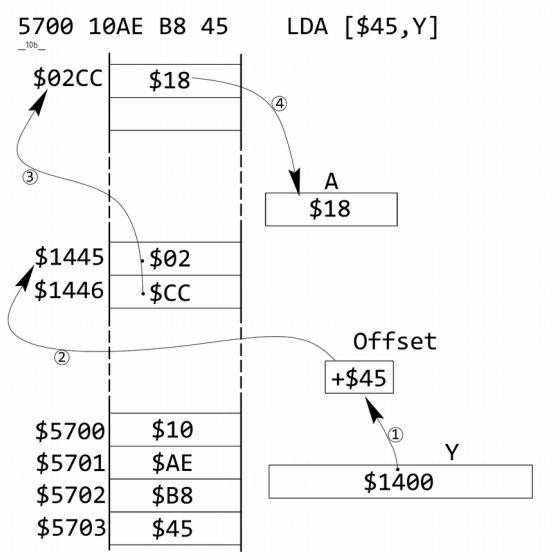
L'adresse de la donnée est obtenue en faisant la somme du contenu du registre d'index et du déplacement codé sur 8 bits. Le déplacement est compris entre -128 et +127.

```
LDA 102,X    ;102 en décimal, en notation en complément à 2
             ;le déplacement est compris entre -128 et +127
LDA $45,Y    ;45 en hexa
```

INDEXE direct



INDEXE indirect



MA : Adressage INDEXE et INDEXE indirect à déplacement 16 bits

Similaire au précédent sauf qu'ici on est sur 16 bits.
Le déplacement est compris entre -32768 et +32767.

```
LDB $2500,U    ; en notation en complément à deux le
               ; déplacement est compris entre
               ; -32768 et +32767
```

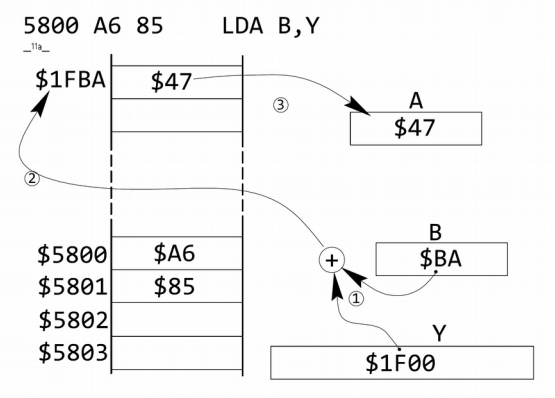
INDEXE direct `LDB $2500,U`
idem croquis ci-dessus

INDEXE indirect `LDB [$2500,U]`
idem croquis ci-dessus

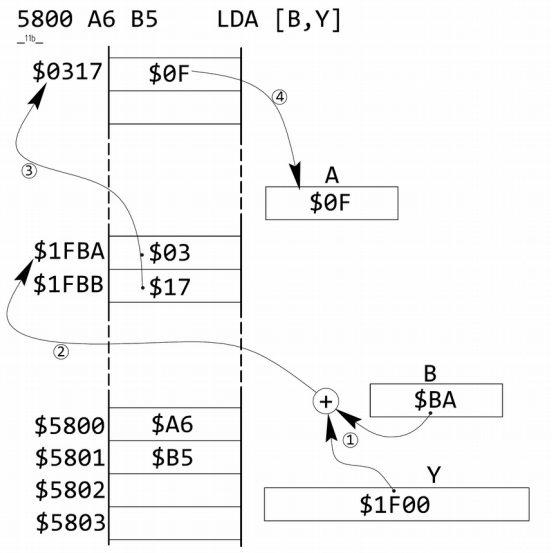
MA : Adressage INDEXE et INDEXE indirect offset par accumulateur

Le contenu des accumulateurs A, B ou D sert de déplacement.
L'adresse de la donnée considérée est obtenue en ajoutant le contenu de l'un des accumulateurs avec le contenu du registre d'index spécifié.
Si l'accumulateur est A ou B, alors la variation n'est que de 256 octets.
Si l'accumulateur est D, alors ou pourra accéder à la totalité de l'espace adressable du µp6809.

INDEXE direct



INDEXE indirect



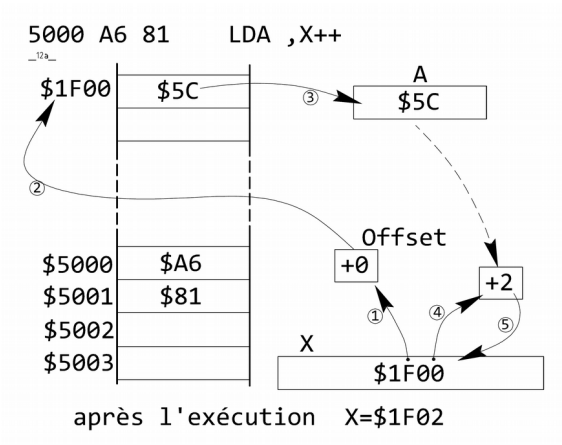
MA : Adressage INDEXE et INDEXE indirect auto-incrémenté

Extrêmement utiles pour l'écriture des boucles.
Le registre contenant l'adresse de l'opérande pointe sur une donnée et effectue le traitement demandé par l'instruction considéré.
Il est **ENSUITE incrémenté automatique de 1 ou 2 unités**, ce qui lui permet de pointer à l'adresse de la donnée suivante. Le contenu registre d'index est donc incrémenté après avoir pointé l'adresse effective.

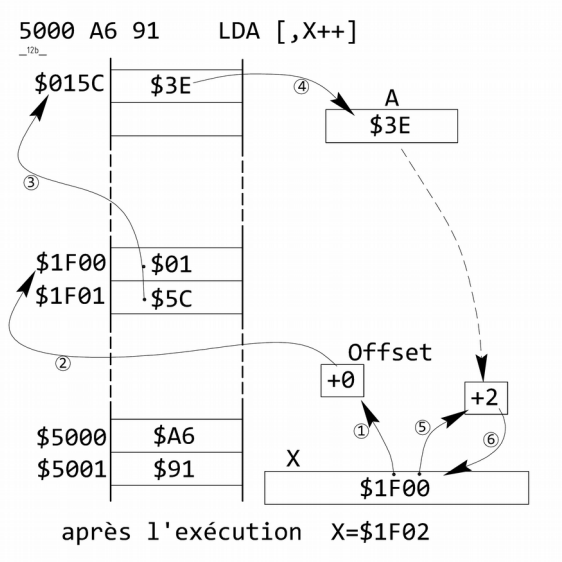
```
STX 0,U+      ; auto-incrémenté de 1 unité
STX 0,U++     ; auto-incrémenté de 2 unités
```

En adressage INDEXE INDIRECT on ne peut pas avoir la décrémentation d'une seule unité, puisqu'il faut 2 octets mémoire pour définir une adresse, donc **`[,-R]` est interdit en Indexé Indirect.**

INDEXE direct



INDEXE indirect



MA : Adressage INDEXE et INDEXE indirect auto-décrémenté

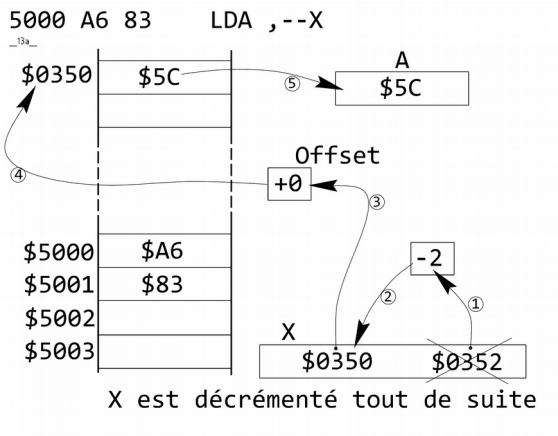
Extrêmement utiles pour l'écriture des boucles.
Le contenu du registre **est TOUT D'ABORD décrémenté de 1 ou 2 unités**, avant de pointer l'adresse effective, le registre contient ensuite l'adresse de la donnée désirée.

```
LDD ,-Y       ; auto-décrémenté de 1 unité
LDA ,--Y      ; auto-décrémenté de 2 unités
```

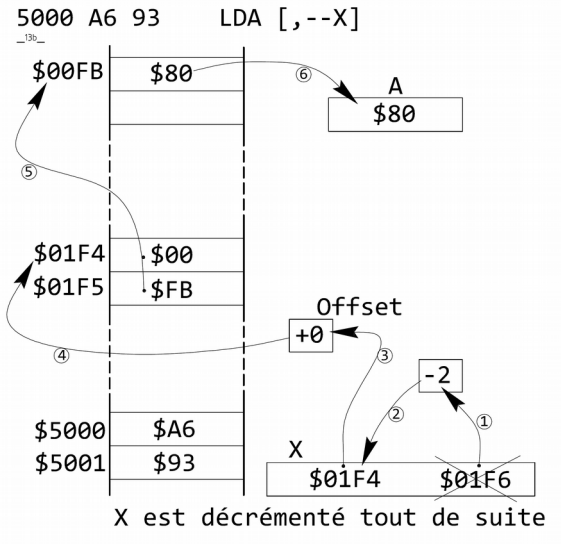
En adressage INDEXE INDIRECT on ne peut pas avoir l'incrémentation d'une seule unité, puisqu'il faut 2 octets mémoire pour définir une adresse donc **`[,R+]` est interdit en Indexé Indirect.**

Pour ces deux adressages. Le post-octet contient le code de 2 bits caractéristique du registre d'index utilisé.

INDEXE direct



INDEX E indirect



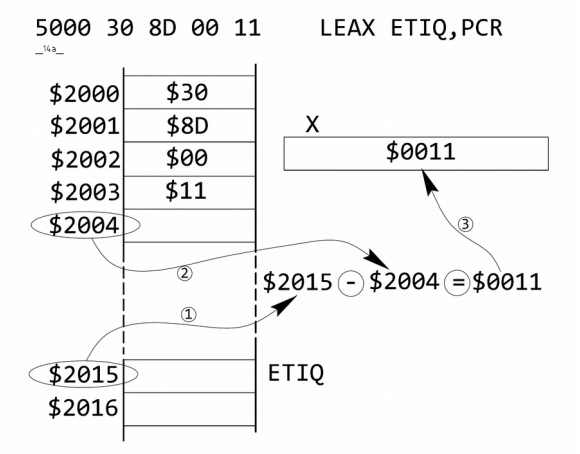
MA : Adressage INDEXE relatif au compteur ordinal PC

Fonctionnement identique aux modes à déplacement constant su 8 ou 16 bits, sauf que le registre est le compteur ordinal PC, il est utilisé comme registre pointeur pour l'adressage.

Avantage : écriture de programme fonctionnant à n'importe quelle adresse mémoire et ceci sans modifications.
Ce mode d'adressage permet de s'affranchir de la localisation du programme : le programme peut être logé à n'importe quelle adresse voir Programme translatable

L'adresse de l'opérande est obtenue par la somme du déplacement (8 ou 16 bits) et le registre PC.

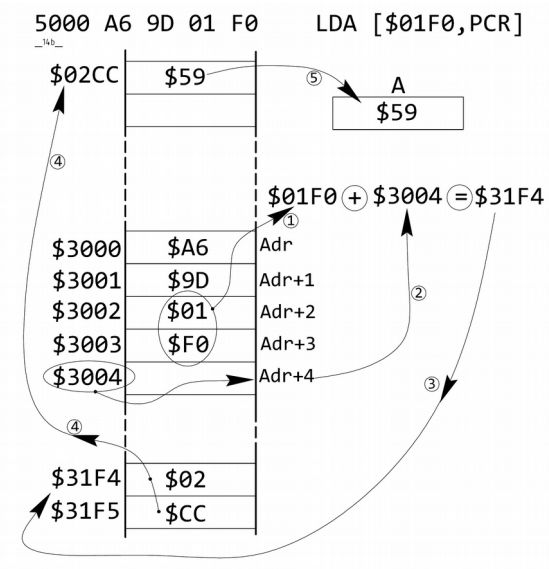
```
        LDA   $26,PCR      ; déplacement sur 8 bits
        LDA   $23F4,PCR    ; déplacement sur 16 bits
ETIQ    EQU   $01FF        ; attribut $01FF à la constante ETIQ
        LDA   ETIQ,PCR     ; déplacement sur 16 bits en fonction d'une Etiquette
        LEAX  ETIQ,PCR     ; charge X avec l'Adresse Effective qui est donnée
                           ; par la position de l'étiquette par rapport au PC
```



Exemple : `LDA $12,PCR` chargement de A avec le contenu de la mémoire située 18 adresses plus loin (18 = $12).

MA : Adressage INDEXE Indirect relatif au compteur ordinal PC

Le contenu du compteur ordinal ajouté à un déplacement constant donne l'adresse de la donnée nécessaire à l'instruction.



MA : Utilisation des modes d'adressage

Cette partie contient des exemples de programmes courts illustrant l'utilisation de plusieurs modes d'adressage.

MA : Utilisation de l'indexation pour des accès séquentiels à un bloc de données

Index | Mode d'Adressage | Liens Rapides | retour au Sommaire

Accès à un tableau de 100 éléments afin de rechercher le caractère @

L'adresse de départ de ce tableau s'appelle BASE.

```
CHERCH  LDX    #BASE      ;
        LDA    #'@        ;
        LDB    #COMPT     ;
TEST    CMPA   ,X+        ; comparaison B avec A et +1 sur X
        BEQ    TROUVE     ; le car recherché est trouvé
        DECB              ; décrémenté B
        BNE    TEST       ; est ce le dernier élément ?
TROUVE                    ;
```

MA : Transfert d'un bloc de données comportant moins de 256 éléments

Mode d'Adressage | Index | Liens Rapides | retour au Sommaire

COMPT est le nombre d'éléments du bloc à déplacer. On suppose ce nombre < 256

La valeur DE est l'adresse de début du bloc, la valeur VERS est l'adresse de début de la zone mémoire où le bloc de donnée doit être transféré.

On déplace un octet à la fois, on garde la trace de l'octet déplacé en stockant sa position dans B.

```
BLKMOV  LDX    #DE       ; init de l'adresse pointeur Source
        LDY    #VERS     ; init de l'adresse pointeur Destination
        LDB    #COMPT    ; B nombre d'élément à transférer
SUITE   LDA    ,X+       ; {adrs X}=>A, puis +1 sur X
        STA    ,Y+       ; A=>{adrs Y}, puis +1 sur Y
        DECB             ; décrémentation du compteur B
        BNE    SUITE     ; tant que B n'est pas à 0 --> SUITE
        .                ; B=0 tous les éléments sont transférés
```

Même programme mais en utilisant le mode d'indexation à déplacement accumulateur. Dans ce cas B set en même temps de valeur de déplacement et de compteur.

Le programme ci-dessous se déroule plus vite car le mode à déplacement par accumulateur nécessite un cycle machine de moins que le mode par incrémentation.

```
BLKMOV  LDX    #DE       ;
        LDY    #VERS     ;
        LDB    #COMPT    ;
SUITE   LDA    B,X       ; ces deux lignes on été
        STA    B,Y       ; modifiées
        DECB             ;
        BNE    SUITE     ;
```

MA : Transfert de bloc de données de plus de 256 éléments

retour au Sommaire | Index | Mode d'Adressage | Liens Rapides

Ce programme transfère 2 octets à la fois, toujours un nombre pairs d'octets.
C'est pourquoi LONG est divisé par 2.

Si LONG, avant la division était impair, le dernier octet ne serait.
Ceci est pris en compte par un test de la retenu après la division.

S'il y a une retenue, on fait +1 sur D.

```
        LDD    #LONG     ;
        LSRA             ; diviser LONG de 16 bits par 2
        RORB             ;
        BCC    PAIR      ;
        ADDD   #1        ; incrémenter D si LONG est impair
PAIR    LDY    #DE       ;
        LDU    #VERS     ;
SUITE   LDX    ,Y++      ; ++ pour 2 octets à transférer à la fois
        STX    ,U++      ; ++
        SUBD   #1        ; pas d'instruction de décrémentation pour D
                         ; alors on soustrait 1 à D
        BNE    SUITE     ;
```

X est utilisé comme registre de transfert Y et U servent de registre d'index

Le code ci-dessus a été optimisé par **(Jacques BRIGAUD** du forum.system-cfg.com) mars 2016.

Il vaut mieux ajouter le 1 avant la division. ***Si c'est pair, le +1 sera effacé.***
Si c'est impair, il sera pris en compte

```
        LDD     #LONG       ;
        ADDD    #1          ;
        LSRA                ; diviser LONG de 16 bits par 2
        RORB                ;
        LDY     #DE         ;
        LDU     #VERS       ;
SUITE   LDX     ,Y++        ; ++ pour 2 octets à transférer à la fois
        STX     ,U++        ; ++
        SUBD    #1          ; pas d'instruction de décrémentation pour D
                            ; alors on soustrait 1 à D
        BNE     SUITE       ;
```

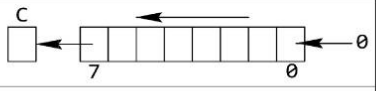
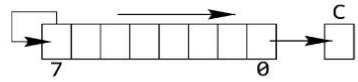
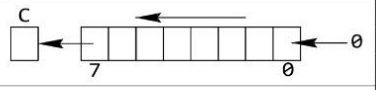
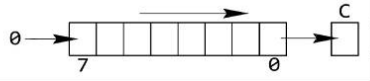
MA : Addition de 2 blocs de données

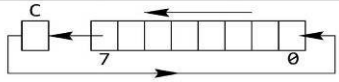
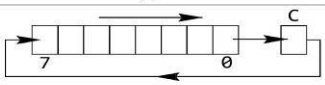
Index retour au Sommaire Mode d'Adressage Liens Rapides

Addition élément par élément, 2 blocs qui débutent respectivement aux adresses BLK1 et BLK2.
Ces 2 blocs ont le même nombre d'élément COMPT.

```
BLKADD  LDX     #BLK1       ;
        LDY     #BLK2       ;
        LDB     #COMPT      ; Nb d'élément à additionner mis dans B
        CLRA                ; RAZ du bit de retenue en prévision de la
                            ; première addition
BOUCLE  LDA     ,X          ; premier élément mis dans A
        ADCA    ,Y+         ; ajout du 2ième élément, puis +1 sur Y
        STA     ,X+         ; résult sauv dans case mém BLK1, +1 sur X
        DECB                ; B décrémenté
        BNE     BOUCLE      ; aussi longtemps que B n'est pas = à 0
```

INS : Tableau Regroupant Toutes les Instructions

	# Immediat			< Direct			Indexé ①			> Etendu			Inhérent			← Modes d'adressage	Bit de CC				
	Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		5 H	3 N	2 Z	1 V	0 C
ABX													3A	3	1	B + X → X					
ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + Mem + C → A	H	N	Z	V	C
ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + Mem + C → B	H	N	Z	V	C
ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + Mem → A	H	N	Z	V	C
ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + Mem → B	H	N	Z	V	C
ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + Mem:Mem+1 → D		N	Z	V	C
ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A ∧ Mem → A		N	Z	0	
ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B ∧ Mem → B		N	Z	0	
ANDCC	1C	3	2													CC ∧ Mem → CC	?	?	?	?	?
ASLA	Idem que												48	2	1		⑧	N	Z	V	C
ASLB	LSL...												58	2	1		⑧	N	Z	V	C
ASL	voir +bas ⇓			08	6	2	68	6+	2+	78	7	3					⑧	N	Z	V	C
ASRA													47	2	1		⑧	N	Z		C
ASRB													57	2	1		⑧	N	Z		C
ASR				07	6	2	67	6+	2+	77	7	3					⑧	N	Z		C
BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3				(Mem ∧ A) seul CC est modifié		N	Z	0	
BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3				(Mem ∧ B) seul CC est modifié		N	Z	0	
CLRA													4F	2	1	0 → A		0	1	0	0
CLRB													5F	2	1	0 → B		0	1	0	0
CLR				0F	6	2	6F	6+	2+	7F	7	3				0 → Mem		0	1	0	0
CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3				(Mem - A) CC modifié	⑧	N	Z	V	C
CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3				(Mem - B) CC modifié	⑧	N	Z	V	C
CMPD	1083	5	4	1093	7	3	10A3	7+	3+	10B3	8	4				(Mem:Mem+1 - D) CC modifié		N	Z	V	C
CMPS	118C	5	4	119C	7	3	11AC	7+	3+	11BC	8	4				(Mem:Mem+1 - S) CC modifié		N	Z	V	C
CMPU	1183	5	4	1193	7	3	11A3	7+	3+	11B3	8	4				(Mem:Mem+1 - U) CC modifié		N	Z	V	C
CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3				(Mem:Mem+1 - X) CC modifié		N	Z	V	C
CMPY	108C	5	4	109C	7	3	10AC	7+	3+	10BC	8	4				(Mem:Mem+1 - Y) CC modifié		N	Z	V	C
COMA													43	2	1	complément(A) → A		N	Z	0	1
COMB													53	2	1	complément(B) → B		N	Z	0	1
COM				03	6	2	63	6+	2+	73	7	3				complément(Mem) → Mem		N	Z	0	1
CWAI	3C	≥20	2													CC ∧ Mem → CC attend interrup	⑦	⑦	⑦	⑦	⑦
DAA													19	2	1	Ajustement Décimal + A → A		N	Z	0	C
DECA													4A	2	1	A - 1 → A		N	Z	V	
DECB													5A	2	1	B - 1 → B		N	Z	V	
DEC				0A	6	2	6A	6+	2+	7A	7	3				Mem - 1 → Mem		N	Z	V	
EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A ⊻ Mem → A		N	Z	0	
EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B ⊻ Mem → B		N	Z	0	
EXG ②	1E	8	2													R1 ⇄ R2					
INCA													4C	2	1	A + 1 → A		N	Z	V	
INCB													5C	2	1	B + 1 → B		N	Z	V	
INC				0C	6	2	6C	6+	2+	7C	7	3				Mem + 1 → Mem		N	Z	V	
JMP				0E	3	2	6E	3+	2+	7E	4	3				Adrs Effective③ → PC					
JSR				9D	7	2	AD	7+	2+	BD	8	3				Saut vers Sous-programme					
LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				Mem → A		N	Z	0	
LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				Mem → B		N	Z	0	
LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				Mem:Mem+1 → D		N	Z	0	
LDS	10CE	4	4	10DE	6	3	10EE	6+	3+	10FE	7	4				Mem:Mem+1 → S		N	Z	0	
LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				Mem:Mem+1 → U		N	Z	0	
LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				Mem:Mem+1 → X		N	Z	0	
LDY	108E	4	4	109E	6	3	10AE	6+	3+	10BE	7	4				Mem:Mem+1 → Y		N	Z	0	
LEAS							32	4+	2+							Adrs Effective③ → S					
LEAU							33	4+	2+							Adrs Effective③ → U					
LEAX							30	4+	2+							Adrs Effective③ → X			Z		
LEAY							31	4+	2+							Adrs Effective③ → Y			Z		
LSLA	Idem que												48	2	1			N	Z	V	C
LSLB	ASL...												58	2	1			N	Z	V	C
LSL	voir +haut ⇑			08	6	2	68	6+	2+	78	7	3						N	Z	V	C
LSRA													44	2	1			0	Z		C
LSRB													54	2	1			0	Z		C
LSR				04	6	2	64	6+	2+	74	7	3						0	Z		C

	# Immediat			< Direct			Indexé ①			> Etendu			Inhérent			← Modes d'adressage	Bit de CC 5	3	2	1	0
	Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		H	N	Z	V	C
MUL													3D	11	1	A x B → D			Z		⑨
NEGA													40	2	1	(Complément à 2 de A) → A	⑧	N	Z	V	C
NEGB													50	2	1	(Complément à 2 de B) → B	⑧	N	Z	V	C
NEG				00	6	2	60	6+	2+	70	7	3				(Complément à 2 de Mem) → Mem	⑧	N	Z	V	C
NOP													12	2	1	Pas d'opération					
ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A ∨ Mem → A		N	Z	0	
ORB	CA	2	2	DA	4	2	EA	4+	2+	FA	5	3				B ∨ Mem → B		N	Z	0	
ORCC	1A	3	2													CC ∨ Mem → CC	⑦	⑦	⑦	⑦	⑦
PSHS	34	5+④	2													Empile registres dans pile S					
PSHU	36	5+④	2													Empile registres dans pile U					
PULS	35	5+④	2													Dépile registres dans pile S					
PULU	37	5+④	2													Dépile registres dans pile U					
ROLA													49	2	1			N	Z	V	C
ROLB													59	2	1			N	Z	V	C
ROL				09	6	2	69	6+	2+	79	7	3						N	Z	V	C
RORA													46	2	1			N	Z		C
RORB													56	2	1			N	Z		C
ROR				06	6	2	66	6+	2+	76	7	3						N	Z		C
RTI													3B	6/15	1	Retour d'interruption	⑦	⑦	⑦	⑦	⑦
RTS													39	5	1	Retour de sous-programme					
SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A - Mem - bit C → A	⑧	N	Z	V	C
SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B - Mem - bit C → B	⑧	N	Z	V	C
SEX													1D	2	1	Extention de signe		N	Z	0	
STA				97	4	2	A7	4+	2+	B7	5	3				A → Mem		N	Z	0	
STB				D7	4	2	E7	4+	2+	F7	5	3				B → Mem		N	Z	0	
STD				DD	5	2	ED	5+	2+	FD	6	3				D → Mem:Mem+1		N	Z	0	
STS				10DF	6	3	10EF	6+	3+	10FF	7	4				S → Mem:Mem+1		N	Z	0	
STU				DF	5	2	EF	5+	2+	FF	6	3				U → Mem:Mem+1		N	Z	0	
STX				9F	5	2	AF	5+	2+	BF	6	3				X → Mem:Mem+1		N	Z	0	
STY				109F	6	3	10AF	6+	3+	10BF	7	4				Y → Mem:Mem+1		N	Z	0	
SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A - Mem → A	⑧	N	Z	V	C
SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B - Mem → B	⑧	N	Z	V	C
SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D - Mem:Mem+1 → D		N	Z	V	C
SWI ⑥													3F	19	1	Interruption logicielle 1					
SWI2⑥													103F	20	2	Interruption logicielle 2					
SWI3⑥													113F	20	2	Interruption logicielle 3					
SYNC													13	≥4	1	Synchro événement extérieur					
TFR ②	1F	6	2													TFR R1,R2 R1 → R2					
TSTA													4D	2	1	Test du contenu de A		N	Z	0	
TSTB													5D	2	1	Test du contenu de B		N	Z	0	
TST				0D	6	2	6D	6+	2+	7D	7	3				Test du contenu de Mem		N	Z	0	

①	Cette colonne donne le nombre de cycle de base et le décompte d'octets. Afin d'obtenir le décompte total, il faut ajouter les valeurs obtenues dans la table Mode d'Adressage Indexé.
②	R1 et R2 paire de registres 8 bits ou paire de registres 16 bits
③	EA est l'Adresse Effective
④	Les instructions PSH et PUL nécessitent cinq cycles plus un cycle pour chaque octet empilé ou dépilé.
⑤	Instructions branchement : 5(6) signifie, 5 cycles si branche non fait, 6 cycles si branchement réalisé
⑥	SWI fixe les bits F et I, SWI2 et SWI3 n'affecte pas les bits F et I
⑦	L'état du registre CC résulte directement de cette instruction.
⑧	Valeur du Flag de demi-retenue non défini.
⑨	Cas particulier : bit C = 1 si le b7 de B est égal à 1

Op	**Op-Code** ou Code instruction en hexadécimal
~	**Nombre de cycle**. Lorsqu'il est suivit d'un + le nombre total de cycles s'obtient en ajoutant la valeur supplémentaire contenue dans le tableau de l'adressage indexés.
#	**Nombre d'octets** traduisant l'encombrement mémoire. Lorsqu'il est suivit d'un + le nombre total de cycles s'obtient en ajoutant la valeur supplémentaire contenue dans le tableau de l'adressage indexés.

⊻	OU Logique exclusif
∧	ET Logique
∨	OU Logique
:	Concaténation

INS : Tableau Regroupant Toutes les Instructions Trié par OpCode (par code opération)

retour au Sommaire | Index | Liens Rapides

OpCode		Mnémo.	Adress.	#
00		NEG	Direct	2
01		---	---	
02		---	---	
03		COM	Direct	2
04		LSR	Direct	2
05		---	---	
06		ROR	Direct	2
07		ASR	Direct	2
08		**ASL, LSL**	Direct	2
09		ROL	Direct	2
0A		DEC	Direct	2
0B		---	---	
0C		INC	Direct	2
0D		TST	Direct	2
0E		JMP	Direct	2
0F		CLR	Direct	2

OpCode		Mnémo.	Adress.	#
10	21	LBRN	Relatif	4
10	22	LBHI	Relatif	4
10	23	LBLS	Relatif	4
10	**24**	**LBCC,LBHS**	Relatif	4
10	**25**	**LBCS,LBLO**	Relatif	4
10	26	LBNE	Relatif	4
10	27	LBEQ	Relatif	4
10	28	LBVC	Relatif	4
10	29	LBVS	Relatif	4
10	2A	LBPL	Relatif	4
10	2B	LBMI	Relatif	4
10	2C	LBGE	Relatif	4
10	2D	LBLT	Relatif	4
10	2E	LBGT	Relatif	4
10	2F	LBLE	Relatif	4
10	3F	SWI2	Inhérent	2
10	83	CMPD	Immediat	4
10	8C	CMPY	Immediat	4
10	8E	LDY	Immediat	4
10	93	CMPD	Direct	3
10	9C	CMPY	Direct	3
10	9E	LDY	Direct	3
10	9F	STY	Direct	3
10	A3	CMPD	Indexé	3+
10	AC	CMPY	Indexé	3+
10	AE	LDY	Indexé	3+
10	AF	STY	Indexé	3+
10	B3	CMPD	Etendu	4
10	BC	CMPY	Etendu	4
10	BE	LDY	Etendu	4
10	BF	STY	Etendu	4
10	CE	LDS	Immediat	4
10	DE	LDS	Direct	3
10	DF	STS	Direct	3
10	EE	LDS	Indexé	3+
10	EF	STS	Indexé	3+
10	FE	LDS	Etendu	4
10	FF	STS	Etendu	4

11	3F	SWI3	Inhérent	2
11	8C	CMPS	Immediat	4
11	83	CMPU	Immediat	4
11	93	CMPU	Direct	3
11	9C	CMPS	Direct	3
11	A3	CMPU	Indexé	3+
11	AC	CMPS	Indexé	3+
11	B3	CMPU	Etendu	4
11	BC	CMPS	Etendu	4

12		NOP	Inhérent	1
13		SYNC	Inhérent	1
14		---	---	
15		---	---	
16		LBRA	Relatif	3
17		LBSR	Relatif	3
18		---	---	
19		DAA	Inhérent	1
1A		ORCC	Immediat	2
1B		---	---	
1C		ANDCC	Immediat	2
1D		SEX	Inhérent	1
1E		EXG	Immediat	2
1F		TFR	Immediat	2

OpCode		Mnémo.	Adress.	#
20		BRA	Relatif	2
21		BRN	Relatif	2
22		BHI	Relatif	2
23		BLS	Relatif	2
24		**BCC,BHS**	Relatif	2
25		**BCS,BLO**	Relatif	2
26		BNE	Relatif	2
27		BEQ	Relatif	2
28		BVC	Relatif	2
29		BVS	Relatif	2
2A		BPL	Relatif	2
2B		BMI	Relatif	2
2C		BGE	Relatif	2
2D		BLT	Relatif	2
2E		BGT	Relatif	2
2F		BLE	Relatif	2

30		LEAX	Indexé	2+
31		LEAY	Indexé	2+
32		LEAS	Indexé	2+
33		LEAU	Indexé	2+
34		PSHS	Immediat	2
35		PULS	Immediat	2
36		PSHU	Immediat	2
37		PULU	Immediat	2
38		---	---	
39		RTS	Inhérent	1
3A		ABX	Inhérent	1
3B		RTI	Inhérent	1
3C		CWAI	Immediat	2
3D		MUL	Inhérent	1
3E		---	---	
3F		SWI	Inhérent	1

40		NEGA	Inhérent	1
41		---	---	
42		---	---	
43		COMA	Inhérent	1
44		LSRA	Inhérent	1
45		---	---	
46		RORA	Inhérent	1
47		ASRA	Inhérent	1
48		**ASLA,LSLA**	Inhérent	1
49		ROLA	Inhérent	1
4A		DECA	Inhérent	1
4B		---	---	
4C		INCA	Inhérent	1
4D		TSTA	Inhérent	1
4E		---	---	
4F		CLRA	Inhérent	1

50		NEGB	Inhérent	1
51		---	---	
52		---	---	
53		COMB	Inhérent	1
54		LSRB	Inhérent	1
55		---	---	
56		RORB	Inhérent	1
57		ASRB	Inhérent	1
58		**ASLB,LSLB**	Inhérent	1
59		ROLB	Inhérent	1
5A		DECB	Inhérent	1
5B		---	---	
5C		INCB	Inhérent	1
5D		TSTB	Inhérent	1
5E		---	---	
5F		CLRB	Inhérent	1

60		NEG	Indexé	2+
61		---	---	
62		---	---	
63		COM	Indexé	2+
64		LSR	Indexé	2+
65		---	---	
66		ROR	Indexé	2+
67		ASR	Indexé	2+
68		**ASL,LSL**	Indexé	2+
69		ROL	Indexé	2+
6A		DEC	Indexé	2+
6B		---	---	
6C		INC	Indexé	2+
6D		TST	Indexé	2+
6E		JMP	Indexé	2+
6F		CLR	Indexé	2+

OpCode		Mnémo.	Adress.	#
70		NEG	Etendu	3
71		---	---	
72		---	---	
73		COM	Etendu	3
74		LSR	Etendu	3
75		---	---	
76		ROR	Etendu	3
77		ASR	Etendu	3
78		**ASL,LSL**	Etendu	3
79		ROL	Etendu	3
7A		DEC	Etendu	3
7B		---	---	
7C		INC	Etendu	3
7D		TST	Etendu	3
7E		JMP	Etendu	3
7F		CLR	Etendu	3

80		SUBA	Immediat	2
81		CMPA	Immediat	2
82		SBCA	Immediat	2
83		SUBD	Immediat	3
84		ANDA	Immediat	2
85		BITA	Immediat	2
86		LDA	Immediat	2
87		---	---	
88		EORA	Immediat	2
89		ADCA	Immediat	2
8A		ORA	Immediat	2
8B		ADDA	Immediat	2
8C		CMPX	Immediat	3
8D		BSR	Relatif	2
8E		LDX	Immediat	3
8F		---	---	

90		SUBA	Direct	2
91		CMPA	Direct	2
92		SBCA	Direct	2
93		SUBD	Direct	2
94		ANDA	Direct	2
95		BITA	Direct	2
96		LDA	Direct	2
97		STA	Direct	2
98		EORA	Direct	2
99		ADCA	Direct	2
9A		ORA	Direct	2
9B		ADDA	Direct	2
9C		CMPX	Direct	2
9D		JSR	Direct	2
9E		LDX	Direct	2
9F		STX	Direct	2

A0		SUBA	Indexé	2+
A1		CMPA	Indexé	2+
A2		SBCA	Indexé	2+
A3		SUBD	Indexé	2+
A4		ANDA	Indexé	2+
A5		BITA	Indexé	2+
A6		LDA	Indexé	2+
A7		STA	Indexé	2+
A8		EORA	Indexé	2+
A9		ADCA	Indexé	2+
AA		ORA	Indexé	2+
AB		ADDA	Indexé	2+
AC		CMPX	Indexé	2+
AD		JSR	Indexé	2+
AE		LDX	Indexé	2+
AF		STX	Indexé	2+

B0		SUBA	Etendu	3
B1		CMPA	Etendu	3
B2		SBCA	Etendu	3
B3		SUBD	Etendu	3
B4		ANDA	Etendu	3
B5		BITA	Etendu	3
B6		LDA	Etendu	3
B7		STA	Etendu	3
B8		EORA	Etendu	3
B9		ADCA	Etendu	3
BA		ORA	Etendu	3
BB		ADDA	Etendu	3
BC		CMPX	Etendu	3
BD		JSR	Etendu	3
BE		LDX	Etendu	3
BF		STX	Etendu	3

OpCode		Mnémo.	Adress.	#
C0		SUBB	Immediat	2
C1		CMPB	Immediat	2
C2		SBCB	Immediat	2
C3		ADDD	Immediat	3
C4		ANDB	Immediat	2
C5		BITB	Immediat	2
C6		LDB	Immediat	2
C7		---	---	
C8		EORB	Immediat	2
C9		ADCB	Immediat	2
CA		ORB	Immediat	2
CB		ADDB	Immediat	2
CC		LDD	Immediat	3
CD		---	---	
CE		LDU	Immediat	3
CF		---	---	

D0		SUBB	Direct	2
D1		CMPB	Direct	2
D2		SBCB	Direct	2
D3		ADDD	Direct	2
D4		ANDB	Direct	2
D5		BITB	Direct	2
D6		LDB	Direct	2
D7		STB	Direct	2
D8		EORB	Direct	2
D9		ADCB	Direct	2
DA		ORB	Direct	2
DB		ADDB	Direct	2
DC		LDD	Direct	2
DD		STD	Direct	2
DE		LDU	Direct	2
DF		STU	Direct	2

E0		SUBB	Indexé	2+
E1		CMPB	Indexé	2+
E2		SBCB	Indexé	2+
E3		ADDD	Indexé	2+
E4		ANDB	Indexé	2+
E5		BITB	Indexé	2+
E6		LDB	Indexé	2+
E7		STB	Indexé	2+
E8		EORB	Indexé	2+
E9		ADCB	Indexé	2+
EA		ORB	Indexé	2+
EB		ADDB	Indexé	2+
EC		LDD	Indexé	2+
ED		STD	Indexé	2+
EE		LDU	Indexé	2+
EF		STU	Indexé	2+

F0		SUBB	Etendu	3
F1		CMPB	Etendu	3
F2		SBCB	Etendu	3
F3		ADDD	Etendu	3
F4		ANDB	Etendu	3
F5		BITB	Etendu	3
F6		LDB	Etendu	3
F7		STB	Etendu	3
F8		EORB	Etendu	3
F9		ADCB	Etendu	3
FA		ORB	Etendu	3
FB		ADDB	Etendu	3
FC		LDD	Etendu	3
FD		STD	Etendu	3
FE		LDU	Etendu	3
FF		STU	Etendu	3

(caractères en gras) un même op-code peut avoir deux instructions différentes ?

Dans les colonnes **#** si il y a un **+** Alors il y a des octets suplémentaires voir la table d'adressage indexé (PostByte)

Les Branchements

Adressage Indexé

INS : Généralité sur les Instructions

De façon générale une instruction spécifie une tâche élémentaire que la machine doit exécuter.
Une séquence ou suite d'instructions placé dans un ordre bien établit constitue un programme.
Pour le µp6809, une instruction est représentée en machine par 1 à 5 octets binaires.

La description des instructions fait intervenir deux entités :

- **Entité Spatiale.**
  Le nombre d'octets affecté à l'instruction caractérise son encombrement en mémoire.
- **Entité Temporelle**.
  Le nombre de cycle machine nécessaire à l'exécution de l'instruction.

Une instruction "courte" est traduite par un nombre réduit d'octets binaires mais ceci ne signifie pas qu'elle soit 'rapide'. Exemple, une instruction d'empilement ou de dépilement n'occupe que 2 octets mais nécessite un nombre important de cycle machine, donc une durée d'exécution relativement longue.

Fort heureusement, dans la majorité des cas, le nombre d'octets et le nombre de cycles machine varient dans le même sens. Concision et rapidité seront donc les maîtres mots des codes optimisés.

Chaque instruction contient un certain nombre d'informations nécessaires à son exécution :

- L'opération proprement dite.
  La partie binaire de l'instruction porte le nom de code opération, code opératoire ou **op-code**.
- L'origine des données.
  Leur nombre peut varier en 0 et 8. Les données peuvent se trouver dans les mémoires externes ou dans les registres internes
- La destination des résultats.
  Elle peut soit être un registre, soit une mémoire externe de 8 ou 16 bits.
  Cette destination est souvent calculée par le µp6809.
  Pour les instructions de branchement, la destination des résultats est tout simplement l'adresse de la prochaine instruction à charger dans le compteur programme PC.

INS : Représentation des instructions en machine

Certaines instructions nécessitent jusqu'à 5 octets (instructions longues dans le mode d'adressage indexé) :

- 2 octets sont utilisés pour le code opération (op-code).
- 1 octet spécial appelé post-octet précise en général l'origine des données et mode d'adressage.
- 2 octets servent au calcul de l'adresse

Des formes à 2, 3 ou 4 octets existent également.
A l'exécution l'UC du µp6809 détermine la nature du premier octet.

Ajout de Jacques BRIGAUD du forum.system-cfg.com *En fait le 6809 possède 3 pages de code op.*
Si le 6809 trouve le CODE de la page 2, alors il va chercher le CodeOp qui suit le 1er octet dans la page 2
Si le 6809 trouve le CODE de la page 3, alors il va chercher le CodeOp qui suit le 1er octet dans la page 3

La reconnaissance du code opération complet dicte alors si un post-octet est nécessaire ou non.

La nature du post-octet renseigne sur le sur le nombre d'octets nécessaires pour compléter entièrement l'instruction.

Le compteur programme s'incrémente à la lecture de chaque octet.

Si l'instruction en cours n'est pas une instruction de branchement, après l'exécution, le µp6809 interprète l'octet suivant comme le début d'une autre instruction et la procédure se répète indéfiniment.

C'est donc l'UC du µp6809 qui détermine la longueur de l'instruction par la lecture du code opération et éventuellement du post-octet.

Le code opération, à part quelques cas particuliers, est composé de 1 ou 2 octets, il contient la nature de l'opération, il précise partiellement le mode d'adressage.

Le reste de l'instruction est appelé opérande et contient 0, 1, 2, ou 3 octets.

Le post octet précise totalement le mode d'adressage, il se situe dans la partie opérande de l'instruction (voir tableau des adressages indexés).

INS : Catégories d'instructions

Dans un premier temps, la classification des instructions du µp6809 peut se faire en 5 catégories :

- Instructions agissant sur les accumulateurs A, B et les mémoires 8 bits.
- Instructions agissant sur l'accumulateur D et les mémoires 16 bits.
- Instructions agissant sur les registres X, Y et les pointeurs de pile S, U.
- Instructions de branchement.
- Instructions spéciales.

La classification ci-dessus fait ressortir les registres mais ignore la nature des opérations portant sur ces registres.
Une autre classification est alors observée, elle s'opère en 4 subdivisions :

INS : Instructions de transformations des données

Instructions arithmétiques

ABX	Addition non signée de B à X
ADCA, ADCB	Addition d'un contenu mémoire à A ou B avec retenue
ADDA, ADDB	Addition d'un contenu mémoire à A ou B
ADDD	Addition d'un contenu mémoire à D
CLR, CLRA, CLRB	Mise à zéro : d'un contenu mémoire, de A ou de B
DAA	Ajustement décimal de A
DEC, DECA, DEB	Décrémentation : d'un contenu mémoire, de A ou de B
INC, INCA, INCB	Incrémentation : d'un contenu mémoire, de A ou de B
MUL	Multiplication non signée de A par B résultat dans D
NEG, NEGA, NEGB	Complémentation à 2 du contenu mémoire ou de A ou de B
SBCA, SBCB	Soustraction du contenu de A ou de B avec retenue
SEX	Extension du signe de l'accumulateur B à A
SUBA, SUBB	Soustraction du contenu de A ou de B
SUBD	Soustraction du contenu de D

Instructions logiques

ANDA, ANDB	ET logique entre un contenu mémoire et A ou B
ANDCC	ET logique entre un opérande et le registre CC
COM, COMA, COMB	Complémentation logique : d'un contenu mémoire, A ou B
EORA, EROB	OU exclusif entre le contenu mémoire et A ou B
ORA, ORB	OU logique entre le contenu mémoire et A ou B
ORCC	OU logique entre un opérande immédiat et le registre CC

Instructions de décalage

ASL, ASLA, ASLB	Décalage à gauche : d'un contenu mémoire, A ou B
LSL, LSLA, LSLB	idem à ASL, ASLA, ASLB
ASR, ASRA, ASRB	Décalage à droite : d'un contenu mémoire, A ou B
ROL, ROLA, ROLB	Décalage circulaire à gauche : d'un contenu mémoire, A ou B
ROR, RORA, RORB	Décalage circulaire à droite : d'un contenu mémoire, A ou B

Instructions de comparaison et de test

BITA, BITB	Test de bit entre un contenu mémoire et A ou B
CMPA, CMPB	Comparaison d'un contenu mémoire avec A ou B
CMPD	Comparaison d'un contenu mémoire avec D
CMPS, CMPU	Comparaison d'un contenu mémoire avec S ou U
CMPX, CMPY	Comparaison d'un contenu mémoire avec X ou Y
TST, TSTA, TSTB	Test : d'un contenu mémoire, A ou B

INS : Instructions de mouvement des données

Instructions de chargement et de stockage

LDA, LDB	Chargement de A ou B avec un contenu mémoire
LDD	Chargement de D avec un contenu mémoire
LDX, LDY, LDS, LDU	Chargement de X, Y, S ou U avec un contenu mémoire
LEAS, LEAU	Chargement de l'adresse effective dans S ou U
LEAX, LEAY	Chargement de l'adresse effective dans X ou Y
STA, STB	Mise en mémoire de A ou de B
STD	Mise en mémoire de D
STS, STU	Mise en mémoire de S ou de U
STX, STY	Mise en mémoire de X ou de Y

Instructions de transfert interne

EXG R1,R2	Echange de 2 registres internes quelconque mais de même longueur
TFR R1,R2	Transfert de R1 dans R2, R1 et R2 doit être de même longueur

Instructions d'empilement et de dépilement

PSHS, PSHU	Empilement d'un nombre quelconque de registres dans S ou U (Sauf le pointeur concerné)
PULS, PULU	Dépilement d'un nombre quelconque de registres dans S ou U (Sauf le pointeur concerné)

INS : Instructions de branchement

Instructions de branchement relatif

BCC, LBCC	Branchement si pas de retenue
BCS, LBCS	Branchement si retenue
BEQ, LBEQ	Branchement si égal ou identique
BGE, LBGE	Branchement si supérieur ou égal à (signe)
BGT, LBGT	Branchement si supérieur (signe)
BHI, LBHI	Branchement si plus haut
BHS, LBHS	Branchement si plus haut ou identique
BLE, LBLE	Branchement si inférieur ou égal (signe)
BLO, LBLO	Branchement si plus bas
BLS, LBLS	Branchement si plus bas ou identique
BLT, LBLT	Branchement si inférieur (signe)
BMI, LBMI	Branchement si négatif
BNE, LBNE	Branchement si non égal ou non identique
BPL, LBPL	Branchement si positif
BRA, LBRA	Branchement si inconditionnel
BRN, LBRN	Non branchement
BVC, LBVC	Branchement si pas de dépassement
BVS, LBVS	Branchement si dépassement
NOP	Sans opération (Sauf d'un octet)

Instructions de branchement absolu

JMP	Saut inconditionnel

Instructions d'appel et de retour de sous-programme

BSR, LBSR	Branchement relatif à un sous-programme
JSR	Branchement absolu à un sous-programme
RTI	Retour de sous-programme d'interruption
RTS	Retour de sous-programme

Interruptions

CWAI	ET logique du registre CC avec un opérande immédiat puis attente d'interruption
SWI	Interruption logiciel (appelée Interruption programmée)
SWI2	Interruption logiciel (appelée Interruption programmée)
SWI3	Interruption logiciel (appelée Interruption programmée)
SYNC	Synchronisation avec la ligne d'interruption

INS : Instructions CLR CLRA CLRB (CL...= CLear)

Remettre à 0 le contenu d'un accumulateur ou d'une case mémoire

CLR **0** à **(Mém)**
CLRA **0** à **(A)**
CLRB **0** à **(B)**

```
                              efhinzvc
Après l'instruction    CC = ____0100      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions LDA LDB (LD...= LoaD)

Charger l'accumulateur A ou B à l'aide d'une case mémoire.

LDA **(Mém)** à **A**
LDB **(Mém)** à **B**

```
LDA #$94      ; après l'instruction A=$94      # indique l'adressage immédiat
              ;          efhinzvc
              ;  et CC = ____100_      (_ = inchangé)
```

L'instruction LDA met la valeur `$94` dans l'accumulateur **A**
$94 étant négatif (car le bit 7=1) et non nul alors le bit N=1 et bit Z=0

Bit Z Dans le cas des instructions LD... :

- Z mis à 1 si A ou B est chargé à $00
- Z mis à 0 si A ou B est chargé avec une valeur <> de 0

Bit N Un octet peut se présenter :

- Soit un nombre positif entre 0 et 255
- Soit un nombre signé entre -128 et +127 (le nombre négatif étant représenté par son complément à 2)

Le bit **N** est tout simplement la recopie du bit 7 de l'octet.

retour au Sommaire
Index Liens Rapides

INS : Instructions LDD LDX LDY LDS LDU (LD...= LoaD)

Chargement les registres 16 bits D, X, Y, S et U avec le contenu de 2 cases mémoire contiguës ou avec une valeur binaire (cas d'adressage immédiat)

(Mém, Mém+1) → Registre

L'octet de poids fort est chargé en premier (contenu de l'adresse Mém)
L'octet de poids faible est chargé en second (contenu de l'adresse Mém+1)

```
                              efhinzvc
Après l'instruction    CC = ____100_      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions LEAS LEAU LEAX LEAY (LEA = Load Effective Adress)

Charge le registre désigné **S**, **U**, **X** ou **Y** avec l'adresse effective, concerne les pointeurs de donnée (pointeur d'index et piles). **AE → Registre** (AE = Adresse Effective)

Le calcul de l'AE (Adresse Effective) ce fait en fonction du seul mode **d'adressage INDEXE** (Direct ou Indirect) et charge cette valeur dans le pointeur désiré (Registre X, Y, S ou U).

Ces instruction ne fonctionnent donc qu'avec **l'adressage INDEXE** (Direct ou Indirect). Puisque ce sont les seuls modes qui nécessitent un calcul d'adresse effective.

L'instruction LEA... charge l'adresse et non pas la donnée pointée par le registre d'adresse. On peut ainsi définir facilement des blocs de données relatifs à d'autres adresses pendant l'exécution d'un programme.

```
LEAS $80,X      ; adressage INDEXE à déplacement 8 bits
                ; si X=$2000 l'adresse effective = $2080
                ; CC ne sera pas affecté
LEAX -1,X       ; décrémentation de X de -1
LEAX 1,X        ; incrémentation de X de +1
```

La séquence

```
STA  ,X
LEAX  +1,X      ; LEAX 0,X équivaut à 2 NOP
DECB            ;          avec une mise à 1 du
                ;          bit Z si (X) = $0000
```

Peut être remplacer par

```
STA   ,X+       ; stockage avec autoincrémentation
DECB
```

Ces instructions LEAS, LEAU, LEAX, LEAY permettent de reloger des programmes.

Voir programme translatable et programme PIC

Voir également : *L'ouvrage n°06 Livre de BUI MINH DUC aux pages IV.23 et III.25.*
L'ouvrage n°03 de Rodnay ZAKS page 206 et 207 chapitre n°5 les programmes translatable.

LEAX et LEAY n'agissent que sur le bit Z, ce bit est mis à 1 quand X ou Y passe par la valeur 0
Donc ces deux instructions peuvent précéder les branchements courts ou longs du type BEQ ou BNE.

LEAU et LEAS n'agissent sur aucun bit du registre CC, ces deux instructions ne peuvent être employées conjointement à des branchements conditionnels.

L'usage des registres S et U comme compteurs est strictement INTERDIT

Les instructions LEA..... servent à mouvoir les pointeurs ou les index de façon très souple à travers tout l'espace mémoire pour rechercher les adresses des données.

```
                                efhinzvc
Après l'instruction    CC = _____Z__      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instruction SEX (Sign Extended) "Extension de signe"

Transforme un nombre de 8 bits signé en complément à 2 en un nombre signé en complément à 2 de 16 bits.

En fait le nombre codé en complément à deux de 8 bits, stocké dans B est tout simplement transformé en nombre en complément à deux, sur 16 bits, pour être stocké dans D. L'ancienne valeur de A est perdue.

Porte uniquement sur A Si bit7 de B égal à **1** (cas d'un nombre négatif) Alors A = $FF
Si bit7 de B égal à **0** (cas d'un nombre positif) Alors A = $00

```
                                efhinzvc
Après l'instruction    CC = ____NZ__      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions STA STB (ST... = STore (ranger)

A à **(Mém)** **B** à **(Mém)**

Pour ces instructions, pas d'adressage immédiat puisque STA et STB range le contenu de l'accumulateur A ou B dans une case mémoire d'adresse définie.

```
STB A,X      ; mode d'adressage Indexé Direct à déplacement accumulateur.
```

X est le registre d'index
A est l'accumulateur utilisé pour le déplacement

Avant l'instruction `STB A,X` si on a : (B) = $32 (A) = $80 (X) = $2000
Après l'instruction la valeur $32 sera stockée à l'adresse $2080

```
                                efhinzvc
Après l'instruction    CC = ____000_      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions STD STX STY STS STU

Registre à **(Mém, Mém+1)**

L'octet de poids fort est chargé en premier Mém
L'octet de poids faible est chargé en second Mém+1

```
STD ,U++     ; adresse double incrémentation
```

Avant l'instruction `STD ,U++` si on a : (U) = `$53F2` (D) = $1000

Après l'instruction, étant donné que (D) = $10 00
$10 (octet de poids fort de D) est mis dans une case mémoire à l'adresse `$53F2`
$00 (octet de poids faible de D) est mis dans une case mémoire à l'adresse `$53F3`
(U) est incrémenté deux fois et passe à `$53F4`

```
                                efhinzvc
Après l'instruction    CC = ____000_      (_ = inchangé)
```

INS : Instructions TFR et EXG

Pour EXG et TFR le contenu de l'octet suivant le code opération est appelé POST-OCTET (PostByte), il précise la paire de registre sur laquelle s'applique les instructions.

TFR	`TFR R1,R2`	Transfert de registre à registre	**R1** à **R2**
EXG	`EXG R1,R2`	Echange de registre	**R1** βà **R2**

Ces deux instructions ne sont que dans le mode d'adressage Immédiat.

```
          PostOctet = AAAA BBBB
                      |    |
       4 bits de poids     4 bits de poids
          Fort pour R1        Faible pour R2
              (Source)        (Destination)
                  |_____    _____|
                        |  |
          Valeur de AAAA ou de BBBB
                        |  |
             Pour D     0000    $0
             Pour X     0001    $1
             Pour Y     0010    $2
             Pour U     0011    $3
             Pour S     0100    $4
             Pour PC    0101    $5
                                $6
                                $7
             Pour A     1000    $8
             Pour B     1001    $9
             Pour CC    1010    $A
             Pour DP    1011    $B
```


INS : Instruction EXG

Permet l'échange de deux registres 8 bits ou deux registres 16 bits.
Le registre de condition CC n'est pas modifié. **R1** βà **R2**

```
EXG R1,R2      ; échange du contenu de registre de même taille
EXG A,B        ; avant A=$10 B=$40
               ; après A=$40 B=$10
                     efhinzvc
Après l'instruction  CC = ________     (_ = inchangé)
```


INS : Instruction TFR

Permet le transfert d'un registre R1 dans un autre registre R2 de même taille.
Le registre de condition CC n'est pas modifié. . **R1** à **R2**

```
TFR R1,R2      ;transfert le contenu de R1 dans R2
                     efhinzvc
Après l'instruction  CC = ________     (_ = inchangé)
```


INS : Instructions Addition ADCA ADCB (ADdition with Carry).

Permet d'ajouter à A ou B le contenu d'une case mémoire ou d'une valeur binaire dans le cas d'adressage immédiat, en plus on ajoute le bit C

(A) + bit C + (Mém) à **(A)**
(B) + bit C + (Mém) à **(B)**

Exemple : `ADCA #$71` Avec initialement : bit C=1 et (A)=$4B

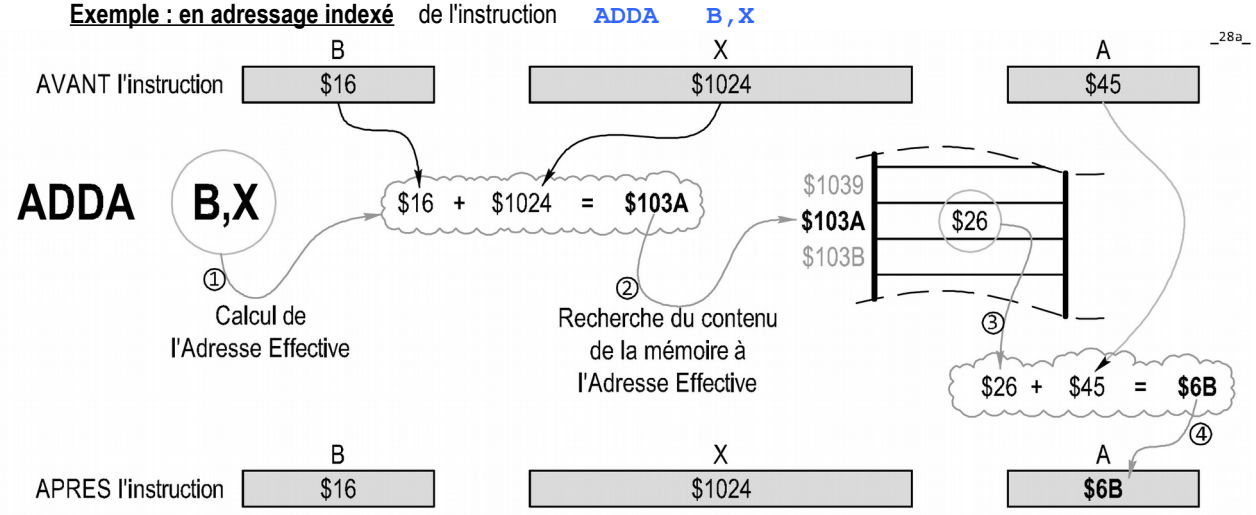
```
   $4B              0100 1011
 + $71            + 0111 0001
 +   1 (bit C)    +          1
----------        -------------
 = $BD            = 1011 1101     Après on aura A=$BD

                     efhinzvc
Après l'instruction  CC = __0_1010     (_ = inchangé)
```

INS : Instructions Addition ADDA ADDB

Fonctionnement identique à **ADCA** et **ADCB** sauf que le bit C n'est pas pris en considération.

(A) + (Mém) à (A)
(B) + (Mém) à (B)

Exemple : en adressage indexé de l'instruction `ADDA B,X`




```
                              efhinzvc
Après l'instruction    CC =  __H_NZVC      (_ = inchangé)
```


INS : Instruction Addition DAA

Utilisé dans les opérations en BCD.
Permet un ajustement décimal sur A. Consiste à faire +06 à A.

Exemple : 2 nombres BCD

```
    8        0000 1000
  + 7      + 0000 0111
 ------    ------------
 = 15     = 0000 1111   = $0F au lieu de 15 en BCD
```

Il faut ajouter 6

```
   $0F        0000 1111
 + $6       + 0000 0110
 -------    ------------
            = 0001 0101   = $15 codé DCB
```

```
                              efhinzvc
Après l'instruction    CC =  ____NZ_C      (_ = inchangé)
```


INS : Instruction Addition ADDD

Permet d'ajouter à D le contenu de la case mémoire ou une valeur sur 16 bits dans le cas d'adressage immédiat.

(D) + (Mém, Mém+1) ⟶ (D)

Exemple :

```
        ADDD $12,X      ; adressage indexé à déplacement sur 5 bits
```

Avant l'instruction (X)=$1000 (D)=$2000 adresse ($1012)=$12 adresse ($1013)=$F4
Après l'instruction (X)=$1000 (D)=$32F4 cc=xxxx 0000

```
                              efhinzvc
Après l'instruction    CC =  ____NZVC      (_ = inchangé)
```

INS : Instruction Addition ABX

Permet d'ajouter les 8 bits Non Signé de B dans le contenu du registre X
(B) + (X) à **(X)**

Exemple : addition BCD sur 16 bits de 2 nombres stocké initialement aux adresses Adr1 et Adr2 le résultat est mis dans Adr3. (Adr1 et Adr2 constitué d'une partie Haute H High et d'une partie Base L Low).

```
LDA    Adr1L    ; charge l'octet poids faible
ADDA   Adr2L    ; additionne l'octet de poids faible
DAA             ; ajustement décimal
STA    Adr3L    ; sauvegarde l'octet de poids faible
                ;
LDA    Adr1H    ; charge l'octet poids fort
ADDA   Adr2H    ; additionne l'octet de poids fort
DAA             ; ajustement décimal
STA    Adr3H    ; sauvegarde l'octet de poids fort
SWI             ;
```

```
                          efhinzvc
Après l'instruction   CC = ________      (_ = inchangé)
```

retour au Sommaire | Index | Liens Rapides

INS : Instructions Soustraction SBCA SBCB (Subtract with Borrow ....)

Soustraction avec retenue. Analogue aux instructions ADCA et ADCB.
(A) – bit C – (Mém) à **(A)**
(B) – bit C – (Mém) à **(B)**

```
                          efhinzvc
Après l'instruction   CC = ____NZVC      (_ = inchangé)
```

retour au Sommaire | Index | Liens Rapides

INS : Instructions Soustraction SUBA SUBB

Soustraction sans retenue. Analogue aux instructions ADDA et ADDB.
(A) – (Mém) à **(A)**
(B) – (Mém) à **(B)**

```
                          efhinzvc
Après l'instruction   CC = ____NZVC      (_ = inchangé)
```

INS : Instruction Soustraction SUBD

(D) – (Mém, Mém+1) à **(D)**

```
                          efhinzvc
Après l'instruction   CC = ____NZVC      (_ = inchangé)
```

retour au Sommaire | Index | Liens Rapides

INS : Instructions Incrémentation INC INCA INCB (INCrement....)

Ajoute 1 à l'accumulateur ou à l'octet mémoire
(Mém) + 1 à **(Mém)**
(A) + 1 à **(A)**
(B) + 1 à **(B)**

Instructions LEA..... S, U, X, Y

Pour Incrémenter les registres X, Y, U ou S voir LEAX, LEAY, LEAU, LEAS

```
                          efhinzvc
Après l'instruction   CC = ____NZV_      (_ = inchangé)
```

retour au Sommaire | Index | Liens Rapides

INS : Instructions Décrémentation DEC DECA DECB (DECrement....)

Soustrait 1 à l'accumulateur ou à l'octet mémoire.
(Mém) - 1 à **(Mém)** **(A) - 1** à **(A)** **(B) - 1** à **(B)**

Instructions LEA..... S, U, X, Y

Pour Décrémenter les registres X, Y, U ou S voir LEAX, LEAY, LEAU, LEAS

```
                          efhinzvc
Après l'instruction   CC = ____NZV_      (_ = inchangé)
```

retour au Sommaire | Index | Liens Rapides

INS : Instruction Multiplication MUL

Multiplication des contenus de A et de B le résultat stocké dans D. Les registres A et B étant considérés comme non signé. Cette instruction est une particularité du µp6809 rarement trouvé sur les microprocesseurs 8 bits du marché de l'époque.
(A x B) à **(D)**

Exemple : au maxi on peut avoir 255 ($FF) x 255 ($FF) = 65025 ($FE01)

```
                              efhinzvc
Après l'instruction    CC =  _____Z_C      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions Négations NEG NEGA NEGB

Donne le complément à 2 du contenu de la case mémoire, de A ou de B.
Autrement dit ces instructions remplacent l'opérande par son opposé.

($\overline{\text{Mém}}$) + 1 à (Mém) **($\overline{\text{A}}$) + 1 à (A)** **($\overline{\text{B}}$) + 1 à (B)**

```
                              efhinzvc
Après l'instruction    CC =  ____NZVC      (_ = inchangé)
```

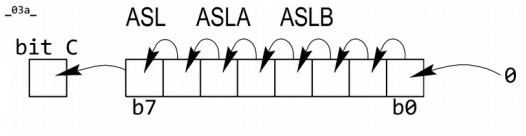
retour au Sommaire
Index Liens Rapides

INS : Instructions De Décalage ASL ASLA ASLB

Décale d'un rang vers la gauche tous les bits de l'accumulateur A ou B, ou d'un octet en mémoire.
Le bit 0 est remplacé par un 0
Le bit 7 est mis dans le bit C du registre de condition CC, ce bit C est appelé bit de retenue (Carry)

Identique aux instructions LSL (même fonction et même OpCode).

ASL = (**A**rithmétique **S**hift **L**eft) Décalage arithmétique vers la gauche.




```
                              efhinzvc
Après l'instruction    CC =  ____NZVC      (_ = inchangé)
```

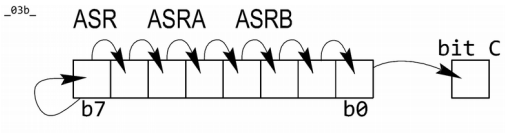
V = N ⊕ C après décalage
⊕ C'est un XOR

retour au Sommaire
Index Liens Rapides

INS : Instructions De Décalage ASR ASRA ASRB

Chaque bit est décalé d'un rang vers la droite
Le bit 0 est mis dans le bit C
Le bit 7 est recopié à la place qu'il occupait précédemment

ASR = (**A**rithmétique **S**hift **R**ight) Décalage arithmétique vers la droite.




```
                              efhinzvc
Après l'instruction    CC =  ____NZ_C      (_ = inchangé)
```

retour au Sommaire Index
Liens Rapides

INS : Instructions "ET" Logiques ANDA ANDB (symbole Λ , & ou parfois *)

J	K	J Λ K
0	0	0
0	1	0
1	0	0
1	1	1

Dans les opérations Binaire le ET s'effectue bit par bit.

Ces instructions ANDA et ANDB effectuent un ET logique entre :
- D'une part le contenu d'une case mémoire ou d'une valeur binaire dans le cas d'adressage immédiat
- Et d'autre part l'un des accumulateurs A ou B.

ANDA **ANDB**

(A) Λ (Mém) à (A)　　(B) Λ (Mém) à (B)
(A) Λ (%xxxxxxxx) à (A)　　(B) Λ (%xxxxxxxx) à (B)

efhinzvc
Après l'instruction CC = ____NZ0_ (_ = inchangé) avec V=0

retour au Sommaire

INS : Instructions "OU" Logiques ORA ORB (symbole V , ≥1 ou parfois +)

Index Liens Rapides

Dans les opérations Binaire le OU inclusif s'effectue bit par bit, entre l'opérande et l'accumulateur A ou B.

J	K	J V K
0	0	0
0	1	1
1	0	1
1	1	1

Ces instructions ORA et ORB effectuent un OU logique entre :

- D'une part le contenu d'une case mémoire ou d'une valeur binaire dans le cas d'adressage immédiat
- Et d'autre part l'un des accumulateurs A ou B.

ORA　　ORB
(A) V (Mém) à (A)　　(B) V (Mém) à (B)
(A) V (%xxxxxxxx) à (A)　　(B) V (%xxxxxxxx) à (B)

efhinzvc
Après l'instruction CC = ____NZ0_ (_ = inchangé) V = 0

retour au Sommaire

INS : Instructions "OU Exclusif" Logiques EORA EORB (symbole ⊻ ou parfois Å)

Index Liens Rapides

Dans les opérations Binaire le OU Exclusif s'effectue bit par bit, entre l'opérande et l'accumulateur A ou B.

J	J	J ⊻ K
0	0	0
0	1	1
1	0	1
1	1	0

EORA　　EROB
(A) ⊻ (Mém) à (A)　　(B) ⊻ (Mém) à (B)
(A) ⊻ (%xxxxxxxx) à (A)　　(B) ⊻ (%xxxxxxxx) à (B)

efhinzvc
Après l'instruction CC = ____NZ0_ (_ = inchangé) V = 0

retour au Sommaire

INS : Instructions de Complémentation COM COMA COMB

Index Liens Rapides

Donne le complément à 1, bit à bit d'un accumulateur ou d'un octet mémoire.

J	$\overline{J}$
0	1
1	0

$(\overline{Mém})$ à (Mém)　　$(\overline{A})$ à (A)　　$(\overline{B})$ à (B)

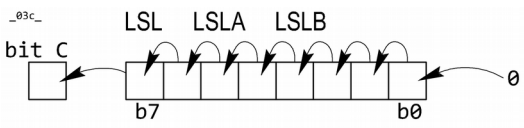
efhinzvc
Après l'instruction CC = ____NZ01 (_ = inchangé) V = 0 C = 1

retour au Sommaire

INS : Instructions De Décalage LSL LSLA LSLB

Index Liens Rapides

LSL = Logical Shift Left (Décalage Logique vers la gauche)



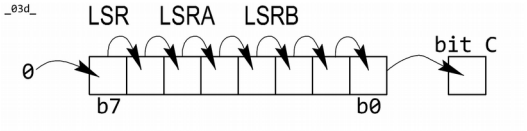
Ces instructions sont identiques aux instructions ASL (Même fonction et même OpCode)

```
                                  efhinzvc
Après l'instruction    CC = ____NZVC      (_ = inchangé)

                          V = N Å C    après décalage
                                Å      C'est un  XOR
```

retour au Sommaire
Index Liens Rapides

INS : Instructions De Décalage LSR LSRA LSRB

LSR = Logical Shift Right (Décalage Logique vers la droite)

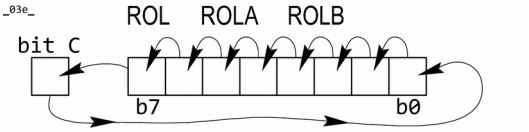



```
                                  efhinzvc
Après l'instruction    CC = ____0Z_C      (_ = inchangé)    N = 0
```

retour au Sommaire
Index Liens Rapides

INS : Instructions De Rotation ROL ROLA ROLB

ROL = ROtate Left Chaque bit est décalé vers la gauche.

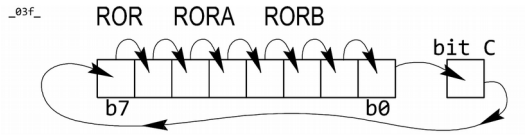



```
                                  efhinzvc
Après l'instruction    CC = ____NZVC      (_ = inchangé)

                          V = N Å C    après décalage
                                Å      C'est un  XOR
```

retour au Sommaire
Index Liens Rapides

INS : Instructions De Rotation ROR RORA RORB

ROR = ROtate Right Chaque bit est décalé vers la droite.




```
                                  efhinzvc
Après l'instruction    CC = ____NZ_C      (_ = inchangé)
```

retour au Sommaire
Index Liens Rapides

INS : Instructions De Test de bits BITA BITB

Effectue un Et logique Λ bit par bit entre le contenu d'un accumulateur A ou B et le contenu de l'opérande.:
Seul le registre de condition CC est modifié.

Et logique	0 Λ 0 = 0	0 Λ 1 = 0	1 Λ 0 = 0	1 Λ 1 = 1

Le bit N Le bit N est positionné à 1 ou 0 selon que le résultat de l'instruction est négatif ou positif.
Le bit Z **Z = 0** si le résulat est non nul
Z = 1 si le résulat est = 0

Exemple : `BITA ,X` type indexé à déplacement constant (nul en l'occurrence).
Avant l'instruction (X) = $0150 (A) = $2A et à l'adresse $0150 on à la valeur $51

```
          $51     0101 0001
          $2A     0010 1010                Le résultat du ET logique est égal à 0
$51 Λ $2A         0000 0000 = $00          alors le bit Z = 1
```

Après l'instruction le **bit Z= 1** **bit N = 0** car le résultat est positif

```
                                  efhinzvc
```

Après l'instruction CC = ____NZ0_ (_ = inchangé) avec V=0

retour au Sommaire Index Liens Rapides

INS : Instructions De Test TST TSTA TSTB

Permet de tester le contenu d'une case mémoire ou d'un accumulateur A ou B.
Après l'instruction la valeur de A n'est pas modifiée
Bit **N = 1** si la valeur est négative
Bit **Z = 1** si la valeur est nulle.
Bit V est mis à 0

Exemple :
```
TSTA ; avec (A) = $B8
```
Le bit Z=0 car $B8 différent de 0
Le bit N=1 car $B8 inférieur à 0 (positif de 0 à 127, négatif de 128 à 255)

```
                    efhinzvc
Après l'instruction CC = ____NZ0_ (_ = inchangé) V = 0
```

retour au Sommaire Index Liens Rapides

INS : Instruction ANDCC

Effectue un ET logique **Λ** entre l'opérande et le registre de condition **CC**

ET logique	0 Λ 0 = 0	0 Λ 1 = 0	1 Λ 0 = 0	1 Λ 1 = 1

Peut être utile pour positionner à 0 certains Bits du registre CC.

(CC) Λ (%xxxxxxxx) à **CC**

Exemple : On souhaite positionner à 0 les bits : 3, 2, 1 et 0 du registre CC
```
    ANDCC   #$F0                ; CC=$12 Avant l'instruction
                                ; CC=$10 Après l'instruction
    (CC) = $12    0001 0010
Opérande = $F0    1111 0000
      $12 Λ $F0 = 0001 0000 = $10
```
les 4 derniers bits de CC ont été remis à 0

```
                    efhinzvc
Après l'instruction CC = __H_NZVC (_ = inchangé) V = 0
```

retour au Sommaire Index Liens Rapides

INS : Instruction ORCC

Effectue un OU logique **V** entre l'opérande et le registre de condition **CC**

OU logique	0 Λ 0 = 0	0 Λ 1 = 1	1 Λ 0 = 1	1 Λ 1 = 1

Exemple : On souhaite positionner à 1 les bits : 3, 2, 2 et 0 de CC
```
    ORCC #$0F                   ; CC=$12 avant l'instruction
                                ; CC=$1F Après l'instruction
    (CC) = $12    0001 0010
Opérande = $0F    0000 1111
      $12 V $0F = 0001 1111 = $1F
```
les 4 derniers bits de CC ont été remis à 1

retour au Sommaire Index Liens Rapides

INS : Instructions De Comparaison CMPA CMPB CMPD CMPS CMPU CMPX CMPY

Afin d'effectuer une comparaison, le contenu de l'opérande est soustrait au contenu du registre spécifié.
Les indicateurs ----NZVC du registre CC sont positionnés suivant le résultat de cette soustraction.

Les contenus de la case mémoire et des registres ne sont pas affectés.
(A) – (Mém) **(B) – (Mém)** **(D) – (Mém, Mém + 1)**
(X) – (Mém, Mém + 1) **(Y) – (Mém, Mém + 1)** **(S) – (Mém, Mém + 1)** **(U) – (Mém, Mém + 1)**

```
                    efhinzvc
Après l'instruction CC = ____NZVC (_ = inchangé)
```

N Å V = 1 à (Reg) < (Mém) en arithmétique signé Å C'est un XOR (ou **V**)
C = 1 à (Reg) < (Mém) en arithmétique non signé
Z = 1 à (Reg) = (Mém) soustraction de 2 valeurs identiques, résultat = 0 donc Z =1

Exemple : Programme qui compare 2 chaînes de caractères. La longueur de ces 2 chaînes est à l'adresse $50.
1ière chaîne stockée à partir de $00 2ième chaîne stockée à partir de $20

```
        LDB    $FF      ; indicateur de chaînes différentes
        LDX    $0000    ; pointe sur la 1ière chaîne
        LDY    $0020    ; pointe sur la 2ième chaîne
COMPT   LDA    ,X+      ; 1er caract de 1ière chaîne
        CMPA   ,Y+      ; 1ier caract de 2ième chaîne
        BNE    FIN      ; non égalité alors terminé
        CMPX   $50      ; dernier caractère ?
        BNE    COMPT    ; non, recommence
        LDB    #$00     ; indicateur de chaîne égales
FIN     STB    $51      ;              ( B ) = $00 si les deux chaînes sont égales
        SWI             ;              ( B ) = $FF si les deux chaînes sont différentes
```

INS : INSTRUCTIONS DE BRANCHEMENTS

Tab Branchements | retour au Sommaire | Index | Liens Rapides

INS : Branchements Inconditionnels

Un branchement est une rupture de séquence. La nouvelle valeur du registre PC s'obtient en ajoutant ou en retranchant une valeur (appelée "Déplacement" ou "Offset") à l'ancien contenu du registre PC.
Le déplacement (ou Offset) est un nombre binaire Signé et peut être sur :

8 bits Branchement COURT : instruction du type **Bxx** (hormis les instructions BITA et BITB)
La plage est de `-128 octets` (déplacement en arrière)
à `+127 octets` (déplacement en avant).

16 bits Branchement LONG : instruction du type **LBxx**
La plage est de `-32768 octets` (déplacement en arrière)
à `+32767 octets` (déplacement en avant).

L'utilisation d'un Assembleur évite le calcul du déplacement : il suffit de mettre un nom dans la colonne étiquette.

INS : Branchements Relatifs

Tab Branchements | retour au Sommaire | Instructions de Branchement | Index | Liens Rapides | Mode d'Adressage

Utilisé uniquement pour les instructions de branchement conditionnel (branchement uniquement si la condition est réalisé). L'équivalent du IF….THEN en basic.
Dans ce mode, l'opérande sera ajoutée ou retranchée (en complément à deux, suivant que l'opérande est positif ou négatif) au compteur ordinal PC.
Rappel : le PC pointe sur la prochaine instruction à exécuter.

INS : Branchements relatifs courts

Tab Branchements | retour au Sommaire | Instructions de Branchement | Index | Liens Rapides | Mode d'Adressage

Le déplacement est codé sur un seul octet. Il permet d'atteindre toute la mémoire par un déplacement de **+127 à -128** octets. Cet adressage permet donc de "brancher" le programme à une instruction se trouvant :

Entre `+ 127 ($7F)` `soit 127 octets` en avant par rapport au PC
Et `- 128 ($80)` `soit 128 octets` en arrière par rapport au PC

Exemple 1 :

```
$00FD
$00FE          BRA $0110     ; branchement à l'adresse $0110.
$0100
```

`$0110 - $0100 = $0010` Cette valeur est ajoutée au PC lors de l'exécution
| |
| Valeur de PC après l'instruction BRA
Adresse de l'instruction BRA

Exemple 2:

```
$0FFC 86 40   LDA #$40
$0FFE 20 xx   BRA $1010  ;  branchement à l'adresse $1010
$1000                    ;  La valeur de xx se calcul à partir du 1er octet suivant.
  .                      ;  On soustrait l'adresse qui suit l'opérande de l'instruction BRA de
  .                      ;  l'adresse de destination $1010 -$1000 = $0010
  .                      ;  on obtient la valeur de l'octet xx=$10 qui se ajouté au compteur
  .                      ;  du programme PC  lors de l'exécution.
$0101 7E 0000  LDX #0    ;  branchement à l'adresse $1010
```

Exemple 3 :

```
        ADDA    $53F8     ; addition de la case mémoire  $53F8  avec A
        BEQ     FIN       ; si le contenu de A=0 après l'addition alors branchement vers FIN
        .                 ; (bit Z de CC est à 1) si non on poursuit après l'instruction  BEQ
FIN     SWI               ;
```

Instructions de Branchement
Tab Branchements | retour au Sommaire
Liens Rapides | Mode d'Adressage | Index

INS : Branchements Relatifs longs

Le "L" devant l'instruction signale le mode relatif long
Fonctionne comme d'adressage relatif court mais le déplacement est codé sur deux octets (16 bits).
Il permet d'atteindre donc toute la mémoire par un déplacement de **+32767 à -32768**.

Entre	`+ 32767 ($7FFF)`	`soit 32767 octets`	en avant par rapport au PC
Et	`- 32768 ($8000)`	`soit 32768 octets`	en arrière par rapport au PC

Il occupe 4 octets mais reste très peu pratique pour la mise en place des routines devant pouvoir s'implanter n'importe où dans la mémoire.

Au niveau du code machine, ce qui différencie l'adressage long de l'adressage court, c'est la présence d'un pré-octet dont la valeur est constante, ce pré-octet est de valeur $10.

Le code opératoire proprement dit est le même dans les deux cas.

Exemple :

```
        LBEQ $2000      ; branche le déroulement à l'adresse $2000
                        ; si le bit Z du registre CC est positionné.
```

Si on utilise une étiquette

```
        BEQ   FIN       ; branchement COURT
        LBEQ  FIN       ; branchement LONG
```

Si le déplacement est connu

```
        BEQ   *+10      ; branchement COURT
        LBEQ  *+$12F6   ; branchement LONG
```

INS : Tableau Regroupant Tous les Branchements

BRANCHEMENTS INCONDITIONNELS

		Mnémonique	Op	~	#	
BRA	**Branchement Toujours** (équivalement à un saut JMP)	BRA $xx	20	3	2	BRanch Always
		LBRA $xxxx	16	5	3	Long BRanch Always
BRN	**Branchement Jamais** (instruction de non opération)	BRN $xx	21	3	2	BRanch Never
		LBRN $xxxx	1021	5	4	Long BRanch Never
BSR	**Branchement à un sous-programme**	BSR $xx	8D	7	2	Branch to SubRoutine
		LBSR $xxxx	17	9	3	Long Branch to SubRoutine

BRANCHEMENTS CONDITIONNELS

	Branchement Si...			Signé		Mnémonique	Op	~	#	
BCC	**Branch si pas de retenue**	si C = 0	bit C	oui		BCC $xx	24	3	2	Branch if Carry Clear
						LBCC $xxxx	1024	5(6)⑤	4	Long Branch if Carry Clear
BHS	**Branch si supérieur ou égal** Reg ⩾ Mém				Non	BHS $xx	24	3	2	Branch if Higher or Same
						LBHS $xxxx	1024	5(6)⑤	4	Long Branch if Higher or Same
BCS	**Branch si retenue**	si C = 1		oui		BCS $xx	25	3	2	Branch if Carry Set
						LBCS $xxxx	1025	5(6)⑤	4	Long Branch if Carry Set
BLO	**Branch si inférieur** Reg < Mém				Non	BLO $xx	25	3	2	Branch if Lower
						LBLO $xxxx	1025	5(6)⑤	4	Long Branch if Lower
BPL	**Branch si positif**	si N = 0	bit N			BPL $xx	2A	3	2	Branch if Plus
						LBPL $xxxx	102A	5(6)⑤	4	Long Branch if Plus
BMI	**Branch si négatif**	si N = 1				BMI $xx	2B	3	2	Branch if Minus
						LBMI $xxxx	102B	5(6)⑤	4	Long Branch if Minus
BVC	**Branch si pas de débordement**	si V = 0	bit V			BVC $xx	28	3	2	Branch if oVerflow Clear
						LBVC $xxxx	1028	5(6)⑤	4	Long Branch if oVerflow Clear
BVS	**Branch si débordement**	si V = 1				BVS $xx	29	3	2	Branch if oVerflow Set
						LBVS $xxxx	1029	5(6)⑤	4	Long Branch if oVerflow Set
BNE	**Branch si différent** Reg ≠ Mém	si Z = 0	bit Z	oui	Non	BNE $xx	26	3	2	Branch if Not Equal
						LBNE $xxxx	1026	5(6)⑤	4	Long Branch if Not Equal
BEQ	**Branch si égal** Reg = Mém	si Z = 1				BEQ $xx	27	3	2	Branch if Equal
						LBEQ $xxxx	1027	5(6)⑤	4	Long Branch if Equal
BLT	**Branch si inférieur** Reg < Mém	N ⊕ V = 1	Ne dépendant pas des bits de CC	oui		BLT $xx	2D	3	2	Branch if Less Than
						LBLT $xxxx	102D	5(6)⑤	4	Long Branch if Less Than
BGT	**Branch si supérieur** Reg > Mém	(N ⊕ V) + Z = 0		oui		BGT $xx	2E	3	2	Branch if Greater Than
						LBGT $xxxx	102E	5(6)⑤	4	Long Branch if Greater Than
BHI		C + Z = 0			Non	BHI $xx	22	3	2	Branch if HIgher
						LBHI $xxxx	1022	5(6)⑤	4	Long Branch if HIgher
BGE	**Branch si supérieur ou égal** Reg ⩾ Mém	N ⊕ V = 0		oui		BGE $xx	2C	3	2	Branch if Greater than Equal to
						LBGE $xxxx	102C	5(6)⑤	4	Long Branch if Greater than Equal to
BLE	**Branch si inférieur ou égal** Reg ⩽ Mém	(N ⊕ V) + Z = 1		oui		BLE $xx	2F	3	2	Branch if Less than or Equal to
						LBLE $xxxx	102F	5(6)⑤	4	Long Branch if Less than or Equal to
BLS		C + Z = 1			Non	BLS $xx	23	3	2	Branch if Lower or Same
						LBLS $xxxx	1023	5(6)⑤	4	Long Branch if Lower or Same

	ET Logique — AND
+	OU Logique — OR
⊕	OU Exclusif Logique — XOR
:	Concaténation
Reg	Registe — (Reg) = contenue du registre
Mém	Mémoire — (Mém) = contenue de la case mémoire
⑤	Instructions branchement : 5(6) signifie, 5 cycles si branche non fait, 6 cycles si branchement réalisé
$xx	Adresse en 8 bits
$xxxx	Adresse en 16 bits

Op	**Op-Code** ou Code instruction en hexadécimal
~	**Nombre de cycle**. Lorsqu'il est suivit d'un + le nombre total de cycles s'obtient en ajoutant la valeur supplémentaire contenue dans le tableau de l'adressage indexés.
#	**Nombre d'octets** traduisant l'encombrement mémoire. Lorsqu'il est suivit d'un + le nombre total de cycles s'obtient en ajoutant la valeur supplémentaire contenue dans le tableau de l'adressage indexés.

INS : Branchement Inconditionnel

INS : Branchement Inconditionnel JMP (JuMP to effective address)

C'est un saut, une rupture de séquence. Il est obtenu en chargeant le compteur ordinal (PC Program Counter) avec l'adresse spécifiée par l'opérande, adresse à laquelle il faut sauter.

Adresse Effective à **PC**

Equivalente au GOTO en Basic, sauf qu'en assembleur l'étiquette peut avoir un nom

```
JMP DEBUT          ;
JMP $F03E          ; charge le PC et effectue le saut vers l'adresse $F03E
```

Pour avoir un programme structuré, cette instruction est à éviter ! Car c'est l'équivalent du GOTO en Basic !

ATTENTION : pour du code relogeable, le JMP est à bannir.

INS : Branchement Inconditionnel Relatif BRA LBRA (BRanch Always) "Branchement toujours"

Tab Branchements | retour au Sommaire | Index | Mode Adressage Relatif | Liens Rapides

Branchement toujours, correspond à JMP avec un adressage Relatif.

Pour avoir un programme structuré, cette instruction est à éviter ! Car c'est l'équivalent du GOTO en Basic !

L'adresse du branchement Relatif est calculée en ajoutant au registre PC un déplacement signé (valeur en complément à deux). **(PC + Déplacement)** à **PC**

```
BRA $xx      (Branch Always)  pour accéder à un espace adressable de 256 octets
LBRA $xxxx   (Long Branch Always) pour accéder à la totalité de l'espace adressable.
```

INS : Branchement Inconditionnel Relatif BRN LBRN (BRanch Never) "Branchement jamais"

Tab Branchements | retour au Sommaire | Index | Mode Adressage Relatif | Liens Rapides

Jamais de Branchement, équivalente à NOP. Cette instruction n'effectue aucune opération.
Utilisé uniquement pour disposer de programmes plus facilement lisibles.
Existe pour maintenir une symétrie dans le jeu d'instructions.

L'adresse du branchement Relatif est calculée en ajoutant au registre PC un déplacement signé (valeur en complément à deux). **(PC + Déplacement)** à **PC**

```
BRN $xx      (Branch Never)  pour accéder à un espace adressable de 256 octets
LBRN $xxxx   (Long Branch Never) pour accéder à la totalité de l'espace adressable_
```

INS : Branchement Inconditionnel Relatif BSR LBSR (Branch to SubRoutine) "....à un sous programme"

Tab Branchements | retour au Sommaire | Index | Mode Adressage Relatif | Liens Rapides

Branchement inconditionnel à un sous-programme, équivalent à JSR avec un adressage Relatif.
L'adresse du branchement Relatif est calculée en ajoutant au registre PC un déplacement signé (valeur en complément à deux). **(PC + Déplacement)** à **PC**

```
BSR $xx      (Branch to Subroutine)  pour accéder à un espace adressable de 256 octets
LBSR $xxxx   (Long Branch to Subroutine pour accéder à la totalité de l'espace adressable
```

Tab Branchements | retour au Sommaire
Index | Mode Adressage Relatif | Liens Rapides

INS : Branchement Conditionnel

Ces branchements sont des ruptures de séquence. Le calcul du branchement ou plutôt du déplacement (dit aussi Offset) sera identique à un branchement inconditionnel mais son exécution sera elle soumise à une condition.

Si la condition est réalisée, le branchement a lieu.
Si la condition n'est pas réalisée le branchement est ignoré, c'est-à-dire que le registre PC (ou PCR) est inchangé.

Les conditions sont relatives à certains bits du registre de condition ou registre d'état, il s'agit du registre CC.

Branchement si	N=1	(bit 3)	**N**égatif	si le résultat ou chargement est négatif
	Z=1	(bit 2)	**Z**éro	si le résultat ou chargement est nul.
	V=1	(bit 1)	o**V**erflow	si il y a dépassement de capacité.
	C=1	(bit 0)	**C**arry	si il y a une retenue.

Exemple 01

```
        LDA   #$FF   ; charge A avec la valeur $FF
TOTO    DECA         ; décrémente A
        BNE   TOTO   ; branch à l'étiquette TOTO si A <> 0
```

Exemple 02

```
        LDA   $B200  ; charge A avec le contenu de la mémoire $B200
        CMPA  #$41   ; compare A avec la valeur $41 en faisant le calcul  A-$41
                     ; (si A=$41 alors A-$41=0) puis positionne le bit Z à 1 si A=$41
        BEQ   TITI   ; test du bit Z et branch à TITI car Z=1
```

Tab Branchements | retour au Sommaire | Index | Mode Adressage Relatif | Liens Rapides

INS : Branchement Conditionnel BEQ BNE BMI BPL BCS BLO BCC BHS BVS BVC

Branchements traduisant une condition simple (dépendant de la valeur d'un bit de CC) Voir tableau ci-dessus.

INS : Branchement Conditionnel BEQ BNE BGT BLE BGE BLT

Branchements opérant sur des nombres signés. Voir tableau ci-dessus.

INS : Branchement Conditionnel BEQ BNE BHI BLS BHS BLO

Branchements qui opèrent sur des nombres non signés Voir tableau ci-dessus.

retour au Sommaire | Index | Liens Rapides

INS : Instructions JSR RTS

JSR (**J**ump to **S**ub**R**outine at effective adress) (≡ au Gosub en Basic)

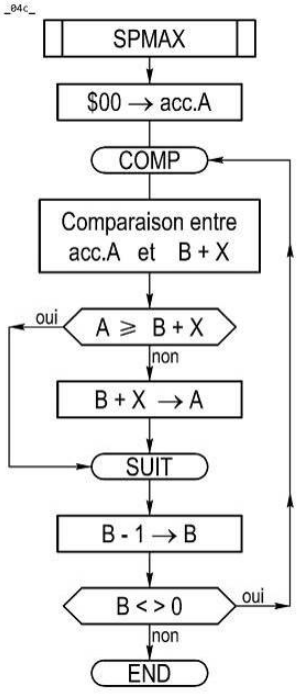
Dès que le µp6809 rencontre **JSR**, il charge le contenu du compteur ordinal (registre PC ou PCR) dans la pile puis se branche à l'adresse spécifié. Le pointeur de pile système est donc décrémenté de 2.

ATTENTION : pour du code relogeable, le JSR est à bannir.

RTS (**R**e**T**urn from **S**ubroutine) (≡ au Return en Basic)

Dès que le µp6809 rencontre **RTS**, il va chercher l'adresse qui se trouve en haut de la pile, l'incrémente et la charge ensuite dans le compteur ordinal. Le pointeur de pile est donc incrémenté de 2.

Exemple : Programme faisant appel un sous-programme permettant de trouver le plus grand élément d'une table de valeurs.



Longueur du bloc à scruter à B
Adresse de début de bloc à X
Le résultat sera sauvegardé à l'adresse $0200

```
        LBB   LONG    ; charge la longueur de la table
        LDX   ADRES   ; charge l'adresse de début
        JSR   SPMAX   ; appel au S-Programme
        STA   $0200   ; sauve le résultat
        SWI           ;
                      ;
SPMAX   LDA   #$00    ; init de la valeur maxi
COMP    CMPA  B,X     ; nouvelle valeur maxi ???
        BGE   SUIT    ; non, on recommence
        LDA   B,X     ; oui, charge ce nouveau maxi
SUIT    DECB          ; bloc terminé ?
                      ; Décrémentation de B
        BNE   COMP    ; non, on continue
        RTS           ;
```

INS : La Pile

Le microprocesseur ne travaille pas toujours avec des emplacements mémoires connus, il se sert très souvent d'une pile. Elle peut être implantée n'importe où dans l'espace mémoire à l'aide des instructions

```
LDS     LDU     TFR X,S     TFR Y,U     EXG D,S     EXG Y,U
```

Ou tout autre mode autorisé de ces 6 instructions, par exemple :

`LDS #$8100` qui définit une pile système à l'adresse $8100

L'expansion de la pile s'effectue par la suite vers les adresses basses en commençant par l'adresse $80FF qui reçoit le nom de "haut de pile".

La **pile S** est gérée ultérieurement par un ensemble d'instructions du type :

`PSHS`	qui "pousse" les données dans la pile S (système)
`PULS`	qui "retire" les données de la pile S
`JSR ou BSR`	qui "poussent" la valeur du PC (compteur programme) dans la pile S pour sauvegarder l'adresse de retour d'un sous-programme.
`RTS`	qui "restitue" l'adresse de retour du programme appelant à la fin d'exécution d'un sous-programme.
`LEAS`	qui "déplace" volontairement le pointeur à n'importe quel autre endroit de la pile, ce déplacement relatif est compté à partir de la position courante du pointeur.
`SWI, SW2, SW3, RTI`	action sur une ligne d'interruption.

La **pile U** est gérée par les instructions du type :

`PSHU`	qui "pousse" les données dans la pile U (utilisateur)
`PULU`	qui "retire" les données de la pile
`LEAU`	qui "déplace" le pointeur relativement à n'importe quel emplacement mémoire.

Il y a aussi `LDA ,U+` `STA ,U+` `LDX ,U++` etc....

La pile est une portion de mémoire vive destinée à la sauvegarde temporaire des informations. Ce sont des emplacements mémoires pour lesquels le CPU possède un générateur d'adresses spécifique.

Ce générateur d'adresses utilise le concept d'empilement et de dépilement sous la règle du dernier entrée, premier sorti. On parle de pile LIFO (Last In, First Out)

La donnée poussée en dernier dans la pile doit être retirée en premier. Cependant, on peut accéder à n'importe quel donnée en déplaçant volontairement les pointeurs avec les instructions du type LEA….

La pile **S** (Système) et la pile **U** (Utilisateur) sont des registres internes au 6809
La présence d'une pile système va faciliter :

- Les retours après interruptions
- Les retours de sous programmes
- Le stockage de données temporaires

La pile commence toujours à l'adresse la plus haute et elle régresse, le pointeur étant décrémenté à chaque empilement.

Le sommet dune pile est en permanence repérée, son pointeur S ou U qui :

- Décrémente avant qu'un octet soit écrit dans la pile.
- Incrémente lorsqu'un octet vient d'être lu dans la pile.

INS : Instructions d'Empilement PSHS PSHU (push ≡ empilement)

Sauvegarde dans la pile. Permet de stocker le contenu d'un ou plusieurs registres internes au sommet de la pile :

Système	pile **S**	`PSHS`
Utilisateur	pile **U**	`PSHU`

Exemple : Sauvegarde des registres X et Y dans la pile Système `PSHS Y,X`
Sauvegarde des registres A, B, et CC dans la pile utilisateur `PSHU A,B,CC`

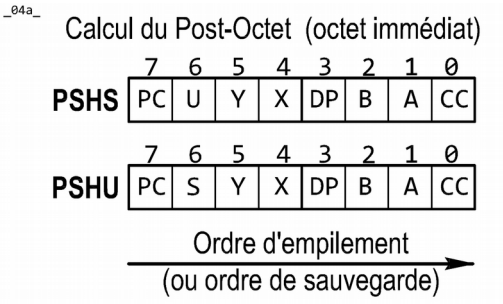
Le code opération est suivi d'un octet dont chaque bit est spécifique d'un registre à sauvegarder.

Le µp6809 connaît donc les registres qu'il doit transférer grâce à un octet supplémentaire appelé le Post-octet.

Selon le positionnement des bits 0 à 7 de cet octet les registres seront affectés.

`PSHU A,B,CC` le post Octet sera = % 0000 0111 = $07 l'OpCode instruction = $36
`PSHU Y,X` le post Octet sera = % 0011 0000 = $30 l'OpCode instruction = $36
`PSHS A,Y,X` le post Octet sera = % 0011 0010 = $32 l'OpCode instruction = $34

_04a_



retour au Sommaire Index Liens Rapides

Exemple d'empilement sur le pointeur de pile S système, `PSHS` (il en serait de même pour le pointeur U)
On commence par le bit de poids fort b7 pour l'ordre d'empilement.

Si b7=1	alors	S – 1 → S	et	PC Low → (S)	Octet de poids faible du registre PC
		S – 1 → S	et	PC Hight → (S)	Octet de poids fort du registre PC
Si b6=1	alors	S – 1 → S	et	U Low → (S)	Octet de poids faible du registre U
		S – 1 → S	et	U Hight → (S)	Octet de poids fort du registre U
Si b5=1	alors	S – 1 → S	et	Y Low → (S)	Octet de poids faible du registre Y
		S – 1 → S	et	Y Hight → (S)	Octet de poids fort du registre Y
Si b4=1	alors	S – 1 → S	et	X Low → (S)	Octet de poids faible du registre X
		S – 1 → S	et	X Hight → (S)	Octet de poids fort du registre X
Si b3=1	alors	S – 1 → S	et	DP → (S)	
Si b2=1	alors	S – 1 → S	et	B → (S)	
Si b1=1	alors	S – 1 → S	et	A → (S)	
Si b0=1	alors	S – 1 → S	et	CC → (S)	

Exemple : (S) = $8100 (X) = $10B6 (Y) = $4F03 Après l'instruction `PSHS X,Y`
S - 4 ($80FC) = $10 correspondant à XH
S - 3 ($80FD) = $B6 correspondant à XL
S - 2 ($80FE) = $4F correspondant à YH
S - 1 ($80FF) = $03 correspondant à YL

Lorsqu'un mélange de plusieurs registres 8 ou 16 bits est mis dans une instruction `PSHS` ou `PULS`, l'ordre d'exécution est celui imposé par le 6809 et non par celui spécifié par le programmeur
Ordre d'empilement pour **PSHS**
`PCL,PCH` `UL,UH` `YL,YH` `XL,XH` `DP` `B` `A` `CC`
Ordre de dépilement pour **PULS**
`CC` `A` `B` `DP` `XH,XL` `YH,YL` `UH,UL` `PCH,PCL`

Ainsi pour pousser les registres A et B dans la pile S on peut écrire :
`PSHS A,B` ou `PSHS B,A` ou `PSHS D` le registre B sera poussé en premier suivi de A

Pour dépiler les données vers les registres X et CC, ont peut écrire indifféremment
`PULS X,CC` ou `PULS CC,X` l'ordre imposé sera CC puis XH puis XL

retour au Sommaire Index Liens Rapides

INS : Instructions de Dépilement PULS PULU (pull ≡ dépilement)

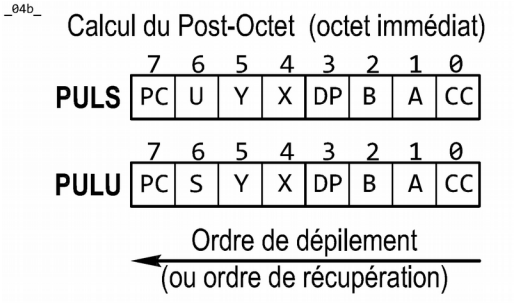
Récupération depuis la pile. Permet de charger un ou plusieurs registres internes à l'aide d'octets stocké dans la pile
- Système pile **S** `PULS`
- Utilisateur pile **U** `PULU`

Restaurer les registres X et Y depuis la pile Système `PULS Y,X`
Restaurer les registres A, B, et CC depuis la pile utilisateur `PULU A,B,CC`

Le code opération est suivi d'un octet dont chaque bit est spécifique d'un registre à sauvegarder.
Le µp6809 connaît donc les registres qu'il doit transférer depuis la pile grâce à un octet supplémentaire appelé le Post-octet.

Selon le positionnement des bits 0 à 7 de cet octet les registres seront affectés.

```
PULU A,B,CC     le post Octet sera = % 0000 0111  =  $07    l'OpCode instruction = $37
PULS Y,X        le post Octet sera = % 0011 0000  =  $30    l'OpCode instruction = $35
PULU A,Y,X      le post Octet sera = % 0011 0010  =  $32    l'OpCode instruction = $37
```



Exemple de dépilement sur le pointeur de pile S système, `PULS` (il en serait de même pour le pointeur U)

On commence par le bit de poids faible b0 pour l'ordre de dépilement.

Si b0=1	alors	S + 1 → S	et	CC → (S)	
Si b1=1	alors	S + 1 → S	et	A → (S)	
Si b2=1	alors	S + 1 → S	et	B → (S)	
Si b3=1	alors	S + 1 → S	et	DP → (S)	
Si b4=1	alors	S + 1 → S	et	X Hight → (S)	Octet de poids fort du registre X
		S + 1 → S	et	X Low → (S)	Octet de poids faible du registre X
Si b5=1	alors	S + 1 → S	et	Y Hight → (S)	Octet de poids fort du registre Y
		S + 1 → S	et	Y Low → (S)	Octet de poids faible du registre Y
Si b6=1	alors	S + 1 → S	et	U Hight → (S)	Octet de poids fort du registre U
		S + 1 → S	et	U Low → (S)	Octet de poids faible du registre U
Si b7=1	alors	S + 1 → S	et	PC Hight → (S)	Octet de poids fort du registre PC
		S + 1 → S	et	PC Low → (S)	Octet de poids faible du registre PC

Pour gagner du temps et de la mémoire

Le code `PULS A,B` peut être

`RTS` remplacer par `PULS A,B,PCR`

retour au Sommaire
Index Liens Rapides

INS : Instruction NOP

Elle ne fait rien, ou presque, elle se contente juste d'incrémenter le Compteur Ordinal.

Elle peut servir en autre à :

- Remplacer une instruction non utile, ce qui permet de ne pas avoir à réécrire tout le programme lors d'un assemblage à la main
- Elle sert pendant la mise au point, en remplaçant une instruction par NOP, pour éviter de recalculer tous les branchements.
- La mise au point d'un programme partie par partie en remplaçant par exemple certains sous-programmes par de NOP
- Provoquer un délai de durée fixe dans l'exécution d'un programme.

Il est évident que le programme définitif devra être débarrassé de ces instructions inutiles afin d'en diminuer le temps d'exécution.

retour au Sommaire Index Liens Rapides

INS : Instruction RTI

Voir le chapitre du Traitement des Interruptions : instruction RTI

retour au Sommaire Index Liens Rapides

INS : Instruction SYNC

Voir le chapitre du Traitement des Interruptions : instruction SYNC

retour au Sommaire Index Liens Rapides

INS : Instruction CWAI

Voir le chapitre du Traitement des Interruptions : instruction CWAI

FEI : Qu'est ce qu'une interruption ?

retour au Sommaire | Vecteurs d'Interruption

C'est un moyen MATERIEL, un signal sur une broche du µp6809 qui change d'état.
Ou une procédure LOGICIEL, une instruction placée dans un programme en cours d'exécution.

Index | Liens Rapides | Sommaire Principal

L'interruption LOGICIEL ou MATERIEL permet :

- d'interrompre un programme en cours
- de traiter prioritairement un programme par rapport à un autre.

Le microprocesseur scrute à chaque cycle, c'est-à-dire toutes les microsecondes, une éventuelle interruption, ce qui permet une réponse instantanée. De ce fait le µp6809 n'a pas besoin de scruter les périphériques, il se contente d'attendre qu'on l'avertisse.

Le µp6809 doit être capable de répondre rapidement aux sollicitations de ces périphériques.
Ces demandes externes au µp6809 sont vues comme des demandes d'interruption, le µp6809 possède pour cela

- **3 Broches d'entrées d'interruption Matérielle NMI, IRQ et FIRQ et une séquence d'initialisation RESET**
- **3 Instructions d'interruption Logicielle SWI, SWI2, SWI3**
- **2 Instructions d'Attente d'Interruptions SYNC CWAI**

Ces méthodes de traitement d'échange (Matérielle, Logicielle ou d'attente) présentent des similitudes :

1. La prise en compte d'une interruption ne se fait jamais pendant l'exécution d'une instruction. Le processeur termine avant tout l'instruction qu'il était entrain d'exécuter.
2. Dans tous les cas, le programme principal est interrompu.
3. Le processeur interdit ensuite les autres interruptions moins prioritaires.
4. Le µp6809 doit sauvegarder tout ou partie du contexte dans une pile, il empile les registres PC et CC et éventuellement d'autres registres
5. Le µp6809 exécute une séquence de traitement d'interruption, en se branchant à l'adresse de la routine d'interruption.
6. L'interruption étant terminée, le processeur dépile les registres empilés et poursuit le programme qui à été interrompu.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Système d'interruption du 6809

Dans les systèmes à base de microprocesseur, on désire souvent intervenir sporadiquement dans le système, même lorsqu'il est en train d'exécuter une tâche quelconque (exemples : arrêt d'urgence pour prévenir d'un danger, arrêt d'une imprimante par manque de papier, arrêt d'un traitement pour traiter un autre programme plus prioritaire).

Un système permettant d'interrompre le déroulement d'un programme n'est réellement utile que si le contexte d'exécution peut être sauvegardé.

Le µp6809 comporte un système de d'interruptions très complet permettant de solutionner la majorité des applications dites "à temps réel".

Par rapport au 6800, le µp6809 possède en plus : une interruption matérielle rapide FIRQ, 2 interruptions logicielles SW2 et SW3, 2 instructions originales CWAI et SYNC.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Différentes catégories d'interruptions

Une interruption est une instruction (logiciel) ou une impulsion (matériel) provoquant un arrêt ordonné du programme en cours, avec sauvegarde partielle ou totale du contexte d'exécution dans la pile système et un branchement du µp6809 vers une séquence spéciale appeler sous-programme d'interruption.

Le sous-programme d'interruption restitue le contexte d'exécution programme principal (à partir de la pile système) dès la rencontre de l'instruction RTI. L'instruction RTI ressemble un peu à RTS, à la différence près qu'il examine de l'Etat du E (bit7 du registre CC) pour connaître l'étendue du dépilage.

Vecteurs d'Interruption

Pour faciliter la description du système d'interruption du µp6809 nous distinguons :

- Les arrêts manuels RESET, HALT
- Les interruptions matérielles IRQ, FIRQ, NMI
- Les interruptions logicielles SWI, SWI2, SWI3
- Les instructions pour attendre une interruption CWAI, SYNC

Une interruption matérielle est provoquée par l'apparition d'une impulsion ou d'un front actif sur l'une des broches d'interruption du µp6809, broches IRQ, FIRQ, NMI.

Mise à part NMI qui ne peut être "masqué", les deux autres IRQ et FIRQ sont masquables par des instructions logiques appropriés.

Au commencement d'un traitement le programmeur spécifie explicitement s'il autorise la prise en compte par le µp6809 des interruptions IRQ ou FIRQ. Si elles sont autorisées elles pourront par la suite apparaître n'importe quel endroit du programme utilisateur. Le µp6809 ne tient compte de leur présence qu'après l'exécution complète de l'instruction en cours.

Avant de passer le contrôle à la séquence d'exécution, le µp6809 effectue une sauvegarde :

- Totale pour IRQ et NMI
- Partielle pour FIRQ

Pour les interruptions logicielles SWI, SWI2, SWI3, elles doivent être insérées dans le programme utilisateur sous forme d'instructions régulières.

Le fonctionnement d'une interruption logicielle s'identifie à celui d'une interruption matérielle. La différence réside dans le fait qu'une interruption logicielle doit être programmée à l'avance, qu'elle est placée dans des endroits bien déterminés, d'où disparition de l'aspect aléatoire rencontré dans une interruption matérielle.

Le fonctionnement des instructions CWAI et SYNC est assez particulier. Elles sont intervenir simultanément les deux aspects matériels et logiciels.

Vecteurs d'Interruption

Le µp6809 comporte aussi deux interruptions matérielles d'un genre particulier :

- RESET qui effectue une initialisation générale du système.
- HALT qui suspend toute activité du µp6809 sans perte de contexte et sans traitement de séquence d'exception.

retour au Sommaire Index
Liens Rapides Vecteurs d'Interruption

FEI : Initialisation générale par RESET

Un niveau bas appliqué sur cette broche provoque une initialisation complète du microprocesseur, les différentes opérations suivantes se déroulent dans l'ordre :

1) Remise à zéro du registre de page directe DP
2) Masquage des interruptions matérielles IRQ et FIRQ
3) Désactivation de NMI
4) Le µp6809 ira chercher l'adresse du programme RESET dans le contenue des mémoires $FFFE et $FFFF.

Le programme RESET est contenu dans une mémoire ROM non volatile. Il sert à démarrer l'ensemble système. D'une façon générale les tâches accomplies dans ce programme sont :

- Définition de l'emplacement de la pile système. Cette instruction portant sur la pile S dégage l'interdiction sur NMI qui rentre en activité après le chargement du pointeur S.
- Chargement des adresses d'exécution des sous-programmes d'exception dans les mémoires spécialement réservées à cet usage, dont la plage va de $FFF0 à $FFFF.
- Initialisation et configuration du mode de fonctionnement des interfaces disponibles sur le système.
- Envoie d'une information l'utilisateur sur la nature du système sous tension.
- Exécution des tâches particulières propre à chaque système, par exemple connexion automatique vers un programme d'application.

retour au Sommaire Index
Liens Rapides Vecteurs d'Interruption

FEI : Arrêt temporaire par HALT

Sur un niveau bas de cette broche, le µp6809 se met au repos. Cette suspension d'activité n'efface pas les contenus des registres.

Le rétablissement ultérieur d'un niveau Haut sur la broche HALT autorise le µp6809 à continuer dans sa tâche sans perdre d'informations.

Cette broche est configurée par un interrupteur ou par l'intermédiaire d'un système de logique électronique.

Cette interruption peut donc être employée à tout instant sans aucun risque de perte d'information à condition que le système soit sous le contrôle de l'unique µp6809 interrompu.

D'autres détails intéressants à connaître sont :

- Le µp6809 n'examine une demande HALT qu'à la fin de l'exécution d'une instruction.
- A l'arrêt le µp6809 ignore les demandes IRQ et FIRQ. Les entrées RESET et NMI sont en revanche mémorisées pour une réponse ultérieure.
- Un sous-programme d'interruption quelconque peut être arrêté par HALT.
- Par sa définition, HALT n'a pas de traitement d'exception, donc ignore la pile.

FEI : Remarques sur la programmation des interruptions

FEI : Remarque 01

Les interruptions ne sont pas très faciles à tester car ils impliquent un minimum de matériel pour produire des niveaux Bas, des impulsions à largeur réglable (SYNC), etc...

D'autre part étant donné le caractère aléatoire des interruptions matérielles, dans certains cas, se posent des problèmes de localisation.

FEI : Remarque 02

Pour les interfaces, les interruptions matérielles ne conduisent pas forcément à une vitesse de transfert maximum étant donné le nombre de registre empilés et dépilés à chaque activation de IRQ.

Certes, l'introduction de FIRQ apporte une amélioration considérable. Le nombre minimal de registres à empiler est laissé à l'appréciation du programmeur. Il peut mettre PSHS au début du traitement d'exception et PULS avant l'exécution de RTI.

Voici quelques indications sur le nombre de cycles machines nécessaires à la prise en compte de l'interruption.

21 cycles	Réponse à NMI à IRQ	6 cycles	Exécution de RTI avec bit E=0
12 cycles	Réponse à FIRQ	15 cycles	Exécution de RTI avec bit E=1
19 cycles	Réponse à SWI	20 cycles	Réponse à SWI2, SWI3
20 cycles	Réponse à CWAI		
9 cycles	Réponse à n'importe quelle interruption après CWAI		
1 cycles	Exécution de SYNC si l'interruption est masquée		
2 cycles	Exécution de SYNC		

FEI : Remarque 03

Lorsque plusieurs sources d'interruptions sont câblées à une même entrée IRQ ou FIRQ, l'identification du logicielle (polling) de la source émettrice de la demande peut présenter quelques inconvénients sur la vitesse de réponse.

D'autre part les adresses des registres des interfaces sont rarement contiguës dans l'espace mémoire du processeur si bien qu'on éprouve parfois quelques difficultés pour employer les modes d'indexés. La vectorisation des interruptions à l'aide des circuits spécialisés peut apporter une solution élégante.

FEI : Remarque 04

Dans les séquences d'exception, le µp6809 permet de manipuler des données de la pile système facilement à l'aide du mode indexé **nn,S**. les offsets sont les suivantes :

```
PCL 11,S    PCH 10,S    UL 9,S    UH 8,S    YL 7,S    YH 6,S
XL 5,S      XH 4,S      DP 3,S    B 2,S     A 1,S     CC 0,S
```

Position du pointeur S à l'entrée du sous-programme d'interruption.

Si les instructions PSHS sont nécessaires dans le sous-programme d'interruption, apporter les corrections nécessaires à ces valeurs d'offset.

Par exemple, si l'on veut interdire les interruptions dans le programme principal après le traitement d'une séquence d'exception, on peut écrire :

```
LDA     0,S             ;
ORA     #%01010000      ; interdire IRQ et FIRQ
STA     0,S             ;
```

Après dépilement, les masques (bit I et bit F du registre de contrôle) se retrouvent avec la valeur 1, ce qui correspond résultat souhaité. Il est également possible de modifier l'adresse de retour.

FEI : Remarque 05

Pour le µp6809, les interruptions sont inhibées quand les bits **F** et **I** du registre d'état sont mis à 1.

Pour les interfaces PIA 6821 et ACIA 6850, c'est exactement le contraire, les bits correspondants des registres de contrôle doivent être mis à 0.

Au RESET général, les modes par interruption du PIA 6821 sont inhibées. Pour l'ACIA 6850, l'initialisation générale qui consiste à charger %00000011 (Master reset) dans le registre le contrôle de l'ACIA 6850 n'affecte pas le bit de contrôle d'interruption CR7.

FEI : Remplissage d'un buffer de commande par une interruption d'une ligne série

Hypothèse d'un terminal écran-clavier connecté au système à travers une ligne série contenant un **ACIA 6850**.

La sortie IRQ du **6850** est connectée à l'entrée IRQ du processeur et le **6850** est configurée en mode réception par interruption (bit CR7 du registre de contrôle égal à 1).

Chaque fois qu'un caractère reçu, il est renvoyé à l'écran par la même interface, selon le mode habituel de test de bit SR1 du registre d'état. Rappelons que se bit passe à 1 si le registre de transmission de le **6850** est disponible.

Le programme débute à l'adresse $8000. Les caractères reçus à partir du clavier seront rangés dans un buffer placé à l'adresse $8100. La fin du programme se termine par la détection du code $0D correspondant à un retour chariot.

Le premier octet du buffer contient l'état de remplissage :

- 0 si le buffer est vide
- 1 si le buffer est plein

Le deuxième octet contient le nombre total de caractères reçus sans compter le caractère "retour chariot".
A partir du troisième octet, se rangent les données proprement dites.

Exemple : si on rentre au clavier MERGE puis "retour chariot" :

```
          ($8100) = $01   indicateur "buffer non vide"
          ($8100) = $05   Cinq caractères reçus
          ($8100) = $4D   'M
          ($8100) = $45   'E
          ($8100) = $52   'R
          ($8100) = $47   'G
          ($8100) = $45   'E
          ($8100) = $0D   'Retour chariot
                    ;*******************************************************
                    ;*   Remplissage d'un buffer de commande par
                    ;*   interruption sur ligne série
                    ;*******************************************************
8000                        ORG    $8000       ;
                    ;------adresses des registres gérant le terminal clavier
          EC80      RCRTER EQU    $EC80       ;  reg. contrôle ACIA
          EC80      RSRTER EQU    $EC80       ;  reg. Etat ACIA
          EC81      RRXTER EQU    $EC81       ;  reg. Réception Clavier
          EC81      RTXTER EQU    $EC81       ;  reg. Transmission écran
                    ;------partie exécutée à l'initialisation générale
8000 10CE 8200              LDS    #$8200      ;  adrs pile système
8004 86   03                LDA    #%00000011  ; reset ACIA
8006 B7   EC80              STA    RCRTER      ;
8009 86   91                LDA    #%10010001  ; interruption réception + 8 bits
                                                ;  + 2 stop + clock 1/16
800B B7   EC80              STA    RCRTER      ;
800E 8E   8080              LDX    #$8080      ;  adrs S/P interruption
8011 BF   FFF8              STX    $FFF8       ;  adrs vecteur IRQ  FFF8 et FFF9
8014 8E   8100              LDX    #$8010      ;  adrs buffer
8017 8D   01   801A         BSR    BUFFER      ;  remplissage buffer
8019 3F                     SWI                ;
                    ;*******************************************************
                    ; S/P remplissage Buffer
                    ; Caractère final "RC"    nom d'appel : BUFFER
                    ; Entrée :     X: adrs point départ buffer
                    ; Sortie :     Aucun registre affecté
                    ; (0,X)=0  buffer vide
                    ; (0,X)=1  buffer plein
                    ; (1,X):   nombre de caractères du buffer sans "RC"
                    ; à partir de (2,X): Contenu du buffer
                    ; Encombrement pile système :  2+5+12+1 = 20 octets
                    ;*******************************************************
801A 34   17        BUFFER PSHS   X,B,A,CC    ;
801C 6F   84               CLR    0,X         ;  buffer vide initialement
801E 6F   01               CLR    1,X         ;  Nb caractères
8020 C6   02               LDB    #2          ;  offset premier caractère
8022 1C   EF               ANDCC  #%11101111  ;  interruption CPU autorisée
8024 6D   84        CARSVT TST    0,X         ;  fin procédure remplissage ???
8026 27   FC   8024        BEQ    CARSVT      ;  sinon caractère suivant
8028 35   97               PULS   PC,X,B,A,CC ;  fin S/P BUFFER
                    ;*******************************************************
                    ; Début S/P d'interruption
                    ; l'adrs $8080 connue doit être chargée au préalable
```

```
                    ; dans ($FFF8),($FFF9) au début de l'appel du S/P BUFFER
                    ; Registres échangés: B (voir explication à la fin du prog
                    ;*********************************************************
8080                        ORG     $8080       ;
8080 B6   EC81              LDA     RRXTER      ;  lire reg. réception
                                                ;  mise à 0 bit interruption SR7
8083 A7   85                STA     B,X         ;  rangement donnée
8085 81   0D                CMPA    #$0D        ;  caractère "RC" return ???
8087 26   07   8090         BNE     VISU        ;  sinon visualiser
8089 6C   84                INC     0,X         ;  mettre indicateur Buffer
                                                ;  plein. Fin remplissage
808B C0   02                SUBB    #2          ;  obtenir Nb caractère
808D E7   01                STB     1,X         ;  stockage
808F 3B                     RTI                 ;  fin S/P interruption
                    ;*********************************************************
8090 34   02        VISU   PSHS    A           ;
8092 B6   EC80              LDA     RSRTER      ;  reg. Etat terminal
8095 85   02                BITA    #%00000010  ;  reg. transmission vide ???
8097 27   F9   8092         BEQ     VISU+2      ;
8099 35   02                PULS    A           ;
809B B7   EC81              STA     RTXTER      ;  transmettre sur écran
809E 6C   62                INC     2,S         ;  avancer valeur de B dans pile
                                                ;  système pour prochaine entrée dans
                                                ;  S/P interrup. (INCB serait une erreur)
80A0 3B                     RTI                 ;  fin S/P interruption
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Quelques explications sur le programme ci-dessus

L'**ACIA 6850** est configuré avec CR7=1, donc la réception fonctionne en interruption. Chaque fois que le registre de réception est plein, le bit SR0 est mis à 1, de même que le bit SR7. La ligne IRQ subi une transition négative et le µp6809 reconnaît une interruption.

Le traitement d'exception débute à l'adresse $8080 par une lecture simple du registre de réception.

Cette lecture provoque une mise à 0 des bits SR0 et SR7 et rétablit un niveau Haut sur la ligne IRQ.

On remarquera qu'ici, le programme ne teste plus l'état du bit de réception SR0.

Le sous-programme BUFFER fonctionne par interruption de type IRQ.

A l'appel de ce sous programme, il est nécessaire de sauvegarder le contenu de $FFF8 et $FFF9 dans un endroit quelconque, puis charger un autre vecteur IRQ avant l'appel de BUFFER. En fin d'exécution du sous-programme, on restituera l'ancienne valeur du vecteur IRQ avant de continuer. Ici, nous avons enlevé cette opération de sauvegarde. Le programmeur pourra la rétablir en écrivant par exemple :

```
LDX     $FFF8       ; sauvegarde vecteur IRQ courant
PSHS    X           ;              dans la pile système
LDX     #$8080      ; nouveau vecteur IRQ pour BUFFER
STX     $FFF8       ; mise en place du nouveau vecteur
LDX     #$8100      ; index du buffer de caractères
BSR     BUFFER      ; appel du sous-programme BUFFER
PULS    X           ; rétablir ancien vecteur IRQ
STX     $FFF8       ;
SWI                 ;
```

La mémoire (0,X) dans le sous-programme BUFFER sert de paramètres de communication entre BUFFER et son sous-programme d'interruption.
La boucle CARSVT se referme jusqu'à ce que (0,X) soit mis à 1 par la séquence d'exception.
(0,X) est mis à 1 quand le caractère reçu est un retour chariot $0D.

Il faut remarquer que l'interruption peut se produire différemment après TST 0,X ou après BEQ CARSVT. Le déclenchement reste aléatoire.

Une autre difficulté est rencontrée pour l'incrémentation de l'offset contenu dans B.
A la 1ière entrée dans la boucle CARSVT, (B)=2.

Après interruption, le contenu de B est poussé dans la pile.

Si l'on incrémente B simplement par INCB, après dépilement par RTI, l'ancienne valeur de B, en l'occurrence 2 sera rétabli dans B et l'offset B restera en réalité inchangé. Pour faire évoluer ce paramètre, il est nécessaire d'actualiser sa valeur par INC 2,S dans la pile S avant l'opération de dépilement RTI.

Ce type d'erreur est difficile à prévoir et à tester. C'est le désavantage des interruptions.

Donc, ne pas employer les registres pour transférer les paramètres lorsqu'il s'agit de sous programme d'interruption, car les registres sont empilés et dépilés automatiquement. Ce qui n'est pas le cas pour les appels BSR, LBSR, JSR, RTS qui n'invoquent que le compteur programme PC.

Si l'on touche aux registres dans la pile système S dans les séquences exception, il y a effectivement transfert de paramètres entre les deux programmes. Or, l'interruption matérielle peut se produire de façon aléatoire, ce qui rend la conception des sous-programmes d'interruption difficile.

Pour s'affranchir de ce problème, il faut s'abstenir d'employer les mêmes registres dans les deux programmes.

Ici, B est employé par la séquence d'exception; ce registre n'est pas employé par la boucle d'attente dans le programme principal. Cette caractéristique est notifiée explicitement dans la partie commentaire.

Dans des cas spécifiques ou tous les registres doivent être mobilisés par les deux programme, le dernier recours consiste à autoriser l'interruption dans des endroits bien précis du programme principal en manipulant adroitement les instructions de masquage ANDCC et ORCC.

FEI : Prog. D'une touche d'arrêt temporaire avec visu des registres 6809 par interruption IRQ.

Etude simultanée d'une interruption par SWI

La mise au point des programmes nécessite des outils logiciels permettant de visualiser l'exécution d'un programme étape par étape.

Cette application ci-dessous, montre comment confectionner un programme traitant une interruption logiciel SWI et comment concevoir un arrêt momentané d'un programme avec la possibilité d'exécution ultérieure sans perte de contexte.

Dans les deux cas, l'ensemble des registres du µp6809 est visualisé sur terminal écran-clavier connecté au système étudié à travers une liaison série gérée par un ACIA 6850.

Nous supposons que le système utilisateur possède un moniteur résident dont l'adresse MONIT est connue par le programmeur.

A la rencontre d'une instruction SWI, le µp6809 entame une séquence d'interruption en sauvegardant l'ensemble des registres dans la pile système S.

Le sous programme de visualisation des registres appelé VISURG doit chercher les valeurs instantanées du programme utilisateur dans la pile système par le mode de indexé nn,S.

Les valeurs binaires sont ensuite converties en caractères ASCII visualisables.

Ces caractères sont rangés ensuite dans une zone mémoire appelée mémoire bloc-notes dont l'adresse est repérée par l'étiquette assembleur BLNOTE.

Au cours du remplissage, le buffer ASCII subit également une opération de formatage qui consiste à glisser les "espaces" entre les nombres, les sauts et retour de ligne pour améliorer la lisibilité.

Ce buffer se termine par un caractère de contrôle ETX de code ASCII $04.

Un autre sous-programme appelé VISU transmet l'ensemble du buffer vers l'écran de visualisation à la fin du module VISURG. Cette organisation en modes indépendants permet un emploi ultérieur de VISURG dans le mode ARRET TEMPORAIRE.

Après la phase de visualisation des registres, le contrôle du µp6809 est transféré moniteur par un branchement inconditionnel `JMP MONIT`.

L'ARRET TEMPORAIRE de l'exécution du programme utilisateur est réalisé par une pression sur la touche "H" du clavier. Les registres instantanés du µp6809 seront visualisés sur l'écran.

Si l'opération désire reprendre l'exécution à l'endroit où s'est produite l'interruption, il suffit d'appuyer sur la touche "C" du clavier.

Au besoin, l'utilisateur peut affecter d'autres codes de contrôle pour réaliser ces deux fonctions.

Si l'opérateur appuie sur des touches autre que "H" et "C", le programme ignore ces touches, car les sous-programmes d'exception savent distinguer les touches et imposent l'ordre "H", "C", "H", "C", etc...

Les touches "H" et "C" interrompent le µp6809 par la ligne IRQ.

Les sous-programmes d'exception correspondants sont répartis volontairement ici en deux niveaux, montrant comment on peut interrompre une séquence d'exception IRQ par une autre interruption IRQ et organiser un retour donné sans perte de contexte.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

---

Dans cette application, le programme principal est parfaitement factice. Il sert simplement à faire varier continuellement les registres pour contrôler la bonne marche des sous-programmes d'interruption, c'est une boucle sans fin.

Pour sortir de cette boucle, il faut effectuer un RESET général ou employer une interruption NMI à condition que cette dernière existe sur le système étudié sous forme de bouton "ABORT" ou tout autre.

Pour tester l'interruption logicielle SWI, il faut enlever la boucle sans fin en effectuant une retouche mineure du code objet. Cette retouche est indiquée dans les commentaires.

Ce programme apparemment complexe est modulaire. Il est conseillé d'isoler en premier lieu des différents sous-programmes :

- VISURG Visualisation des registres qui appelle BINHEX et VISU.
- BINHEX Conversion d'un octet binaire en deux caractères ASCII
- VISU Transmission du buffer ASCII sur écran
- SPINT1 Sous-programme d'interruption par IRQ du 1er niveau
- SPINT2 Sous-programme d'interruption par IRQ du 2ième niveau
- SPSWI Sous-programme d'interruption par SWI

L'usage des couleurs peut améliorer la lisibilité.

Isolé également la boucle sans fin simulant le programme principal qu'on veut interrompre.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
                        ;*********************************************************
                        ;   Arrêt Temporaire avec Visualisation des registres
                        ;   du 6809 à l'écran. Interruption logicielle par SWI
                        ;   Interruption Matérielle par IRQ      ouvrage 06 pVII.18
                        ;*********************************************************
8000                            ORG     $8000       ;
                        ;------Adrs Registre gérant le terminal écran-clavier
          EC80          RSCR1   EQU     $EC80       ; Reg. contrôle ACIA
          EC80          RSET1   EQU     $EC80       ; Reg. Etat ACIA
          EC81          RSRD1   EQU     $EC81       ; Reg. Réception Clavier
          EC81          RSTD1   EQU     $EC81       ; Reg. Transmission Ecran
          F000          MONIT   EQU     $F000       ; Adrs Moniteur
          8040          SPSWI   EQU     $8040       ; Adrs S/P SWI
          8080          SPINT1  EQU     $8080       ; Adrs 1er niveau d'interrup par IRQ
          8050          SPINT2  EQU     $8050       ; Adrs 2èm niveau d'interrup par IRQ
          8300          BLNOTE  EQU     $8300       ; Adrs mémoire bloc-notes
                        ;------Partie exécutée à l'initialisation générale
8000 10CE 8400                  LDS     #$8400      ; Adrs de la pile système
8004 86   03                    LDA     #%00000011  ; Master Rest de l'ACIA
8006 B7   EC80                  STA     RSCR1       ;
8009 86   91                    LDA     #%10010001  ; interruption réception + 8 bits
                                                    ; + 2 stops + clock 1/16
800B B7   EC80                  STA     RSCR1       ;
800E 8E   8080                  LDX     #SPINT1     ; adrs S/P IRQ
8011 BF   FFF8                  STX     $FFF8       ;
8014 8E   8040                  LDX     #SPSWI      ; adrs S/P SWI
8017 BF   FFFA                  STX     $FFFA       ;
```

retour au Sommaire | Index | Liens Rapides

```
                        ;*********************************************************
                        ;   la séquence suivante s'exécute indéfiniment.
                        ;   Appuyer de temps à autre sur la touche H pour
                        ;   arrêter le déroulement du programme et de visualiser
                        ;   les registres du 6809 à l'écran
                        ;   Pour continuer, appuyer sur la touche C
                        ;*********************************************************
```

```
801A 4F                    CLRA              ;
801B 5F                    CLRB              ;
801C 8E   0000             LDX    #0         ;
801F 1F   12               TFR    X,Y        ;
8021 1F   13               TFR    X,U        ;
8023 1C   EF               ANDCC  #%11101111 ; autoriser IRQ
8025 4C             TSTINT INCA              ;
8026 5C                    INCB              ;
8027 30   01               LEAX   1,X        ;
8029 31   21               LEAY   1,Y        ;
802B 33   41               LEAU   1,U        ;
802D 20   F6   8025        BRA    TSTINT     ; enlever pour tester SWI
802F 3F                    SWI               ;
                    ;*************************************************
                    ;   S/P interruption par SWI. Pour tester cette séquence,
                    ;   remplacer l'op-code de BRA  TSTINT par NOP NOP $12 $12
                    ;*************************************************
8040                       ORG    SPSWI      ; implantation S/P SWI
8040 17   00BD 8100        LBSR   VISURG     ; VISUaliser les ReGistres
8043 1C   AF               ANDCC  #%10101111 ; Autoriser IRQ et FIRQ
8045 32   6C               LEAS   12,S       ; rétablir position pointeur
8047 7E   F000             JMP    MONIT      ; retour au moniteur
                    ;------S/P interruption IRQ 1er niveau
8080                       ORG    SPINT1     ; adrs d'implantation
8080 B6   EC81             LDA    RSRD1      ; lire caractère reçu
8083 81   48               CMPA   #'H        ; mode "arrêt temporaire"
8085 27   01   8088        BEQ    ARRET      ; si oui vers ARRET
8087 3B                    RTI               ; sinon retour au programme départ
8088 8D   76   8100 ARRET  BSR    VISURG     ; visualiser registres
                    ;------définition d'un 2ième niveau d'interruption
808A 8E   8050             LDX    #SPINT2    ; adrs 2ième niveau IRQ
808D BF   FFF8             STX    $FFF8      ; Charger vecteur IRQ
8090 3C   EF               CWAI   #%11101111 ; enlever le masquage sur IRQ et
                                             ; attendre arrivée autre caractère
                    ;------A cet endroit, 6809 attend une interruption type IRQ
                    ;      Le S/P d'interruption du 2ième niveau vérifie s'il
                    ;      s'agissait bien d'un caractère C provenant de
                    ;      l'ACIA1, puis retourne le contrôle au 1er niveau
                    ;------(voir S/P interruption 2ième niveau)
8092 8E   8080             LDX    #SPINT1    ; rétablir le vecteur IRQ niveau 1
8095 BF   FFF8             STX    $FFF8      ;
8098 3B                    RTI               ; retour au prog principal
                    ;------S/P interruption IRQ 2ième niveau
8050                       ORG    SPINT2     ; adrs d'implantation
8050 7D   EC80             TST    RSET1      ; examiner bit SR7 de ACIA 1
8053 2A   0E   8063        BPL    ATTEN      ; si non ACIA1 attendre
8055 B6   EC81             LDA    RSRD1      ; lecture registre réception
8058 81   43               CMPA   #'C        ; mode "Continuer" ???
805A 26   07   8063        BNE    ATTEN      ; sinon attendre autre caractère
805C 8E   838A             LDX    #BLNOTE+128+10  ; index sur chaîne EXECUTION
805F 17   01AA 820C        LBSR   VISU       ; visualiser
8062 3B                    RTI               ; caractère C identifié
                                             ; continuer S/P 1er niveau
                    ;------dans l'un des 2 cas suivans : IRQ non issue de
                    ;      l'ACIA 1 ou caractère différent de C, revenir
                    ;------au niveau 1 mais sur l'instruction CWAI
8063 AE   6A        ATTEN  LDX    10,S       ; adrs retour dans pile S
8065 30   1E               LEAX   -2,X       ; 2 octets en avant
8067 AF   6A               STX    10,S       ; mettre dans la pile avant
                                             ; dépilement par RTI
8069 3B                    RTI               ;
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
                    ;*************************************************
                    ;    S/P Visualisation des registres du 6809
                    ;    nom d'appel : VISURG
                    ;    Entrée: sans paramètre
                    ;    Sortie: aucun registre affecté
                    ;    sous programme appelés :  BINHEX,  VISU
                    ;    encombrement pile système : 9+7 = 16 octets
                    ;*************************************************
8100                       ORG    $8100      ;
8100 34   37        VISURG PSHS   Y,X,B,A,CC  ;
                    ;------A cause de cette opération de sauvegarde le pointeur S
                    ;      se déplace de 9 octets vers le bas. Ajouter +9 pour
                    ;      retrouver les offsets des registres sauvegardés à
                    ;------l'interruption
```

```
8102 8E   8300            LDX    #BLNOTE     ; Adrs mémoire bloc-notes
8105 CC   5043            LDD    #$5043      ; car. PC $50="P" et $43="C"
8108 ED   81              STD    ,X++        ;
810A 86   3A              LDA    #$3A        ; car. $3A=":"
810C A7   80              STA    ,X+         ;
810E A6   E8 13           LDA    19,S        ; mot poids fort PC
8111 17   00DF 81F3       LBSR   BINHEX      ; conversion
8114 ED   81              STD    ,X++        ;
8116 A6   E8 14           LDA    20,S        ; mot poids faible PC
8119 17   00D7 81F3       LBSR   BINHEX      ;
811C ED   81              STD    ,X++        ;
811E CC   2020            LDD    #$2020      ; deux espaces
8121 ED   81              STD    ,X++        ;
8123 CC   533A            LDD    #$533A      ; S:
8126 ED   81              STD    ,X++        ;
8128 1F   40              TFR    S,D         ; position courante pointeur S
812A C3   0015            ADDD   #21         ; 9 pour compenser VISURG
                                             ; + 12 pour compenser interruption
812D 1F   02              TFR    D,Y         ; sauvegarde provisoire
812F 17   00C1 81F3       LBSR   BINHEX      ; conversion poids fort S
8132 ED   81              STD    ,X++        ;
8134 1F   20              TFR    Y,D         ; obtenir poids faible S
8136 1E   89              EXG    A,B         ;
8138 17   00B8 81F3       LBSR   BINHEX      ;
813B ED   81              STD    ,X++        ;
813D CC   2020            LDD    #$2020      ; deux espaces
8140 ED   81              STD    ,X++        ;
8142 CC   553A            LDD    #$553A      ; U:
8145 ED   81              STD    ,X++        ;
8147 A6   E8 11           LDA    17,S        ; mot poids fort U
814A 17   00A6 81F3       LBSR   BINHEX      ;
814D ED   81              STD    ,X++        ;
814F A6   E8 12           LDA    18,S        ; mot poids faible U
8152 17   009E 81F3       LBSR   BINHEX      ;
8155 ED   81              STD    ,X++        ;
8157 CC   2020            LDD    #$2020      ; deux espaces
815A ED   81              STD    ,X++        ;
815C CC   593A            LDD    #$593A      ; Y:
815F ED   81              STD    ,X++        ;
8161 A6   6F              LDA    15,S        ; mot poids fort Y
8163 17   008D 81F3       LBSR   BINHEX      ;
8166 ED   81              STD    ,X++        ;
8168 A6   E8 10           LDA    16,S        ; mot poids faible Y
816B 17   0085 81F3       LBSR   BINHEX      ;
816E ED   81              STD    ,X++        ;
8170 CC   2020            LDD    #$2020      ; deux espaces
8173 ED   81              STD    ,X++        ;
8175 CC   583A            LDD    #$583A      ; X:
8178 ED   81              STD    ,X++        ;
817A A6   6D              LDA    13,X        ; mot poids fort X
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
817C 8D   75   81F3       BSR    BINHEX      ;
817E ED   81              STD    ,X++        ;
8180 A6   6E              LDA    14,S        ; mot poids faible X
8182 8D   6F   81F3       BSR   BINHEX       ;
8184 ED   81              STD    ,X++        ;
8186 86   0D              LDA    #$D         ; retour chariot
8188 A7   80              STA    ,X+         ;
818A 86   0A              LDA    #$A         ; saut de ligne
818C A7   80              STA    ,X+         ;
818E CC   4450            LDD    #$450       ; lettre  DP
8191 ED   81              STD    ,X++        ;
8193 86   3A              LDA    #$3A        ; signe ":"
8195 A7   80              STA    ,X+         ;
8197 A6   6C              LDA    12,S        ; registre DP
8199 8D   58   81F3       BSR    BINHEX      ;
819B ED   81              STD    ,X++        ;
819D CC   2020            LDD    #$2020      ; deux espaces
81A0 ED   81              STD    ,X++        ;
81A2 CC   423A            LDD    #$423A      ; B:
81A5 ED   81              STD    ,X++        ;
81A7 A6   6B              LDA    11,S        ; registre B
81A9 8D   48   81F3       BSR    BINHEX      ;
81AB ED   81              STD    ,X++        ;
81AD CC   2020            LDD    #$2020      ; deux espaces
81B0 ED   81              STD    ,X++        ;
```

```
81B2 CC   413A            LDD    #$413A       ; A:
81B5 ED   81              STD    ,X++         ;
81B7 A6   6A              LDA    10,S         ; registre A
81B9 8D   38   81F3       BSR    BINHEX       ;
81BB ED   81              STD    ,X++         ;
81BD CC   2020            LDD    #$2020       ; deux espaces
81C0 ED   81              STD    ,X++         ;
81C2 CC   4343            LDD    #$4343       ; lettre  CC
81C5 ED   81              STD    ,X++         ;
81C7 86   3A              LDA    #$3A         ; signe ":"
81C9 A7   80              STA    ,X+          ;
81CB A6   69              LDA    9,S          ; registre CC
81CD 108E 0008            LDY    #8           ; Nb d'opération
81D1 C6   30              LDB    #$30         ; 0 ASCII
81D3 E7   80       BINBIN STB    ,X+          ; initialiser à $30
81D5 48                   LSLA                ; bit nul ???
81D6 24   02   81DA       BCC    *+4          ; si oui, bit suivant
81D8 6C   1F              INC    -1,X         ; sinon, mettre $31
81DA 31   3F              LEAY   -1,Y         ; Cpt. Opération - 1
81DC 26   F5   81D3       BNE    BINBIN       ;
81DE CC   0D0A            LDD    #$0D0A       ; retour et saut de ligne
81E1 ED   81              STD    ,X++         ;
81E3 86   04              LDA    #$04         ; caractère ETX
81E5 A7   80              STA    ,X+          ;
          81E7      ADRCR  SET    *            ; mémoriser adrs courante
                                              ;
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
                  ;------Stockage des chaînes de caractères à l'assemblage
8380                      ORG    BLNOTE+128   ; 128 octets au-delà du bolc-notes
8380      0D0A            FDB    $0D0A        ; retour chariot saut ligne CRLF
8332      50              FCC    /PAUSE/      ; Chaîne à écrire
8387      2020            FDB    $2020        ; deux espaces
8389      04              FCB    $04          ; caractère ETX
838A      0D0A            FDB    $0D0A        ; retour chariot saut ligne CRLF
838C      45              FCC    /EXECUTION/  ;
8395      0D0A            FDB    $0D0A        ; retour chariot saut ligne CRLF
8397      04              FCB    $04          ; caractère ETX
                                              ;
81E7                      ORG    ADRCR        ; Suite de VISURG
                  ;------visualisation de la chaîne PAUSE
81E7 8E   8380            LDX    #BLNOTE+128  ; adresse chaîne à transmettre
81EA 8D   20   820C       BSR    VISU         ;
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
                  ;------Visualiser les caractères contenus dans le
                  ;------bloc-note jusqu'à la rencontre de ETX
81EC 8E   8300            LDX    #BLNOTE      ; début du bloc-notes
81EF 8D   1B   820C       BSR    VISU         ; visualiser
81F1 35   B7              PULS   PC,Y,X,B,A,CC   ; fin S/P VISURG
                  ;*******************************************************
                  ;        S/P Convertion d'un octet binaire en 2
                  ;        caractères ASCII hexadécimaux
                  ;   Nom: BINHEX
                  ;   Entrée: A: donnée à convertir
                  ;   Sortie: A: code ASCII 4 bits poids fort
                  ;           B: code ASCII 4 bits poids faible
                  ;   Exemple: $3A sera converti en $33="3" $41="4"
                  ;   Encombrement pile système: 2 octets
                  ;*******************************************************
81F3 1F   89      BINHEX  TFR    A,B          ;
81F5 44                   LSRA                ; obtenir 4 bits poids fort
81F6 44                   LSRA                ;
81F7 44                   LSRA                ;
81F8 44                   LSRA                ;
81F9 81   0A              CMPA   #$A          ; < 10 ???
81FB 25   02   81FF       BLO    *+4          ;
81FD 8B   07              ADDA   #7           ;
81FF 8B   30              ADDA   #$30         ;
8201 C4   0F              ANDB   #%00001111   ; 4 bits poids faible
8203 C1   0A              CMPB   #$A          ; < 10 ???
8205 25   02   8209       BLO    *+4          ;
8207 CB   07              ADDB   #7           ;
8209 CB   30              ADDB   #$30         ;
820B 39                   RTS                 ; Fin S/P BINHEX
```

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

```
                  ;*******************************************************
```

```
                        ;   S/P visualisation d'une zone mémoire bloc-note
                        ;   jusqu'à la rencontre d'un ETX
                        ;   les registres de l'ACIA1 sont employés ici
                        ;   Nom Appel: VISU
                        ;   Entrée: X: Adresse début bloc
                        ;   Sortie: aucun registre affecté
                        ;   Encombrement pile système: 7 octets
                        ;*******************************************************
820C 34   17          VISU  PSHS   X,B,A,CC     ;
820E A6   80                LDA    ,X+          ; caractère à transmettre
8210 81   04                CMPA   #4           ; ETX ???
8212 26   02    8216        BNE    *+4          ;
8214 35   97                PULS   PC,X,B,A,CC  ; fin S/P VISU
8216 F6   EC80              LDB    RSET1        ;
8219 54                     LSRB                ; registre réception libre ???
821A 54                     LSRB                ;
821B 24   F9    8216        BCC    *-5          ; sinon, attendre
821D B7   EC81              STA    RSTD1        ; transmettre
8220 81   0A                CMPA   #$A          ; saut de ligne ???
8222 27   06    822A        BEQ    TDNUL        ; si oui, transmettre des nuls
8224 81   0D                CMPA   #$D          ; retour chariot ???
8226 27   02    822A        BEQ    TDNUL        ;
8228 20   E4    820E        BRA    VISU+2       ; caractère suivant
822A C6   1E          TDNUL LDB    #30          ; 30 nuls à transmettre
822C B6   EC80              LDA    RSET1        ;
822F 44                     LSRA                ;
8230 44                     LSRA                ;
8231 24   F9    822C        BCC    TDNUL+2      ;
8233 4F                     CLRA                ;
8234 B7   EC81              STA    RSTD1        ;
8237 5A                     DECB                ;
8238 26   F2    822C        BNE    TDNUL+2      ;
823A 20   D2    820E        BRA    VISU+2       ; caractère suivant
....                        ....
....                        ....
823C 39                     RTS                 ; fin S/P Visu
```

retour au Sommaire | Index | Liens Rapides

FEI : Quelques explications sur le programme ci-dessus

- Toutes les étiquettes assembleur sont définies au début du programme. Au besoin adapter les adresses au système sous test avant de procéder à une compilation. La pile système est mise ici à $8400.

- Après avoir configuré l'interface de communication sur le mode interruption en réception et non en transmission, le programme de charge les vecteurs d'interruption SWI et IRQ dans les adresses réservées du µp6809. Avant d'entrer dans la boucle sans fin, le µp6809 autorise l'interruption par IRQ.

- Examinons tout d'abord l'interruption logicielle SWI. Pour la rendre opérationnelle, il est nécessaire de remplacer BRA TSTINT par 2 instructions NOP puis lancer l'exécution à partir de $8000.

- A l'entrée dans la séquence d'exception SWI, les interruptions par IRQ et FIRQ sont inhibées. L'ensemble des registres est sauvegardé dans la pile système et le pointeur S décroît de 12 unités.

  Après la visualisation des registres par l'appel de VISURG, la séquence d'exception autorise les interruptions par IRQ et FIRQ.

  Ceci n'est pas tout à fait nécessaire, puisque le contrôle est donné au moniteur

  L'instruction LEAS 12,S rétablit la position du pointeur pour simuler les conditions du programme principal juste avant la rencontre de SWI.

  Rappelons que cette séquence d'exception ne charge aucuns registres du µp6809 à part CC et PC. Donc, à l'entrée du moniteur par JMP MONIT, on retrouve les conditions du programme principal si le moniteur du système sous test sait sauvegarder le contexte.
  Au besoin, supprimer les instructions ANDCC #%10101111 et LEAS 12,S.

- Pour tester les interruptions par IRQ, rétablir l'instruction BRA TSTINT. L'interruption peut se produire de façon aléatoire à l'intérieur de la boucle TSTINT. Chaque fois qu'une donnée est rentrée au clavier, l'impulsion sur IRQ provoque une séquence d'interruption.

  Si la donnée reçue n'est pas la touche "H", la séquence d'exception du premier niveau retourne le contrôle programme principal en restituant l'ensemble des registres.

- Lorsque la touche "H" est d'identifiée, le µp6809 visualise les registres selon le format suivant :
  `PAUSE PC:.... S:... U:.... Y:.... X:.... DP:.... B:....`
  `A:.... CC:........`

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

Après la transmission de l'ensemble du buffer vers l'écran, la séquence d'exception du premier niveau modifie le vecteur IRQ à l'adresse $FFF8, $FFF9 avant d'exécuter CWAI et autoriser simultanément IRQ.

L'opérateur peut alors rentrer une autre commande. Chaque impulsion sur IRQ provoque un traitement de 2ième niveau.

Si l'identification révèle une touche autre que "C", le contrôle est rendu au 1er niveau mais sur l'instruction CWAI.

Ce détail implique une modification de la valeur du compteur programme dans la pile système avant de dépilement.

Si le caractère reçu est "C", le sous-programme du 2ième niveau affiche le message "EXECUTION" puis retourne au 1er niveau, à l'instruction juste après CWAI.

Avant de rendre le contrôle au programme principal, le 1er niveau rétablir encore une fois le vecteur IRQ correspondant.

Donc, il est important de souligner qu'on peut imbriquer un grand nombre d'interruptions en rétablissant chaque fois correctement le vecteur IRQ du niveau correspondant.

Cette manipulation est normalement interdite puisque le masque bit I est mis à 1 chaque fois qu'on actionne la ligne IRQ. Néanmoins, le programmeur peut concevoir son système particulier, le µp6809 possède toutes les ressources logicielles pour satisfaire le programmeur exigeant.

FEI : Trois broches d'entrées d'interruption Matérielle :

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

Le µp6809 possède trois interruptions matérielles (NMI, IRQ et FIRQ) plus une séquence d'initialisation RESET, cette dernière est traitée comme l'interruption la plus prioritaire du système.

FEI : Fonctionnement de la Broche NMI | (Non Masquable Interrupt) Interruption non masquable

retour au Sommaire | Index | Les Interruptions | Liens Rapides | Vecteurs d'Interruption

(Broche n°2) Interruption Matérielle
Une interruption matérielle provient d'un signal externe.

C'est une interruption mon masquable sensible à un front descendant ↓ de la broche (NMI|) entraîne une séquence d'interruption NON MASQUABLE.

Cette interruption ne peut être interdite d'où son nom. Le programmeur ne peut donc pas interdire son fonctionnement par programme et possède une priorité supérieure à FIRQ| ou à IRQ| ou aux interruptions logicielles SWI, SWI2, SWI3. L'interruption NMI| est la plus prioritaire après la séquence RESET.

Cette interruption est destinée à configurer des séquences de sauvegarde de l'ensemble du système en présence de défaut d'alimentation électrique.

Eventuellement, on peut se servir de cette entrée pour installer un arrêt manuel du type "abort". Dans la plupart des moniteurs, ce bouton joue à peu près le même rôle que SWI. Cependant l'intervention demeure parfaitement aléatoire.

Décrivons ici la réponse à NMI :

1) L'indicateur bit E du registre d'état, est mis à 1 pour mémoriser l'étendue de l'opération d'empilement.

2) Le µp6809 empile tous les registres dans la pile système S dans l'ordre suivant : PCL, PCH, UL, UH, YL, YH, XL, XH, DP, B, A, CC. Le pointeur S décroît de 12 cases mémoire.

3) Les interruptions IRQ et FIRQ sont inhibées par masquage des bits I et F du registre d'état CC.

4) Le µp6809 viens chercher l'adresse de la séquence d'exception dans les mémoires $FFFC et $FFFD. L'exécution devrait se terminer par RTI pour rendre le contrôle au programme initial.

Le µp6809 sauvegarde tous ses registres internes et se branche automatiquement à un sous-programme dont l'adresse est stockée dans les adresses :

{$FFFC} + {$FFFD}
LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

Cette entrée NMI| est souvent réservée aux traitements devant résulter d'une défaillance d'alimentation, sauvegarde dans une mémoire CMOS alimentée par une pile.

Cette broche est dévalidées par la broche RESET| et n'est pas revalidée d'après un changement du registre S (système) en mode d'adressage immédiat.

Le contenu de la totalité des registres du µp6809 est sauvegardé dans la pile système.

Le bit E de CC est mis à 1 avant son chargement dans la pile système, pour indiquer que l'état complet du processeur a été sauvegardé (dans l'ordre PC bas, PC haut, U bas, U haut, Y bas, Y haut, X bas, X haut, DP, B, A puis CC).

FEI : Fonctionnement de la Broche IRQ

(Interrupt ReQuest), interruption masquable

(Broche n°3) Interruption Matérielle. Une interruption matérielle provient d'un signal externe.

Cette interruption peut ou non être autorisée, elle est la moins prioritaire des interruptions matérielles. IRQ et un mode d'interruption très populaire. IRQ| à donc une priorité plus basse que les broches FIRQ| ou NMI|.

Le déroulement de la routine engendré par IRQ| peut-être interrompue par les broches FIRQ| ou NMI| Tous les registres sont empilés

Sur un niveau bas sur la broche IRQ| et après la fin de l'instruction en cours, le µp6809 sauvegarde tous ses registres internes (l'état complet du µp6809) et se branche automatiquement à un sous-programme dont l'adresse est stockée dans les adresses $FFF8 + $FFF9 à condition que le bit I du registre CC soit à 0.

{$FFF8} + {$FFF9}
LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

Le sous-programme de traitement de l'interruption doit libérer la source de l'interruption avant de d'exécuter l'instruction RTI

IRQ| est masquable par l'intermédiaire du bit I de CC.

- Lorsque le **bit I = 1** l'interruption est interdite.
- Lorsque le **bit I = 0** l'interruption est autorisé.

La mise à 1 de ce bit I, part une instruction du type ORCC #%00010000 rend le µp6809 insensible à l'état électrique de la broche IRQ.
Le bit I peut-être remis à zéro par une opération du type ANDCC #%11101111.

Examinons la réponse du µp6809 à une interruption IRQ lorsque cette dernière n'est pas masquée :

1) l'indicateur E est mis à 1, d'où empilement de l'ensemble des registres.

2) Le µp6809 effectue une sauvegarde de l'ensemble des registres dans l'ordre suivant : PCL, PCH, UL, UH, YL, YH, XL, XH, DP, B, A, CC. Le pointeur S décroît de 12 cases mémoire.

3) L'indicateur I et mis à 1 inhibant ainsi toutes les autres instructions de type IRQ survenues ultérieurement durant le traitement de la séquence d'exception.

4) L'indicateur F n'est pas touché, donc pendant le traitement du son programme d'interruption IRQ, le µp6809 accepte les interruptions du type FIRQ.
Il est tout à fait possible d'autoriser les interruptions multiples par IRQ réparties en niveaux.
Pour ce faire à l'entrée de chaque séquence d'exception il faut mettre le bit I à 0 par ANDCC # %11101111 et manipuler avec soin les contenus des adresses $FFF8 et $FFF9.

5) Le µp6809 charge le contenu des adresses $FFF8 et $FFF9 dans le compteur programme PC, puis exécute la séquence d'exception qui doit se terminer par une instruction RTI.

De façon générale, dans un système simple, tous les périphériques sont câblés à IRQ à travers une porte OU à plusieurs Entrées. Le plus souvent les circuits ont une sortie à collecteur ouvert, ce qui évite l'ajout de logique supplémentaire.

Donc, chaque fois que le µp6809 reçoit un niveau bas sur cette entrée, il ignore totalement quel est le périphérique qui a émis cette demande.

Donc dans la conception des logiciels le début de la séquence d'exceptions relatives à une interruption IRQ contient une identification de la source émettrice. Ceci explique l'utilité des bits d'état des registres d'interface 6821 et 6850.

Il peut arriver que plusieurs périphériques émettent des demandes d'interruption dans le laps de temps écoulé entre la première demande et le début du traitement d'exception.

Dans une telle situation, l'ordre avec lequel le programmeur teste les différents périphériques détermine l'ordre de priorité.

Il existe une façon matérielle autorisant une identification rapide de la source interruption. Le procédé s'appelle une vectorisation des interruptions, qui nécessite l'implantions d'un certain nombre de circuits spécialisés. La méthode d'identification de la source par logiciel s'appelle "identification logicielle" "polling".

FEI : Fonctionnement de la Broche FIRQ (Fast Interrupt ReQuest), interruption Rapide

(Broche n°4) Interruption Matérielle. Une interruption matérielle provient d'un signal externe.

Dans certains cas, le µp6809 doit traiter les interruptions très rapidement de qui pose un problème avec les interruptions classiques où la sauvegarde totale du contexte microprocesseur prend un certain temps (1 cycle d'horloge par octet sauvegardé).

Cette interruption FIRQ et plus rapide puisqu'il n'y a que de compteur programme et le registre de condition CC qui sont sauvegardés, ce qui fait 3 octets au lieu de 12 en temps normal.

Ici le bit E est positionné à 0 de manière à indiquer que seuls les registres PC bas, PC haut et le registre CC sont sauvegardés dans la pile système.

Un niveau bas sur cette broche FIRQ| entraîne la séquence FIRQ à condition que le bit F du registre CC soit à 0.

Cette interruption a priorité par rapport à une demande d'interruption standard la broche IRQ|.
L'adresse de départ du sous-programme de traitement des FIRQ est donnée par le contenu des adresses :

Vecteurs d'Interruption

```
{$FFF6} + {$FFF7}
```

LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

Le sous-programme de traitement des interruptions doit libérer la source de l'interruption avant l'exécution de l'instruction RTI.

L'opération de sauvegarde effectuée lors d'une interruption IRQ est relativement longue et retarde le traitement de la séquence d'exception.

FIRQ n'effectue qu'une sauvegarde partielle (PC et CC) et permet d'accélérer la procédure en économisant 9 cycles machine en comparaison du temps mis par le µp6809 pour répondre à IRQ. C'est à l'utilisateur d'empiler les autres registres qui va altérer.

FIRQ et également masquable par le bit F, bit b6 du registre d'état CC. Sous l'angle matériel, son implantation est identique à IRQ.

La prise en compte de FIRQ par le µp6809 quand le bit F=0 se déroule de la manière suivante :

1) Le µp6809 met le bit E à 0 indiquant qu'il va empiler seulement le compteur programme PC et le registre d'état CC.

2) La sauvegarde partielle s'effectue dans l'ordre PCL, PCH puis CC. Le pointeur S décroît de 3 cases mémoire à la fin de l'opération.

3) Le masquage simultané des bits I et F interdit toute interruption ultérieure sur les lignes correspondantes.
On remarquera que la prise en compte de FIRQ désactive à la fois FIRQ et IRQ, alors que la prise en compte d'IRQ n'exclut pas une interruption ultérieure par FIRQ.
On dit parfois que FIRQ est prioritaire par rapport à IRQ.

4) Le µp6809 viens chercher l'adresse du sous-programme d'exception à l'adresse $FFF6 et $FFF7. Le contrôle est rendu au programme initial également par une instruction RTI. Dans la séquence d'exception au besoin on peut effectuer une sauvegarde partielle des registres utile par l'instruction PSHS. Dans un tel cas, ne pas oublier de les dépiler juste avant la rencontre de l'instruction RTI, car cette dernière ne dépilera que les registres PC et CC.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Trois instructions d'interruption Logicielle

Instructions permettant l'arrêt du programme en cours (pour voir le détail de ces instructions) :

SWI (SoftWare Interrupt)
SW2 (SoftWare Interrupt 2)
SWI3 (SoftWare Interrupt 3)

retour au Sommaire | Vecteurs d'Interruption | Index | Liens Rapides

FEI : Traitement des Interruptions Logicielles SWI SWI2 SWI3

Dès que le µp6809 rencontre l'une des trois instructions ci-dessous SWI, SWI2, SWI3, celui-ci sauvegarde l'état complet des registres internes dans la pile système. Dans l'ordre suivant :

```
S - 1 à S       PC bas à (S)
S - 1 à S       PC haut à (S)
S - 1 à S       U bas à (S)
S - 1 à S       U haut à (S)
S - 1 à S       Y bas à (S)
S - 1 à S       Y haut à (S)
S - 1 à S       X bas à (S)
S - 1 à S       X haut à (S)
S - 1 à S       DP à (S)
S - 1 à S       B à (S)
S - 1 à S       A à (S)
S - 1 à S       CC à (S)
```

Puis le µp6809 continu son déroulement.

retour au Sommaire | Les Interruptions | Vecteurs d'Interruption | Index | Liens Rapides

FEI : Instruction d'interruption logicielle SWI (SoftWare Interrupt)

Avant la sauvegarde des registres internes le bit E de CC est positionné à 1, pour indiquer que l'état total du µp6809 est sauvegardé dans la pile S.
L'interruption logiciel SWI est une instruction assembleur possédant un code opératoire `$3F`.

Après la sauvegarde des registres internes, les bits I et F de CC sont positionné à 1 (ce qui signifie que les interruptions matérielles sont interdites).

Cette interruption logicielle est plus prioritaire que FIRQ| et IRQ| car son traitement entraîne le masquage de FIRQ| et IRQ|.

Généralement, SWI est utilisée dans un moniteur de mise au point de programme, pour faire des arrêts sur adresses.
Le PC est chargé avec le contenu des adresses :

```
{$FFFA} + {$FFFB}
```

LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

A la rencontre de cette instruction SWI le µp6809 déclenche la série d'opération suivante :

1) Mise à 1 du bit E avertissant que l'ensemble des registres sont sauvegardés.

2) sauvegarde de l'ensemble des registres dans l'ordre suivant : PCL, PCH, UL, UH, YL, YH, XL, XH, DP, B, A, CC. Le pointeur S décroît de 12 cases mémoire à la fin de cette opération.

3) Interdiction d'IRQ (bit I=1) et FIRQ (bit F=1). La séquence d'exception SWI ne sera pas interrompue par les sources connectées à IRQ et FIRQ.

4) Chargement du contenu des adresses $FFFA et $FFFB dans le compteur programme. Cette séquence d'exception doit se terminer également par l'instruction RTI qui restitue le contexte initial.

SWI est une interruption logicielle conçue avec une priorité importante puisqu'elle interdit IRQ et FIRQ.

Pour cette raison, SWI trouve son utilité dans les utilitaires systèmes destinés à la mise au point des programmes utilisateurs. Ca permet également de faire appel à un OS, à des routines systèmes.
L'intérêt est que l'utilisateur n'a pas à connaître l'adresse, c'est l'OS qui s'occupe de tout.

Par exemple, pour poser un point d'arrêt à une adresse donnée, le moniteur remplace le premier octet de l'instruction pour le code $3F.

A la rencontre d'un code, le sous-programme d'interruption SWI visualise les registres du µp6809 et arrête momentanément exécution en entrant dans une phase d'attente.

Par la suite, si l'opérateur désire continuer, le moniteur établit l'op-code correct sauvegardé dans la zone système, positionne le prochain point d'arrêt, puis exécuter le programme utilisateur à partir de l'ancien point d'arrêt.

Cette méthode implique que le programme sous test doit se trouver dans un espace mémoire lecture écriture.

retour au Sommaire | Les Interruptions | Vecteurs d'Interruption | Index | Liens Rapides

FEI : Instruction d'interruption logicielle SWI2

Fonctionnement similaire à SWI, sauf que les masques d'interruptions bits I et F de CC ne sont pas affectés.

Autrement dit, SWI2 et SWI3 ont un fonctionnement identique à SWI, mais elles peuvent être interrompue par toutes autres interruptions du µp6809 et sont par conséquent les moins prioritaires

Le PC est chargé avec le contenu des adresses :

`{$FFF4} + {$FFF5}`

LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

Les interruptions matérielles sont donc autorisées durant l'exécution du sous-programme d'interruption logicielle SWI2, elle a une priorité inférieure à SWI.

FEI : Instruction d'interruption logicielle SWI3

retour au Sommaire | Index | Les Interruptions | Liens Rapides | Vecteurs d'Interruption

Fonctionnement similaire à SWI2, sauf que l'adresse du sous-programme de traitement est contenue des adresses :

`{$FFF2} + {$FFF3}`

LSB (Least Signifiant Bit) poids Faible
MSB (Most Signifiant Bit) poids Fort

FEI : Interruptions logicielles SWI2 et SWI3

retour au Sommaire | Index | Les Interruptions | Liens Rapides | Vecteurs d'Interruption

Elles sont réservées à l'usage du programme utilisateur.

Leur fonctionnement reste semblable à SWI à la différence près que SWI et SWI ne masque pas les interruptions IRQ et FIRQ (les bits I et F conservent leur valeur au commencement du traitement d'exception), donc les sous-programmes correspondants demeurent interruptibles.

On dit que SWI2 et SWI3 ont une priorité inférieure à celle de SWI. Les adresses des vecteurs d'interruption sont :

- $FFF4 et $FFF5 pour SWI2
- $FFF2 et $FFF3 pour SWI3

Si l'instruction SWI est configurée différemment d'un système à l'autre, SWI2 et SWI3 peuvent faire partie d'un logiciel "portable".

Les sous-programmes d'exception correspondants doivent accompagner les modules livrés, avec gestion des adresses absolues $FFF2 à $FFF5.

FEI : Instructions d'Attente d'Interruptions SYNC CWAI

FEI : Instruction CWAI (**C**lear **WA**it **I**nterrupt) Attente d'interruption

retour au Sommaire | Index | Les Interruptions | Liens Rapides | Vecteurs d'Interruption

Le µp6809 doit pouvoir se mettre en attente ou se synchroniser sur un évènement extérieur dont la présence est signalée par une ou des broches d'entrée d'interruptions (Voir instructions SYNC et CWAI).

L'instruction CWAI synchronise le µp6809 sur un évènement externe par le biais d'une interruption.

CWAI est une instruction possédant un opérande écrit dans le mode immédiat.

(CC %xxxxxxxx) *à* **CC avec bit E=1**

CWAI fonctionne dans sa première phase comme ANDCC (elle effectue un ET logique entre l'octet mémoire immédiat et le registre de conditions CC).

Ceci a pour but de positionner à 0 certains bits de CC, on peut effacer ici les masques d'interruptions. Ensuite le bit E est mis à 1 ce qui a pour but de provoquer une sauvegarde totale du contexte (états de tous les registres internes) dans la pile matérielle.

Cet état est donc sauvegardé, le µp6809 se met en position d'attente d'interruption (il n'exécute donc plus d'instructions).

Lorsqu'une interruption intervient, le µp6809 se branche à la routine de gestion d'interruption et l'exécute.

Lorsque le µp6809 rencontre une instruction RTI (retour d'interruption le contexte est restauré puisque le bit E de CC sauvegardé précédemment a été positionné à 1.

Avant l'interruption CWAI :

- Si le bit F=1 avant CWAI, seule IRQ est activée.
- Si le bit F=0 avant CWAI, FIRQ et IRQ peut interrompre le µp6809.

NMI peut également interrompre le µp6809 bien que ce mode de fonctionnement ne soit pas très habituel.
Exemple d'attente d'interruption NMI

```
CWAI    #$FF          ; car IRQ et FIRQ sont masqué.
```

Par exemple CWAI #%11101111 effectue un ET logique du registre CC avec l'opérande demandé, sauvegarde l'ensemble des registres dans la pile le système S puis se met en attente d'interruption.

CWAI permet au µp6809 de synchroniser le traitement de la séquence d'exception sur une interruption IRQ ou FIRQ.

La réponse est rapide puisque le µp6809 a déjà sauvegardé l'ensemble des registres. Il ne lui reste plus qu'à charger le vecteur d'interruption pour le compteur programme, soit à partir des mémoires :

- $FFF8 et $FFF9 pour IRQ
- $FFF6 et $FFF7 pour FIRQ

Vecteurs d'Interruption

Notons toutefois que le bit E est égal à 1, par conséquent, même si interruption est un FIRQ, la sauvegarde est totale et non partielle.

Au dépilement par l'instruction RTI, l'ensemble des registres sera restitué, d'où un mouvement du pointeur S de + ou - 12 unités.

On trouve CWAI dans des applications l'on a besoin d'une interruption simultanée de toutes les unités de traitement sur une seule impulsion de départ. CWAI force toutefois le µp6809 à se brancher vers une séquence d'exception, ce qui n'est pas le cas avec SYNC.

Dans d'autres circonstances, CWAI permet aux programmeurs de sélectionner un endroit précis dans son programme pour lancer un traitement d'exception, même lorsqu'il s'agit d'une interruption matérielle. C'est pour la raison pour laquelle nous l'appelons "interruption matériel programmée" alliant les deux concepts matériel et logiciel.

CWAI écarte en quelque sorte de l'aspect aléatoire d'une interruption matérielle.

FEI : Instruction SYNC (**SYNC**hronize to external event) Attente d'une synchronisation externe

retour au Sommaire | Index | Les Interruptions | Liens Rapides | Vecteurs d'Interruption

Le µp6809 doit pouvoir se mettre en attente ou se synchroniser sur un évènement extérieur dont la présence est signalée par une ou des broches d'entrée d'interruptions (Voir instructions SYNC et CWAI).

L'instruction SYNC permet de synchroniser le µp6809 sur un évènement extérieur, grâce aux broches d'interruption NMI|, IRQ| et FIRQ|.

Utile par exemple dans une application biprocesseur où les taches sont partagées.

Elle permet également de réaliser des synchronisations rapides avec les périphériques, cette méthode permet éventuellement d'éviter l'utilisation d'un circuit d'accès direct mémoire.

Dès la rencontre de cette instruction le µp6809 s'arrête et attend qu'une interruption se produise.

Cette interruption peut être masquée par le biais des bits I ou F de CC.

Dès d'une interruption apparaît le µp6809 reprend son programme et exécute les instructions suivant l'instruction SYNC.

C'est une interruption matérielle programmée. A la rencontre d'une telle instruction le µp6809 se met en attente d'interruption qui peut provenir de IRQ ou de FIRQ ou éventuellement de NMI.

Si une interruption quelconque est inhibée par un masquage (sauf pour NMI), ou si le niveau Bas appliqué dure moins de trois cycles machine, le µp6809 continue l'exécution normale de l'instruction suivant SYNC.

Si le niveau Bas appliqué se prolonge au-delà de trois cycles machine avec l'indicateur correspondant non masqué (égal à 0), alors le µp6809 entame le traitement habituel de l'interruption sollicitée.

On remarque que :

- Tous niveaux Bas appliqué sur l'une des entrées IRQ, FIRQ ou NMI, quelle que soit sa durée, quel que soit l'état des bits I et F, provoque un redémarrage du µp6809 et le sort de l'état HALT.

- Si une impulsion dure moins de trois cycles machine, son rôle essentiel est de redémarrer le µp6809 sur un programme commençant immédiatement après l'instruction SYNC, et ceci très rapidement car il n'y a ni opération de sauvegarde, ni vectorisation.
  A ce titre, on peut se servir de SYNC et d'une impulsion externe courte pour synchroniser le traitement du µp6809 sur un événement externe, d'où nom de cette instruction.

- Par comparaison avec l'instruction CWAI, l'instruction SYNC peut sortir le µp6809 de son état latent sans orienter ce dernier vers une séquence interruption, alors que CWAI oblige le µp6809 à se diriger vers un traitement d'exception. SYNC ne contient pas d'opérande permettant d'agir sur les indicateurs du registre d'état comme CWAI.

retour au Sommaire | Les Interruptions | Vecteurs d'Interruption | Index | Liens Rapides

FEI : Instruction RTI (**R**e**T**urn from **I**nterrupt)

Toutes les séquences de traitement d'interruption doivent se terminer par l'instruction RTI pour éviter toute mauvaise manipulation de la pile.
L'utilisation de cette instruction comme les instructions CWAI, SYNC, … sera très rare.

L'appel à un sous programme peut se faire de 2 façons :

- Par logiciel par l'utilisation de JSR et le retour par RTS
- Par le matériel par une interruption, l'instruction de retour dans ce cas là est RTI (voir le détail dans le chapitre des interruptions)

Dès que le µp6809 rencontre cette instruction, il teste tout d'abord la valeur du bit E de CC, registre CC est restauré en premier :

- **Bit E = 1** le restant des registres sont restauré dans l'ordre A, B, DP, X haut, X bas, Y haut, Y bas, U haut, U bas, PC haut puis PC bas)
  Autrement dit : Si E=1 alors l'empilement était total. Le dépilement correspond à l'ordre connu pour l'opération PSHS, soit CC, A, B, DP, XH, XL, YH, YL, UH, UL, PCH, PCL avec une variation du pointeur S de + ou - 12 unités à la fin de l'instruction RTI.

- **Bit E = 0** Seul le registre PC est restauré dans l'ordre PC haut puis PC bas (le registre CC ayant été restauré en premier lieu).
  Autrement dit : Si E=0 alors il s'agissait d'un empilement partiel au début de l'exécution de l'interruption. Le µp6809 dépile dans l'ordre CC, PCH, PCL et décroît aussi le pointeur de 3 unités

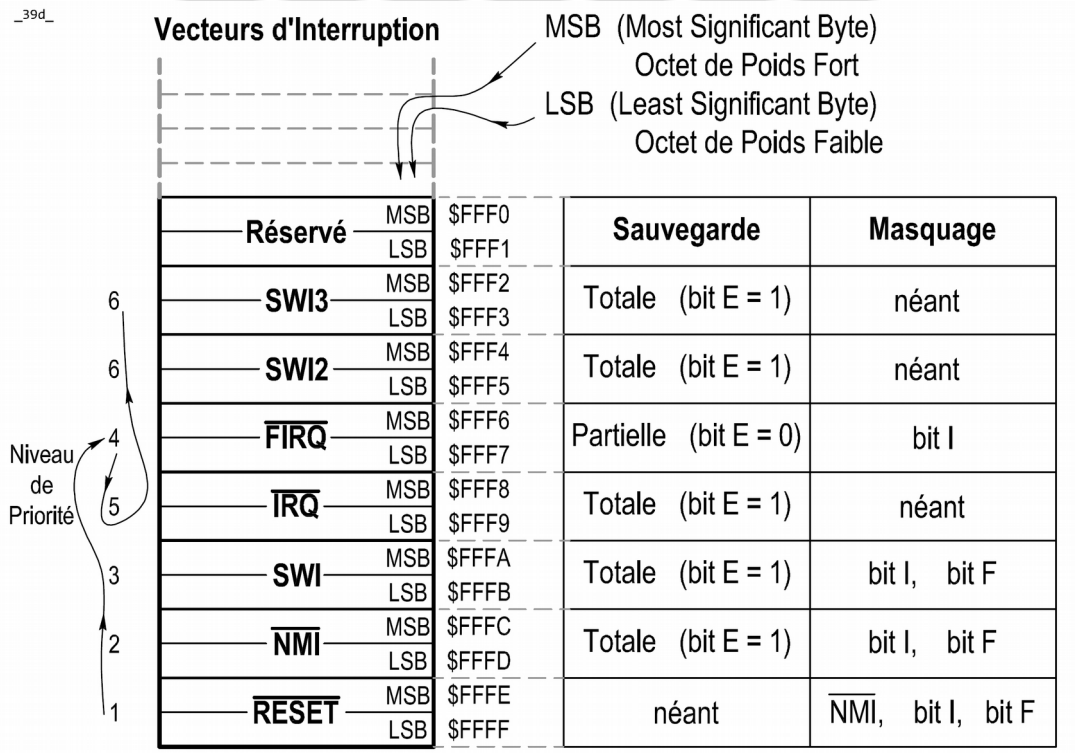
RTI est la dernière instruction d'un sous programme d'interruption, à sa rencontre le µp6809 dépile d'abord le registre d'état CC, il examine la valeur du bit E pour connaître l'étendue du dépilement.

retour au Sommaire | Les Interruptions | Index | Liens Rapides

FEI : Tableau des Vecteurs d'Interruption

Les interruptions du µp6809 font apparaître des niveaux de priorité, généralement liées à la structure Matérielle.
Lors d'une interruption le µp6809 se positionne automatiquement à une adresse contenue dans deux octets (MSB et LSB).
Cette adresse représente l'adresse du programme de traitement de l'interruption demandée, en voici le tableau.

SWI3 | SWI2 | FIRQ | IRQ | SWI | NMI | RESET



			Sauvegarde	Masquage
Réservé	MSB	$FFF0		
	LSB	$FFF1		
SWI3	MSB	$FFF2	Totale (bit E = 1)	néant
	LSB	$FFF3		
SWI2	MSB	$FFF4	Totale (bit E = 1)	néant
	LSB	$FFF5		
$\overline{\text{FIRQ}}$	MSB	$FFF6	Partielle (bit E = 0)	bit I
	LSB	$FFF7		
$\overline{\text{IRQ}}$	MSB	$FFF8	Totale (bit E = 1)	néant
	LSB	$FFF9		
SWI	MSB	$FFFA	Totale (bit E = 1)	bit I, bit F
	LSB	$FFFB		
$\overline{\text{NMI}}$	MSB	$FFFC	Totale (bit E = 1)	bit I, bit F
	LSB	$FFFD		
$\overline{\text{RESET}}$	MSB	$FFFE	néant	$\overline{\text{NMI}}$, bit I, bit F
	LSB	$FFFF		

retour au Sommaire | Index | Les Interruptions | Liens Rapides

FEI : Modes de Traitement

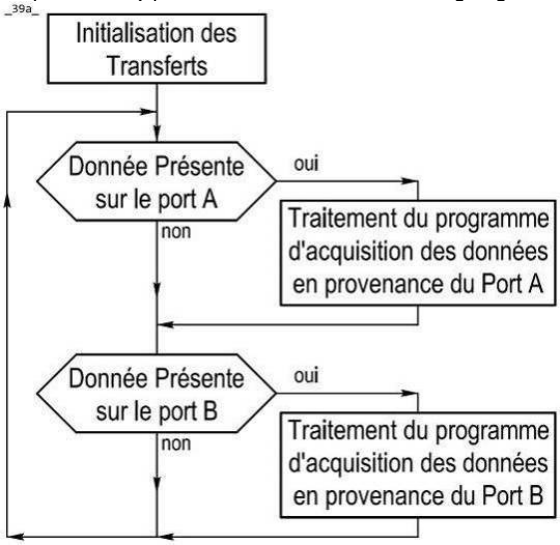
Le but de cette partie est de mettre en évidence les méthodes de gestion des requêtes d'entrée-sortie des périphériques du µp6809.

retour au Sommaire | Index | Liens Rapides

FEI : Scrutation Systématique

Dans ce cas le µp6809 initialise les transferts, puis exécute une boucle d'attente d'événement extérieur en testant systématiquement les indicateurs d'état de chacun des périphériques.

Pour une application comportant un µp6809 et un PIA 6821, voici l'organigramme de la boucle de scrutation.



L'immobilisation du µp6809 pénalise fortement cette méthode, le temps d'activité du µp6809 est très faible devant celui des périphériques connectés (une imprimante par exemple).

Pour remédier à cela, le fonctionnement en interruptions apporte une solution appréciable.

FEI : Principe de Fonctionnement en interruption

retour au Sommaire | Index | Liens Rapides

On interrompt le programme en cours et on exécute un sous-programme de traitement d'interruption.
Le sous-programme de traitement contient les routines de transfert, il tient compte du niveau de priorité des interruptions.

Ce mode de transfert permet au µp6809 d'exécuter des traitements indépendants.
Le µp6809 est sollicité seulement lorsque l'interface est prête à dialoguer en vue de transférer des informations à la mémoire centrale.

FEI : Gestions des Interruptions

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

En général, l'entrée IRQ| du µp6809 doit supporter plusieurs interfaces. Les interfaces possèdent des niveaux de priorité égaux.

Pour définir un niveau de priorité d'une interface par rapport à une autre il existe plusieurs méthodes :

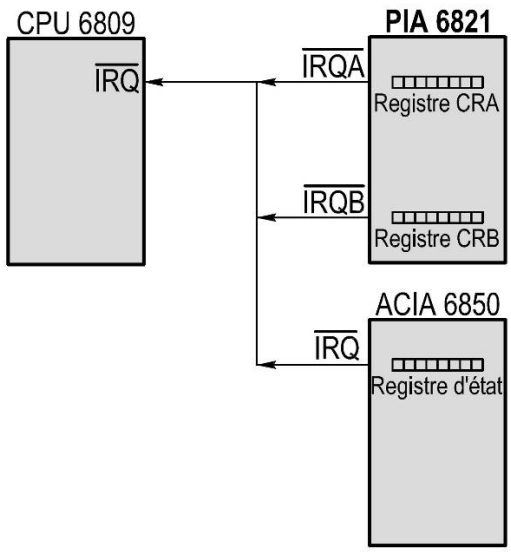
- La méthode logicielle
- La méthode matérielle

FEI : Méthode logicielle

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

Toutes les lignes des interfaces sont câblées en "OU" et reliées à l'entrée demande d'interruption du µp6809.

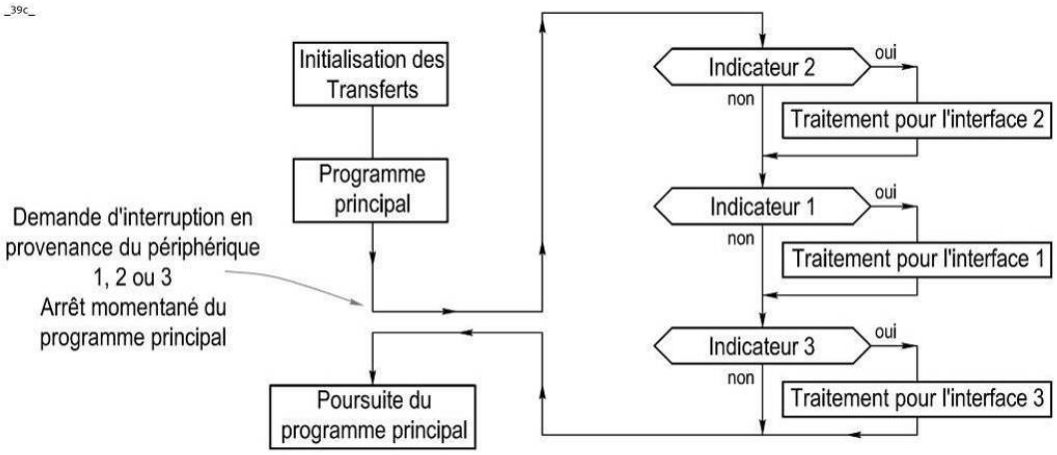
_39b_



Lorsque l'entrée IRQ| du µp6809 est activée, le µp6809 fournit l'adresse du vecteur d'interruption (pour le µp6809 le vecteur pour IRQ| est une adresse de 16 bits stockée en $FFF8 et $FFF9.

C'est l'informaticien qui détermine la priorité pour tester chacun des indicateurs d'état, afin de savoir qui a créé l'interruption. La première interface testée dans le sous-programme sera la plus prioritaire.

La figure suivante montre que l'interface n°2 est la plus prioritaire. La moins prioritaire étant la n°3 dans cet exemple. Une scrutation (ou "POLLING") est réalisée seulement après une demande d'interruption.



retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Méthode matérielle

On peut établir la hiérarchie entre les interfaces en utilisant une logique électronique externe qui permet d'identifier directement l'origine de la demande.

Cette méthode permet au µp6809 d'accéder directement au vecteur d'interruption correspondant à la demande, il n'y a donc plus besoin de scrutation logicielle des interfaces.

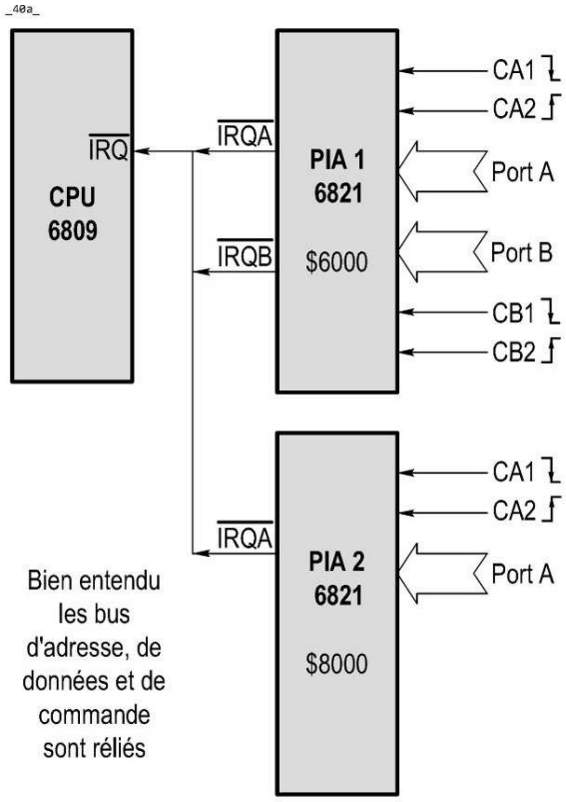
Il existe des circuits intégrés spécialisés dont le rôle est de contrôler les priorités d'interruption.

FEI : Exemple 01 à base de 2 PIA 6821

Dans cet exemple, une application est à base d'un µp6809 et de deux PIA assurant les liaisons avec la périphérie.

retour au Sommaire | Index | Liens Rapides

FEI : Ex01 Structure de l'Application



Les ports A et B du PIA1 et le port A du PIA2 sont en entrée.

Toutes les lignes de dialogue sont programmées en entrées.

Les lignes CA1 et CB1 sont actives sur des fronts descendants.
Les lignes CA2 et CB2 sont actives sur des fronts montants.

A chaque ligne de dialogues (CA1, CB1, CB1, CB2) correspond un périphérique différent, et par conséquent un sous-programme de traitement approprié.

Le S/Prog CA1 traite l'interruption générée par la ligne CA1 du PIA1

Le S/Prog CA2 traite l'interruption générée par la ligne CA2 du PIA1

L'entrée d'interruption IRQ| du µp6809 est reliée aux lignes IRQA et IRQB des PIA (voir schéma ci-dessus).

FEI : Ex01 Fonctionnement

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

On ne s'occupe dans cet exemple que des informations en provenance de l'extérieur.
Comme toutes les lignes d'interruption sont reliées entre elles, le contrôle de priorité est réalisé par logiciel.

Le programme de gestion des PIA se décompose en 3 parties :

- Initialisation du transfert
- Scrutation des interfaces
- Exécution du transfert.

Un front actif sur CA1 du PIA1 entraîne le positionnement le bit indicateur d'état CRA7, une interruption validée par le bit CRA0 = 1 est envoyé vers le µp6809.

Le µp6809 termine l'instruction en cours puis exécute le sous-programme de traitement des interruptions.

L'indicateur est réinitialisé, les données sont transférées, puis le processeur reprend le programme principal.

FEI : Ex01 Organisation de la mémoire

retour au Sommaire | Index | Liens Rapides

Chacun des PIA occupe 4 octets mémoire.

Les 6 sous-programmes de traitement des transferts d'information sont implantés entre les adresses $3000 et $4000.

L'ensemble des adresses de base des sous-programmes de traitement et des octets PIA est regroupé dans une table à partir de l'adresse $2000.

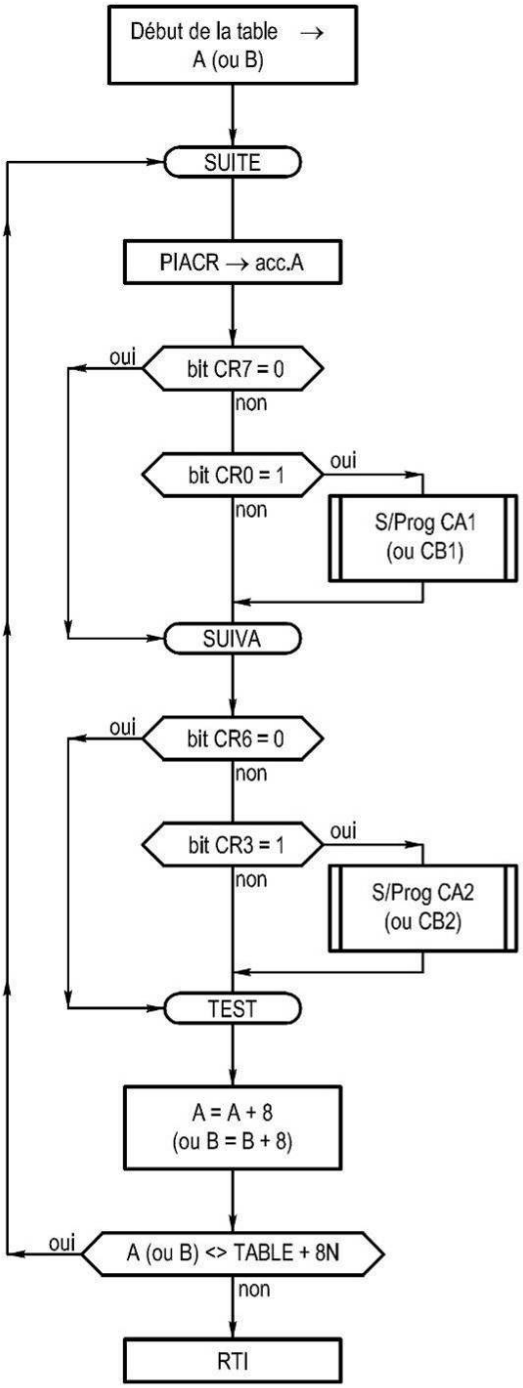
Le sous-programme de scrutation est implanté à l'adresse $0100.

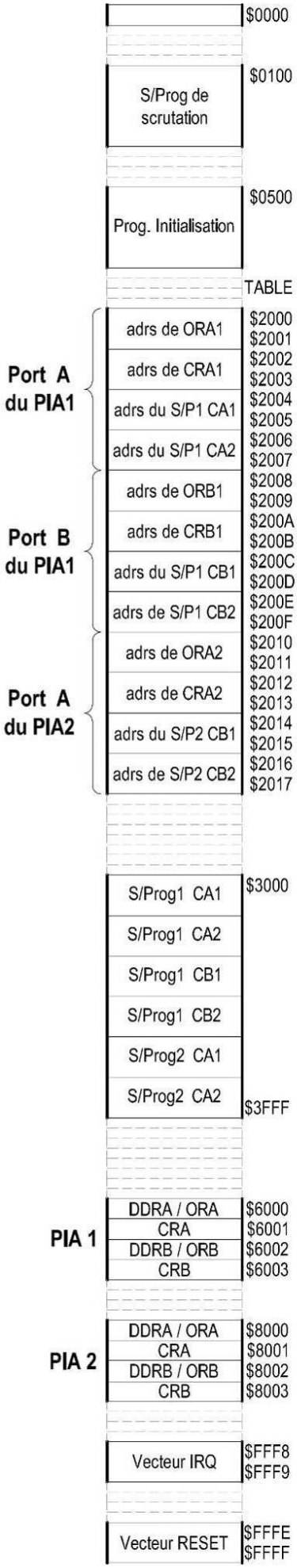
Le sous-programme d'initialisation à l'adresse $0500.

Voir l'organisation de la mémoire apposé contre l'organigramme suivant.

FEI : Ex01 Organigramme de la scrutation

_4eb_





FEI : Ex01 Programmation

FEI : EX01 Initialisation du transfert

Cette partie permet d'initialiser le PIA
Il faut programmer les ports PIA1 A, PIA1 B et PIA2 A en entrées en initialisant les registres DDRA et DDAB.
Le fonctionnement est défini ensuite par le contenu des registres de contrôle

```
MEM     EQU     $A000       ;
PILE    EQU     $F3FF       ;
;---- Définition des registres de programmation
ADPIA1  EQU     $6000       ;
ADPIA2  EQU     $8000       ;
CRA1    EQU     ADPIA1+1    ;
ORA1    EQU     ADPIA1      ;
DDRA1   EQU     ADPIA1      ;
CRB1    EQU     ADPIA1+3    ;
ORB1    EQU     ADPIA1+2    ;
DDRB1   EQU     ADPIA1+2    ;
CRA2    EQU     ADPIA1+1    ;
ORA2    EQU     ADPIA2      ;
DDRA2   EQU     ADPIA2      ;
CRB2    EQU     ADPIA2+3    ;
ORB2    EQU     ADPIA2+2    ;
DDRB2   EQU     ADPIA2+2    ;
        ORG     $500        ; ---------------------------------------
        CLR     CRA1        ;}
        CLR     CRB1        ;} tous les registres de contrôle
        CLR     CRA2        ;} sont à zéro
        CLR     CRB2        ;}
        CLR     DDRA1       ; port A du PIA 1 en entrée
        CLR     DDRB1       ; port B du PIA 1 en entrée
        CLR     DDRA2       ; port A du PIA 2 en entrée
        LDA     #$FF        ; le port B PIA2 en sortie
        STA     DDRB2       ;
        LDA     #%00011101  ;}
        STA     CRA1        ;}
        STA     CRB1        ;}—initialisation de CRA et CRB
        STA     CRA2        ;}
        LDA     #%00101101  ;}
        STA     CRB2        ;}
        LDX     #$0100      ; init du vecteur
        STX     $FFF8       ; d'interruption IRQ ($FFF8 et $FFF9)
```


FEI : Ex01 Scrutation des interfaces

La scrutation consiste dans ce cas à venir tester tous les indicateurs d'états de chaque interface.

```
TABLE   EQU     $2000       ;
        LDX     #ORA1       ; port A du PIA1
        STX     TABLE       ;
        LDX     #CRA1       ;
        STX     TABLE+2     ;
        LDX     #$3000      ; adrs S/Prog 1 CA1
        STX     TABLE+4     ;
        LDX     #$3200      ; adrs S/Prog 1 CA2
        STX     TABLE+6     ;
        LDX     #ORB1       ; port B du PIA1
        STX     TABLE+8     ;
        LDX     #CRB1       ;
        STX     TABLE+10    ;
        LDX     #$3400      ; adrs S/Prog 1 CB1
        STX     TABLE+12    ;
        LDX     #$3600      ; adrs S/Prog 1 CB2
        STX     TABLE+14    ;
        LDX     #ORA2       ; port A du PIA2
        STX     TABLE+16    ;
        LDX     #CRB2       ;
        STX     TABLE+18    ;
        LDX     #$3800      ; adrs S/Prog 2 CA1
        STX     TABLE+20    ;
        LDX     #$4000      ; adrs S/Prog 2 CA2
        STX     TABLE+22    ;
```

On commence à travailler sur le port A du PIA 1

```
;------programme d'interruption scrutation des interfaces.
        ORG     $0100       ;
        LDX     TABLE       ; initialisation de la recherche
SUITE   LDA     [,X]        ; lecture de CRA1
        BPL     SUIVA       ; test sur CA1, on va à SUIVA si CA1 inactive
```

```
        BITA    #$1         ; l'interruption est-elle valide ?
        BEQ     SUIVA       ; branche à SUIVA si l'interrup est masquée
        JSR     [4,X]       ; chercher l'adresse du S/Prog de traitement
                            ; d'interruption due à Cx1 des PIA
SUIVA   ROLA                ; rotation à gauche → test sur CA2
        BPL     TEST        ; test sur CA2 on va à TEST si CA2 inactive
        ANDA    #$10        ; l'interruption est elle valide
        BEQ     TEST        ; branchement à TEST si l'interrup est masquée
        JSR     [6,X]       ; chercher l'adresse du sous-programme de
                            ; traitement d'interruption due à Cx2
```

On travaille ensuite sur les autres ports : N = nombre de ports (N=3 dans cet exemple)

```
TEST    LEAX    8,X         ; on change de port
        CMPX    #TABLE+8*N  ; test pour voir si la scrutation est terminée
                            ; si elle est en cours, on se branche à SUITE
        BNE     SUITE       ;
        RTI                 ;
```

FEI : EX01 Exécution du transfert

Le programme d'exécution du transfert consiste à lire le contenu du registre de sortie du port concerné puis à le transférer dans la mémoire.
Pour le S/Prog SP1 CA1, dont l'adresse de départ est contenue dans la table à l'adresse TABLE + 4.

```
;------programme de transfert exemple sur S/Prog 1 CA1
        ORG     $3000       ;
        LDA     ORA1        ; lecture port A du PIA1 (CRA7=0 pour le PIA1)
        STA     MEM         ; le contenu est transféré à l'adresse MEM
        RTS                 ; on retourne au sous-programme de scrutation
        END                 ;
```

FEI : Exemple 02 Interfaçage d'un clavier Hexadécimal

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Ex02 Sujet

C'est l'exemple de l'interfaçage d'un clavier hexadécimal à travers un PIA en utilisant la ligne IRQ|.
Le clavier est composé de 16 boutons poussoirs organisés en matrice 4x4.

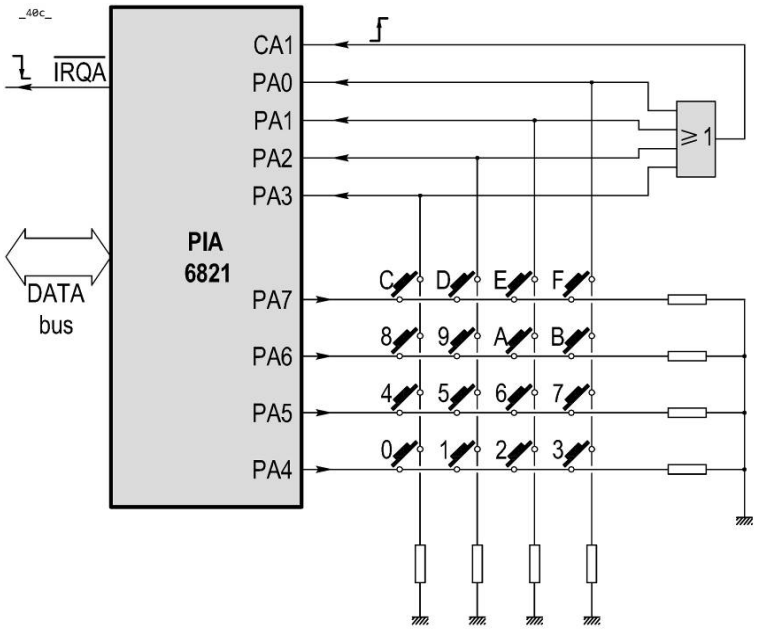
Il s'agit de générer dans l'accumulateur **A** le code hexadécimal correspondant à la touche enfoncée.
Exemple si c'est la touche B il faudra générer %0000 1011.

De plus la technologie des boutons poussoirs étant sommaire ils fourniront un effet de rebondissements qu'il faudra éliminer par logiciel.

Enfin, si l'on enfonce plusieurs touches simultanément, on souhaite appeler un sous-programme ERROR (non développé ici), il pourra par exemple commander l'écriture d'un message ERROR sur l'affichage associé.

retour au Sommaire | Index | Liens Rapides | Vecteurs d'Interruption

FEI : Ex02 Schéma



FEI : Ex02 Principe

Au repos, aucune touche n'étant enfoncée, l'entrée CA1 du PIA est au niveau Bas.
Si on presse une touche quelconque, un front montant sera envoyé sur la broche CA1, déclenchant le processus d'interruption.

A l'état d'initialisation du PIA les broches PA4 à PA7 sont en sortie et sont toutes au niveau Haut. Les broches PA0 à PA3 sont en entrée.

Lorsque l'on presse une touche du clavier, le programme de traitement d'interruption lira d'abord sur PA0 à PA3 le numéro de la colonne de la touche enfoncée.

Puis l'on inverse les fils d'entrée-sortie sur le port A, on lit alors sur PA4 à PA7 le numéro de rangée de la touche enfoncée tandis que les lignesPA0 à PA3 sont en sortie et au niveau Haut.

On remarque qu'à chaque touche, correspond un numéro de colonne et un numéro de rangée particuliers. C'est à partir de ces numéros que l'on peut trouver différents procédés pour le codage en hexadécimal de chacune des touches.

On a créée une table comportant 16 codes à l'aide des instructions FCB correspondant aux 16 touches du clavier.

Chaque code est composé :

- D'un premier chiffre pour le n° de colonne
- D'un deuxième chiffre pour le n° de rangée

Le code généré par la touche enfoncée est comparé à la table en partant de la première position de cette table et jusqu'à trouver l'identité des codes.

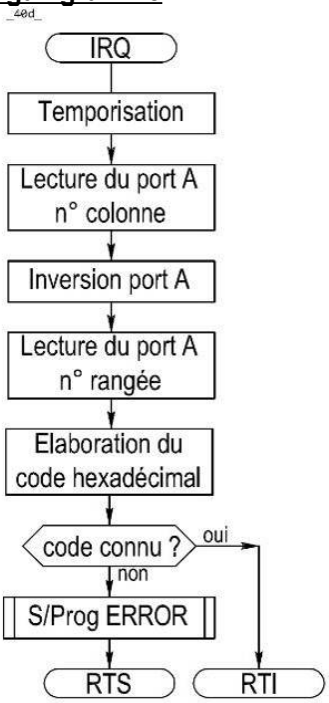
Un compteur initialisé à zéro, est incrémenté après chaque comparaison non satisfaisante.
Lorsqu'il y aura égalité ce compteur contiendra le code hexadécimal recherché.

Si l'on n'a pas trouvé le code après avoir balayé toute la table, il est vraisemblable que plusieurs touches ont été pressées simultanément, on appelle alors le sous-programme ERROR.

Ce programme étant un programme d'interruption, l'adresse de départ $1000 aura au préalable été chargé dans les vecteurs de l'interruption IRQ à savoir $FFF9 et $FFF8 par la séquence suivante :

```
LDX     #$1000
STX     $FFF8
```


FEI : Ex02 Organigramme



```
        ORG    $100        ;
ERROR   EQU    $1040       ;
PIACRA  EQU    $8005       ;
PIADRA  EQU    $8004       ;
PIAORA  EQU    PIADRA      ;
RAM0    EQU    $0000       ; 1ère adresse de la mémoire RAM.
RAM1    EQU    $0001       ; 2ème adresse de la mémoire RAM.
                           ;
; Initialisation du port A du PIA
; Interruption sur le front montant de CA1
; 4 broches LSB en entrée
; 4 broches MSB en sortie état Haut
                           ;
INIT    CLRA               ;
        STA    PIACRA      ;
        LDA    #$F0        ; %11110000
        STA    PIADRA      ;
        LDB    #$07        ; %00000111
        STB    PIACRA      ;
        STA    PIAORA      ;
        RTS                ;
                           ;
; programme d'interruption pour décodage
; de clavier hexadécimal
                           ;
        ORG    $1000       ;
;------tableau des codes du clavier
TAB     FCB    $18,$14,$12,$11
        FCB    $28,$24,$22,$21
        FCB    $48,$44,$42,$41
        FCB    $88,$84,$82,$81
                           ;
;------début de la tempo anti-rebond du clavier
TEMPO   LDA   #$04         ; = 2µs
        STA   RAM0         ; = 4µs
BCL4    LDA   #$02         ; = 2µs
        STA   RAM1         ;  = 4µs
BCL3    DEC   RAM1         ;                              } 6µs
        BNE   BCL3         ;  si [RAM1]≠0 alors BCL3 } 3µs   9µs x 02 = 18µs
        DEC   RAM0         ;                                 6µs
        BNE   BCL4         ;  si [RAM0]≠0 alors BCL4         3µs 18µs x 04 =72µs
                           ;
;------fin de la tempo
        LDB   PIAORA       ;  lecture colonne
        LDA   #$03         ;
        STA   PIACRA       ;  inversion port A
        LDA   #$0F         ;  %0000FFFF 4 MSB en entrée
        STA   PIADRA       ;            4 LSB en sortie
        STA   PIACRA       ;  état haut
        STA   PIAORA       ;
        ANDB  PIAORA       ;  lecture rangée
        LDX   #TAB         ;  code touche dans acc.B
        CLRA               ;  registre comptage
ENCORE  CMPB  0,X          ;
        BEQ   FIN          ;
        LDA   ,X+          ; pour incrémenter X de 1
                           ; (remplace l'instruction INX du vieux 6800)
        INCA               ;
        CMPA  #$10         ;  code non trouvé
        BEQ   ERROR        ;  aller à ERROR
        BRA   ENCORE       ;
                           ;
;------résultat dans acc.A,  Retour interruption
FIN     JSR   INIT         ;  réinitialisation PIA
        RTI                ;  pour nouveau codage
        END                ;
```

E/S : GENERALITES

Le µp6809 nécessite de communiquer avec l'extérieur :

Les périphériques d'entrées permettent de recevoir des données provenant par exemple de : clavier, disque …

Les périphériques de sorties permettent d'envoyer des données vers par exemple : écran, imprimante, disque….

Le µp6809 ne possède pas d'instruction spéciale pour les E/S.

Les entrées-sorties sont traitées comme de simples cases mémoires placées dans l'espace adressable du µp6809.

On peut donc utiliser ces cases mémoires particulières avec toutes les instructions du µp6809.

Il existe des composants ineffaçables avec le µp6809 tel que :

6821	PIA	interface parallèle programmable	(Peripheral Interface Adaptor)
6828	PIC	Contrôleur de priorité d'interruption	(Priority Interrupt Controler)
6829	MMU	Interface de gestion mémoire	
6830		ROM de 1 Ko par 8 bits	
6839		ROM mathématique	
6840	PTM	3 temporisateurs programmables	
6843	FDC	Contrôleur de disque souple simple densité	
6844	DMAC	Contrôleur d'accès direct en mémoire	
6845	CRTC	Contrôleur de visualisation	
6846	COMBO	ROM 2Ko port parallèle 8 bits (avec un Timer)	
6850	ACIA	Interface série asynchrone (RS 232)	
6852	SSDA	Interface série synchrone	
6854	ADLC	Contrôleur de transmission avec protocole	
6855	DMA		(Direct Memory Assces)
68488	GPIA	interface IEEE-488	
9364		Contrôleur d'écran	

Le PIA 6821 (Peripheral Interface Adapter) permet la liaison parallèle entre le µp6809 et le mode extérieur.
Il communique avec le µp6809 par l'intermédiaire des bus de données (8 bits), de certains fils du bus d'adresse provenant du 6809 et du bus contrôle.

6821 : Port A :

La charge maximale d'une entrée représente 1,5 charge TTL standard.
Les broches du Port A peuvent être lues par le µp6809 à la seule condition de respecter les niveaux de tensions.

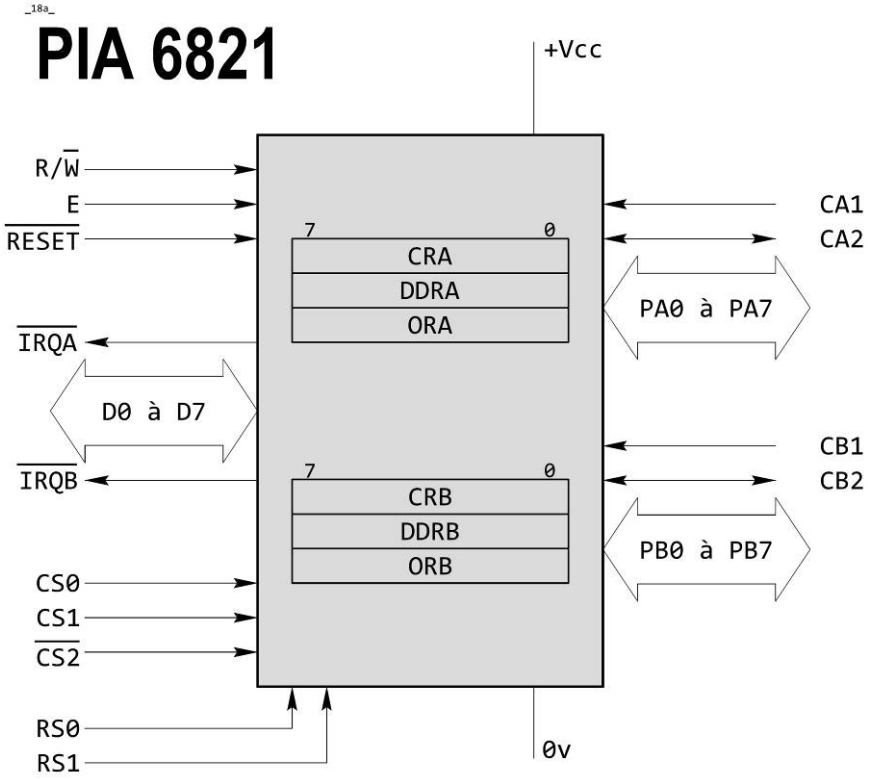
- U > 2 volts pour un 1 logique
- U < 0,8 volts pour un 0 logique

6821 : Port B :

Broches en logique trois états ce qui permet de les mettre en haute impédance lorsque le PIA n'est pas sélectionné.
Les sorties du port B (PB0 à PB7) sont compatibles TTL et peuvent fournir jusqu'à 1 mA sous 1,5 volts

6821 : Organisation Interne

retour au Sommaire | Liens Rapides | Index



Le PIA est divisé en 2 parties indépendantes A et B. Le 6821 possède :

- Un port de 8 bits bidirectionnel
- 2 lignes de contrôle par port (soit 4 lignes de contrôle au total)
- 3 registres internes par port (soit 6 registres pour l'ensemble du PIA)

Le PIA se comporte comme seulement 4 positions mémoire, bien qu'il comporte 6 registres internes.

Les registres DDRx et ORx ont la même adresse, le bit 2 du registre de contrôle CRA ou CRB permettra la distinction entre ces deux registres.

Il en résulte qu'avant de programmer les registres (DDRA ou DDRB) et (ORA ou ORB) il faudra programmer le registre CRA ou CRB, quitte à les modifier par la suite.

retour au Sommaire | Liens Rapides | Index

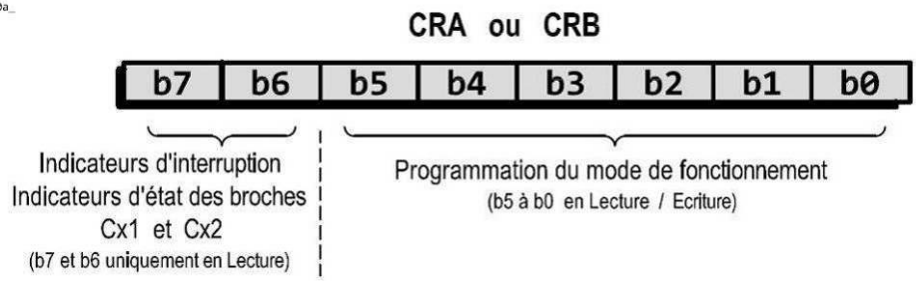
Pour voir la Sélection des registres internes

Le 6821 peut gérer la génération automatique du signal STROBE (validation de données) dans une application mettant en œuvre un protocole d'échange de données de type CENTRONICS.

6821 : Registres CRA et CRB (Control Register A et B) Registre de contrôle des ports A et B

Registres 8 bits qui contiennent les paramètres de fonctionnement :

_20a_

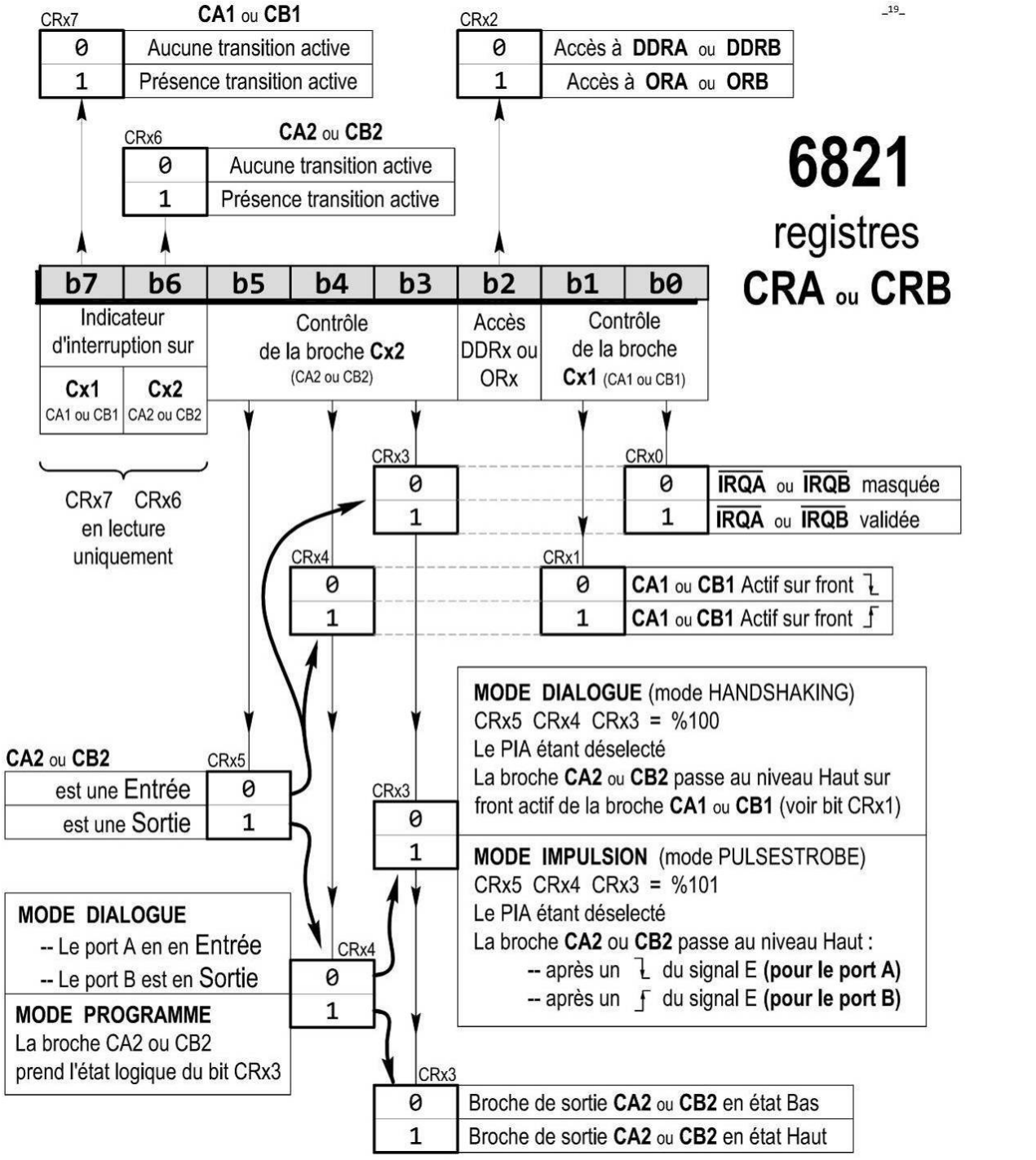


Ce registre concerne la programmation des broches spéciales CA1, CA2 pour le port A et CB1, CB2 pour le port à B.
b7 et b6 : Les indicateurs d'état accessibles en lecture, c'est le PIA qui mettra à jour ces bits en fonction de l'activité sur les broches Cx1 et Cx2.
b5 à b0 : Les paramètres de fonctionnement accessible en lecture et en écriture.

6821 : Vue complète du registre CRA ou CRB

(Convention d'écriture x sera mis pour A ou B)

_19_



6821 : Détail du registre CRA ou CRB

6821 : CRx0

Autorise ou non l'envoi d'une interruption sur IRQx|, lorsqu'on a reçu la bonne transition active sur la broche Cx1. Ce bit CRx0, valide la répercussion des interruptions vers le µp6809 par l'intermédiaire des signaux IRQA| (broche 38) et IRQB| (broche 37).

= 0 n'autorise pas IRQx|
= 1 autorise l'envoi d'une IRQx|

Si une interruption arrive sur la broche Cx1 alors que la répercussion vers le µp6809 est invalidée (cas ou le bit CRx0 est à 0) et si le programme positionne le bit CRx0 à 1
Alors le signal de sortie IRQx| (broches 38 ou 37) basculera à l'état Bas pour signifier au µp6809 qu'une interruption avait déjà été détectée.

retour au Sommaire | Liens Rapides | Index

6821 : CRx1

Précise le sens de la transition active attendu sur la broche Cx1
Les bits CRx1 (CRA1 et CRB1) permettent ainsi de sélectionner le front actif des entrées d'interruption des broches CA1 ou CB1.

= 0 on attend un front descendant sur la broche Cx1
= 1 on attend un front montant sur la broche Cx1

retour au Sommaire | Liens Rapides | Index | Sélection Registres Internes

6821 : CRx2 Adressage du 6821

Utilisé pour l'adressage, permet la distinction entre les registres DDRx et ORx

= 0 accès aux registres **DDRx** (DDRA ou DDRB) registres sens de transfert
= 1 accès aux registres **ORx** (ORA ou ORB) registres de données

Il faut néanmoins respecter la configuration des broches RS0 et RS1

PIA 6821		A15 à A2 (voir la logique de décodage)			A1	A0	bit n°2 du registre CRx		
		CS0	CS1	$\overline{CS2}$	RS1	RS0	bit CRA2	bit CRB2	Adresse
Partie A	**ORA**	1	1	0	0	0	**1**	0 ou 1	**Adr**
	DDRA	1	1	0	0	0	**0**	0 ou 1	**Adr**
	CRA	1	1	0	0	1	0 ou 1	0 ou 1	**Adr + 1**
Partie B	**ORB**	1	1	0	1	0	0 ou 1	**1**	**Adr + 2**
	DDRB	1	1	0	1	0	0 ou 1	**0**	**Adr + 2**
	CRB	1	1	0	1	1	0 ou 1	0 ou 1	**Adr + 3**

retour au Sommaire | Liens Rapides | Index

6821 : CRx5 CRx4 CRx3

Définissent et régissent le fonctionnement des broches **Cx2**, broches CA2 et CB2 (broches programmables en Entrée ou en Sortie).

Si CRx5 = 0 le mode de fonctionnement des lignes Cx1 et Cx2 est identique. C'est des entrées dont les états électriques sont repérés par les bits CRx7 et CRx6.

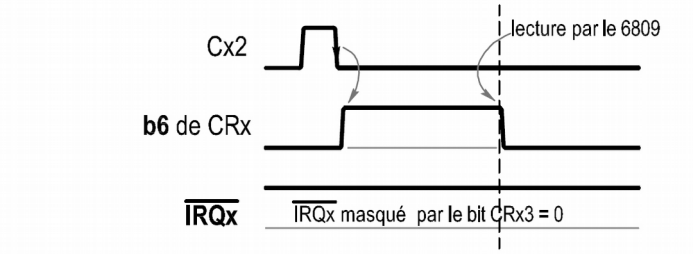
6821 : CRx5 = 0 la broche Cx2 est en ENTREE (en entrée d'interruption)

Les broches CA2 et CB2 sont respectivement analogues aux broches CA1 et CB1.

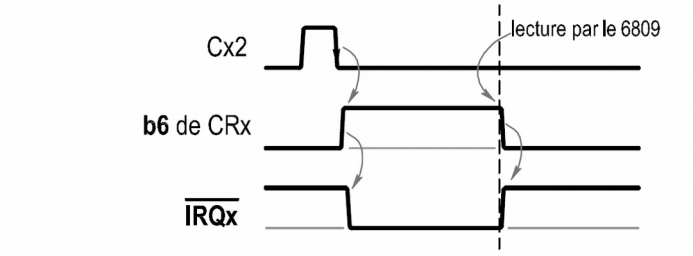
Les bits **CRx4 CRx3** : jouent un rôle identique aux bits CRx1 et CRx0 mais pour la broche Cx2 qui est programmé en entrée.

_2eb_

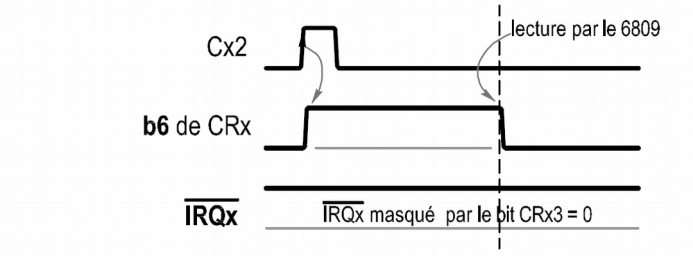
CRx
b5 b4 b3 = %000 **CA2 ou CB2 en Entrée** (b5 = 0)



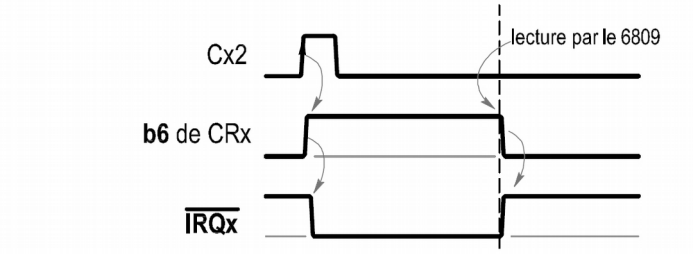
CRx
b5 b4 b3 = %001 **CA2 ou CB2 en Entrée** (b5 = 0)



CRx
b5 b4 b3 = %010 **CA2 ou CB2 en Entrée** (b5 = 0)



CRx
b5 b4 b3 = %011 **CA2 ou CB2 en Entrée** (b5 = 0)



retour au Sommaire | Liens Rapides | Index

6821 : CRx5 = 1 la broche Cx2 est en SORTIE (en sortie de commande)

La programmation en sortie de commande des broches CA2 et CB2 peut par exemple trouver son utilité lors du contrôle de transmission des données parallèle vers la périphérie.
La sortie de commande étant gérée comme signal de validation de données (STROBE).
Le fonctionnement côté A et côté B est différent

Trois MODES de Fonctionnement :

- 1er Mode HANDSHAKE ou mode Dialogue
- 2ième Mode SET-RESET ou mode Programmé
- 3ième Mode PULSE-STROBE ou mode Impulsion

retour au Sommaire | Liens Rapides | Index

6821 : 1er Mode HANDSHAKE ou mode Dialogue CRx5, CRx4, CRx3 = %100

Dans ce 1er mode les ports A et B ont un fonctionnement différent :

- Le port A travaille en ENTREE
- Le port B travaille en SORTIE

La broche Cx2 repasse à l'état Haut ⌐ Lorsque que la broche Cx1 recevra le prochain front actif.
La broche Cx2 passe à l'état Bas ¬

Pour le port A

Sur le premier front Descendant du fil E (Enable) qui suit un ordre de lecture du registre ORA.

- Passage de la broche CA2 à 0, sur le front négatif (front descendant) de l'horloge broche E qui suit une lecture du registre ORA.
- Passage de la broche CA2 à 1, quand le bit CRA7 est positionné à 1 lors d'une détection d'un front actif sur la broche CA1.

Pour le port B

Sur le premier front Montant de la broche E (Enable) qui suit un ordre d'écriture du registre ORB.
Le port B est adapté au dialogue en Sortie, sur le point de vue du matériel électronique, le port B est plus puissant que celui du port A.

- Passage de la broche CB2 à 0, sur le front positif (front montant) de l'horloge broche E qui suit une écriture du registre ORB.
- Passage de la broche CB2 à 1, quand le bit CRB7 est positionné à 1 lors d'une détection d'un front actif sur la broche CB1.

retour au Sommaire | Liens Rapides | Index

6821 : 2ième Mode SET-RESET ou mode Programmé CRx5, CRx4, CRx3 = %110 ou %111

La broche Cx2 suit l'état du bit CRx3
Il suffit donc de modifier le contenu du registre CRx (bit CRx3) pour changer l'état de la broche Cx2.

Si bit **CRx3 = 1** alors la broche Cx2 = 1
Si bit **CRx3 = 0** alors la broche Cx2 = 0

Quand CRx5, CRx4, CRx3 = %111, la broche Cx2 passe à 1 quand le bit CRx3 = 1 par écriture dans le registre CRx.

6821 : 3ième Mode PULSE-STROBE ou mode Impulsion CRx5, CRx4, CRx3 = %101

retour au Sommaire | Liens Rapides | Index

Pour le port A

La broche CA2 passe à l'état Bas sur le premier front négatif (front Descendant) du fil E (Enable) qui suit un ordre de lecture du registre ORA.

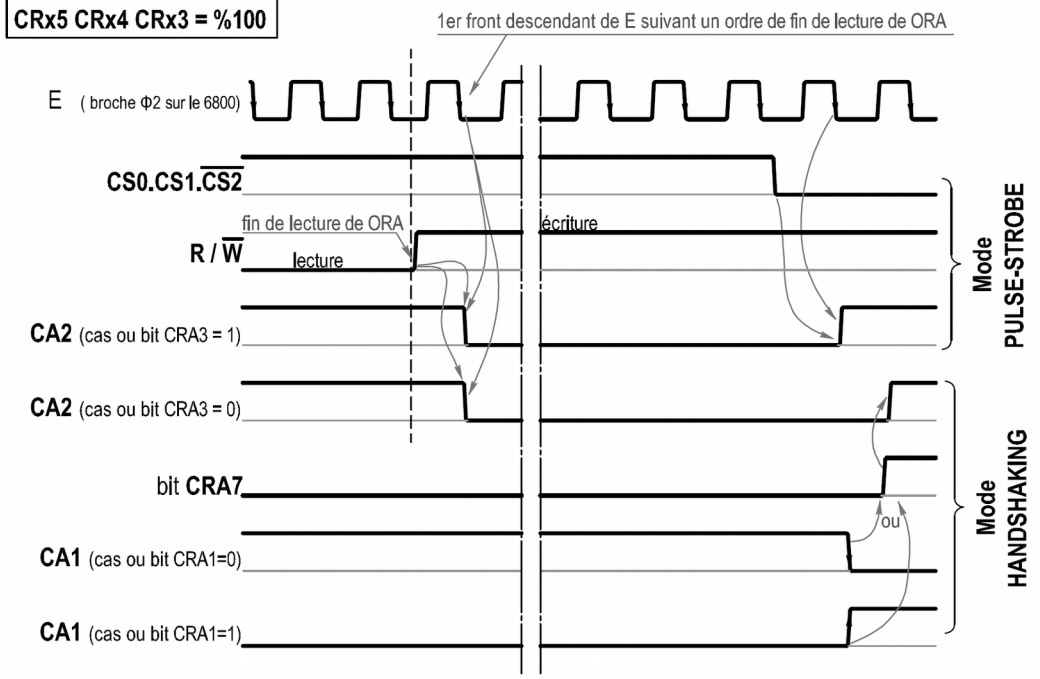
La broche CA2 passe à l'état Haut sur le front négatif (front descendant) alors que le µp6809 est désélectionné.

_20c_

6821 : le Port A (travaille en ENTREE)

CRx5 CRx4 CRx3 = %101
CRx5 CRx4 CRx3 = %100

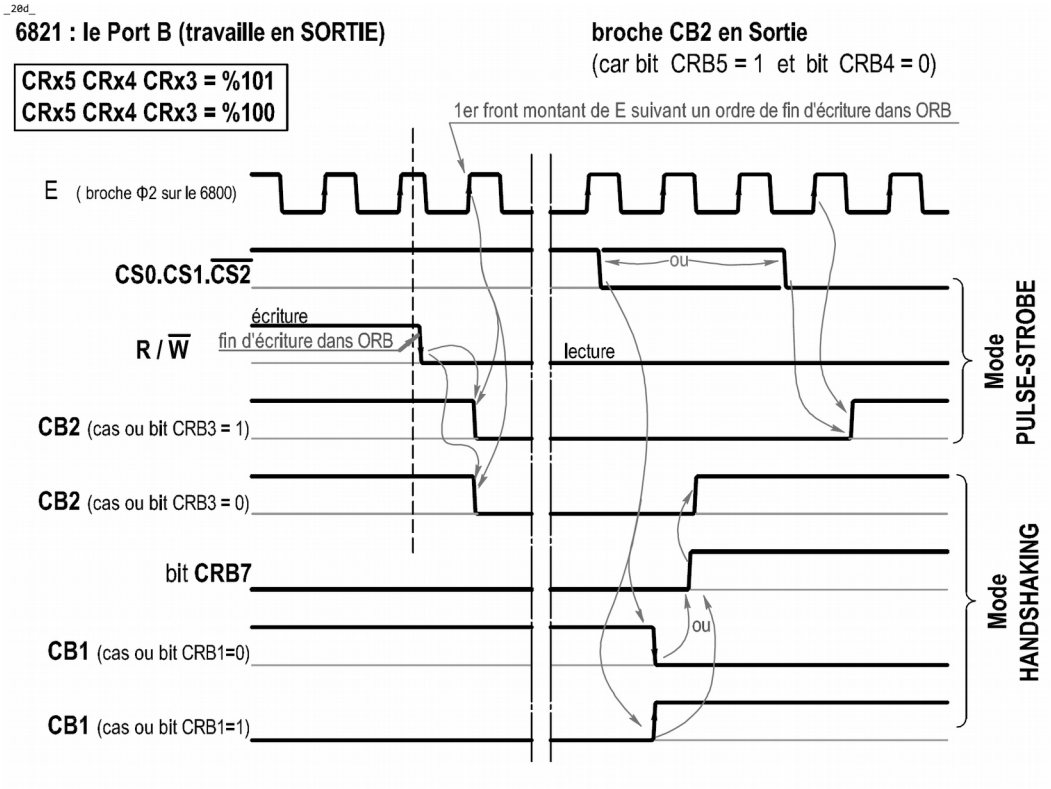
broche CA2 en Sortie
(car bit CRA5 = 1 et bit CRA4 = 0)



Pour le port B

La broche CB2 passe à l'état Bas ↓ sur le premier front Montant de la broche E (Enable) qui suit un ordre d'écriture dans le registre ORB.

La broche CB2 repasse à l'état Haut ↑ sur le premier front Montant de la broche E (Enable) alors que le 6821 est désélectionné (et donc lorsque que Cx1 recevra le prochain front actif).



6821 : Les bits CRx7 et CRx6

Ils sont des drapeaux internes d'interruption qui indique le passage à l'état Bas des broches IRQA| et IRQB| en fonction des 4 broches spéciales CA1, CA2 et CB1, CB2 et de leur programmation.

Ces indicateurs peuvent alors être mis en œuvre lors d'un masquage d'interruption.
Ils sont RAZ lors de chaque lecture du registre de données ORA ou ORB.
Après une telle réinitialisation, la prochaine interruption qui pourra être pris en compte devra intervenir au moins un cycle d'horloge E plus tard.

Après un RESET
CRx6 et CRx7 sont à 0
Les broches CA2 et CB2 sont en entrée
L'on n'autorise pas d'interruption
Les ports A et B sont en entrée

6821 : CRx6

Flag d'interruption lorsque les broches Cx2 sont en Entrée (en lecture uniquement) :
Si les broches Cx2 sont en Sortie alors ce bit CRx6 est forcé à 0 et non affecté par les transitions sur les broches Cx2.
= 0 par lecture du registre ORx
= 1 lorsqu'on reçoit la transition active attendue sur Cx2

6821 : CRx7

Flag d'interruption lorsque les broches Cx1 sont en Entrée (en lecture uniquement) :
Si les broches Cx1 sont en Sortie alors ce bit CRx7 est forcé à 0 et non affecté par les transitions sur les broches Cx1.
= 0 par lecture du registre ORx
= 1 lorsqu'on reçoit la transition active attendue sur Cx1

Attention ! : Ce bit CRx7 est mis à zéro après une lecture par le µp6809 : ←A VERIFIER !!!!

- Du registre de contrôle CRx
- Du registre de donnée

6821 : Programmation des broches CA1, CA2, CB1 et CB2 en ENTREE

retour au Sommaire | Liens Rapides | Index

Lorsque les broches **CA1** **CA2** **CB1** ou **CB2** sont programmé en entrée d'interruption, il est nécessaire qu'au moins une fois le signal E ait été à l'état haut pendant que le signal externe devant déclencher l'interruption était actif.

Les broches d'entrées **CA1** et **CB1**
sont initialisées par le contenu des bits CRx1 et CRx0.
Ces broches positionnent l'indicateur CRx7 quand elles sont actives.

Les broches d'entrées-sorties **CA2** et **CB2** (travaillent en entrées quand CRx5 est à 0),
sont initialisées par le contenu des bits CRx4 et CRx3.
Ces broches positionnent l'indicateur CRx6 quand elles sont actives.

6821 : Broches CAx CBx : Modes Automatiques

retour au Sommaire | Liens Rapides

Lorsque les lignes CB1 et CB2 sont programmées comme des sorties, les modes dits "automatiques" produisent des impulsions sur ces lignes sans intervention du µp6809 lors d'une opération d'entrée-sortie (envoi ou réception d'une donnée). En contre partie, la durée et polarité des impulsions échappement au contrôle du programmeur.

Les sorties Cx2 remplissent des fonctions différentes :

- La partie A set à la réception des données.
- La partie B sert à la transmission des données.

Dans chaque cas juste au dessus, les 2 lignes de contrôle de la partie correspondante (CA1, CA2 pour la partie A et CB1, CB2 pour la partie B) sont mobilisées pour l'appel-réponse.

6821 : Broches CAx CBx : Modes Manuels

Dans les modes dits "Manuels", la polarité de l'impulsion peut-être programmée par le bit CRx3.
La durée de l'impulsion est également réglable.

6821 : CA1 et CB1

retour au Sommaire | Liens Rapides | Index

Ces broches sont toujours en entrées.
Ces broches sont initialisées par les bits CRA1, CRA0 ou CRB1, CRB0.
Ces broches positionnent les bits indicateurs CRA7 ou CRB7 quand elles sont actives.

_17b_

CRx1	CRx0	$\overline{IRQx}$	CRx7
0	0	inhibée $\overline{IRQA}$ = 1 $\overline{IRQB}$ = 1	Mis à 1 sur ↓ appliquée sur CA1 ou CB1
0	1	autorisée	Mis à 1 sur ↓ appliqué sur CA1 ou CB1 simultanément $\overline{IRQA}$ ou $\overline{IRQB}$ ↓
1	0	inhibée $\overline{IRQA}$ = 1 $\overline{IRQB}$ = 1	Mis à 1 sur ↑ appliquée sur CA1 ou CB1
1	1	autorisée	Mis à 1 sur ↑ appliqué sur CA1 ou CB1 simultanément $\overline{IRQA}$ ou $\overline{IRQB}$ ↓

CA1 pour le port A CB1 pour le port B.
Actifs sur un front montant ou descendant suivant la programmation du bit CRx1 et le positionnement du bit CRx7.

CA1 et CB1 ne peuvent qu'être en entrées
Deux broches en entrée d'évènement plus spécialisées dans la détection de ce qui devrait être des interruptions.

Ces broches positionnent directement les indicateurs d'interruption des registres CRx, soit les bits CRx7.

6821 : CA2 et CB2

Ces broches peuvent travailler en entrées, quand les bits CRA5 ou CRB5 sont à 0.
Ces broches sont initialisées par les bits CRA4, CRA3 ou CRB4, CRB3.
Ces broches positionnent les bits indicateurs CRA6 ou CRB6.

6821 : Broche CA2

- Associée à la partie A
- Peut être programmée en entrée ou en sortie.
- Permet de contrôler l'échange à travers le PIA.
- Son rôle est déterminé en fonction des bits CRA3, CRA4 et CRA5
- Elle positionne en entrée, un flag d'interruption bit CRA6
- CA2 en entrée représente une charge TTL
- CA2 en sortie peut alimenter une charge TTL

6821 : Broche CB2

- Associée à la partie B
- Peut être programmée en entrée ou en sortie
- Permet de contrôler l'échange à travers le PIA
- Son rôle est déterminé en fonction des bits CRB3, CRB4 et CRB5.
- Elle positionne en entrée, un flag d'interruption bit CRB6
- CB2 en entrée est compatible TTL
- CB2 en sortie est compatible TTL et peut fournir jusqu'à 1mA sous 1,5 volt au système périphérique connecté.
- Les broches CA2 et CB2 peuvent être programmées par logiciel, en entrée d'interruption à l'image des broches CA1 et CB1.
- Les broches CA2 et CB2 peuvent aussi être programmées en sorties et être alors destinées à commander tout ou partie d'un système périphérique au même titre que l'une des 16 broches des ports A et B.

6821 : Programmation des broches CA2 et CB2 en SORTIE

Ces broches peuvent travailler en sorties, quand les bits CRA5 ou CRB5 sont à 1.
Le bit CRA4 ou CRB4 permet alors de choisir deux modes de fonctionnement :

6821 : CRA4 ou CRB4 = 0 Mode Dialogue

Dans ce mode et contrairement aux autres modes les ports A et B ont un fonctionnement différent :

Le port A travaille en ENTREE. La lecture du registre ORA entraîne le passage à l'état Bas de la broche CA2 (voir le bit CRA4 = 0)

Le port B travaille en SORTIE. L'écriture dans le registre ORB entraîne le passage à l'état Bas de CB2

Dans ces deux cas ci-dessus, c'est le contenu du bit CRA3 ou CRB3 qui détermine le mode de retour à l'état initial des broches CA2 ou CB2.

Si le bit CRA3 ou CRB3 = 0 Alors les broches CA2 ou CB2 repassent à l'état Haut quand un front actif est détecté sur la broche CA1 ou CB1.

Si le bit CRA3 ou CRB3 = 1 Alors les broches CA2 ou CB2 reprennent leur état initial quand le circuit est désélecté sur : Un front descendant pour CA2
Un front montant pour CB2,

6821 : CRA4 ou CRB4 = 1 Mode Programmé

Les broches CA2 et CB2 prennent l'état du bit CRA3 ou CRB3.

Par exemple pour CRA3 = 0 la broche CA2 est à l'état Bas, l'écriture de CRA3 = 1 entraîne le passage de CA2 à l'état Haut. Le fonctionnement est identique pour la broche CB2 avec CRB3.

6821 : Registres de direction de données DDRA et DDRB (Data Direction Register A et B)

Ils définissent (bit à bit) le sens de travail de chacune des broches des ports A et B. Accessible en lecture comme en écriture.

Px0 à Px7 des ports A et B **= 0** la broche est une ENTREE
= 1 la broche est en SORTIE

Exemple : définir le sens de travail de chacune des broches PA0 à PA7 et PB0 à PB7

= $00 toutes les 8 broches sont en Entrée
= $FF toutes les 8 broches sont en Sortie
Nota : un front actif sur l'entrée RESET initialise les ports en entrée.

Exemple : on veut que sur le port A Les broches PA0, PA1, PA2 et PA3 soient en entrée
Les broches PA4, PA5, PA6 et PA7 soient en sortie

Il faudra écrite dans DDRA la valeur $F0 soit %11110000

```
LDAA   #$F0
STA    PIADRA      ; à condition que bit CRA2 = 0
```

6821 : Registres de Sortie ORA et ORB (Output Register A et B)

Ils mémorisent les informations envoyées à l'extérieur sur les ports A et B.

En sortie :

Si une des broches PA0 à PA7 ou PB0 à PB7 est en sortie, ce sera le contenu du bit correspondant au registre ORA ou ORB qui influencera cette broche. Dans ce cas le µp6809 devra écrire dans le registre ORx.

Si on écrit une donnée dans le registre ORx, celle-ci se trouvera automatiquement sur les broches du port x (A ou B). Permettant de mémoriser une donnée lors d'une écriture.

Si les lignes de données fonctionnent comme des sorties, une lecture des registres de données est possible. Elle fournit en fait l'ancienne valeur.

En entrée :

Si une des broches PA0 à PA7 ou PB0 à PB7 est en entrée, ce sera l'état du signal présent sur la broche qui influencera le contenu du registre ORx. Dans ce cas le µp6809 devra lire dans le registre ORx.

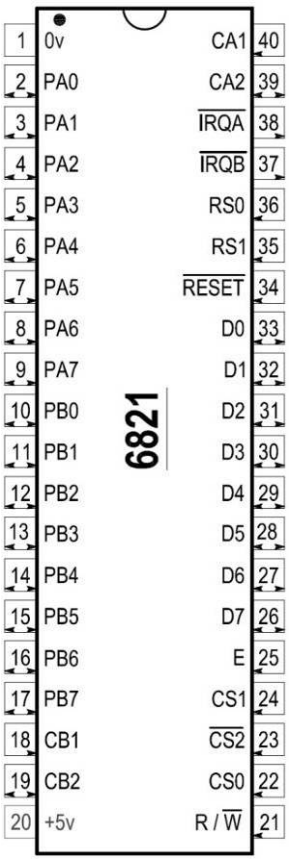
Si les broches Px0 à Px7 reçoivent une donnée alors on aura la possibilité de lire le registre ORx pour récupérer la donnée représentant les états du port.

Les données présentes sur les ports A et B sont prises en compte par une lecture de ORA et ORB mais ne sont pas mémorisées dans ces registres. Il faut donc que ces données soient présentes suffisamment longtemps pour être lue.

Si les lignes de données fonctionnent comme des entrées, une écriture dans les registres de données n'a pas de sens.

6821 : Organisation Externe

6821 : Brochage



6821 : Liaison avec le bus de données du 6809

De **D0 à D7**, ces 8 lignes sont bidirectionnelles directement reliées au bus de données du µp6809.

Elles assurent l'échange des données. La circuiterie de sortie des broches D0 à D7 est à trois états.
Lorsque le composant n'est pas sélectionné CS0.CS1.CS2| = 0 alors ces 8 broches ne sont pas utilisées, elles sont dans l'état haute impédance.
Le sens de transfert des informations est donné par l'état de la broche R/W|

6821 : Liaison avec le bus d'adresses du 6809

6821 : Broches CS0, CS1 et CS2 | (Chip Select Line) Sélection de boîtier :

Permettent l'adressage physique du boîtier, si ces trois lignes CS0. CS1. CS2| = %110 alors le 6821 est sélectionné.

On appliquera les broches adresses haute A15, A14, A13 sur les entrées CS0. CS1. CS2| lorsqu'il s'agit d'un petit système. On pourrait également mettre A12, A11, A10 pour un autre plan d'adressage.

L'état des signaux CS0, CS1 et CS2| doit être stabilisé pendant toute la durée du niveau Haut du signal E lorsqu'un transfert d'informations doit être effectué.

CS0, CS1, CS2| connectées directement au µp6809 ou en passant par un circuit assurant un décodage d'adresse.

6821 : Broches RS0 et RS1 (Register Select Line)

Permettent de sélectionner (d'adresser) les registres internes (4 positions en mémoire).
On leur appliquera nécessairement les broches d'adresse base A0 et A1.

Sélection de registre RS0, RS1 (broches en entrée). Ces 2 broches sont utilisées conjointement aux registres de contrôles CRA et CRB afin d'accéder à la totalité des 6 registres internes du 6821.

Ces deux broches RS0 et RS1 ne permettant théoriquement que l'adressage de quatre registres. Un bit de chacun des registres de contrôle CRx assure l'aiguillage vers les 2 registres manquant au décodage primaire assuré par les broches RS0 et RS1.

L'état des signaux RS0 et RS1 doit être stabilisé pendant toute la durée du niveau Haut de la broche E lorsqu'un transfert d'informations entre le 6821 et µp6809 est à mettre en œuvre.

6821 : Liaison avec le bus de contrôle du µp6809

6821 : Broche E (Enable)

Signal d'activation des échanges, assure la synchronisation des transferts d'information entre le 6821 et le µp6809, la synchronisation de tous les autres signaux du composant sont réalisés à partir des seuls fronts montant ou descendant de ce signal E.

Reçoit un signal d'horloge, généralement connecté à la broche E du µp6809 (anciennement **Φ2** pour le 6800) pour assurer des échanges synchrones.

6821 : Broche RESET| broche en entrée

Sensible au niveau bas, reçoit le signal RESET général de la carte qui initialise le PIA.
Cette broche doit être à l'état haut au moins une micro secondes avant la première sélection du PIA et donc avant d'adresser le PIA.

Remise à zéro des registres internes et de ce fait, programme en entrée toutes les broches des ports A et B ainsi que les 2 broches spéciales CA2 et CB2 (sachant que CA1 et CB1 ne peuvent qu'être en entrées).

De plus durant le RESET, les interruptions sont masquées, ainsi le µp6809 ne risque pas d'être interrompu par inadvertance.

Lors d'un RESET

- Tous les bits du registre de contrôle sont à 0
- Les registres adressables immédiatement après le RESET sont les registres de contrôle CRx et le registre de sens de transfert de données DDRx
- Les interruptions IRQA| et IRQB| sont inhibées.

Après un RESET

- Les bits b7 et b6 du registre CRx sont à 0
- Les broches Cx2 sont en Entrée
- On n'autorise pas d'interruption
- Les ports A et B sont en entrée

Le fait que toutes les broches en liaison avec la périphérie sont programmées en entrée, cela sécurise les organes périphériques, il ne risque pas d'être actionnés de manière intentionnelle.

6821 : Broche R/W |: (Read / Write |)

Lecture / Ecriture. Fixe le sens des transferts.

R/W | = 0

Pour un transfert du µp6809 → 6821, écriture des informations dans le 6821, le bus de données du PIA est en entrée (si le boîtier est sélectionné et que la broche E est à l'état haut).

R/W | = 1

Pour un transfert du 6821→ µp6809, lecture des informations du 6821, le bus de données du PIA est en sortie (si le boîtier est sélectionné et que la broche E est à l'état haut).

6821 : Broches IRQA | et IRQB |

(IR... = Interrupt Request) broches en sortie

2 lignes d'interruption supportant le OU câblé.
Reliées à IRQ|, FIRQ| ou NMI| du µp6809, ces lignes permettent d'interrompre l'exécution du programme en cours et d'appeler un sous-programme de traitement d'interruption.

L'interruption peut se faire soit directement, soit par l'intermédiaire d'une circuiterie de priorité d'interruption.

Ce type de signal peuvent être traditionnellement être connecté en "OU câblé"; ils peuvent donc être reliés les uns aux autres et même à d'autres signaux de même type issus d'autres composants (ces sorties sont à drain ouvert).

Chaque ligne IRQ est associée à un port : IRQA| au port A et IRQB| au port B et aussi respectivement aux bits 6 et 7 des registres de contrôles CRA et CRB.

6821 : Liaison avec la périphérie : lignes de transfert

6821 : Broches PA0 à PA7

Permettent de transmettre ou de recevoir des informations sur 8 bits.
Suivant la programmation du registre DDRA :

- **= 0** la broche est en Entrée
- **= 1** la broche est en Sortie

Exemple : Si le bit DDRA3=0 la broche PA3 sera une entrée

6821 : Broches PB0 à PB7

Permettent de transmettre ou de recevoir des informations sur 8 bits.
Suivant la programmation du registre DDRB :

- **= 0** la broche est en Entrée
- **= 1** la broche est en Sortie

Contrairement au port A, ces 8 lignes sont en logique 3 états
Quand le port B est mis en entrée, il est vu comme de la haute impédance. Il n'y a pas de résistances de Pull Up comme sur le port A.
Exemple : Si le bit DDRB5=1 la broche PB5 sera une sortie

6821 : Fonctionnement

6821 : Transfert d'une donnée Périphérie --> µp6809

Exemple : En mettant le registre DDRA = $00, alors PA0-PA7 toutes les broches du port A sont en ENTREE.
La donnée disponible sur le port A est transmise à l'amplificateur de bus de données.
Cette donnée ne transite pas par ORA, il n'y a donc pas de mémorisation des données en entrée.
Ce transfert se fait sous le contrôle du registre CRA.
Un signal actif sur CA1 sera validé sur IRQA| si et seulement si le contenu du registre de contrôle le permet.
La fonction est identique pour la partie B

6821 : Transfert d'une donnée µp6809 --> Périphérie

Exemple : En mettant le registre DDRB = $FF, alors PB0-PB7 toutes les broches du port B sont en SORTIE.
La donnée disponible sur le bus de donnée du µp6809 est chargée dans le registre de sortie B.
La donnée est disponible tant qu'une nouvelle écriture n'est pas intervenue.

6821 : Sélection des registres internes

Le µp6809 accède au registre interne du PIA par l'intermédiaire des lignes de sélection de boîtier CS0, CS1 et CS2| et par la sélection de registre RS0 et RS1.

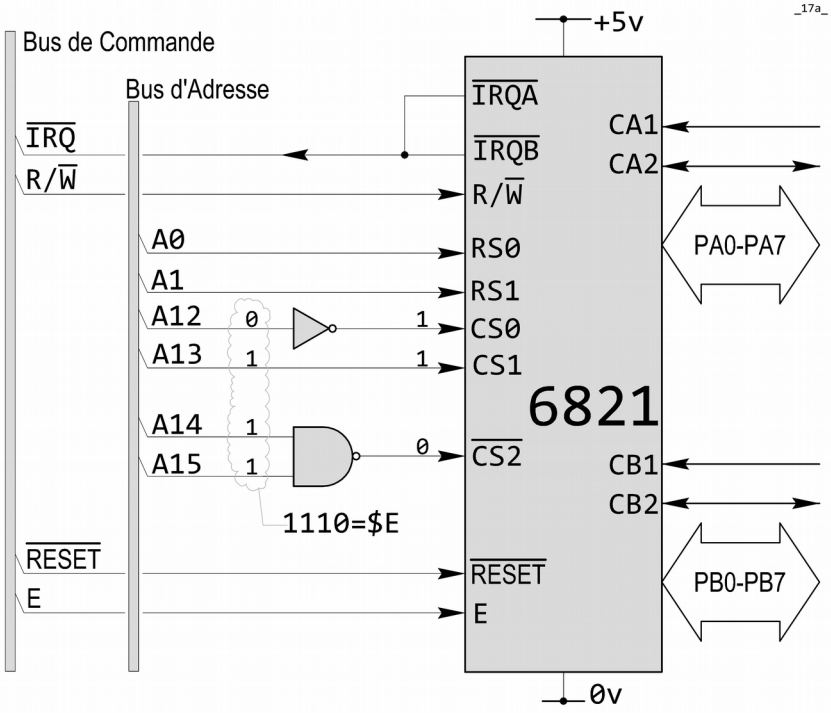
6 registres internes pour 4 positions mémoires. C'est le rôle du bit CRx2 qui permettra de distinguer les registres DDRx et ORx.

Autrement dit, pour adresser les 6 registres avec 2 broches RS0 et RS2, on utilise en plus l'état du bit CRx2.

Adressage du 6821

Voir également le bit CRA2 et CRB2, sélection des registres et adressage du 6821

Les broches A2 à A15 sont reliées à une logique de décodage qui détermine l'adresse de base du PIA.
Les broches A0 et A1 sont reliées à RS0 et RS1 pour accéder aux adresses Adr, Adr + 1, Adr + 2, Adr + 3



Exemple d'un 6821 implanté en $E000

_17c_

A15	A14	A13	A12									A1	A0		
1	1	1	0									0	0	$E000	DDRA / ORA
1	1	1	0									0	1	$E001	CRA
1	1	1	0									1	0	$E002	DDRB / ORB
1	1	1	0									1	1	$E003	CRB
$E				$0				$0							

retour au Sommaire | Liens Rapides | Index

6821 : Méthode de programmation du PIA

Compte tenu du fait que les registres DDRx et ORx ont la même adresse, il faut programmer le registre CRx afin de pouvoir y accéder.

6821 : Exemple de programme 01, Port A en Entrée, Port B en Sortie

Dans cet exemple, on n'utilise pas les lignes de commandes, et il n'y a pas eu de RESET.
On désire lire la donnée présente sur le port A et l'afficher sur le port B

```
        CLRA            ; raz de A   (A=$00)
        STA    PIACRA   ; $00->PIACRA
        STA    PIACRB   ; $00->PIACRB
        STA    PIADRA   ; port A en entrée
        COMA            ; A passe de $00 à $FF
        STA    PIADRB   ; $FF->PIADRB   port B en sortie
        STA    PIACRA   ; $FF->PIACRA
        STA    PIACRB   ; $FF->PIACRB   b2 de CRB = 1
ENCORE  LDA    PIAORA   ; lecture port A
        STA    PIAORB   ; affichage port B
        BRA    ENCORE   ; branch.toujours vers ENCORE
        END             ;
```

retour au Sommaire | Liens Rapides | Index

6821 : Exemple de programme 02, Utilisation des lignes de commande CA1 et CB2

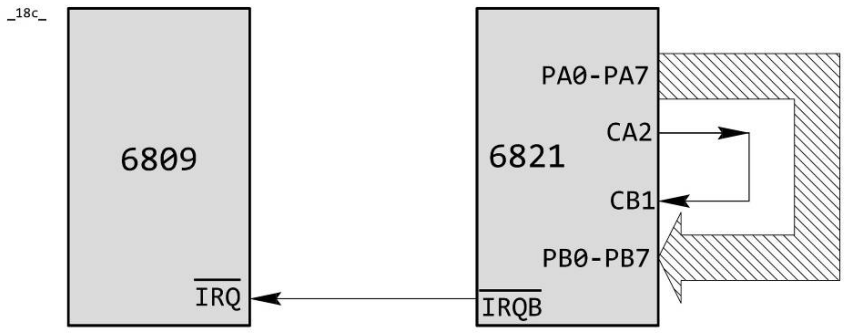
Dans cet exemple, on effectue la lecture du port A après avoir eu un front descendant sur CA1.
Puis on écrit sur le port B après avoir eu un front montant sur CB2.
Le port B recopie le port A.

```
        CLRA            ; raz de A   (A=$00)
        STA   PIACRA    ; $00->PIACRA
        STA   PIACRB    ; $00->PIACRB
        STA   PIADRA    ; port A en entrée
        COMA            ; A passe de $00 à $FF
        STA   PIADRB    ; $FF->PIADRB   port B en sortie
        LDA   #$04      ; $04 = %0000 0100
        STA   PIACRA    ; $04->PIACRA
        LDA   #$14      ; $14 = %0001 0100
        STA   PIACRB    ; $14->PIACRB
                        ;
ATCA1   LDA   PIACRA    ;  | attente front descendant de CA1
        BPL   ATCA1     ;  |_test de b7 de CRA
        LDA   PIAORA    ; lecture port A
                        ;
ATCB2   LDB   PIACRB    ;  | | attente front montant de CB2
        ROLB            ;  | | décalage du b6 en position 7
        BPL   ATCB2     ;  | |_test de b6 de CRB, branch si > 0
        STA   PIAORB    ; lecture port B
        LDA   PIAORB    ; lecture fictive de ORB pour raz de b6
        BRA   ATCA1     ; branche toujours vers ATCA1
        END             ;
```

retour au Sommaire | Liens Rapides | Index

6821 : Exemple de programme 03, Simulation d'un dialogue entre 2 microprocesseurs

On désire simuler un dialogue entre deux microprocesseurs par l'intermédiaire d'un PIA.
On utilise pour cela un seul PIA connecté sur une structure à base de µp6809.
Le port A travaille en sortie, et est reliée au port B programmé en entrée.
La ligne de dialogue CA2 programmé en sortie est reliée à CB1, ceci assurant la synchronisation du transfert.
La ligne IRQB| est reliée à IRQ| du µp6809.



Le programme se décompose en trois parties indépendantes :

- Programme d'initialisation du PIA
- Sous-programme de sortie d'un octet vers le port A
- Sous-programme d'interruption

6821 : Exemple de programme d'initialisation du PIA

retour au Sommaire | Liens Rapides | Index

Il consiste à initialiser les registres internes du PIA,
pour obtenir un mode de fonctionnement répondant aux caractéristiques ci-dessus :

- Le port A en sortie DDRA = $FF
- Le port B en entrée DDRB = $00

```
CRA   x x 1 1   1 1 x x   = $3C  =  %0011 1100
          | |   | | |_|__indifférent (CA1 non utilisé)
          | |   | |_____accès direct à ORA
          | |   |_______CA2 est à l'état Haut
          | |___________CA2 travaille en mode programmable
          |_____________CA2 est une sortie

CRA   x x 1 1   0 1 x x   = $34  =  %0011 0100
          | |   | | |_|__indifférent (CA1 non utilisé)
          | |   | |_____accès direct à ORA
          | |   |_______CA2 est à l'état Bas
          | |___________CA2 travaille en mode programmable
          |_____________CA2 est une sortie
```

```
CRB   x x x x   x 1 0 1   = $05  =  %0000 0101
          | |   | | | |__interruption IRQB validé
          | |   | | |____CB1 active sur front descendant
          | |   | |______accès direct à ORB
          | |   |____________indifférent CB2 non utilisé
          | |________________indifférent CB2 non utilisé
          |____________________indifférent CB2 non utilisé
```

6821 : Le programme d'initialisation

retour au Sommaire | Liens Rapides | Index

Doit également contenir l'initialisation de la pile et celle des vecteurs d'interruptions.

```
;------Définition des registres de programme-----------
       ORG  $1000    ;
PILE   EQU  $3FFF    ;
SPIRQ  EQU  $2000    ; Sous Prog d'interruption
ADPIA  EQU  $F400    ; Adresse du PIA
ORA    EQU  ADPIA    ; ---------port A    $F400
DDRA   EQU  ADPIA    ;                    $F400
CRA    EQU  ADPIA+1  ;                    $F401
ORB    EQU  ADPIA+2  ; ---------port B    $F402
DDRB   EQU  ADPIA+2  ;                    $F402
CRB    EQU  ADPIA+3  ;                    $F403
;------Initialisation du PIA-------------------
       LDS  #PILE    ; init du pointeur de la pile
       LDX  #SPIRQ   ; init du vecteur d'interruption
       STX  $FFF8    ; vecteur pour IRQ | FFF8 et FFF9
       CLRA          ; RAZ de A
       STA  CRA      ; accès à DDRA bit CRA2=0
       STA  CRB      ; accès à DDRB bit CRB2=0
       STA  DDRB     ; port B en Entrée car  A=$00->DDRB
       COMA          ; pour A passage de $00 à $FF
       STA  DDRA     ; port A en Sortie car  A=$FF->DDRA
       LDA  #$3C     ; %0011 1100 voir ci-dessus
       STA  CRA      ; init de CRA
       LDA  #$05     ; %0000 0101 voir ci-dessus
       STA  CRB      ; init de CRB
```

retour au Sommaire | Liens Rapides | Index

6821 : A la fin du programme d'initialisation

le PIA est prêt à transférer des octets du port A vers le port B

Il faut écrire l'octet à transmettre dans le registre ORA, puis activer CA2 pour lancer la procédure de transfert.

```
;------Routine de transfert d'un octet dans le port A-----------
;------Programme principal
       LDA  OCTET    ; charg registre ORA avec l'octet à transférer
       STA  ORA      ;
       LDA  #$34     ; %00110100 CA2 passe à 0 ça active en même temps CB1
                     ; CA2 en sortie, mode SET-RESET
                     ; CA2 passe de H à B ==> front descendant sur CB1
       STA  CRA      ;
       NOP           ; }--attente de prise en compte du front
       NOP           ; }                    actif sur CB1
       NOP           ; }
       BRA  *        ; bouclage du prog sur lui-même puis branch
                     ; à $2000 (routine traitement d'interruption)
                     ; si interruption IRQ voir SPIRQ
```

6821 : Cette routine de transfert est interrompue

par l'interruption IRQ due au front actif sur CB1

retour au Sommaire | Liens Rapides | Index

Le microprocesseur interrompt le déroulement de la routine de transfert, puis exécute le programme d'interruption approprié. Dans le cas où plusieurs périphériques sont connectés sur la ligne IRQ du microprocesseur, il faut tester les bits d'état de ces périphériques, afin de déterminer l'origine de l'interruption.

```
;------Routine d'interruption-------------------
       ORG  $2000    ;
       LDA  CRB      ; test du bit CRB7
       BITA #$80     ; %1000 0000
       BNE  GISSE    ;
       NOP           ; }--tester d'autre PIA
       NOP           ; }
       NOP           ; }
```

On teste ensuite, les différents registres d'état des autres périphériques, le niveau de priorité des uns par rapport aux autres est donné par le logiciel. La routine GISSE permet de lire le port B

```
       NOP           ; }
GISSE  LDA  ORB      ; lecture du port B
       STA  MEMOIR   ;
       LDA  #$3C     ; %0011 1100
```

```
                        ; CA2 en sortie, Accès à ORA
                        ; CA2 passe de B à H
                        ; CA2 repasse à l'état initial ainsi que CB1
              STA  CRA  ;
              RTI       ; retour au programme principal
                        ;
OCTET         FCB  $12  ; origine ?
MEMOIR        RMB  1    ; destination ?
              END       ;
```

La lecture du port B, ramène l'indicateur d'état bit CRB7 à sa valeur initiale.

retour au Sommaire | Liens Rapides | Index

6821 : Exemple de programme 04, Génération d'un système d'impulsion corrélées

On désire générer 4 signaux du type de la figure ci-dessous. Chaque signal est décalé d'un quart de cycle par rapport au signal adjacent. On peut facilement les programmer symétriques ou non symétriques.

Une solution consiste à décaler un registre 8 bits comportant un groupement de 4 bit à 1, les autres étant à 0, comme dans la figure ci-dessous.

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	1
1	1	0	0	0	0	1	1
1	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	1	1	1	0	0

Il suffit alors de prélever les signaux sur les sorties paires ou impaire de la partie A ou B.

Pour régler la période de globale, on modifie le délai entre deux décalages. L'intérêt de cette solution réside en l'absence d'un test systématique à la fin de chaque période.

```
                        ;************************************************
                        ;    Génération de 4 signaux décalés les uns par
                        ;    rapport aux autres d'un quart de cycle
                        ;************************************************
8000                          ORG    $8000       ;
                        ;------Adresse des registres de la partie B
          ED0C          RCRB    EQU    $ED0C       ;  Reg. de contrôle partie B
          ED08          RDDRB   EQU    $ED08       ;  Reg. sens transfert partie B
          ED08          ROR_B   EQU    $ED08       ;  Reg. Donnée partie B
8000 7F   ED0C                CLR    RCRB        ;  adr reg. sens transf CRB2= 0
8003 86   FF                  LDA    #$FF        ;  Déf. Toutes lignes B comme sorties
8005 B7   ED08                STA    ROR_B       ;
8008 86   04                  LDA    #%00000100  ; adrs reg.données CRB2 = 1
800A B7   ED0C                STA    RCRB        ;
                        ;------Charger mot définissant impulsions dans reg. données
800D 86   F0                  LDA    #%11110000  ;
800F B7   ED08                STA    ROR_B       ;
                        ;------Boucle sans fin de génération d'impulsions
8012 8D   05    8019  GNIMPU  BSR    DELAI       ;  Délai réglable
8014 79   ED08                ROL    ROR_B       ;
8017 20   F9    8012          BRA    GNIMPU      ;
                        ;------S/P DELAI permettant de régler la fréquence
8019 86   64           DELAI  LDA    #100        ;  Valeur de réglage
801B 4A                       DECA               ;
801C 26   FD    801B          BNE    DELAI+2     ;  si A =/= 0 décrémenter A
801E 39                       RTS                ;  fin du S/P DELAI
```

retour au Sommaire | Liens Rapides | Index

6821 : Quelques explications sur le programme ci-dessus

On peut envisager un deuxième système de signaux dont les différentes composantes ne sont pas liées entre elles. Dans ce cas, la solution consiste à ranger au préalable des mots dans une table qui seront ensuite transmis un par un dans le registre de données du 6821.

Les différents mots inscrits dans un tableau auquel le µp6809 peut accéder rapidement en mode indexé.

Le phénomène est cyclique, la fin du tableau correspond à la fin d'un cycle.
Le µp6809 doit détecter cette éventualité et se reporter de nouveau au début du tableau.

Quelques cycles machines nécessaires à l'actualisation de l'index doivent être comptabilisés dans le décompte des durées des impulsions.

```
                    ;***********************************************
                    ;    Génération d'un système de signaux dont les formes
                    ;    sont définies par les valeurs inscrites dans un tableau
                    ;***********************************************
8000                        ORG    $8000       ;
                    ;------Adresse des registres de la partie B
          ED0C      RCRB    EQU    $ED0C       ;  Reg. de contrôle partie B
          ED08      RDDRB   EQU    $ED08       ;  Reg. sens transfert partie B
          ED08      ROR_B   EQU    $ED08       ;  Reg. Donnée partie B
8000 7F   ED0C              CLR    RCRB        ;  adr reg. sens transf CRB2= 0
8003 86   FF                LDA    #$FF        ;  Déf. Toutes lignes B comme sorties
8005 B7   ED08              STA    ROR_B       ;
8008 86   2C                LDA    #%00101100  ; CRB2 = 1 adressage reg. Données.
                                                ;  Brève impulsion sur la sortie CB2
                                                ;  à chaque écriture
800A B7   ED0C              STA    RCRB        ;
                    ;------Initialisation à chaque début de cycle
800D 108E 8100      GNIMPU  LDY    #$8100      ;  adrs début tableau
8011 C6   41                LDB    #65         ;  Nb de découpages élémentaire à
                                                ;  l'intérieur d'un cycle. Lg tableau
8013 A6   A0        VALSVT  LDA    ,Y+         ;  charger une valeur tableau,
                                                ;  VALeur SuiVanTe
8015 B7   ED08              STA    ROR_B       ;
8018 8D   05   801F         BSR    DELAI       ;  Délai réglable
801A 5A                     DECB               ;  Fin Tableau ?
801B 26   F6   8013         BNE    VALSVT      ;  sinon, valeur suivante
801D 20   EE   800D         BRA    GNIMPU      ;  si oui, revenir début tableau
                    ;------S/P DELAI permettant de régler la durée de
                    ;------l'impulsion élémentaire
801F 8E   07D0      DELAI   LDX    #2000       ;  valeur de réglage
8022 30   1F                LEAX   -1,X        ;
8024 26   FC   8022         BNE    DELAI+3     ;  si (X) =/= 0 décrémenter (X)
8026 39                     RTS                ;  Fin S/P DELAI
```

retour au Sommaire | Liens Rapides | Index

6821 : Quelques explications sur le programme ci-dessus

Le deuxième exemple est plus adapté ou séquenceur électronique, les durées des impulsions sont beaucoup plus longues. Pour obtenir des durées doubles, triples, il suffit d'appeler plusieurs fois le sous-programme DELAI.

Les séquences présentées s'effectuent indéfiniment. Pour les arrêter, il faut utiliser les réserves propres du système utilisateur, c'est-à-dire les interruptions manuelles.

Pour s'adresser au registre sens de transfert RDDRB nous employons CLR RCRB qui agit en fait sur 6 bits CRB0 à CRB5.

Comme les lignes CB1 et CB2 ne sont pas affectées à ce niveau, cette instruction est tolérable.

Dans le premier exemple, seules les sorties impaires sont utilisées.
Si l'on écrit LDA #%01010101 au lieu de LDA #$FF, les sorties paires peuvent éventuellement jouer le rôle d'entrées.

Dans le second exemple, à chaque écriture dans le registre de données de la partie B, une impulsion est générée automatiquement sur la sortie CB2.

Dans le premier exemple, l'instruction qui permet de changer d'état est ROL ROR_B. Elle implique une lecture de l'ancienne valeur puis une écriture de la nouvelle valeur.

Son emploi dans ce cas particulier évite l'écriture des séquences de branchement à la fin de chaque cycle.

Ces séquences rendent le système d'impulsion non symétrique, surtout aux fréquences élevées.
La fréquence la plus élevée sera celle obtenu en enlevant le branchement vers DELAI.
Le sous-programme DELAI doit laisser à l'organe électromécanique suffisamment de temps pour commuter.

retour au Sommaire | Liens Rapides | Index

6821 : Exemple de programme 05, Transmission et réception de données en mode parallèle

Un système processeur-interface PIA peut-être programmé en transmetteur ou récepteur de données en mode parallèle.

Dans une transmission ou réception de données par interface série ACIA 6850, les bits SR0 et SR1 informe le µp6809 sur la disponibilité du récepteur ou du transmetteur. Dans le cas d'une transmission parallèle les bits CRx7 jouent à peu près le même rôle. Ils sont liés aux états électriques des lignes Cx1.

Dans le PIA 6821, il existe quelques possibilités supplémentaires concernant ce mode de programmation des signaux de dialogue.

retour au Sommaire | Liens Rapides | Index

6821 : Ex-prog 05 : En réception

Supposons que les lignes PA0 à PA7 sont programmées comme des entrées. Lorsque le transmetteur externe a déposé une donnée sur les entrées du port A, il avertit le récepteur que la donnée est prête en actionnant l'entrée de dialogue CA1.

La transition active peut-être positive (front montant) ou négative (front descendant) selon la valeur du bit CRA1. Ce signal sur CA1 positionne le bit d'état CRA7 à 1.

Le µp6809 reconnaît CRA7=1, lit la donnée disponible dans le registre de données RDDRA.

Si le mode de fonctionnement de CA2 est "Automatique", une brève impulsion sera générée sur la sortie CA2 par la lecture même du registre de données.

Si le mode de fonctionnement de CB2 est "Manuel" le µp6809 doit modifier le bit CRA5 pour produire une impulsion sur la ligne CB2.

Cette impulsion avertit en retour le transmetteur que la donnée a été reçue avec succès, et qu'un autre cycle de transmission peut-être entamé.

Il est possible de se servir de la partie B pour effectuer une réception de données. Cependant, après chaque lecture du registre de données RDDRB, le µp6809 doit effectuer une écriture factice dans ce même registre pour produire l'impulsion de dialogue sur la ligne CB2, cette servitude ralentit la vitesse d'acquisition de l'ensemble.

retour au Sommaire | Liens Rapides | Index

6821 : Ex-prog 05 : En transmission

La partie B est conçue pour ce fonctionnement. Elle possède également un mode "automatique" et un mode "manuel".

Au début d'un cycle de transfert, le µp6809 lit l'état électrique de la ligne CB1 à travers le bit CRB7 pour savoir si le récepteur est prêt à recevoir.

Si le bit CRB7=1, la donnée peut-être déposée dans le registre données RDDRB. Cette écriture produit automatiquement une impulsion sur la sortie CB2 si un tel mode est sollicité.
On a la possibilité de produire cette impulsion manuellement en jouant sur le bit CRB5.

Cette impulsion avertit le récepteur externe qu'une donnée est déposée sur les lignes PB0 à PB7. Après lecture, le récepteur actionne de nouveau l'entrée CB1 et autre cycle recommence.

On peut éventuellement se servir de la partie A du PIA 6821 pour effectuer une transmission. Cependant, après chaque écriture dans le registre de données, le µp6809 doit effectuer une lecture factice du même registre pour produire une brève impulsion sur la sortie CA2, ce qui ralentit la vitesse d'acquisition.

retour au Sommaire | Liens Rapides | Index

Dans cette application, nous nous servons du même PIA 6821 pour tester simultanément une transmission et une réception avec une poignée de main complète.

Il faut effectuer au préalable les liaisons électriques suivantes :

- Relier PB0...PB7 à PA0...PA7
- Relier CB2 à CA1
- Relier CB1 à CA2

L'application suivante comporte en réalité 3 programmes différents :

- Les 2 premiers se servent de la partie B comme transmetteur et de la partie A comme récepteur.
- Dans le 3ième programme, B joue le rôle de récepteur et A de transmetteur.

La programmation n'est pas identique si l'on considère le mode génération des impulsions de dialogue sur les lignes CA2 et CB2.

En étudiant c'est programmes, il faut imaginer la présence de 2 PIA physiquement séparés pour pouvoir saisir l'utilité réelle des signaux de dialogue.

A l'exécution, chaque programme transfère une zone mémoire vers une autre pour simuler la transmission et la réception de données.

```
                    ;*************************************************************
                    ;    Transmission et Réception de donnée par PIA   page VI.44
                    ;*************************************************************
                    ;------Adresses registres Port A
          ED04      RCRA    EQU     $ED04       ;  Reg. Contrôle A
          ED00      RDDRA   EQU     $ED00       ;  Reg. Sens tranfert A
          ED00      ROR_A   EQU     $ED00       ;  Reg. Donnée A
                    ;------Adresses registres Port B
          ED0C      RCRB    EQU     $ED0C       ;  Reg. Contrôle B
          ED08      RDDRB   EQU     $ED08       ;  Reg. Sens tranfert B
          ED08      ROR_B   EQU     $ED08       ;  Reg. Donnée B
                    ; *************************************************************
                    ;------1ier PROGRAMME
                    ;------Port B = transmetteur  Port A = Récepteur
                    ;------Brève impulsion en mode automatique
8000                        ORG     $8000       ;  adrs du 1er programme
8000 7F   ED04              CLR     RCRA        ;  CRA2=0 adressage RDDRA
8003 7F   ED0C              CLR     RCRB        ;  CRB2=0 adressage RDDRB
8006 4F                     CLRA                ;
8007 B7   ED00              STA     RDDRA       ;  port.A  8 entrées
800A 4A                     DECA                ;
800B B7   ED08              STA     RDDRB       ;  port.B  8 sorties
800E 86   2C                LDA     #%00101100  ; impulsion automatique CB2 +
                                                ;  adressage reg. donnée
                                                ;  CRB7=1 sur transmition négative
8010 B7   ED0C              STA     RCRB        ;
8013 B7   ED04              STA     RCRA        ;  impulsion automatique CB2 +
                                                ;  adressage reg. donnée
                                                ;  CRA7=1 sur transmition négative
8016 108E 9000              LDY     #$9000      ;  adrs zone à transférer
801A 8E   1000              LDX     #$1000      ;  bloc de 4096 octets
801D CE   A000              LDU     #$A000      ;  Adrs de rangement
8020 B6   ED00              LDA     ROR_A       ;  initialiser le transfert en
                                                ;  mettant CRB7=1, récepteur prêt
                    ;------Boucle de transfert. ATTENTION
                    ;------CRB7 est remis à 0 après chaque écriture dans ROR_B
                    ;------CRA7 est remis à 0 après chaque lecture de ROR_A
8023 7D   ED0C      TRRP1   TST     RCRB        ;  récepteur prêt ?
8026 2A   FB   8023         BPL     TRRP1       ;  sinon attendre
8028 A6   A0                LDA     ,Y+         ;  donnée à transmettre
802A B7   ED08              STA     ROR_B       ;  Envoyer et produire impulsion
                                                ;  sur CB2 vers CA1
802D 7D   ED04      ATTEN1  TST     RCRA        ;  donnée reçue sur port A ?
8030 2A   FB   802D         BPL     ATTENT1     ;  si CRA7=0 attendre
8032 B6   ED00              LDA     ROR_A       ;  lire donnée puis production
                                                ;  impulsion sur CA2 vers CB1
8035 A7   C0                STA     ,U+         ;  rangement donnée
8037 30   1F                LEAX    -1,X        ;  bloc épuisé ?
8039 26   E8   8023         BNE     TRRP1       ;  sinon continuer
803B 3F                     SWI                 ;
```



```
                    ;*************************************************************
                    ;------2ième PROGRAMME
                    ;------Port B = transmetteur  Port A = Récepteur
                    ;------production manuelle des impulsions
                    ;--------------sur les sortie CA2 et CB2
803C 7F   ED04              CLR     RCRA        ;  CRA2=0 adressage RDDRA
803F 7F   ED0C              CLR     RCRB        ;  CRB2=0 adressage RDDRB
8042 4F                     CLRA                ;
8043 B7   ED00              STA     RDDRA       ;  port.A  8 entrées
8046 4A                     DECA                ;
8047 B7   ED08              STA     RDDRB       ;  port.B  8 sorties
804A 86   36                LDA     #%00110110  ;  CB1 et CB2 initialement bas
                                                ;  adressage reg. données +
                                                ;  CRA7=CRB7=1 sur transmition positive
804C B7   ED0C              STA     RCRB        ;
804F B7   ED04              STA     RCRA        ;
8052 108E 9000              LDY     #$9000      ;  adrs zone à transférer
8056 8E   1000              LDX     #$1000      ;  bloc de 4096 octets
8059 CE   A000              LDU     #$A000      ;  Adrs de rangement
805C 8A   08                ORA     #%00001000  ;  porter CA2 Haut, donc CRB7=1
                                                ;  pour initialiser le transfert
805E B7   ED04              STA     RCRA        ;
8061 84   F7                ANDA    #%11110111  ;  remettre CA2 bas
```

```
8063 B7   ED04             STA    RCRA        ;
                  ;----------------Boucle de transmission
8066 7D   ED0C     TRRP2   TST    RCRB        ;  récepteur prêt ?
8069 2A   FB   8066        BPL    TRRP2       ;  sinon, attendre
806B A6   A0               LDA    ,Y+         ;  Donnée à transmettre
806D B7   ED08             STA    ROR_B       ;  Envoi
8070 B6   ED0C             LDA    RCRB        ;  Mot de configuration
8073 8A   08               ORA    #%00001000  ;  porter CB2 haut, donc CRA7=1
                                               ;  pour avertir récepteur
8075 B7   ED0C             STA    RCRB        ;
8078 84   F7               ANDA   #%11110111  ;  remettre CB2 Bas
807A B7   ED0C             STA    RCRB        ;
                  ;---------------Boucle de réception
807D 7D   ED04     RECEP2  TST    RCRA        ;  donnée reçue sur port A ?
8080 2A   FB   807D        BPL    RECEP2      ;  sinon attendre
8082 B6   ED00             LDA    ROR_A       ;  lire donnée
8085 A7   C0               STA    ,U+         ;  rangement donnée
8087 B6   ED04             LDA    RCRA        ;  Mot configuration port A
808A 8A   08               ORA    #%00001000  ; porter CA2 haut, donc CRB7=1
                                               ;  pour avertir transmetteur
808C B7   ED04             STA    RCRA        ;
808F 84   F7               ANDA   #%11110111  ; remettre CA2 Bas
8091 B7   ED04             STA    RCRA        ;
                                               ;
                  ;----------------Test fin de la séquence transmission-réception
8094 30   1F               LEAX   -1,X        ;  bloc épuisé ?
8096 26   CE   8066        BNE    TRRP2       ;  sinon continuer
8098 3F                    SWI                ;
```

retour au Sommaire | Liens Rapides | Index

```
                  ; ***********************************************************
                  ;------3ième PROGRAMME
                  ;------Port B = récepteur   Port A = Transmetteur
                  ;------production des impulsions par des lectures et écritures
                  ;------factices des registres de données
8099 7F   ED04             CLR    RCRA        ;  CRA2=0 adressage RDDRA
809C 7F   ED0C             CLR    RCRB        ;  CRB2=0 adressage RDDRB
809F 4F                    CLRA               ;
80A0 B7   ED08             STA    RDDRB       ;  port.B  8 entrées
80A3 4A                    DECA               ;  (A)=$FF
80A4 B7   ED00             STA    RDDRA       ;  port.A  8 sorties
80A7 86   24               LDA    #%00100100  ;  impulsion sur sortie CA2
                                               ;  et CB2 + adressage reg. donnée
                                               ;  + CR7 = 1 sur transition négative
80A9 B7   ED04             STA    RCRA        ;
80AC B7   ED0C             STA    RCRB        ;
80AF 108E 9000             LDY    #$9000      ;  adrs zone à transférer
80B3 8E   1000             LDX    #$1000      ;  bloc de 4096 octets
80B6 CE   A000             LDU    #$A000      ;  Adrs de rangement
                  ;------Produire une impulsion vers le transmetteur
                  ;------port A pour indiquer que le récepteur est
                  ;------prêt et pour initialiser le transfert
80B9 B7   ED08             STA    ROR_B       ;  écriture factice dans le registre
                                               ;  donnée récepteur pour générer
                                               ;  un état bas sur CB2 (CA1). Le
                                               ;  bit CRA7 est mis à 1
                  ;------Boucle de transmission avec port A
80BC 7D   ED04     TRRP3   TST    RCRA        ;  réception prêt ?
80BF 2A   FB   80BC        BPL    TRRP3       ;  sinon attendre
80C1 A6   A0               LDA    ,Y+         ;  donnée à transmettre
80C3 B7   ED00             STA    ROR_A       ;  envoi de donnée
80C6 B6   ED00             LDA    ROR_A       ;  lecture factice pour produire
                                               ;  impulsion avertissant récepteur
                  ;------Boucle de réception avec port B
80C9 7D   ED0C     RECEP3  TST    RCRB        ;  donnée présente sur port B ?
80CC 2A   FB   80C9        BPL    RECEP3      ;  si CRB7=0, attendre
80CE B6   ED08             LDA    ROR_B       ;  lire donnée
80D1 B7   ED08             STA    ROR_B       ;  écriture factice pour mettre
                                               ;  CB2 état bas avertissant le
                                               ;  transmetteur (CRA7=1)
80D4 A7   C0               STA    ,U+         ;  rangement donnée
80D6 30   1F               LEAX   -1,X        ;  bloc épuisé ?
80D8 26   E2   80BC        BNE    TRRP3       ;  sinon continuer
80DA 3F                    SWI                ;
```

retour au Sommaire | Liens Rapides | Index

6821 : Quelques explications sur le programme ci-dessus

Le premier type de transfert est le plus rapide, ce type de transfert émettant de brèves impulsions automatiquement après chaque lecture ou chaque écriture sur les lignes CA2 et CB2.

Cependant, si l'un des deux systèmes est plus lent que l'autre, il faut prolonger "manuellement" les impulsions. C'est l'esprit du second programme.

Dans le 3ième programme, à cause de la conception du PIA 6821, on est obligé d'écrire des instructions factices de lecture et d'écriture dans les registres de données pour produire des impulsions sur les sorties CA2 et CB2.

On a souvent tendance à les supprimer car elles ne jouent apparemment aucun rôle, d'où l'importance d'une documentation très explicite.

On peut confier le rôle de "maître" soit au transmetteur, soit au récepteur. Dans les deux cas, l'organe assurant le contrôle doit détecter le nombre maximum de données requises et mettre en veilleuse l'autre pour transférer éventuellement d'autres données ultérieurement.

On remarque dans les trois programmes, la présence d'une séquence d'initialisation indispensable au démarrage du transfert.

retour au Sommaire | Liens Rapides | Index

6850 : Organisation des données sérielles

retour au Sommaire | Index | Sommaire Principal | Liens Rapides

ACIA (**A**synchronous **C**ommunications **I**nterface **A**daptor) permet de réaliser la liaison série entre le µp6809 et la périphérie, il s'adapte au standards EIA-RS-232-C et CCITT-V24 (transfert de données par le mode série).

Il existe 2 types de communication série Synchrone
Asynchrone

Il est nécessaire d'adjoindre au 6850, deux circuits convertisseurs de niveaux (et éventuellement quelques amplificateurs de lignes) : Le circuit **1489** pour la conversion TTL - RS232 Il existe aussi
Le circuit **1488** pour la conversion RS232 – TTL le circuit **MAX232**

Les signaux délivrés sur une ligne RS232 obéissent à une logique négative, les tensions fluctuent entre :
+12 volts (pour le niveau 0, niveau Bas)
-12 volts (pour le niveau 1, niveau Haut).

6850 : Protocole Start-Stop

retour au Sommaire | Index | Liens Rapides

Le transfert sur une ligne RS 232 débute par un bit de départ, qui amène la ligne de -12v (position d'attente) à un +12v, indiquant ainsi au récepteur le début d'un transfert.

La ligne est maintenue à +12v durant le bit de départ, soit 833 µs pour une vitesse de 1200 bauds.

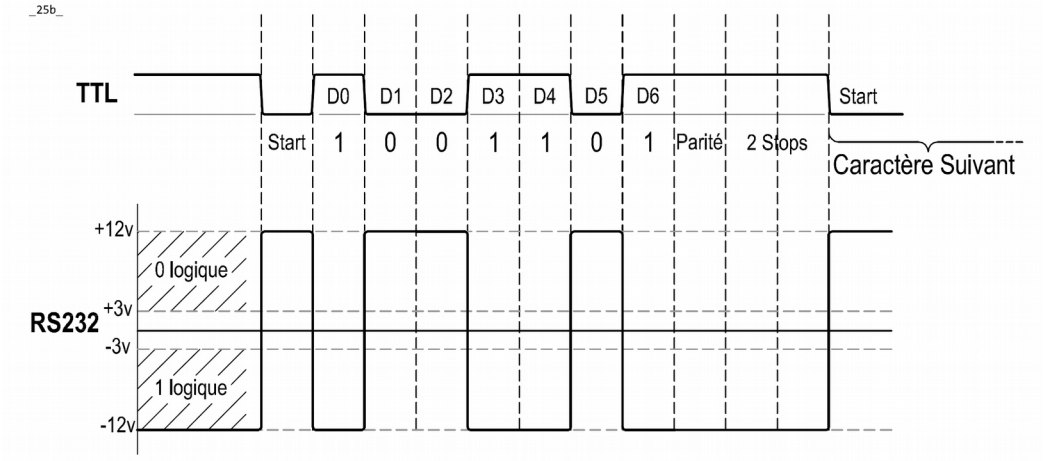
Le récepteur échantillonne le bit de départ vers son milieu pour s'assurer qu'il s'agit bien d'un bit de départ, éliminant ainsi les éventuelles impulsions parasites de courte durée sur la ligne.

Le récepteur lit ensuite l'état de la ligne toutes les 833 µs et vers le milieu d'un bit. Il est bien évident que l'émetteur et le récepteur doivent être réglés initialement sur la même vitesse de transfert.

Après la lecture de 7 ou 8 bits de données (suivant la configuration), le récepteur échantillonne un éventuel bit de parité puis détecte le dernier bit qui est un bit d'arrêt (possibilité de mettre 2 bits d'arrêt)

On remarquera que le bit de départ est toujours au +12V, tandis que les bits d'arrêt sont toujours au niveau -12V

A la détection du front de montée du bit de départ suivant, le récepteur corrige la synchronisation de son horloge interne pour la lecture de la donnée adjacente, on parle ici de transmission asynchrone.



6850 : Protocoles DTR, XON-XOFF et ETX-ACK

retour au Sommaire | Index | Liens Rapides

Dans la majorité des situations, l'émetteur est beaucoup plus rapide que le récepteur et la durée que met le récepteur pour exploiter les données est très longue.

Le récepteur doit signaler d'une façon ou d'une autre à l'émetteur qu'il faut arrêter momentanément l'envoi et attendre la prochaine disponibilité du buffer de réception.

Ces procédures d'appel-réponse sont souvent qualifiées de "poignées de main" (Handshaking)

Voir aussi l'exemple dans le sommaire→ Autres Exemples

6850 : Protocol DTR (Data Terminal Ready)

retour au Sommaire | Index | Liens Rapides

On emploi une ligne de contrôle DTR issue d'une sortie du récepteur, qui sera connecté à l'entrés DCD de l'émetteur. Cette ligne DCD est ramenée sur l'entrée DCD du 6850 au travers d'un convertisseur de ligne qui inverse également le signal.

Quand le buffer de réception est presque vide, le récepteur met la ligne DTR à +12V et le transfert est autorisé.

Lorsque le buffer de réception est presque plein, le récepteur force la ligne DTR à – 12 volts pour avertir l'émetteur qu'il faut arrêter momentanément l'envoi des données.

Dans ce protocole, la poignée de main est assurée au **niveau matériel** par l'implantation d'une ligne supplémentaire.

Voir aussi l'exemple dans le sommaire → Autres Exemples

6850 : Protocole XON – XOFF

retour au Sommaire | Index | Liens Rapides

La poignée de main est exécutée au **niveau logiciel**.
Quand le buffer de réception est presque plein, le récepteur renvoie vers l'émetteur, via la ligne de transmission le code DC3 (Device Control 3) caractère ASCII $13. Dans ce cas l'émetteur suspend l'envoi des données.

Quand le buffer de réception est presque vide le caractère renvoyé vers l'émetteur est le code DC1 caractère ASCII $11. L'émetteur reprendra le transfert de données.

Ce protocole suppose que l'émetteur doit surveiller constamment le retour possible d'un caractère DC1 ou DC3.

On remarque que l'interface fonctionne simultanément en transmission et en réception, on dit qu'il fonctionne en FULL-DUPLEX. Par opposition le mode HALF-DUPLEX implique soit une transmission, soit une réception, mais jamais une simultanéité des deux.

Voir aussi l'exemple dans le sommaire → Autres Exemples

6850 : Protocole ETX-ACK

retour au Sommaire | Index | Liens Rapides

Le buffer d'émission doit être plus petit que le buffer de réception (taille calculée en octets).
L'émetteur envoie un bloc entier de données, avec un caractère spécial marquant la fin du bloc. C'est le code ASCII **`$03`** (ETX - End Of Text).

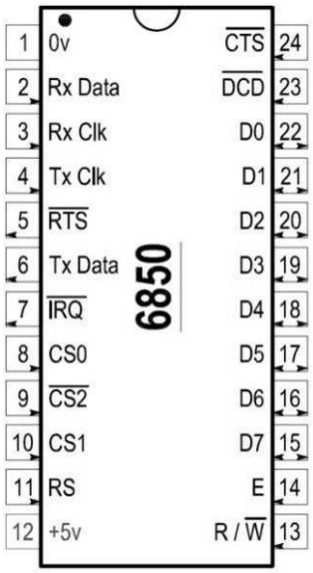
L'émetteur arrête ensuite l'envoi des caractères.

De son côté, le récepteur exploite les données à sa propre vitesse. Arrivant au caractère ETX le récepteur retourne à l'émetteur le caractère ASCII **`$06`** (ACK – acknowledge) indiquant à l'émetteur que le récepteur est prêt à accepter un autre bloc de données.

Voir aussi l'exemple dans le sommaire → Autres Exemples

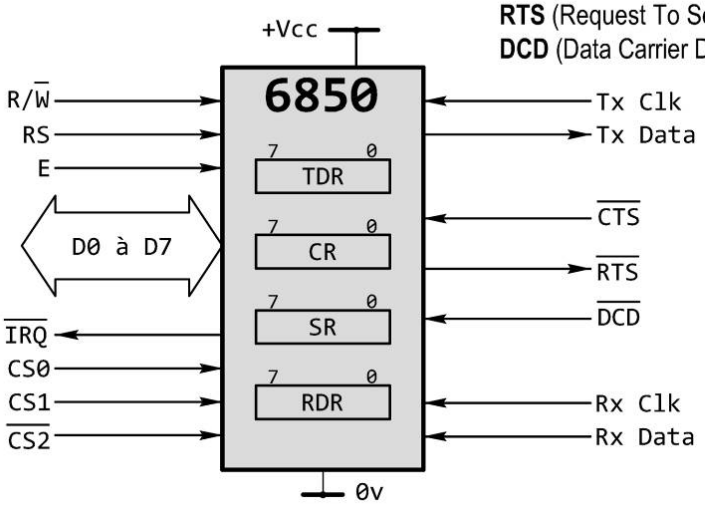
6850 : Brochage

retour au Sommaire | Index | Liens Rapides



6850 : Organisation Interne

_21a_



CTS (Clear To Send) Inhibition de l'émetteur
RTS (Request To Send) Demande d'émission par le récepteur
DCD (Data Carrier Detect) Perte de porteuse de données

TDR (Transmit Data Register)
Registre de Transmission **(En écriture)**

CR (Control Register)
Registre de Contrôle **(En écriture)**

SR (Status Register)
Registre d'Etat **(En lecture)**

RDR (Receive Data Register)
Registre de Réception **(En lecture)**

6850 : Les échanges avec le µp6809 se font par :

- Bus de données d0 à d7, ces 8 broches bidirectionnelles sont reliées au µp6809, elles assurent l'échange des données entre le µp6809 et le 6850. Si elles ne sont pas utilisées ces broches sont en haute impédance. Ce bus assure en autre :
  - Programmer le registre de contrôle
  - Ecrire dans le registre transmission
  - Lire dans le registre réception
  - Lire le registre d'état

- Bus d'adresses 4 lignes allant sur les broches :
  - 3 Lignes de validation de boîtier **CS0, CS1, CS2 |** (Chip Select Line), qui permettent l'adressage physique du boîtier. On appliquera des bits d'adresse haute sur ces 3 broches lorsque l'on a affaire à un petit système.
    CS0, CS1 Sont validées au niveau Haut, CS2 **|** est validé au niveau bas, d'où la combinaison de sélection pour CS0**, CS1, CS2 | = %110**
  - Une entrée de sélection de registre **RS** (Register Select Ligne) qui permet de sélectionner les registres internes.

- Bus de contrôle
  - Une entrée **E** (Enable Line) signal d'activation des échanges, reçoit un signal de l'horloge généralement E du µp6809 (anciennement Φ2 sur le 6800)
  - **R/W |** (Read/Write) : lecture écriture
  - **IRQ |** (Interrupt ReQuest line) : reliée à IRQ|, FIRQ| ou à NMI| du µp6809, cette sortie permet d'interrompre le µp6809.

6850 : Les échanges avec les périphériques se font par :

- Une broche de transmission de données **TxD** (Transmit Ted Data Line). Cette broche de sortie assure la transmission des données en série.

- Une broche de réception de données **RxD** (Receive Data Line). Cette broche d'entrée réceptionne les données série en provenance de la périphérie.

- 3 broches de contrôle assurant la synchronisation des transferts :

  - **CTS|** : (Clear To Send) Permet l'inhibition de l'émetteur.
    Broche d'entrée, permet le contrôle automatique de la fin de transmission par un MODEM.
    Non utilisée, elle doit être au niveau Bas.

  - **RTS|** : (Request To Send) Permet une demande d'émission.
    Broche de sortie, permet de piloter un périphérique ou un modem.

  - **DCD|** : (Data Carrier Detect) Perte de la porteuse de données.
    Broche d'entrée, permet le contrôle de la réception.
    Non utilisée, elle doit être au niveau Bas.

- 2 entrées d'horloge permettant de fixer les cadences de transmission et de réception.
  - **TxCk** (Transmit Clock)
    Entrée horloge de transmission, elle sert de référence pour la transmission des données.
  - **RxCk** (Receive Clock)
    Entrée horloge de réception, elle est utilisée pour la synchronisation des informations reçues.

Les échanges avec le périphérique se font suivant le mode START-STOP, adapté pour les transmissions de 50 bps à 500 kbps (bps = bit par seconde). Chaque caractère en informant de :

- 7 ou 8 bits.
- Suivi d'un bit de parité paire ou impaire.
- Suivi d'un ou deux bits de Stop (suivant la cadence désirée).

Les bits de Start et Stop on pour but de synchroniser deux caractères consécutifs.

retour au Sommaire | Index | Liens Rapides

6850 : Sélection des Registres Internes

Bien que le 6850 ait 4 registres internes de 8 bits (2 à lecture seule, 2 à écriture seule), le µp6809 voit cette interface comme s'ils occuperaient seulement 2 positions mémoires.

- **CR** (Control Register) registre de contrôle.
  En Ecriture seule, contiennent les paramètres de fonctionnement.

- **SR** (Statut Register) registre d'état.
  En Lecture seule, contiennent les informations sur les opérations en cours.

- **TDR** (Transmit Data Register) registre de transmission de données.
  En Ecriture seule, contient le mot de 8 bits à émettre.

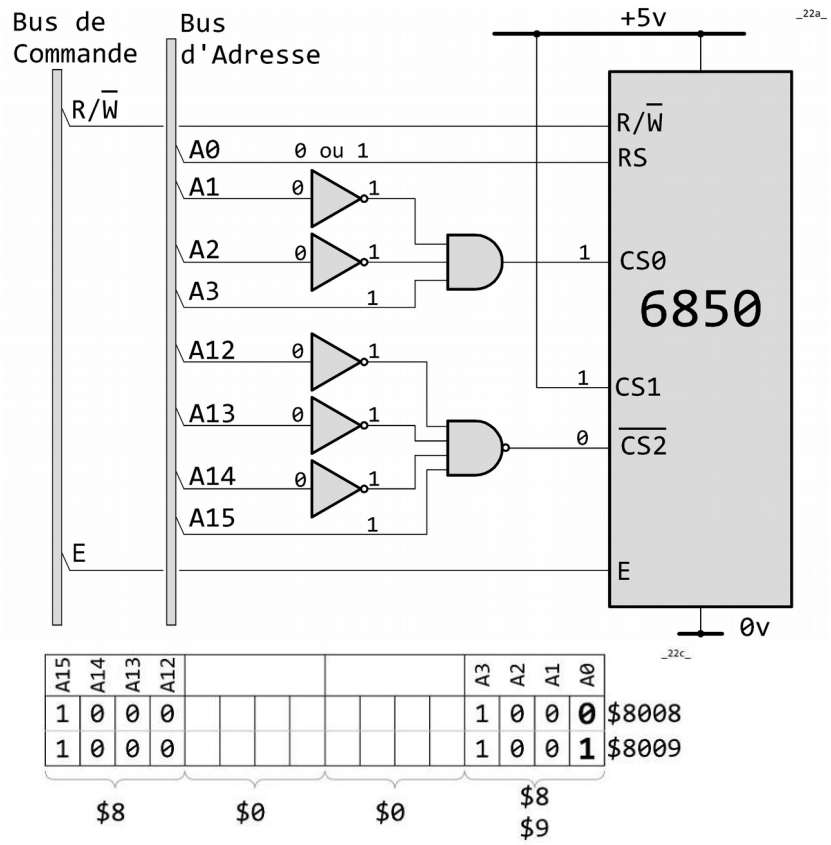
- **RDR** (Receive Data Register) registre de réception de données.
  En Lecture seule, reçoit le mot de 8 bits en provenance de la périphérie.

Dès la validation du boîtier par CS0, CS1 et CS2|, l'adressage des 4 registres du 6850, se fera en conjonction des broches R/W| et RS.
Le boîtier sera sélectionné si CS0.CS1.CS2| = %110
L'entrée RS recevra le bit A0 pour que les adresses soient consécutives.

_22b_

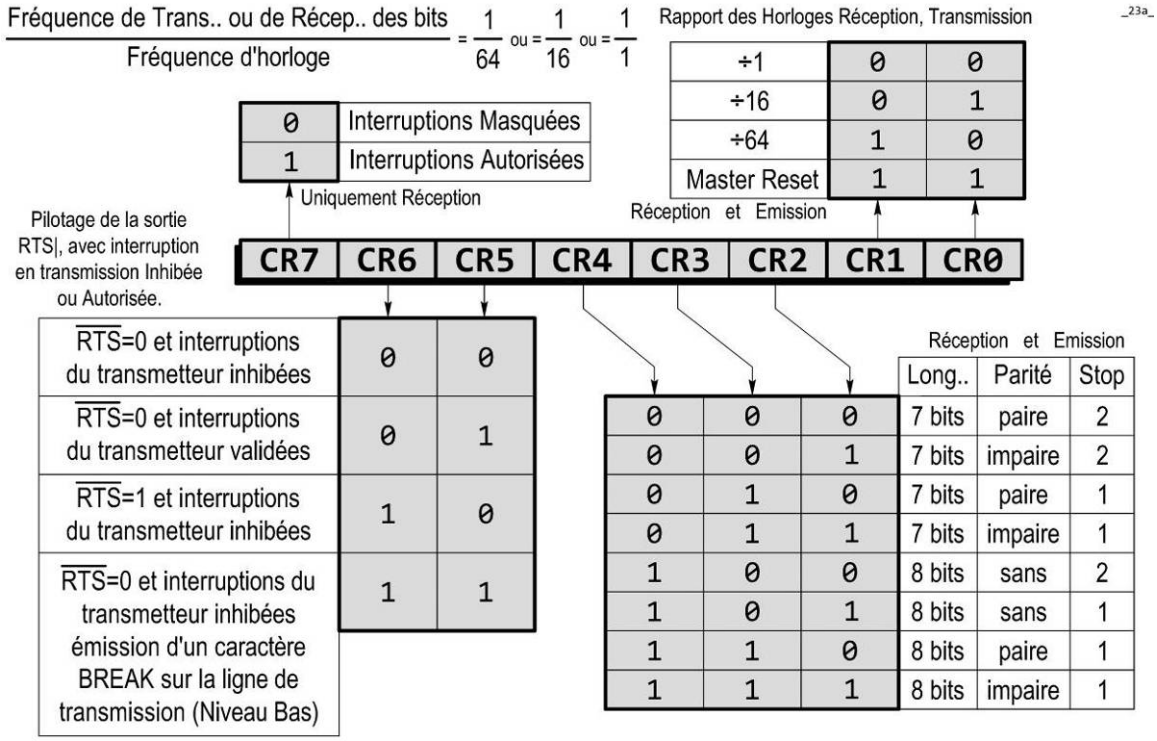
ACIA 6850	Broches du 6809 → / Broches du 6850 →	Logique de décodage A15 à A1			A0	R/W		
		CS0	CS1	CS2	RS	R/W	Adresse	
Ecriture dans le registre de contrôle	**CR**	1	1	0	**0**	**0**	**Adr**	Reçoit les paramètres du fonctionnement
Lecture du registre d'état ou de statut	**SR**	1	1	0	**0**	**1**	**Adr**	Permet au 6809 de connaître l'état du registre d'émission, du registre de réception, du CTS 106, du DCD 109, de IRQ
Ecriture dans le registre Transmission de donnée	**TDR**	1	1	0	**1**	**0**	**Adr + 1**	L'octet à écrire va dans le registre TDR puis il est transmis par le 6850 au registre TSR pour être transmit sur la broche TxD 103 C'est le ↓ de la broche E qui charge les 8 bits à émettre dans TDR, le bit 0 est émis en premier
Lecture du registre Réception de donnée	**RDR**	1	1	0	**1**	**1**	**Adr + 1**	Reçoit l'octet à lire, c'est le ↓ de la broche E qui permet la lecture du registre RDR L'octet reçu par la broche RxD 104 est stocké bit par bit dans le registre RSR puis le 6850 transmets cet octet au registre RDR

Exemple : d'implantation d'un 6850 (implanté à l'adresse $8008 et $8009)



6850 : Registre CR (control Register) registre de contrôle

Contient les paramètres de fonctionnement (format, vitesse, ...) de transmission de la réception.
Précise le mode fonctionnement : De la partie réception et de la partie émission
Des interruptions
De la broche RTS|



6850 : Bits CR1 CR0 :

Détermine les facteurs de division des horloges de transmission de réception.
= %11 Est utilisé pour l'initialisation programmée du registre CR,

Cette combinaison est appelée **MASTER RESET**.

retour au Sommaire Liens Rapides

6850 : Bits CR4 CR3 CR2 :

Détermine le format du mot transmis ou reçu : - Longueur du mot
- Parité
- Nombre de bits d'arrêt

retour au Sommaire Liens Rapides

6850 : Bits CR6 CR5 :

Contrôle la partie transmission. Actifs en transmission seulement, ces bits déterminent la méthode de transfert. L'envoi d'un caractère vers la périphérie est toujours précédé d'un test sur le bit TDR du registre SR (lecture du bit SR1). Pour s'assurer que celui-ci est vide.
Les 4 combinaisons permettent :

- De fixer l'état de sortie de la demande d'envoi RTS
- De valider ou non une autorisation d'interruption TIE (Transmit Interrupt Enable).
  Si TIE est validé alors le bit SR7 suivra SR1 du même registre d'état.
  Donc si le bit SR1 égal à 1, alors le registre de transmission est vide, on générera une interruption de transmission (broche IRQ| au niveau bas) et le bit SR7 sera positionné à 1.
- De générer un BREAK (niveau bas sur la ligne)

retour au Sommaire Liens Rapides

6850 : Bit CR7 :

Valide ou non une autorisation d'interruption de réception RIE (RIE = Receiver Interrupt Enable).
Si RIE est validé, alors le bit SR7 suivra le bit SR0 du même registre d'état.
Donc si le bit SR0=1 (indiquant que le registre réception est plein), alors la sortie IRQ| passe au niveau Bas et le bit SR7 sera positionné à 1.

retour au Sommaire Index Liens Rapides

6850 : Initialisation programmée (MASTER RESET)

A la différence du PIA 6821 qui possède une broche RESET|, le 6850 a son propre circuit de mise sous tension qui le maintient dans son état inhibé jusqu'à son initialisation programmée, ceci afin d'éviter la transmission de d'informations erronées.

Pas de broche RESET sur le 6850 due à la limitation du nombre de broches 24 broches pour le 6850. Le MASTER RESET est obtenu par programme en mettant b1 b0 du registre CR à %11, ce qui impose la broche RTS| à l'état haut (donc sans action), les interruptions du transmetteur son inhibées.

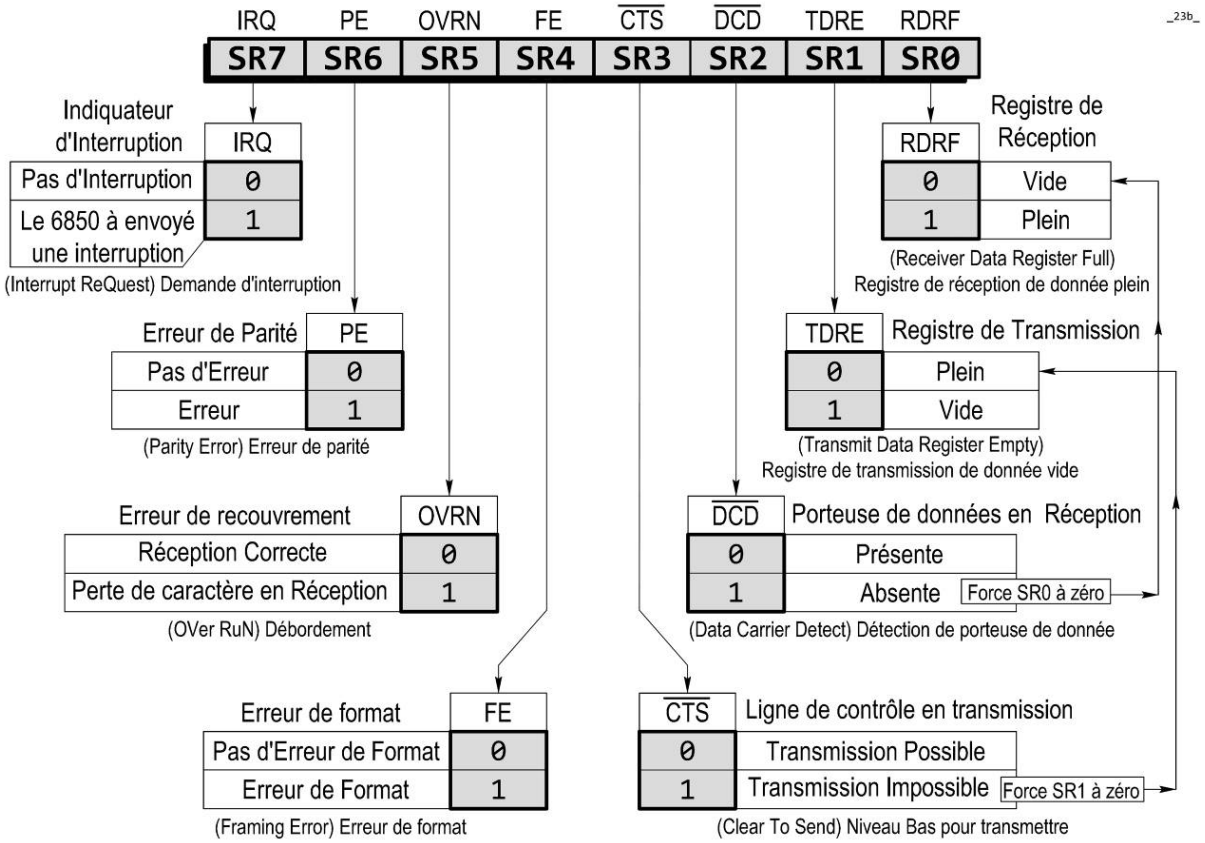
Après un MASTER RESET les bits b6 b5 du registre CR ne sont plus d'inhibés, le registre CR peut alors être programmé. D'autre part, tous les bits du registre SR sont mis à zéro sauf b3 et b2.

Avant de programmer un mot le 6850, la mise sous tension doit être suivie d'un MASTER RESET (initialisation programmée).

retour au Sommaire Index Liens Rapides

6850 : Registre SR (Status Register) Registre d'états

8 bits à lecture seule, le registre SR appelé registre d'état, contient le mot d'état qui renseigne le µp6809 sur les opérations en cours.
Contient les informations d'état en provenance : Du registre de transmission, Du registre de réception, Des lignes de commande.



Pour la suite, il est intéressant de remarquer qu'en général les bits du registre d'état sont remis à 0 par :

- Une lecture du registre de réception.
- Une écriture dans le registre de transmission.

Ce procédé comporte un avantage car il accélère le transfert : les bits SR0 et SR1 sont automatiquement "préparés" pour le transfert du caractère suivant.

Pour les autres bits, la remise à 0 s'effectue de façon indirecte car on ne peut pas écrire dans le registre d'état. Dans un programme assembleur apparaissent parfois des instructions dites factices du type

`LDA RGRX` ou `STA RGTX`.

Les valeurs lues ou écrites ne sont pas utiles, on cherche tout simplement à agir sur les bits du registre d'état.

Pour une lecture factice, si l'on ne veut pas affecter la valeur dans les accumulateurs, on peut employer `TST RGRX` dans le mode étendu ou indexé pour obtenir le même effet. Cette instruction affecte néanmoins le registre d'état CC du µp6809.

6850 : SR0 : RDRF (Receiver Data Register Full) registre de réception de donnée plein

retour au Sommaire | Index | Liens Rapides

Un test sur ce bit permet de connaître l'état du registre de réception.
La lecture de ce bit réinitialise SR0, il passe à 0.

= 0 registre de réception est vide (ou broche d'entrée DCD| = 1)
= 1 registre de réception est plein, et que les autres bits SR6 SR5 SR4 (PE, OVRN, FE) sont positionnés.

Une lecture du registre de réception remettra ce bit SR0 (RDRF) à zéro.
Ce bit est aussi affecté par la broche DCD| et le MASTER RESET.

Si la broche DCD| est à un niveau Haut (indiquant une perte de porteuse) alors ce bit SR0 est forcé à 0 jusqu'à ce que DCD| retrouve son niveau bas.

Pendant un MASTER RESET ce bit SR0 est également forcé à 0.

retour au Sommaire | Index | Liens Rapides

6850 : SR1 : TDRE (Transmit Data Register Empty) Registre de transmission de donnée vide

Le 6850 a fini d'envoyer un octet. Ce bit permet de connaître l'état du registre de transmission.
L'écriture dans ce dernier fait passer le bit SR1 de 1 à 0.

SR1 = 0 Indique que le registre de transmission TDR est **PLEIN** Un signal de transfert fera passer la donnée du registre TDR au registre TSR, remettant ce bit SR1 à 1.
SR1 = 1 Indique que le registre de transmission TDR est **VIDE**.
Donc qu'une donnée peut être envoyée par le µp6809. Une nouvelle donnée peut y être inscrite. Après écriture, ce bit SR1 est remis à 0.

Si la broche CTS| est à un niveau haut ce bit SR1 est forcé à 0.

retour au Sommaire | Index | Liens Rapides

6850 : SR2 : DCD (Data Carrier Detect) Détection de la perte de la porteuse de donnée

Ce bit est lié à la broche DCD| DCD| = 0 (broche à l'état Haut)
DCD| = 1 (broche à l'état Bas)

SR2 = 0 indique d'une porteuse de donnée est présente.
SR2 = 1 indique une perte de porteuse. (si la broche DCD| repasse à l'état Bas).

Si perte de porteuse alors les bits SR7 et SR2 sont à 1

IRQ| = 0 lors de la transition ⌐ de la broche DCD| (si les interruptions du récepteur sont autorisées) le bit CR7 = 1.

Si la broche DCD| est employée pour l'appel-réponse alors le bit SR2 passe à 1 par une mise à l'état bas de la broche DCD| par le récepteur.

Ce bit SR2 est remis à 0 par une lecture du registre d'état SR suivie d'une lecture du registre de réception, à condition que la ligne DCD soit remise au préalable à l'état Haut.
Lors d'un MASTER RESET le bit SR2 = 0 et le bit SR7 = 0

Dans ce cas, si les interruptions de réception sont autorisées (bit CR7 à 1) alors bit SR7 passe à 1 et un niveau Bas est envoyé sur la broche IRQ|.

Le registre de réception est toujours considéré comme vide, aucune donnée n'est prise en considération.

Pour le remettre à Zéro, il est nécessaire :

- Soit de faire une lecture du registre SR, suivie d'une lecture du registre de réception.
- Soit de faire un MASTER RESET.

Si pendant l'opération de RAZ du bit SR2, la broche DCD| reste à l'état Haut.
Ce bit SR2 repassera à 0 dès que la broche DCD| retrouve l'état Bas.

retour au Sommaire | Index | Liens Rapides

6850 : SR3 : CTS (Clear To Send) Niveau Bas pour transmettre

Ce bit indique l'état de la broche CTS| en provenance du modem.
Inhibition de l'émetteur. Si CTS| est à 1, la transmission n'est pas possible, on force le bit SR1 (TDRE) à 0.

Le registre TDR (registre de transmission de données) est toujours considéré comme étant plein.
SR3 = 0 Broche d'entrée CTS| à l'état Haut, le modem est prêt à émettre
SR3 = 1 Broche d'entrée CTS| à l'état Bas, le registre de transmission est inhibé, le bit SR1 (TDRE) mis à 0 et le bit SR7 (IRQ) mis à 0.

L'application d'un niveau haut sur la broche CTS| entraîne l'inhibition des bit SR1 et SR7 (SR1 = 0 et SR7 = 0).

Ce signal s'emploi dans le mode modem. Il indique que l'organe récepteur désire émettre et suspend l'activité du transmetteur.

La broche d'entrée CTS| n'a pas d'effet sur un caractère en cours de transmission ou placé dans le registre de transmission. Il y a seulement non initialisation de l'émission.

La broche d'entrée CTS| doit être mise à zéro en cas d'inutilisation.

Le bit CTS| n'est pas affecté par MASTER RESET.

retour au Sommaire | Index | Liens Rapides

6850 : SR4 : FE (Framing Error) Erreur de format

Ce bit est positionné durant le transfert de données reçues et le restera positionné tant que le défaut subsiste. Ce bit b4 est remis à zéro par un MASTER RESET ou un niveau Haut sur la broche DCD|.

SR4 = 0 Format correct, indique qui n'y a pas d'erreur de format.
Les deux organes de dialogue (Emetteur et Récepteur) sont réglés sur un format identique.
SR4 = 1 Indique une erreur de format, qui peut provenir d'une erreur (absence de premier bit STOP) ou perte de synchronisation ou d'une réception défectueuse ou de la réception d'un BREAK.

retour au Sommaire | Index | Liens Rapides

6850 : SR5 : OVRN (OVer RuN) Débordement

Indique que certains caractères reçus n'ont pas été lus par le µp6809 ou un ou plusieurs caractères on été perdues.
Ceci se produit lorsqu'un ou plusieurs caractères ont été reçus avant la lecture du caractère précédent dans le registre de réception.

SR5 = 0 Réception correcte sans surcharge, aucun caractère n'a été perdu.
SR5 = 1 Indique une surcharge en réception. L'évacuation des données reçues est trop lente. Après la suppression de la surcharge, ce bit SR5 peut être remis à 0 par une seconde lecture factice du registre de réception.

Cependant la mise à 1 n'intervient que lorsque la lecture du caractère précédent la surcharge a eu lieu et elle se produit à partir du milieu du dernier bit du deuxième caractère reçu sans lecture du registre de réception.

Le bit SR0 (RDRF) reste à 1 jusqu'à ce que la condition de surcharge soit supprimée.
Un niveau haut sur la broche DCD| ou un MASTER RESET entraîne le SR5 (OVRN) à zéro.

Le bit SR5 peut également être réinitialisé par une seconde lecture du registre RDR (registre de réception de données) Lors d'une nouvelle lecture le bit SR0 = 0 (RDRF) et le bit SR5 = 0 (OVRN)

retour au Sommaire | Index | Liens Rapides

6850 : SR6 : PE (Parity Error) Erreur de parité

Ce bit sera positionné si l'on n'a programmé un contrôle parité par bits CR4 CR3 CR2, alors le bit SR6 sera activé lors du transfert interne.

SR6 = 0 Parité correcte, indique qui n'y a pas erreur de parité.
SR6 = 1 Indique une erreur de parité en réception.
Ce bit est actif seulement si le récepteur est configuré avec une détection de parité (voir les bits SR4, SR3, SR2).

Un niveau Haut sur la broche DCD| ou un MASTER RESET met ce bit SR6 à 0.

retour au Sommaire | Index | Liens Rapides

6850 : SR7 : IRQ (Interrupt Request) Demande d'interruption

Ce bit passe à 1 entraînant le passage à 0 de la sortie IRQ| du 6850.
S
R7 = 0 Absence d'interruption, il n'a pas eu d'interruption.
SR7 = 1 Demande d'interruption, une interruption est positionnée.

Ce bit SR7 Indique la présence d'une demande d'interruption qui provient (3 sources sont possibles) :

1. Du récepteur (donc en réception), si le "mode par interruption" du récepteur est autorisé le bit CR7 = 1, le bit SR7 reproduit l'état du bit de réception SR0. (RDRF)
2. Du transmetteur (donc en émission), si les bits CR6 CR5 = %01 alors les interruptions du transmetteur sont autorisées, le bit SR7 reproduit l'état du bit SR1. (TDRE)
3. D'une perte de porteuse, lorsque le bit CR7 = 1, les interruptions du récepteur sont autorisées. Le bit SR7 passe également à 1, lorsque la broche DCD| est forcée à l'état Bas par une condition externe (broche d'entrée DCD| = 1).
   Lorsque la broche DCD| (109) passe à un niveau haut  alors il y a génération d'une interruption (comme pour le bit SR2).

Le bit SR7 est remis à 0 par une lecture du registre de réception (interruption provenant du récepteur) ou par une écriture dans le registre de transmission (interruption provenant du transmetteur). Dans tous les cas, la broche DCD| doit être forcée à l'état Haut au préalable (entrée DCD| = 0).

retour au Sommaire | Index | Liens Rapides

6850 : Transmission

Registre de transmission de donnée reçoit du µp6809, par l'intermédiaire du bus de donnée, le mot à transmettre qui sera transféré dans le registre TSR (Registre à décalage de transmission) pour être ensuite sérialisé.
La transmission d'un caractère doit être précédée de la lecture du registre d'état SR, afin de connaître l'état du bit SR1 (TDRE). On pratique par interruption ou par boucle d'attente (polling).

Si le bit SR1 (TDRE) est à 1, le caractère transmettre et chargé dans le registre TDR (registre de transmission de données) sur une commande d'écriture (front descendant de Φ2), le bit SR1 (TDRE) passe alors à 0 indiquant le que le registre TDR n'est pas libre.

Ensuite les données sont transférées du registre TDR dans le registre TSR (Registre à décalage de transmission) pendant une absence de transmission avec une durée correspondant à un bit série.
Le fond descendant du signal de transfert remet le bit SR1 (TDRE) à 1, ainsi un autre caractère peut être immédiatement chargé dans le registre TDR.

Le registre à décalage de sortie TSR, transmet sur la broche TxData le caractère en synchronisme avec une horloge interne dont la fréquence peut t'être 1/1, 1/16 ou 1/64 de la fréquence d'horloge appliquée sur la broche TxClk.

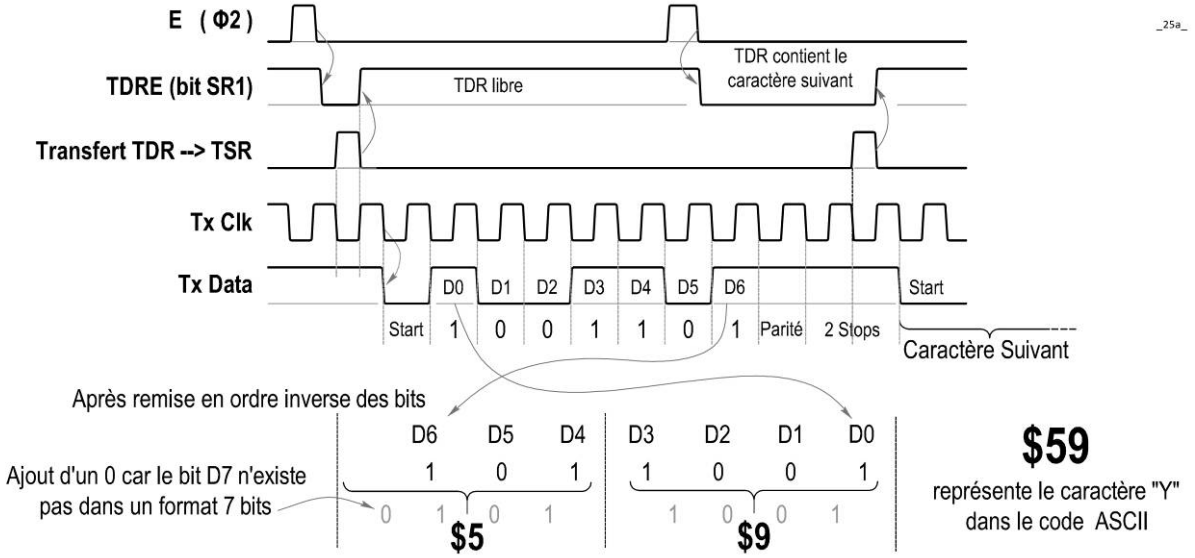
Le caractère est transmis bit par bit en commençant par D0 précédé d'un bit START, D7 étant suivi éventuellement par le bit de parité puis par un ou deux bites de STOP suivant la programmation du registre de contrôle. Les bits sont transmis sur la transition négative de l'horloge interne.

Pendant une opération de transmission la longueur d'un caractère et le nombre de bits d'arrêt peuvent être modifiés sans affecter le caractère en cours de transmission (sauf si la transition à lieu pendant le temps de transfert interne). La sélection de parité affecte immédiatement le caractère en cours de transmission.

Toute modification sur la partie transmission affecte également la partie réception. Si celle-ci est dommageable, on attendra une absence de réception.

Exemple de chronogramme de transmission

On transmet la lettre **"Y"** codée en ASCII (59), donc sur 7 bits avec bit de parité paire et 2 bits de STOP.
On remarque que le deuxième caractère doit être inscrit dans le registre TDR avant le dernier bit de STOP du caractère précédent si mon il y a absence de transmission (IDLIND TIME).



retour au Sommaire | Index | Liens Rapides

6850 : Réception

Le registre de réception de donnée RDR reçoit le mot dé-sérialisé en provenance du registre à décalage de réception RSR.

Dans les systèmes de transmission asynchrone, les données sont transmises sans signal de synchronisation, ce sont les bits START et STOP qui vont permettre une synchronisation des bits du caractère aussi par rapport à la broche d'entrée RxClk (horloge de réception).

Cette broche RxClk synchronise une horloge interne dont le facteur de division 1/16 ou 1/64 est choisi par la programmation du registre de contrôle CR.

Pour le rapport 1/1 la synchronisation doit être externe.

retour au Sommaire | Index | Liens Rapides

6850 : Rapports 1/16 et 1/64

La synchronisation est assurée par la première transition négative (front descendant) suivant une période de repos.

Le bit de départ START est échantillonné durant les transitions positives (front montant) de l'horloge externe RxClk.

Si la broche d'entrée RxData est au niveau Bas pour 9 échantillons dans le mode 1/16 ou pour 33 échantillons dans le mode 1/64, ce qui représente plus de 50% de la durée d'un bit, le bit reçu et considéré comme bit START.

Ce bit START est rangé dans le registre à décalage RSR sur le front descendant de l'horloge interne.

Une fois que le bit de départ est détecté, la synchronisation de bit et de caractère est faite, les autres bits suivant le bit START sont décalés dans le registre RSR à peu près au milieu de leur durée.

Si la broche d'entrée RxData retourne à l'état Haut pendant la période d'échantillonnage du bit de départ, ce faux bit de départ est ignoré et le récepteur se place en attente d'un bit de départ correct.

retour au Sommaire | Index | Liens Rapides

6850 : Rapport 1/1

Dans ce mode, il n'y a pas de synchronisation de bit à l'intérieur du récepteur, donc l'horloge externe doit être synchronisée par la donnée.

Dès l'apparition de la première transition négative (front descendant), après une période de repos du signal reçu, l'échantillonnage ce produit sur le front montant de l'horloge externe la broche RxClk et le bit START est chargé sur le front descendant suivant de l'horloge.

Pour améliorer la sécurité de détection, le front positif de l'horloge externe doit intervenir dans le milieu d'un bit.

retour au Sommaire | Index | Liens Rapides

6850 : Fonctionnement général du récepteur

La validité du caractère reçu est contrôlée pendant la réception et positionne les bits concernés du registre d'état. Il y a test de la parité, du format et du débordement.

La réception complète du caractère provoque la mise en 1 du bit SR0 (RDRF).

Le caractère est transféré dans le registre RDR (registre de réception de données) après suppression des bits de départ, de parité, de STOP, c'est à ce moment que les indicateurs d'état sont positionnés.

Le premier bit série reçu correspondra sur le bus de données à D0.

Il est possible de lire un caractère dans le registre RDR (registre de réception de données) pendant qu'un autre caractère est mémorisé dans le registre RSR (registre à décalage en réception).

Si pendant la réception il y a absence du premier bit STOP, une erreur de format est signalée par le positionnement à 1 du bit SR4 (FE), sans qu'il y ait perte de synchronisation de caractère.

retour au Sommaire | Index | Liens Rapides

6850 : Programmation Routine d'initialisation

L'ACIA doit être initialisé avant de transmettre ou de recevoir des données.

Le premier état d'initialisation, qui se produit automatiquement à la mise sous tension doit être supprimé par un MASTER RESET. Ceci se produit en mettant CR1 CR0 à %11.

Puis l'on programme le registre de contrôle pour utilisation désirée.

On peut à tout moment, configurer à nouveau l'interface pour obtenir un autre protocole de dialogue :

```
RGCR    EQU     $EC00       ;  Adrs reg.Contrôle
RGSR    EQU     $EC00       ;  Adrs reg.Etat, on peut écrite RGSR EQU RGCR
RGTX    EQU     $EC01       ;  Adrs reg.Transmission
RGRX    EQU     $EC01       ;  Adrs reg.Réception
                            ;
        LDA     #%00000011  ; init programmée "Master Reset"
        STA     RGCR        ;
        LDA     #%00000001  ; config pour un cas particulier
        STA     RGCR        ;
```

Après avoir fixé le protocole de transfert, on peut effectuer une transmission ou une réception de donnée, ou les deux simultanément. En inhibant les "modes par interruptions" :

retour au Sommaire | Liens Rapides

6850 : Programme d'initialisation pour une émission

La séquence débute par un test du bit SR1 :

- Si SR1 = 1 le registre de transmission est vide, le µp6809 peut alors placer la donnée à transmettre dans le registre de transmission RGTX
- Si SR1 = 0 la donnée précédente est en cours de transmission, il faut attendre.

```
        PSHS    A           ;  sauvarde de la donnée à transmettre
TXDATA  LDA     RGSR        ;  lesture du reg.Etat
        BITA    #%00000010  ;  test bit SR1
```

```
        BEQ     TXDATA      ;  Attendre que le reg.TX est vide
        PULS    A           ;  récup donnée à transmettre
        STA     RGTX        ;  transmettre
```

Si le registre B est disponible le programme ci-dessus devient (en enlevant PSHS et PULS)

```
TXDATA  LDB     RGSR        ;  lecture reg.Etat
        LSRB                ;  glisser le bit SR1 dans la
        LSRB                ;     retenue (bit C) du reg CC du 6809
        BCC     TXDATA      ;  sit bit SR1 = 0 alors attendre en bouclant
        STA     RGTX        ;  transmettre donnée
```

6850 : Programme d'initialisation pour une réception

retour au Sommaire | Liens Rapides

Il faut tester la validité d'une donnée reçue en veillant à l'absence d'erreur, en testant ces bits :

- SR4 pour les erreurs de format
- SR5 pour les erreurs de surcharge
- SR6 pour les erreurs de parité

Pour l'instant supposons que l'Emetteur et le Récepteur sont réglés sur un même protocole, dans ce cas la séquence de réception débute par un test de la présence d'une donnée dans le registre de réception RGRX à l'aide du bit SR0 du registre d'état.

```
RXDATA  LDA     RGSR        ;  lesture reg.Etat
        LSRA                ;  mettre le bit SR0 dans la retenu bit C
        BCC     RXDATA      ;  si SR0=0 attendre en bouclant
        LDA     RGRX        ;  lecture du reg.Réception
```

6850 : Programmation Routine de transmission

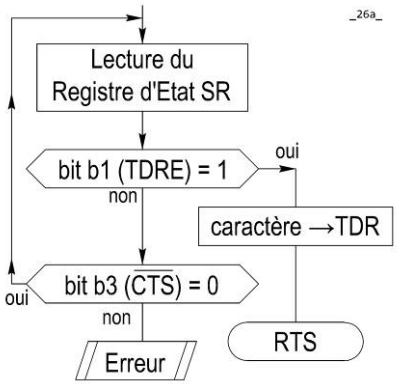
retour au Sommaire | Index | Liens Rapides

6850 : 1er exemple de transmission

Dans un premier temps, on teste que le registre de transmission est vide, si non il faut vérifier, avant de tester de nouveau le bit b1 (TDRE), que le bit b3 (CTS|) n'est pas à 1 ce qui inhiberait le bit b1 (TDRE).

Dans le cas d'une liaison avec un modem, cela signifierait une perte de porteuse.

```
DEBUT   LDA    ACIASR ;registre d'état -->Acc.A
        ASRA          ;décal droite bits, mise dans flag C
        ASRA          ;décal droite bits, mise dans flag C
        BCS    TRANS  ;test du bit b1 TDRE
        ASRA          ;décal droite bits, mise dans flag C
        ASRA          ;décal droite bits, mise dans flag C
        BCC    DEBUT  ;test du bit b3 CTS
        JMP    ERROR  ;
TRANS   STB    ACIATR ;caractère --> registre TDR
        RTS           ;
```



6850 : 2ième exemple de transmission

retour au Sommaire | Index | Liens Rapides

On veut envoyer une donnée stockée à l'adresse $1000 vers un télétype.
On ne travaille pas dans ce cas en interruption, il faut tester le contenu du registre de transmission afin de s'assurer qu'un caractère peut être envoyé.

1ière Solution `Liv6809ModeInterface` (test sur la transmission)

```
        LDA    #$03         ; %0000 0011 Master Reset b1 b0 = %11
        STA    ACIACR       ;
        LDA    #$45         ; %0100 0101 initialisation du CR
        STA    ACIACR       ;
        LDA    #%00000010   ;
ATENT   BITA   ACIASR       ;test sur bit b1 TDRE
        BEQ    ATENT        ;TDR est plein, on recommence le test
        STA    ACIATD       ;TDR est vide on charge le contenu
```

```
                        ;        de la mémoire $1000 dans TDR
```

2ième Solution `Liv6809et9365-66` (test sur la réception)

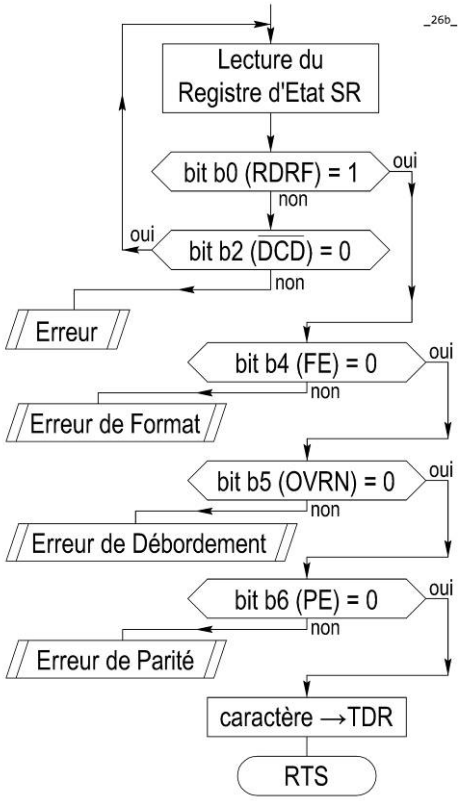
```
        LDA   #$03       ; %0000 0011 Master Reset b1 b0 = %11
        STA   ACIACR     ;
        LDA   #$45       ; %0100 0101 initialisation du CR
        STA   ACIACR     ;
GISSE   LDA   ACIASR     ; lecture du registre d"état
        LSRA             ; test sur SR0 RDR est plein
        BCC   GISSE      ; RDR est vide SR0=0, on recommence le test
        LDA   ACIARD     ; le contenu du registre de réception est
        STA   $1000      ; transféré à l'adresse $1000
```

Dans le cas d'une transmission, il n'y a pas bien sûr de test sur les bits PE, FE et OVRN.
La transmission se fait automatiquement à partir du chargement de TDR.

6850 : Exemples de programmation en réception

6850 : 1er exemple de Réception

```
DEBUT   LDA   ACIASR  ; registre d'état --> Acc.A
        ASRA          ; décal droite bits, mise dans flag C
        BCS   TESTFE  ; test du bit b0 RDRF
        ASRA          ; décal droite bits, mise dans flag C
        ASRA          ; décal droite bits, mise dans flag C
        BCC   DEBUT   ; test du bit b2 DCD
        BRA   ERROR   ;


TESTFE  ASRA          ; décal droite bits, mise dans flag C
        ASRA          ; décal droite bits, mise dans flag C
        BCC   TESTOV  ; test du bit b4 FE
        BRA   ERFORM  ;
TESTOV  ASRA          ;
        BCC   TESTPE  ; test du bit b5 OVRN
        BRA   ERROR   ;

TESTPE  ASRA          ;
        BCC   LECTUR  ; test du bit b6 PE
        BRA   ERPAR   ; branchement vers SPgm Err Parité
LECTUR  LDA   ACIADR  ; lecture du caractère
        RTS           ;
```



6850 : 2ième exemple de Réception

On veut recevoir d'un télétype une donnée et la stocker en mémoire à l'adresse $1000.

Il faut commencer par initialiser le 6850. Après une initialisation logicielle (MASTER RESET), on détermine le contenu du registre de contrôle CR en fonction des caractéristiques de la ligne 7 ou 8 bits, la parité, le nombre de bits de stop, etc.…

La fréquence RxClk est de 1760 Hz, il faut donc utiliser le diviseur par 16 pour obtenir la vitesse de 110 bauds (b1 b0 du registre CR = à %01).

Le télétype transmet des mots de 7bits, parité impaire et 2 bits de stop (b4 b3 b2 du registre CR = à %001).

La ligne de transmission est inactive, les interruptions du transmetteur sont inhibées (b6 b5 du registre CR = à %10).

Les interruptions du récepteur sont inhibées (b7 du registre CR = à %0).

Le programme d'initialisation du 6850 est donc le suivant :

```
        LDA    #%00000011   ; Master Reset
        STA    ACIACR       ;
        LDA    #%01000101   ; initialisation du CR
        STA    ACIACR       ;
```


Ce programme ne travaille pas en interruption, il faut tester le registre de réception pour savoir si une donnée a été envoyée.

Ce test se fait sur le bit b0 du registre d'état SR.

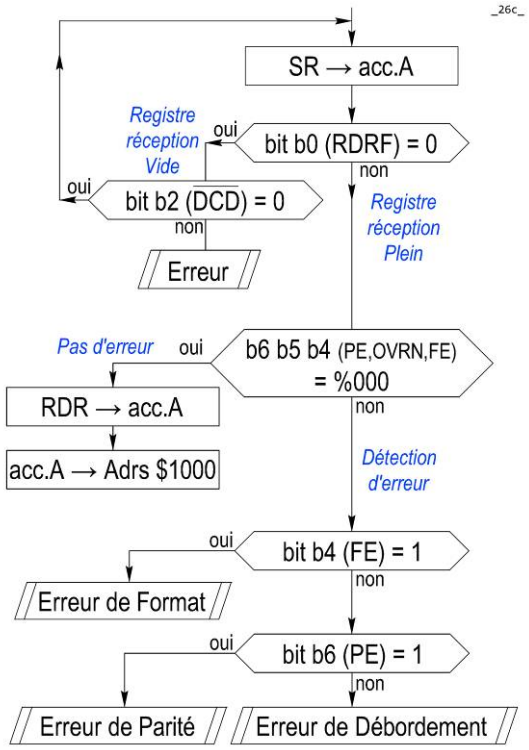
Le µp6809 reste en attente tant que le registre de réception est vide.

```
GISSE   LDA    ACIASR    ; test registre réception b0 SR = 1 alors RDR plein
        LSRA             ; décalage 1 bit vers la droite, mise dans le flag C
        BCC    GISSE     ; lag C = 0 alors RDR est vide, on recommence le test
        LDA    ACIARD    ; contenu registre de réception transféré
                         ;                     à l'adresse $1000
        STA    $1000     ;
```

On suppose ici que le message reçu ne présente pas d'erreur, dans certains cas il est nécessaire de faire des tests sur le registre d'état.
L'organigramme d'un tel programme est le suivant :

Le programme est alors plus complexe.

```
GISSE   LDA    ACIASR     ;test registre réception
                          ;    SR0=1 alors RDR plein
        LSRA               ;décal d'un bit --> droite
                          ;    et mise dans flag C
        BCC    LAINE      ;branch routine LAINE si
                          ;    bit b0 = zéro (RDR vide)
        BITA   #$70       ;test bits b6 b5 b4 PE,OVRN,FE
        BNE    SPERR      ;branch à SPER un
                          ;    des 3 bits est à 1
        LDA    ACIARD     ;le contenu du registre de
                          ;    réception est transferé
        STA    $1000      ;    à l'adresse $1000
LAINE   BITA   #4         ;Test sur le bit b2 DCD |
        BEQ    GISSE      ;DCD|=1, recommence le test
        BRA    SPDCD      ;DCD|=0, branche au SPgm SPDCD
                          ;
SPERR   BITA   #$10       ;test sur l'erreur de format
        BNE    SPFE       ;branche au SPgm SPFE
        BITA   #$40       ;test sur l'erreur de parité
        BNE    SPPE       ;branche au SPgm SPPE
        BRA    SPOVRN     ;branche au SPgm SPOVRN
```



retour au Sommaire | Index | Liens Rapides

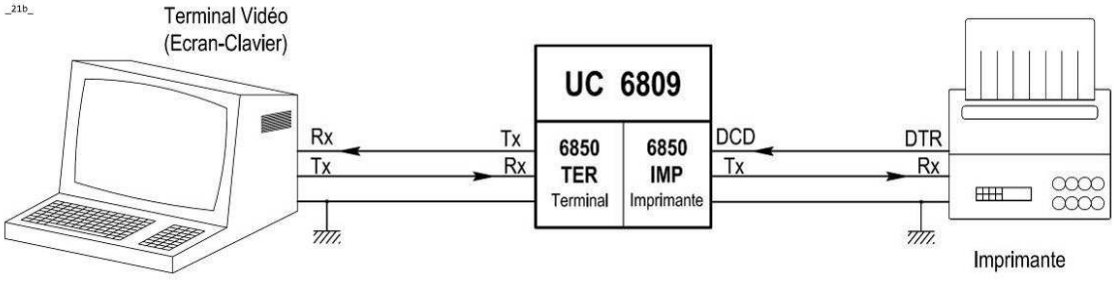
6850 : Exemple de Transmission des caractères Clavier vers Imprimante

La plupart des imprimantes ayant une liaison série n'ont pas tous les signaux de la norme RS-232-C.
L'imprimant peut être :
- Soit à réception seule.
- Soit d'un type plus complet possédant un clavier intégré. Dans ce cas les programmes en mode FULL-DUPLEX sont plus complexes.

Dans cet exemple nous utilisons l'imprimante comme une machine à écrire. L'opérateur envoi des données ASCII à partir d'un clavier vers une liaison série.

Après avoir analysé le caractère reçu du clavier, le µp6809 renvoie le caractère, éventuellement modifié ou codé différemment, vers l'imprimante.
Deux interfaces série : S0 fonctionnera en Transmission
S1 fonctionnera en Réception

Comme la vitesse de l'opérateur est très inférieure à celle réglée habituellement pour les lignes série, les protocoles de dialogues sont réduits à leur plus simple expression, c'est-à-dire au protocole START-STOP.



retour au Sommaire | Index | Liens Rapides

Du point de vue électrique :

- Les lignes "masse signal" et "masse châssis" seront normalement connectées.
- Une seule ligne de réception est nécessaire pour S1 (Réception provenant du terminal Ecran-Clavier).
- Une seule ligne de transmission est nécessaire pour S0 (Emission vers l'imprimante).
- Les broches DCD| doivent être maintenues constamment à l'état haut +12 volts. C'est-à-dire les entrées DCD| =0 pour ne pas inhiber les récepteurs. Cette condition doit être vérifiée s'il y a un mauvais fonctionnement.

Le programme de gestion ci-dessous s'occupe également de l'envoi d'un caractère à l'écran.

```
8000                    ORG     $8000       ;  adrs de chargement
                  ;------Adresses des registres ACIA IMP (imprimante)
          EC00    RCRIMP  EQU     $EC00       ;  reg CR Contrôle IMP
          EC00    RSRIMP  EQU     $EC00       ;  reg.SR Etat IMP
          EC01    RTXIMP  EQU     $EC01       ;  reg.TX Transmission IMP
```

```
                    ;------Adresses des registres ACIA TER (Terminal)
          EC80      RSRTER EQU    $EC80       ;  reg. SR Etat TER
          EC81      RRXTER EQU    $EC81       ;  reg. RX Réception clavier TER
          EC81      RTXTER EQU    $EC81       ;  reg. TX Transmission TER
                    ;------Configuration de l'imprimante
8000 C6   03                LDB    #%00000011  ;  Init Programmé
8002 F7   EC00              STB    RCRIMP      ;  Registre contrôle imprimante
8005 C6   01                LDB    #%00000001  ;  7bits, Paire, 2 Stop, 1/16
8007 F7   EC00              STB    RCRIMP      ;
                    ;------Attendre un caractère du clavier
800A F6   EC80      ATCLAV LDB    RSRTER      ;  regitre Etal Terminal
800D 54                     LSRB               ;  examen bit réception
800E 24   FA   800A         BCC    ATCLAV      ;  bit SR0=0 alors boucle d'attente
8010 B6   EC81              LDA    RRXTER      ;  donnée
8013 81   03                CMPA   #$03        ;  ETX ou CTRL C ?
8015 27   1C   8033         BEQ    FIN         ;  si oui alors FIN
                    ;------Envoi caractère vers imprimante 6850 IMP
8017 F6   EC00      TXIMP  LDB    RSRIMP      ;  registre Etat Transmission
801A C5   02                BITB   #%00000010  ;  examen bit transmission
801C 27   F9   8017         BEQ    TXIMP       ;  bit SR1 = 0 alors bouclage
                                               ;                  d'attente
801E B7   EC01              STA    RTXIMP      ;  caract à transm dans reg.trans
                    ;------Cette partie est optionnelle, utile pour clavier
                    ;------autonome fonctionnant sans écho sur l'écran
8021 81   20                CMPA   #$20        ;  caractère visualisable
8023 24   02   8027         BCC    *+4         ;
8025 86   2E                LDA    #$2E        ;  Mettre un point
                    ;------Transmette caractère sur écran
8027 F6   EC80      TXTER  LDB    RSRTER      ;  registre TER terminal
802A C5   02                BITB   #%00000010  ;
802C 27   F9   8027         BEQ    TXTER       ;
802E B7   EC81              STA    RTXTER      ;  transmettre
8031 20   D7   800A         BRA    ATCLAV      ;  Attendre un autre caractère
8033 3F             FIN     SWI                ;
```

retour au Sommaire | Index | Liens Rapides

Dans ce programme le branchement côté TERminal n'est pas initialisé, ni configuré car il doit l'être déjà la mise sous tension du système. S'il n'en est pas ainsi, aucune information ne pourrait être reçue à partir du clavier. Cependant, si le besoin s'en fait sentir, ce branchement peut être configuré à nouveau exactement comme pour le branchement côté IMPrimante (voir croquis ci-dessus).

La réception d'un caractère du clavier s'effectue sans vérification d'erreur à la réception.

Le caractère ETX ou CRTL-C de code $03 sert ici comme caractère de sortie permettant de quitter le programme. Il peut être changé en un autre caractère quelconque.
Si l'ACIA TER est configuré pour recevoir des mots codés sur 8 bits, tous les caractères ASCII de $00 à $FF peuvent être reconnus par le µp6809.

Comme l'ACIA IMP gérant l'imprimante est configuré sur un mode 7 bits, le 8^{ième} bit de donnée ne sera pas pris en compte et l'intervalle vu par l'imprimante se réduit à [$00 , $7F], $03 exclu.

Dans cet intervalle [$00 , $7F] on trouve :	Les caractères de contrôle	de : $00 à $1F
	Les caractères visualisables	de : $20 à 7F

La plupart des imprimantes reconnaissent les caractères de contrôle. Le programme ci-dessus envoie ces derniers normalement sur l'imprimante mais les remplace par des points sur l'écran de visualisation.

retour au Sommaire | Index | Liens Rapides

6850 : Exemple de Transmission des caractères vers Imprimante, Protocole DTR

Une imprimante qui imprime des caractères à une vitesse de 100 caractères par secondes nécessite une certaine vitesse de réception donnée. Si la vitesse de transfert est supérieure à celui du débit de l'imprimante, le buffer de réception de l'imprimante est très vite remplie par le µp6809.

Le µp6809 peut alors effectuer d'autres tâches en attendant la disponibilité du buffer. Ceci explique qu'un µp6809 peut gérer en parallèle plusieurs imprimantes. A notre échelle, tout se passe comme si elles fonctionnaient toutes simultanément.

Pour exécuter le protocole DTR, il faut connecter la broche DTR| de l'imprimante à la broche DCD| de l'interface du µp6809 (voir le schéma de l'exemple précédent).

Lorsque le buffer de réception l'imprimante est rempli, la broche DTR| passe à l'état Bas et positionne le bit SR2 du registre d'état.

Le programme de transmission doit arrêter l'envoi des données et attendre que cette broche DTR| bascule à l'état Haut avant le vidage complet du buffer.

Pour tester cet exemple, il faut disposer d'un texte quelconque dont la taille doit être supérieure à celle du buffer de l'imprimante, pour avoir au moins un changement d'état de la broche DTR|.

Comme le nombre de caractères à transmettre sert à arrêter le programme, ce paramètre doit être connu à l'avance.

Ici un sous-programme GRAPH crée cette zone de données à transmettre. L'arrangement de ces données permet d'examiner le graphisme des caractères de l'imprimante.

La figure ci-dessous donne les formes de caractères obtenues par l'exécution du programme ci après.

retour au Sommaire | Index | Liens Rapides

Pour des raisons d'espace et de lisibilité la représentation ci-dessous sous 3 colonnes ne correspond pas à ce que va sortir le programme, ce dernier établit une impression sur 2 colonnes, espacées de 6 caractères espaces :
- La première pour les codes de $20 à $4F.
- La seconde colonne pour les codes $50 à $7F.

```
20                  40  @@@@@@@@@@@@@@@@    60  ````````````````
21  !!!!!!!!!!!!!!!!    41  AAAAAAAAAAAAAAAA    61  aaaaaaaaaaaaaaaa
22  """"""""""""""""    42  BBBBBBBBBBBBBBBB    62  bbbbbbbbbbbbbbbb
23  ################    43  CCCCCCCCCCCCCCCC    63  cccccccccccccccc
24  $$$$$$$$$$$$$$$$    44  DDDDDDDDDDDDDDDD    64  dddddddddddddddd
25  %%%%%%%%%%%%%%%%    45  EEEEEEEEEEEEEEEE    65  eeeeeeeeeeeeeeee
26  &&&&&&&&&&&&&&&&    46  FFFFFFFFFFFFFFFF    66  ffffffffffffffff
27  ''''''''''''''''    47  GGGGGGGGGGGGGGGG    67  gggggggggggggggg
28  ((((((((((((((((    48  HHHHHHHHHHHHHHHH    68  hhhhhhhhhhhhhhhh
29  ))))))))))))))))    49  IIIIIIIIIIIIIIII    69  iiiiiiiiiiiiiiii
2A  ****************    4A  JJJJJJJJJJJJJJJJ    6A  jjjjjjjjjjjjjjjj
2B  ++++++++++++++++    4B  KKKKKKKKKKKKKKKK    6B  kkkkkkkkkkkkkkkk
2C  ,,,,,,,,,,,,,,,,    4C  LLLLLLLLLLLLLLLL    6C  llllllllllllllll
2D  ----------------    4D  MMMMMMMMMMMMMMMM    6D  mmmmmmmmmmmmmmmm
2E  ................    4E  NNNNNNNNNNNNNNNN    6E  nnnnnnnnnnnnnnnn
2F  ////////////////    4F  OOOOOOOOOOOOOOOO    6F  oooooooooooooooo
30  0000000000000000    50  PPPPPPPPPPPPPPPP    70  pppppppppppppppp
31  1111111111111111    51  QQQQQQQQQQQQQQQQ    71  qqqqqqqqqqqqqqqq
32  2222222222222222    52  RRRRRRRRRRRRRRRR    72  rrrrrrrrrrrrrrrr
33  3333333333333333    53  SSSSSSSSSSSSSSSS    73  ssssssssssssssss
34  4444444444444444    54  TTTTTTTTTTTTTTTT    74  tttttttttttttttt
35  5555555555555555    55  UUUUUUUUUUUUUUUU    75  uuuuuuuuuuuuuuuu
36  6666666666666666    56  VVVVVVVVVVVVVVVV    76  vvvvvvvvvvvvvvvv
37  7777777777777777    57  WWWWWWWWWWWWWWWW    77  wwwwwwwwwwwwwwww
38  8888888888888888    58  XXXXXXXXXXXXXXXX    78  xxxxxxxxxxxxxxxx
39  9999999999999999    59  YYYYYYYYYYYYYYYY    79  yyyyyyyyyyyyyyyy
3A  ::::::::::::::::    5A  ZZZZZZZZZZZZZZZZ    7A  zzzzzzzzzzzzzzzz
3B  ;;;;;;;;;;;;;;;;    5B  [[[[[[[[[[[[[[[[    7B  {{{{{{{{{{{{{{{{
3C  <<<<<<<<<<<<<<<<    5C  \\\\\\\\\\\\\\\\    7C  ||||||||||||||||
3D  ================    5D  ]]]]]]]]]]]]]]]]    7D  }}}}}}}}}}}}}}}}
3E  >>>>>>>>>>>>>>>>    5E  ^^^^^^^^^^^^^^^^    7E  ~~~~~~~~~~~~~~~~
3F  ????????????????    5F  ________________    7F
```

retour au Sommaire | Index | Liens Rapides

```
8000                        ORG     $8000       ;  adrs de chargement
                    ;------Adresses des registres ACIA IMP (imprimante)
          EC00      RCRIMP  EQU     $EC00       ;  reg CR Contrôle IMP
          EC00      RSRIMP  EQU     $EC00       ;  reg Etat IMP
          EC01      RTXIMP  EQU     $EC01       ;  reg Transmission IMP
          EC01      RRXIMP  EQU     $EC01       ;  reg Réception IMP
                    ;------Création de la zone des données à transmettre
8000 8E   8100              LDX     #$8100      ;  Adrs basse zone des données
8003 8D   28   802D         BSR     GRAPH       ;  branch S/P créat tab données
                    ;------Configuration de l'imprimante
8005 C6   03                LDB     #%00000011  ;  init programmée Master Reset
8007 F7   EC00              STB     RCRIMP      ;  reg.Conrôle IMP
800A C6   01                LDB     #%00000001  ;  7bits, Paire, 2 Stop, 1/16
800C F7   EC00              STB     RCRIMP      ;
                    ;------Corps principal du programme de gestion
800F A6   80        CARSVT  LDA     ,X+         ;  code à transmettre, CAR SuiVanT
8011 F6   EC00      VRDCD   LDB     RSRIMP      ;  reg.Etat, VéRif bit DCD
8014 C5   04                BITB    #%00000100  ;  état bit DCD ?
8016 27   08   8020         BEQ     TXIMP       ;  si bit SR2=0 transmission
                                                ;  pointe vers TransIMP. Séquence de
                                                ;  mise à 0 du bit DCD au prochain
                                                ;  passage à état Haut de la lgn DCD
8018 F6   EC00              LDB     RSRIMP      ;  lecture reg.Etat
801B F6   EC01              LDB     RRXIMP      ;  lecture reg.Réception
```

```
801E 20   F1   8011        BRA    VRDCD        ;  tester de nouveau bit DCD
8020 C5   02        TXIMP   BITB   #%00000010   ;  état bit de transmission ?
8022 27   ED   8011        BEQ    VRDCD        ;  si SR1=0 boucle attente
8024 B7   EC01              STA    RTXIMP       ;  transmettre
8027 8C   8D00              CMPX   #$8D00       ;  fin buffer de transmission ?
802A 26   E3   800F         BNE    CARSVT       ;  sinon, car suivant
802C 3F                     SWI                 ;
```

retour au Sommaire | Index | Liens Rapides

```
                    ;*******************************************************
                    ;  S/P de création d'une zone de données pour étudier le
                    ;  graphisme des caractères d'une imprimante
                    ;    Nom d'appel : GRAPH
                    ;    Entrée :   X: adrs basse zonne de données
                    ;    Sortie :   aucun registre affecté
                    ;               longueur données : 3072 = $0C00 octets
                    ;  Extension pile système : 11 = $0B octets
                    ;*******************************************************
802D 34   17        GRAPH   PSHS   X,B,A,CC     ;
802F 32   7E                LEAS   -2,S         ;  réserve données temporaire
8031 C6   30                LDB    #$30         ;  Nbre de ligne du texte
8033 E7   E4                STB    0,S          ;
8035 86   20                LDA    #$20         ;  code départ
8037 A7   61                STA    1,S          ;
8039 C6   10        LNSVT   LDB    #16          ;  Nbre d'espace en début de ligne
                                                ;  LigNe SuiVanTe
803B 86   20                LDA    #$20         ;  code car espace
803D A7   80                STA    ,X+          ;
803F 5A                     DECB                ;
8040 26   FB   803D         BNE    *-3          ;
8042 A6   61                LDA    1,S          ;  Code à transmettre
8044 8D   49   808F         BSR    BINHEX       ;  conversion Binaire-Hexa
8046 ED   81                STD    0,X++        ;  stockage 2 car. Convertis
8048 CC   2020              LDD    #$2020       ;  2 espaces entre les colonnes
804B ED   81                STD    0,X++        ;
804D C6   10                LDB    #16          ;  Nbre Car. A visualiser
804F A6   61                LDA    1,S          ;  code à visualiser
8051 A7   80                STA    0,X+         ;
8053 5A                     DECB                ;
8054 26   FB   8051         BNE    *-3          ;
8056 C6   06                LDB    #6           ;  6 espaces entre les deux tableau
8058 86   20                LDA    #$20         ;  code car. Espace
805A A7   80                STA    0,X+         ;
805C 5A                     DECB                ;
805D 26   FB   805A         BNE    *-3          ;
```

voir ci-après pour ces 4 lignes

retour au Sommaire | Index | Liens Rapides

```
                    ;------la deuxième colonne commence à partir
                    ;------du code $50 (code première colonne + $30)
805F A6   61                LDA    1,S          ;  code deuxième colonne
8061 8B   30                ADDA   #$30         ;
8063 8D   2A   808F         BSR    BINHEX       ;  conversion Binaire-Hexa
8065 ED   81                STD    0,X++        ;
8067 CC   2020              LDD    #$2020       ;  2 car espace
806A ED   81                STD    0,X++        ;
806C C6   10                LDB    #16          ;
806E A6   61                LDA    1,S          ;  code 1ière colonne
8070 8B   30                ADDA   #$30         ;  code à visualiser
8072 A7   80                STA    0,X+         ;
8074 5A                     DECB                ;
8075 26   FB   8072         BNE    *-3          ;
                    ;------Mettre en fin de ligne "CR" return et "LF" line feed
8077 86   0D                LDA    #$0D         ;  Car Return
8079 A7   80                STA    0,X+         ;
807B 86   0A                LDA    #$0A         ;  Car Line Feed
807D A7   80                STA    0,X+         ;
                    ;------Test de fin de programme (48 lignes)
807F A6   61                LDA    1,S          ;  Actualiser code pour lgn suivante
8081 4C                     INCA                ;
8082 A7   61                STA    1,S          ;
8084 E6   E4                LDB    0,S          ;  Actualiser Nbre de lgn restant
8086 5A                     DECB                ;
8087 E7   E4                STB    0,S          ;
8089 26   AE   8039         BNE    LNSVT        ;  écrire données ligne suivante
808B 32   62                LEAS   2,S          ;
808D 35   97                PULS   PC,X,B,A,CC  ; fin du S/P GRAPH
```

```
                    ;*********************************************************
                    ;  S/P conversion d'un octet binaire en 2 car. ASCII en Hexa
                    ;  Nom d'appel : BINHEX
                    ;     Entrée :   A : Donnée à convertir
                    ;     Sortie :   A : Code ASCII hexa 4 bits poids fort
                    ;                B : Code ASCII hexa 4 bits poids faible
                    ;                (ex : $3A sera converti en $33 $41
                    ;*********************************************************
808F 1F   89        BINHEX TFR    A,B          ;
8091 44                    LSRA                ; ----obtenir 4 bits poids fort
9092 44                    LSRA                ; -- |
9093 44                    LSRA                ; -- |
9094 44                    LSRA                ; -- |
8095 81   0A               CMPA   #$A          ;  < 10 ?
8097 25   02   809B        BLO    *+4          ;
8099 8B   07               ADDA   #7           ;
809B 8B   30               ADDA   #$30         ;
809D C4   0F               ADDB   #$0F         ;  4 bits poids faible
809F C1   0A               CMPB   #$A          ;  < 10 ?
80A1 25   02   80A5        BLO    *+4          ;
80A3 CB   07               ADDB   #7           ;
80A5 CB   30               ADDB   #$30         ;
80A7 39                    RTS                 ;  fin du S/P BINHEX
```

Le programme comporte 2 parties : - Gestion des transferts de données suivant le protocole DTR.
- Sous-programme de création de la base de données.

Dans le corps principal du programme de gestion, on observe un test systématique du bit SR2 du registre d'état pour détecter un éventuel changement d'état de la broche DCD| (transition de +12 volts → -12 volts).

Lorsque le bit SR1 = 1 la séquence aux adresse $8018 et $801B :

```
     LDB    RSRIMP      ;  lecture reg.Etat
     LDB    RRXIMP      ;  lecture reg.Réception
```

Effectue une remise à 0 "potentielle" du bit DCD. Ce bit sera effectivement remis à 0 lorsque la broche DCD| change d'état (transition de -12 volts → +12 volts) indiquant que le buffer de réception est à nouveau prêt pour recevoir d'autres données.

Durant la période d'attente, on peut orienter le µp6809 vers le traitement d'une autre tâche, c'est ce qui est effectivement réalisé dans les systèmes multitâches.

Les quatre lignes du programme précédent

```
8031 C6   30                LDB    #$30         ;  Nbre de ligne du texte
8033 E7   E4                STB    0,S          ;
8035 86   20                LDA    #$20         ;  code départ
8037 A7   61                STA    1,S          ;
```

Peuvent être Optimisé par les deux lignes suivantes

```
                            LDD    #$3020       ; Nbre de ligne du texte
                            STD    0,S          ;          + code départ
```


6850 : Exemple de Transmission des caractères vers Imprimante, Protocole ETX-ACK

Dans le protocole ETX-ACK (End of text, Acknowledge), les données sont envoyées par blocs.
La taille du blog transmis doit-être en tout cas inférieure à celle du buffer de réception de l'imprimante.

Le dernier caractère d'un bloc doit-être le code ETX $03.
De son côté, l'imprimante exploite les caractères reçus à sa propre vitesse.

A la détection du code ETX $03, le récepteur renvoie au transmetteur le code à ACK $06 lui indiquant sa disponibilité.

Pour tester cette application, seule les broches de transmission Tx et de réception Rx doivent être connectées électriquement.

L'entrée DCD| du système processeur-interface doit être maintenue à 0 (ligne DCD au niveau Haut +12 volts).

Pour la majorité des imprimantes, cette ligne peut être connectée à la broche DTR|.

Ici, on surveille l'état du bit de réception SR0 dans le registre d'état, qui atteste la présence d'une donnée en retour.

Rappelons que dans le protocole DTR, la poignée de main et basée sur l'état du bit SR2 qui traduit l'état de l'entrée DCD|.


```
                        ;*******************************************************
                        ;    Transmission de blocs de données vers une imprimante
                        ;    suivant le protocole ETX/ACK
                        ;*******************************************************
8000                            ORG     $8000       ;  adrs de chargement
                        ;------Adresses des registres ACIA IMP (imprimante)
          EC00          RCRIMP  EQU     $EC00       ;  reg CR Contrôle IMP
          EC00          RSRIMP  EQU     $EC00       ;  reg Etat IMP
          EC01          RTXIMP  EQU     $EC01       ;  reg Transmission IMP
          EC01          RRXIMP  EQU     $EC01       ;  reg Réception IMP
                        ;------Configuration de l'imprimante
8000 C6   03                    LDB     #%00000011  ; init programmée Master Reset
8002 F7   EC00                  STB     RCRIMP      ;  reg.Conrôle IMP
8005 C6   01                    LDB     #%00000001  ; 7bits, Paire, 2 Stop, 1/16
8007 F7   EC00                  STB     RCRIMP      ;
                        ;------transmission. Les données à transmettre se
                        ;------trouve à partir de l'adresse $8100
800A 8E   8100                  LDX     #$8100      ;
800D C6   06                    LDB     #6          ;  Nb. Blocs à transmettre
800F 8D   01    8012            BSR     TXIMP       ;  Transmission
8011 3F                         SWI                 ;
                        ;*******************************************************
                        ;  S/P de transmission de blocs de données
                        ;  suivant le protocole ETX/ACK    Nom d'appel : TXIMP
                        ;  Entrée :   X: adresse basse de données
                        ;             B: Nombre de blocs de 512 octets
                        ;  Sortie :   aucun registre modifié
                        ;  Extention pile système: 12= $0C octets
                        ;*******************************************************
8012 34   37            TXIMP  PSHS    Y,X,B,A,CC  ;
8014 32   7F                    LEAS    -1,S        ;  réserve pour compteur blocs
8016 E7   E4                    STB     0,S         ;  compteur blocs
8018 B6   EC01          RAZSR0 LDA     RRXIMP      ;  remise à 0 du but SR0, tester la
                                                   ;  disponibilité de l'imprimante
801B 86   03                    LDA     #$03        ;  caractère ETX
801D 8D   25    8044            BSR     CARIMP      ;  transmettre un caractère
801F B6   EC00          BLCSVT LDA     RSRIMP      ;  reg.Etat, BLoCs SuiVanT
8022 44                         LSRA                ;  examen bit SR0
8023 24   FA    801F            BCC     BLCSVT      ;  si SR0=0 bouclage BLoCs SuiVanT
8025 B6   EC01                  LDA     RRXIMP      ;  donnée reçue
8028 81   06                    CMPA    #$06        ;  caract ACK ?
802A 26   EC    8018            BNE     RAZSR0      ;  sinon, nouvel essai,
                                                   ;  imprim prète, trnasmission
802C 108E 0200                  LDY     #512        ;  longueur d'un bloc en octets
8030 A6   80            CARSVT LDA     ,X+         ;  code à transmettre
8032 8D   10    8044            BSR     CARIMP      ;  transmettre un caractère
8034 31   3F                    LEAY    -1,Y        ;  Nb Car du bloc épuisé ?
8036 26   F8    8030            BNE     CARSVT      ;  sinon, CARactère SuiVanT
8038 86   03                    LDA     #$03        ;  caractère ETX
803A 8D   08    8044            BSR     CARIMP      ;  transmettre
803C 6A   E4                    DEC     0,S         ;  compteur blocs épuisé ?
803E 26   DF    801F            BNE     BLCSVT      ;  sinon, BLoCs SuiVanT
8040 32   61                    LEAS    1,S         ;
8042 35   97                    PULS    PC,X,B,A,CC ; Fin du S/P TXIMP
                        ;*******************************************************
                        ;  S/P transmission d'un caractère vers imprimante
                        ;  Nom d'appel : CARIMP
                        ;  Entrée :   A: Code à transmettre
                        ;  Sortie :   aucun registre modifié
                        ;  encombrement pile système : 3 octets
                        ;*******************************************************
8044 34   02            CARIMP PSHS    A           ;
8046 B6   EC00                  LDA     RSRIMP      ;  reg.Etat
8049 85   02                    BITA    #%00000010  ; examen bit transmission
804B 27   F9    8046            BEQ     *-5         ;  si bit SR1=0, attendre
804D 35   02                    PULS    A           ;  code à transmettre
804F B7   EC01                  STA     RRXIMP      ;  registre transmission IMP
8052 39                         RTS                 ;  Fin S/P CARIMP
```


Après les opérations habituelles de sauvegarde, le sous-programme TXIMP effectue une lecture factice du registre de réception, pour mettre à 0 le bit SR0, qui peut-être mit accidentellement à 1 par un autre programme.

Par précaution, le système teste d'abord la disponibilité du récepteur en envoyant le caractère ETX.

Le caractère renvoyé doit-être un à ACK. S'il n'en n'est pas ainsi, on peut s'orienter vers une séquence d'avertissement.

Ici le programme est simplifié, la boucle BNE RAZSR0 se referme indéfiniment si le caractère retourné n'est pas un à ACK.

Si le buffer de l'imprimante possède une taille supérieure à 512 octets, il faudra modifier la ligne
```
802C 108E 0200            LDY    #512        ;
```

La transmission se termine par l'envoi d'un caractère ETX à la fin de chaque bloc. Le test de fin de programme est basé sur le nombre total de blocs à transmettre.

retour au Sommaire | Index | Liens Rapides

6850 : Exemple de Transmission des caractères vers Imprimante, Protocole XON-XOFF

Dans le protocole DTR, la disponibilité ou l'indisponibilité du buffer de réception sont traduites par l'activation d'une broche électrique DTR|.

Dans le protocole XON-XOFF le transmetteur arrête le transfert à la réception du code XOFF ou DC3 (Device Control 3 : code ASCII $13).

Lorsque le buffer de réception est prêt à recevoir, le code DC1 (code ASCII $11) est retourné au transmetteur.

Comme dans le protocole ETX-ACK, le transfert met en œuvre une broche de transmission et une broche de réception pour le retour des codes de contrôle.

La différence est qu'ici la longueur du bloc de données est déterminée automatiquement par le récepteur et on n'a pas besoin de la fixer dans le programme assembleur comme pour le cas de ETX-ACK.

Le fonctionnement de XON-XOFF se rapproche plutôt du protocole DTR; seulement ici, le transmetteur doit reconnaître une donnée issue du récepteur au lieu de lire l'état électrique d'une broche.

retour au Sommaire | Index | Liens Rapides

```
                    ;******************************************************
                    ;    Transmission de blocs de données vers une imprimante
                    ;    suivant le protocole XON-XOFF ou DC1/DC3
                    ;******************************************************
8000                        ORG    $8000       ;  adrs de chargement
                    ;------Adresses des registres ACIA IMP (imprimante)
          EC00      RCRIMP EQU    $EC00       ;  reg CR Contrôle IMP
          EC00      RSRIMP EQU    $EC00       ;  reg Etat IMP
          EC01      RTXIMP EQU    $EC01       ;  reg Transmission IMP
          EC01      RRXIMP EQU    $EC01       ;  reg Réception IMP
                    ;------Configuration de l'imprimante
8000 C6   03                LDB    #%00000011  ; init programmée Master Reset
8002 F7   EC00              STB    RCRIMP      ;  reg.Conrôle IMP
8005 C6   01                LDB    #%00000001  ; 7bits, Paire, 2 Stop, 1/16
8007 F7   EC00              STB    RCRIMP      ;
                    ;------Transmission. Les données à transmettre se trouvent
                    ;------à partir de l'adresse $8100
800A 8E   8100              LDX    #$8100      ;
800D 108E 0C00              LDY    #$C00       ;  Nb Car à transmettre
8011 8D   01   8014         BSR    TXIMP       ;  transmission
8013 3F                     SWI                ;
                    ;******************************************************
                    ;  S/P de transmission de donnée
                    ;  Protocole XON / XOFF      Nom d'appel: TXIMP
                    ;  Entrée :   X: Adresse basse zone de données
                    ;             Y: Nombre de car à transmettre
                    ;  Sortie : Aucun registre modifié
                    ;  Extension pile système : 9 octets
                    ;******************************************************
8014 34   37        TXIMP  PSHS   Y,X,B,A,CC  ;
8016 B6   EC01              LDA    RRXIMP      ; mise à zéro bit réception
                    ;------tranqmission avec surveillance du retour de
                    ;------code XOFF (DC3  $13) avertissant buffer plein
8019 F6   EC00      CARSVT LDB    RSRIMP      ; reg.Etat CARactère SuiVanT
801C 56                     RORB               ; examen bit réception
801D 25   0E   802D         BCS    STOPTD      ; si bit SR0=1, arrêter la trans.
801F 56                     RORB               ; examen bit transmission
8020 24   F7   8019         BCC    CARSVT      ; si bit SR1=0, recommencer test
```

```
8022 A6   80              LDA    ,X+          ; code à transmettre
8024 B7   EC01            STA    RTXIMP       ; transmettre
8027 31   3F              LEAY   -1,Y         ; Nb Caractères épuisé ?
8029 26   EE   8019       BNE    CARSVT       ; sinon, continuer
802B 35   B7              PULS   PC,Y,X,B,A,CC   ; Fin S/P TXIMP
```

retour au Sommaire | Index | Liens Rapides

```
                  ;------détection d'un code XOFF
802D F6   EC01    STOPTD LDB    RRXIMP       ; Effacer bit réception
                  ;------Attente du code XON (Disponibilité)
8030 F6   EC00    ATXON  LDB    RSRIMP       ; reg.Etat
8033 54                  LSRB                ; examen bit réception
8034 24   FA   8030      BCC    ATXON        ; si bit SR0=0, ATtente XON
8036 F6   EC01           LDB    RRXIMP       ; lecture reg réception
8039 C1   11             CMPB   #$11         ; XON ?
803B 27   DC   8019      BEQ    CARSVT       ; code correct, continer.
                                              ; Si le code reçu est incorrect
                                              ; -> possibilité d'écrire un S/P traitant
                                              ; cette erreur. Ceci n'est utile que pour
                                              ; une IMP munie d'un clavier Ici, tout
                                              ; code reçu est accepté
803D 20   DA   8019      BRA    CARSVT       ; instruct. à modifier éventuellement
```

retour au Sommaire | Index | Liens Rapides

Dans ce programme, lors de la rencontre d'un code en retour en cours de transmission, ce code est interprété automatiquement comme un XOFF et il n'y a pas de contrôle.
Ceci n'est nullement gênant si l'imprimante doit travailler en mode réception seule.

Pour les imprimantes dotées d'un clavier, on peut intervenir dans le système à partir du clavier. Dans ce cas spécifique, chaque code retourné au transmetteur comporte une signification précise et une identification systématique s'impose. Pour XON également, le contrôle s'avère inutile pour les mêmes raisons.

retour au Sommaire | Index | Liens Rapides

6850 : Exemple de Réception des données, avec diverses vérifications, Contrôle ligne RTS

Dans les applications précédentes, la séquence de réception est présentée dans sa forme la plus simple, sans aucune vérification. Dans la majorité des systèmes préréglés, elle suffit amplement.

Pour les autres applications particulières, le 6850 possède de nombreuses options de contrôle réception :

Surcharge : La condition de surcharge se produit lorsque les données arrivent en flots et le programme assembleur se chargeant de la réception ne lit pas rapidement les données reçues pour la vitesse réglée.

Parité et Format : La parité est le format du mot reçu sont automatiquement échantillonnée par l'interface et toute erreur sera signalée par la mise à 1 des bits SR6 ou SR4.
Le bit de parité n'apparaît jamais dans le registre de réception, même avec un format 7 bits. Le contrôle est effectué au niveau "matériel" appelé aussi "mode câblé" pour simplifier la programmation au niveau utilisateur.

Etat de la ligne DCD : Elle traduit l'état de la broche DCD| dans un mode de transmission par modem (DCD Data Carrier Detect traduis par "Perte porteuse de données").
Il avertit le récepteur de l'absence de la porteuse modulée. Cette de entrée est utile en transmission avec le protocole DTR. Elle sert de ligne de dialogue principale au mode "poignée de main câblée". L'usage de cette broche d'entrée DCD et du bit associé SR2 est en réalité très générale. Elle sert dans toutes circonstances où l'on a besoin d'avertir le µp6809 d'un changement quelconque dans les conditions externes.

retour au Sommaire | Index | Liens Rapides

De son côté, le 6850 peut piloter un périphérique avec la broche RTS| en programmant les bit CR5 et CR6.

Dans le programme qui suit le 6850 joue le rôle de récepteur. Il supervise le transfert avec la broche de sortie RTS|. Lorsque RTS| = 0, le récepteur est prêt à recevoir. Il arrêtera le transfert en avertissant l'émetteur avec RTS| = 1.

```
                  ;*****************************************************
                  ;  réception de donnée avec ligne série
                  ;  vérifications diverses (format, parité, surcharge)
                  ;  contrôle du transmetteur par ligne RTS
                  ;*****************************************************
8000                      ORG    $8000        ;
                  ;------adresses des registres du 6850
          EC00    RCRIMP  EQU    $EC00        ;  reg. contrôle
          EC00    RSRIMP  EQU    $EC00        ;  reg. Etat
          EC00    RRXIMP  EQU    $EC01        ;  reg. Réception
8000 C6   43              LDB    #%01000011   ; initialisation programmée avec
                                              ;  RTS=1 (ligne RTS à -12v)
8002 F7   EC00            STB    RCRIMP       ;  registre contrôle
8005 8E   8100            LDX    #$8100       ;  adrs rangement données
```

```
8008 108E 000A            LDY    #10          ;  nb d'octet à recevoir
800C 31   3F              LEAY   -1,Y         ;  écarter dernière donnée
                    ;------Configurer le mode de réception et actionner ligne RTS
800E C6   01              LDB    #%00000001   ; 7 Bits,parité paire,2 stop, clock 1/16
                                              ; + RTS |=0 (ligne RTS= +12v)
8010 F7   EC00            STB    RCRIMP       ;
                    ;------Réception données avec vérifications diverses
8013 F6   EC00      ATDON LDB    RSRIMP       ;  reg. Etat
8016 C5   01              BITB   #%00000001   ; Examen bit réception
8018 26   06   8020      BNE    VERIF        ;  SR0=1, alors vérification
801A C5   04              BITB   #%00000100   ; examen bit DCD
801C 27   F5   8013      BEQ    ATDON        ;  SR2=0, alors ATtente DONnée
801E 20   2F   804F      BRA    TMDCD        ;  --> TraiteMent DCD
```

retour au Sommaire | Index | Liens Rapides

```
                    ;------Vérification format, parité, surcharge
8020 C5   70        VERIF BITB   #%01110000   ; test global
8022 27   0E   8032      BEQ    RXDON        ;  sans erreur, --> Rx réception donnée
8024 C5   10              BITB   #%00010000   ; erreur format ?
8026 27   02   802A      BEQ    *+4          ;  sinon test suivant adrs +4
8028 20   22   804C      BRA    ERFMT        ;  --> erreur format
802A C5   40              BITB   #%01000000   ; erreur parité ?
802C 27   02   8030      BEQ    *+4          ;  sinon --> erreur surcharge
802E 20   1D   804D      BRA    ERPAR        ;  --> erreur PARité
8030 20   1C   804E      BRA    ERSCH        ;  --> erreur SurCHarge
                    ;------Aucune erreur détectée, lecture donnée
8032 B6   EC01      RXDON LDA    RRXIMP       ;  lecture reg. réception
8035 A7   80              STA    ,X+          ;  rangement donnée épuisé ?
8037 31   3F              LEAY   -1,Y         ;  Nb données épuisé ?
8039 26   D8   8013      BNE    ATDON        ;
                    ;------Inhibition de l'émetteur
803B C6   41              LDB    #%01000001   ; RTS |=1 (ligne RTS à -12v)
803D F7   EC00            STB    RCRIMP       ;  reg. contrôle
                    ;------Lecture dernière donnée en cours
8040 F6   EC00            LDB    RSRIMP       ;  reg. Etat
8043 56                   RORB                ;
8044 24   FA   8040      BCC    *-4          ;
8046 B6   EC01            LDA    RRXIMP       ;
8049 A7   80              STA    ,X+          ;  rangement
804B 3F                   SWI                 ;
                    ;------Si une ou plusieurs erreurs sont présentes,
                    ;------elles sont indiquées dans le registre B
804C 3F             ERFMT SWI                 ;
804D 3F             ERPAR SWI                 ;
804E 3F             ERSCH SWI                 ;
804F 3F             TMDCD SWI                 ;
```

retour au Sommaire | Index | Liens Rapides

Après l'initialisation programmée de l'interface avec RTS| = 1 et après changement des paramètres d'acquisition, le récepteur autorise le transfert en mettant la broche de sortie RTS| = 0 (broche RTS| à +12volts).

Le récepteur teste ensuite le bit de réception et doit trouver normalement SR0=0 indiquant l'absence de donnée. La broche d'entrée DCD| doit être maintenue constamment à 0 (broche DCD| à +12v) pour ne pas inhiber le bit de réception. Le récepteur entre normalement la phase d'attente de réception.

L'opérateur peut alors lancer le programme de transfert du côté de transmetteurs. Ce dernier détecte l'autorisation de transfert en examinant l'état de la broche RTS|.

Si la donnée est disponible, le récepteur vérifie le format, la parité et la surcharge.
Le test est d'abord global. Si une ou plusieurs erreurs sont présentes, la discrimination s'effectue avec une priorité accordée au format, puis à la parité.
Dans ce programme la détection d'une erreur provoque une interruption logicielle SWI.

Il faut vérifier alors le mot de configuration de l'interface et la division de l'horloge à la fois du côté transmetteur et récepteur.

Si la donné est correcte, le µp6809 lit le registre de réception, range la donnée en mémoire puis vérifie si le nombre de caractères demandé est épuisé.

Lorsque le nombre de caractères, moins 1, est atteint, le récepteur actionne la broche RTS| pour avertir le transmetteur. Or, tout ceci se produit pendant l'envoi de la dernière donnée dans le registre à décalage du récepteur, d'où l'utilité de la dernière séquence.

6840 : Généralités

retour au Sommaire | Index | Sommaire Principal | Liens Rapides

Le principe du 6840 est simple : un compteur 16 bits initialement mis à une certaine valeur se décrémente au rythme d'une horloge externe ou provenant du 6809.
Le 6840 est un temporisateur programmable, qui peut être utilisé comme :

- Générateur d'interruption.
- Générateur de signaux périodiques : Multivibrateur Astable (train d'impulsions de durée et de période programmable).
- Générateur de signaux non périodique : Monostable.
- Chronomètre : mesure d'intervalle de temps (Fréquencemètre).
- Fréquencemètre : mesure de durée d'impulsion (largeur d'impulsion).
- Compteur d'événements.

Il comporte essentiellement 3 compteurs à 16 bits, dont le fonctionnement est commandé par 3 registres de commande.

Le 6840 est mono tension (0, +5v), sa consommation est de 500 mW environ.

Les 3 temporisateurs peuvent fonctionner simultanément, ce qui donne beaucoup de souplesse circuit.
Ces trois temporisateurs peuvent être bouclés les uns sur les autres.

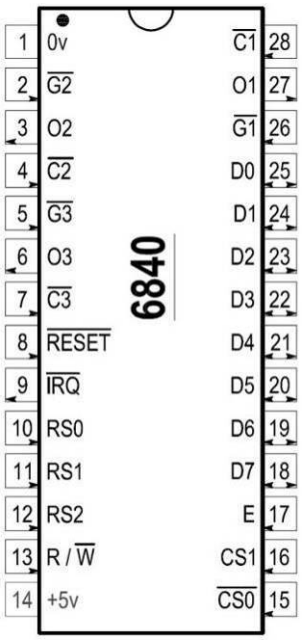
On peut ainsi simultanément générer un signal carré, générer un signal unique et faire une mesure de durée.

Autres caractéristiques essentielles

- Fonctionnement à partir de l'horloge du µp6809 ou d'une horloge externe.
- 3 entrées C| pour horloges externes.
- 3 entrées G| de déclenchement sont synchronisées à l'intérieur du timer.
- Fréquence maximum externe 4 Mhz (uniquement sur le timer 3).
- 3 sorties masquables.
- Les compteurs accessibles par lecture indiquent le temps qui sépare de la fin de la période programmée.

6840 : Brochage

retour au Sommaire | Index | Liens Rapides



6840 : Organisation Externe

retour au Sommaire | Index | Liens Rapides

Les échanges avec le µp6809 se font par l'intermédiaire :

6840 : D0 à D7 :

Du bus de données, si ces broches ne sont utilisées elles se retrouvent dans l'état haute impédance.

Ce bus est utilisé pour programmer :
- Les 3 registres de contrôle CR1, CR2 et CR3
- Les registres tampon LSB et MSB de chaque timer
- Lire les registres tampon ou le registre d'état.

6840 : Les broches CS0 | et CS1 :

De 2 lignes de validation de boîtier qui permettent l'adressage physique du boîtier.

6840 : Les broches RS0, RS1, RS2 :

3 entrées de sélection de registre.
Les combinaisons de ces 3 broches permettent de sélectionner les registres internes (8 positions mémoire).

6840 : La broche E :

De l'entrée (Enable) qui reçoit l'horloge Φ2 du µp6809, afin de synchroniser et d'activer les échanges.

6840 : La broche R/W | :

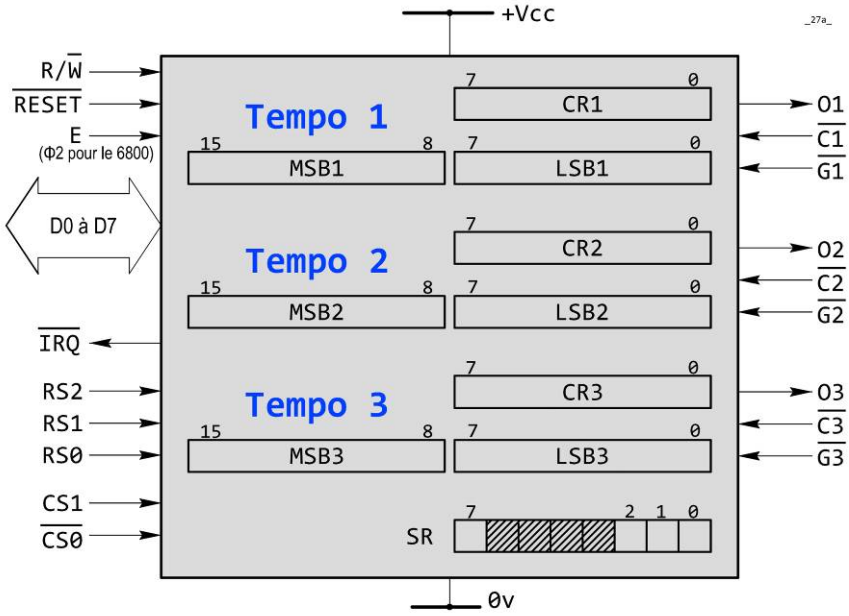
De l'entrée Lecture / écriture (Read / Write) qui fixe le sens des transferts, écriture du temporisateur ou lecture.

6840 : La broche IRQ | :

D'une ligne d'interruption à drain ouvert, donc supportant le OU câblé, et qui permet d'interrompre l'exécution d'un programme.
Reliée à la broche IRQ|, FIRQ| ou NMI| du µp6809.

retour au Sommaire | Index | Liens Rapides

6840 : Organisation Externe Schématique



6840 : Organisation Externe Liaisons avec la périphérie

retour au Sommaire | Liens Rapides | Index

6840 : Entrées horloges externes : C1 |, C2 |, C3 |

Ces entrées sont compatibles avec les niveaux TTL et peuvent varier du continu à la valeur de l'horloge du µp6809.

La fréquence appliquée peut aller du continu à la fréquence d'horloge appliquée sur la broche E (ENABLE).

Les compteurs internes du 6840 peuvent être activés par l'horloge du µp6809 ou par des horloges externes. Chaque temporisateur possède sa propre entré d'horloge externe.

L'utilisation de ces entrées nécessite impérativement la prise en considération des paramètres dynamiques du 6840. L'horloge du temporisateur numéro 3 peut être divisée par 8, mais elle est traitée de façon identique à C1| ou C2|

6840 : Entrées GATE | : G1 |, G2 |, G3 |

Ces entrées sont compatibles avec des signaux asynchrones TTL, donc non synchrone de l'horloge du µp6809.

Trois impulsions d'horloge sont nécessaires pour les prendre en compte.
Ces entrées sont directement liées aux compteurs 16 bits.

retour au Sommaire | Liens Rapides | Index

6840 : Sorties des temporisateurs : O1, O2, O3

Chaque sortie peut commander 2 charges TTL et délivre un signal défini quand le temporisateur travaille en astable ou en monostable.

Une sortie non validée reste à 0

En mode intervalle de temps (chronomètre ou fréquencemètre), des signaux apparaissent en sortie si CRx7 = 1, mais leur forme est imprévisible.

6840 : Organisation Interne

Le µp6809 doit donc adresser 10 registres de 8 bits différents :

- 6 registres de 8 bits pour les timers **MSB1, LSB1, MSB2, LSB2, MSB3, LSB3**.
- 3 registres de 8 bits **CR1 CR2 CR3** pour les contrôles
- 1 registre d'état **SR**

6840 : MSB1, LSB1, MSB2, LSB2, MSB3, LSB3 :

Elle comprend essentiellement 3 compteurs de 2 x 8 bits (permettant la génération de signaux de rapport cyclique 1/2).
Chaque timers est divisé en 2 registres de 8 bits, correspondants aux poids forts MSB et aux poids faibles LSB.
Ces registres tampon, peuvent fonctionner en 16 bits, contiennent les paramètres de comptage.
Ces timers permettant de générer des signaux de rapport cyclique variable.
Les données sont transférées des registres tampon dans les compteurs proprement dits lors d'un cycle d'initialisation des compteurs.
Les compteurs sont décrémentés à chaque impulsion d'horloge (interne ou externe).
Suivant le mode de fonctionnement spécifié, le compteur s'arrête ou recommence un nouveau cycle lorsqu'il arrive zéro.

6840 : CR1 CR2 CR3 :

Trois registres de contrôle en 8 bits définissent le mot de fonctionnement de chacun des compteurs : mode astable, mode monostable, comparateur de fréquence, comparateur de largeur d'impulsions.
Ces registres CR1 CR2 CR3 ne sont accessibles qu'en écriture.

6840 : SR registre :

Un registre d'état en 8 bits, à lecture seule, il nous fournit les indications d'interruption de chacun des timers (interruptions indépendantes) et de n'importe lequel d'entre deux bits SR7.

retour au Sommaire | Index | Liens Rapides

6840 : Fonctionnement

Pour chaque temporisateur, il faut programmer le registre de contrôle CR afin de définir lequel des trois modes (astable, monostable ou intervalle de temps) on va utiliser.

Le contenu du registre CR permet également de valider la sortie et les interruptions générées par le 6840, de choisir l'horloge d'activation et le mode de fonctionnement des compteurs.

Il faut ensuite charger le registre tampon associé au temporisateur choisi.

Dans le cas de fonctionnement en interruption, un test sur le registre d'état SR est nécessaire, afin de connaître le temporisateur qui est à l'origine de cette interruption.

retour au Sommaire | Index | Liens Rapides

6840 : Adressage, Sélection Du Boîtier

Le bit b0 du registre de contrôle 2 (registre CR20) permettra de différencier les registres de contrôle 1 et 3 qui ont donc même adresse.

Les broches RS0, RS1, RS2 combinées aux broches R/W|, CS0| et CS1 permettent de sélectionner les registres internes, mais ne suffisent pas pour adresser les 10 registres internes.
Le 6840 occupe 8 octets mémoire pour 10 registres interne.

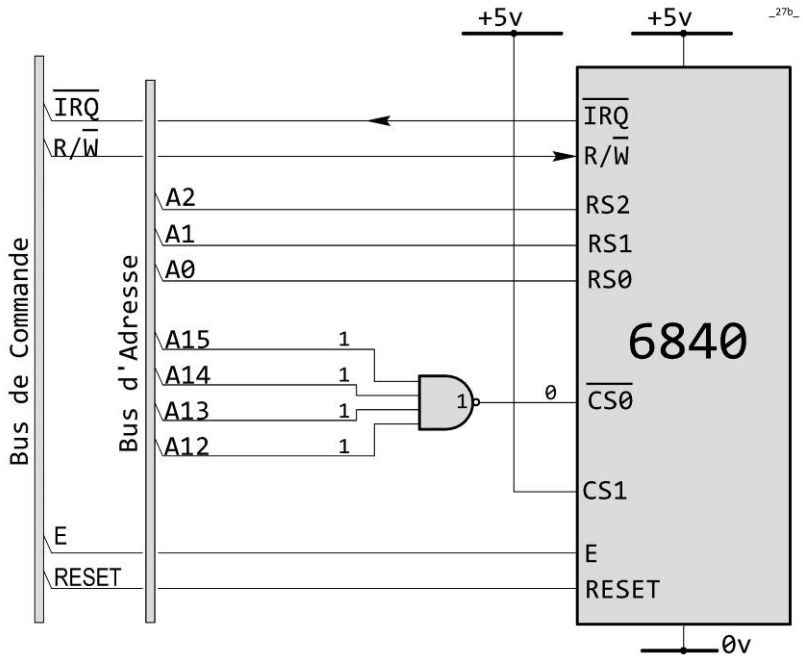
Ces entrées RS0, RS1 et RS2 reçoivent nécessairement les bits A0, A1 et A2 du bus d'adresse pour que le µp6809 voie le temporisateur comme 8 positions mémoires consécutives.

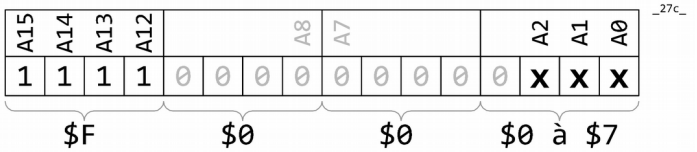
Pour adresser les registres CR1 ou CR3 il faut tenir compte de l'état du bit b0 du registre CR2.

PTM 6840	Broches du 6809 →	A15 ... A3		A2	A1	A0	En lecture R / $\overline{W}$ = 1	En écriture R / $\overline{W}$ = 0	CR20 c'est le bit 0 de CR2
	Broches du 6840 →	CS1	$\overline{CS0}$	RS2	RS1	RS0			
Adresses	**Adr**	1	0	0	0	0	**Pas de lecture possible**	Ecriture registre **CR3**	si CR20 = 0
								Ecriture registre **CR1**	si CR20 = 1
	Adr + 1	1	0	0	0	1	Lecture registre d'état **SR**	Ecriture registre **CR2**	
	Adr + 2	1	0	0	1	0	Lecture compteur **MSB1**	Ecriture registre tampon **MSB1**	
	Adr + 3	1	0	0	1	1	Lecture compteur **LSB1**	Ecriture registre tampon **LSB1**	
	Adr + 4	1	0	1	0	0	Lecture compteur **MSB2**	Ecriture registre tampon **MSB2**	
	Adr + 5	1	0	1	0	1	Lecture compteur **LSB2**	Ecriture registre tampon **LSB2**	
	Adr + 6	1	0	1	1	0	Lecture compteur **MSB3**	Ecriture registre tampon **MSB3**	
	Adr + 7	1	0	1	1	1	Lecture compteur **LSB3**	Ecriture registre tampon **LSB3**	

retour au Sommaire | Index | Liens Rapides

Exemple d'un 6840 implanté à l'adresse $F000





En affectant un 0 sur les bits d'adresse non connectés, le 6840 aura pour adresses :

LSB (Least Significant Bit) octet de poids faible.
MSB (Most Significant Bit) octet de poids fort.

$F000 : registres CR des tempos 1 et 3 suivant le bit b0 du registre CR de la tempo 2.
$F001 : registres CR de la tempo 2 et registre SR suivant la broche R/W|.
$F002 : poids Fort MSB du registre tampon 1 et compteur num 1 suivant la broche R/W|
$F003 : poids Faible LSB du registre tampon 1 et compteur num 1 suivant la broche R/W|
$F004 : poids Fort MSB du registre tampon 2 et compteur num 2 suivant la broche R/W|
$F005 : poids Faible LSB registre tampon 2 et compteur num 2 suivant la broche R/W|
$F006 : poids Fort MSB du registre tampon 3 et compteur num 3 suivant la broche R/W|
$F007 : poids Faible LSB registre tampon 3 et compteur num 3 suivant la broche R/W|

Exemple : Autrement dit si un 6840 se trouve dans la zone partant de l'adresse $7000,
On aura alors :

$7000	écriture de CR1 ou CR3 suivant le bit 0 de CR2									
$7001	lecture de SR ou écriture de CR2 suivant la broche R/W									
$7002	lecture ou écriture de MSB1			en fonction de la broche R/W						
$7003	"	"	"	de LSB1	"	"	"	"	"	"
$7004	"	"	"	de MSB2	"	"	"	"	"	"
$7005	"	"	"	de LSB2	"	"	"	"	"	"
$7006	"	"	"	de MSB3	"	"	"	"	"	"
$7007	"	"	"	de LSB3	"	"	"	"	"	"

retour au Sommaire | Index | Liens Rapides

6840 : Registres de commande CRx

CR1, CR2 et CR3 précisent le mode de fonctionnement de chaque temporisateur.

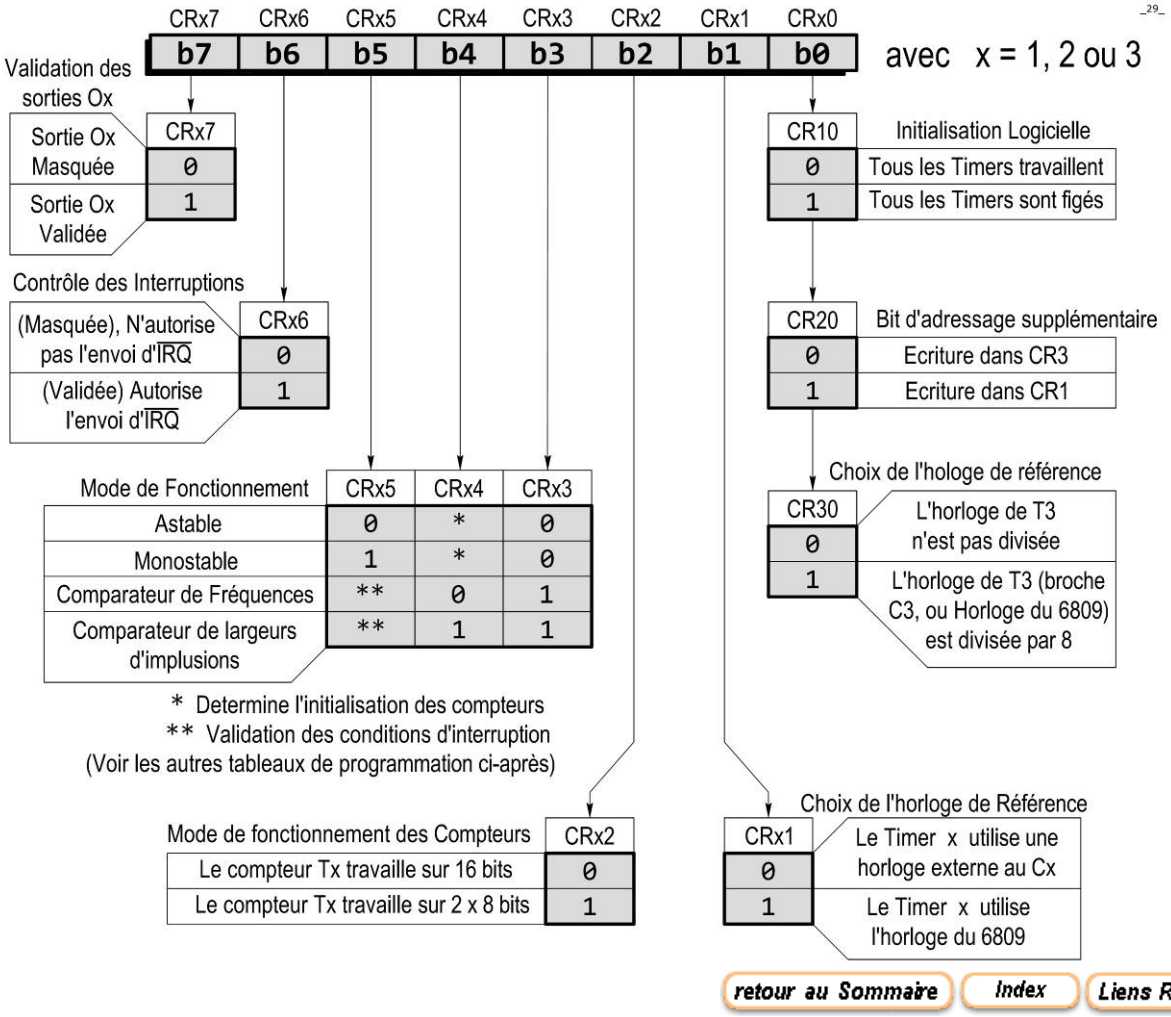
Les différents bits de ces registres remplissent un rôle identique SAUF le bit 0 :

- **Le bit 0** du CR1 sert de RESET logiciel.
- **Le bit 0** du CR2 sert de bit d'adressage supplémentaire.
- **Le bit 0** du CR3 permet de diviser ou non l'horloge du compteur 3 par 8.

Pour ces 3 registres, les autres bits permettent de choisir :

- **Le bit 1** : l'horloge de référence.
- **Le bit 2** : le mode de décrémentation des compteurs
- **Les bits 3, 4 et 5** : le mode de travail des temporisateurs
- **Le bit 6** : les interruptions.
- **Le bit 7** : les sorties de la broche Ox

De plus comme la table des adresses le montre, seul le registre CR2 (registre de commande numéro 2) est adressable directement; pour l'écriture dans les registres de commande numéro 1 et 3 il faut auparavant positionner le bit 0 du registre CR numéro 2.



retour au Sommaire | Index | Liens Rapides

6840 : **CR10** **= 0** tous les temporisateurs sont autorisés à fonctionner
= 1 tous les temporisateurs sont figés dans leur état présent.

6840 : **CR20** **= 0** accès au registre CR num 3
= 1 accès au registre CR num 1

6840 : **CR30** (Horloge externe 4 Mhz maxi)
= 0 facteur de division : 1
= 1 facteur de division : 8.

6840 : **CRx1** Définit si l'on utilise :
= 0 utilisation d'une horloge externe (appliquée sur la broche d'entrée Cx correspondante)
= 1 utilisation d'une horloge du microsystème.

6840 : **CRx2** Définit la taille du compteur
= 0 compteur 16 bits
= 1 compteur 2 x 8 bits

6840 : **CRx3 CRx4 CRx5**
0 x 0 Astable
0 x 1 Monostable
1 0 x Comparaison de fréquences
1 1 x Comparaison de largueurs d'impulsions
x est utilisé pour modifier l'initialisation du compteur et sa validation, ou les conditions d'interruption.

6840 : **CRx6** Valide ou non les interruptions
= 0 n'autorise pas l'envoi d'IRQ| (niveau haut)
= 1 autorise l'envoi IRQ|

6840 : **CRx7** Valide ou non la sortie correspondante
= 0 sortie masquée
= 1 sortie validée

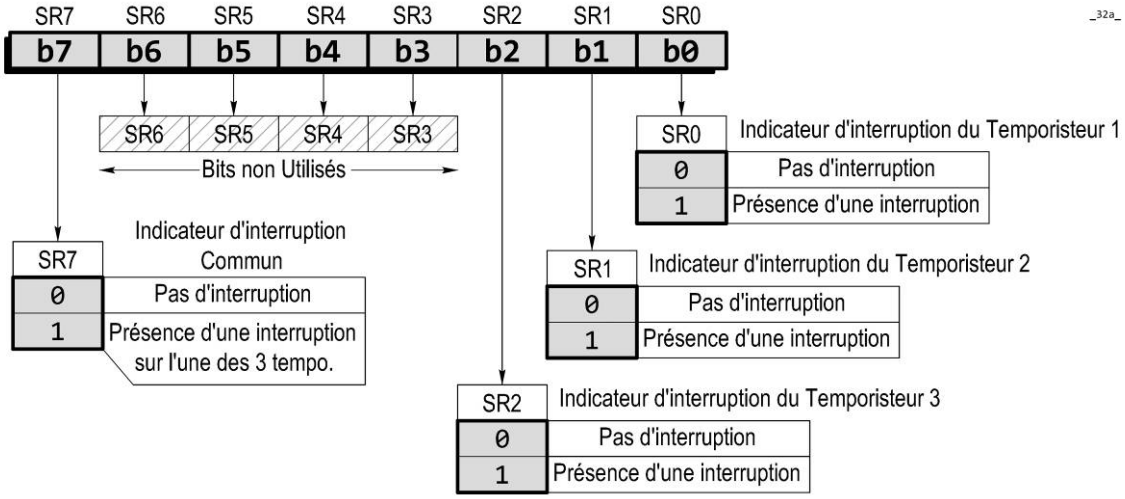
retour au Sommaire | Index | Liens Rapides

6840 : Registre d'Etat SR

C'est un registre à lecture seule. Seul 4 sur les 8 bits sont utilisés comme indicateurs d'interruption.

- Le bit SR0 est associé au temporisateur 1
- Le bit SR1 est associé au temporisateur 2
- Le bit SR2 est associé au temporisateur 3

Le bit SR7 est le bit d'interruption commun aux 3 temporisateurs. Il est positionné à 1 lorsque n'importe quel bit indicateur indépendant sera à 1, à condition que le bit CRx6 soit égal à 1 (validation des interruptions).



Un indicateur d'interruption bit SR2, SR1 ou SR0 est remis à zéro :

- soit par un niveau actif sur la broche RESET|
- soit par CR10 = 1
- soit par une lecture du compteur du temporisateur à condition que le registre d'état ait été lu auparavant et l'indicateur d'interruption positionné.

Cette condition liant la lecture du registre d'état SR et la lecture du compteur a été prévu pour éviter l'oubli d'interruptions qui pourraient se produire après la lecture du registre d'état mais avant la lecture du compteur.

6840 : Rôle des registres Tampons (Initialisation)

Chaque temporisateur indépendant comprend un registre tampon de 16 bits (en écriture) associé à un compteur 16 bits (en lecture).

La durée de comptage d'un compteur dépend du contenu du registre tampon. Ce contenu est calculé en fonction de signaux désignés en sortie, voir les divers modes de fonctionnement du 6840.

Comme les compteurs sont en 16 bits et que le bus de données est en 8 bits, il est nécessaire de stocker dans un BUFFER les 8 bits de plus fort poids.

L'octet de poids fort est préalablement stocké dans le registre MSB Buffer, il sera automatiquement transféré dans le registre tampon MSB lors de l'écriture du registre LSB (poids faible).

C'est pour cela que l'on doit d'abord écrire le MSB puis le LSB. Cet ordre est le même pour la lecture.

Il serait bon de préconiser des opérations 16 bits avec le 6809 pour respecter l'ordre de lecture et d'écriture
Pour l'écriture : STD, STX, ou STY
Pour la lecture : LDD, LDX, ou LDY

L'initialisation d'un compteur peut se faire :

- En appliquant le niveau actif sur RESET| ou CR10=1
- Lors de la commande écriture des registres tampons
- Par application d'une transition descendante sur l'entrée Gx| (G=Gate)

6840 : Rôle des compteurs (Initialisation)

L'initialisation d'un compteur est définie comme transfert d'une donnée du registre tampon dans le compteur. Avec pour conséquence l'effacement de l'indicateur d'interruption associé au compteur.

L'initialisation du compteur se produit dans le cas :

- D'un RESET externe (broche RESET| à 0)
- D'un RESET interne CR10 = 0 (dépendant du mode de fonctionnement).
- Lors d'une écriture dans le registre tampon associé d'une transition négative sur la broche gâchette Gx| (dépendant du mode de fonctionnement).

Une fois initialisé, le compteur est automatiquement décrémenté à la vitesse de l'horloge d'activation choisie.

A chaque fois que le compteur passe par une valeur nulle, il est automatiquement réinitialisé avec le contenu du registre tampon qui lui est associé.

Les compteurs sont accessibles en lecture, comme pour les tampons, il faut toujours lire l'octet de poids fort MSBx avant l'octet de poids faible LSBx.

La lecture de MSBx entraîne le transfert automatique le LSBx dans LSB Buffer.
La lecture de LSBx consiste à lire le contenu de LSB Buffer.

6840 : Différents modes de fonctionnement

6840 : Mode Astable (aussi appelé Multivibrateur Astable ou Mode continu)

(Mode, 01, 02, 03 et 04 voir tableau ci-dessous)

Chacun des 3 temporisateurs peut travailler en multivibrateur Astable CRx5 = 0 CRx3 = 0 avec la broche de sortie Ox est valide on obtient :
- Soit un signal carré CRx2 = 0 (on travaille en 16 bits).
- Soit un signal asymétrique CRx2 = 1 (on travaille en 2 x 8 bits).

Le compteur peut donc travailler :
- Soit en 16 bits.
- Soit en 2 x 8 bits.

L'initialisation d'un compteur peut se faire par remise à zéro du temporisateur de trois façons suivantes :
- Application d'un niveau Bas sur la broche RESET|
- Mise à 1 de b0 du registre CR1.
- Application d'un front descendant sur la broche d'entrée Gx| (G=Gate)

D'autre part, si le bit b4 des registres CRx (x =1, 2 ou 3) est à 0, alors on aura une initialisation du compteur à chaque commande d'écriture dans le registre tampon.

Autres Remarques

- Il est indispensable pour le fonctionnement des compteurs que l'entrée Gx| (G=Gate) soit maintenue à l'état Bas.
- En fonctionnement 16 bits, la sortie du compteur, si elle est validée est à l'état Bas pendant toute la phase de mise à l'état initial et y restera ensuite pendant tout le temps de décrémentation de compteur TO (Time Out)
- Cette sortie passera ensuite à l'état Haut et s'y maintiendra pendant la même période de temps, et ainsi de suite.

retour au Sommaire | Index | Liens Rapides

6840 : Mode Astable Fonctionnement en 2 x 8 bits

L'utilisateur calcule la valeur de **M** (valeur contenue dans les 8 bits MSB) et la valeur de **L** (valeur contenue dans les 8 bits LSB) en fonction du signal désiré et l'horloge d'activation utilisée.

Une fois initialisé, le compteur décrémente **M** et **L**.

Suite au passage à zéro des poids faibles LSB, l'octet de poids fort MSB est décrémenté. Suite au passage à zéro des poids forts, la sortie Ox passe à l'état haut.

La sortie passera à l'état Haut au début de la prochaine impulsion d'horloge, elle restera à l'état Haut jusqu'à ce que les compteurs LSB et MSB soit tous deux à zéro. Autrement dit, le compteur LSB étant utilisé en décompteur, chaque fois que celui ci passe à 0, le compteur MSB est décrémenté d'une unité.

Quand le LSB=0, le MSB est inchangé; sur le coup de l'horloge suivant, le LSB est remis à sa valeur initiale, chargée dans le registre tampon, puis le MSB est décrémenté.

Chaque fois que le compteur passe à zéro, l'indicateur individuel d'interruption est positionné à 1, une interruption sera transmise au µp6809 si CRx6 = 1.

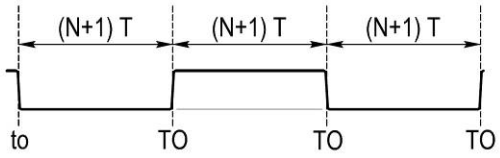
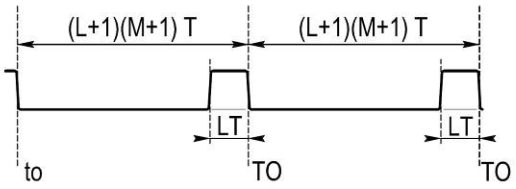
La réinitialisation du compteur entraîne celle de l'indicateur d'interruption, et passage à l'état Bas de la sortie Ox

Le bit CRx4 de chaque temporisateur permet de choisir le type d'initialisation des compteurs.

_30a_

Mode Astable CRx7=1 CRx5=0 CRx3=0

Condition nécessaire de fonctionnement du compteur ($\overline{G}$ = 0) la broche $\overline{GATE}$ soit maintenu à l'état Bas

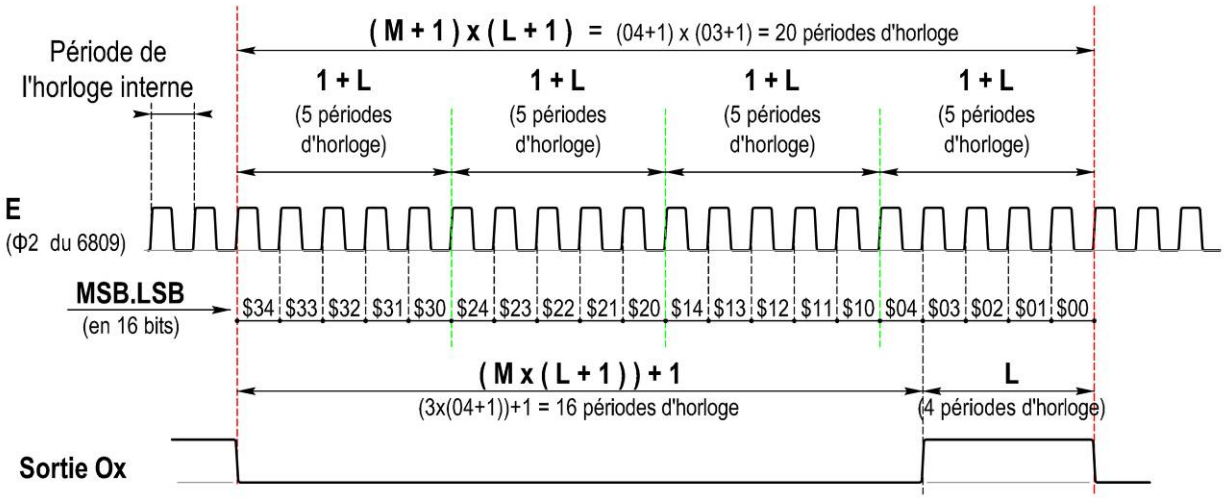
	CRx4	CRx2	Initialisation compteur	Signal en broche de sortie Ox
Fonctionnement sur 16 bits	0	0	(01) $\overline{G}$↓ ou W ou R	
	1	0	(02) $\overline{G}$↓ ou R	
Fonctionnement sur 2 x 8 bits	0	1	(03) $\overline{G}$↓ ou W ou R	
	1	1	(04) $\overline{G}$↓ ou R	

_31a_

$\overline{G}$↓	Font descendant sur la broche d'entrée $\overline{Gx}$ ($\overline{GATE}$)
W	Commande d'Ecriture des registres tampon MSB puis LSB
R	Reset du Timer (CR10=1 ou broche $\overline{RESET}$ niv. Bas)
N	Donnée sur 16 bits contenue dans les registres tampons du compteur
L	Donnée sur 8 bits contenue dans le registre tampon LSB du compteur
M	Donnée sur 8 bits contenue dans le registre tampon MSB du compteur
T	Front descendant de l'horloge appliquée au compteur
to	Initialisation du compteur
TO	Time Out : Temps de décrémentation des compteurs

Exemple de signal en sortie du 6840 en mode Astable, 2 x 8 bits utilisant l'horloge interne

Contenu de M = MSB = $ 03 Contenu de L = LSB = $ 04 _32b_



retour au Sommaire | Index | Liens Rapides

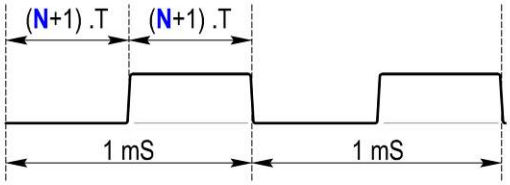
6840 : Mode Astable : Exemple 01

En reprenant la configuration du tableau ci-dessus (Mode Astable), **on désire obtenir simultanément** :

- En sortie du temporisateur 1 un signal carrée de période 1 ms
- En sortie du temporisateur 2 un signal rectangulaire de période 0,5 ms et de rapport cyclique 1/4.

On utilisera l'horloge du µp6809. On utilisera de ce fait :
- Le temporisateur 1 sur 16 bits
- Le temporisateur 2 sur 2 x 8 bits

Temporisateur 1 (en 16 bits)

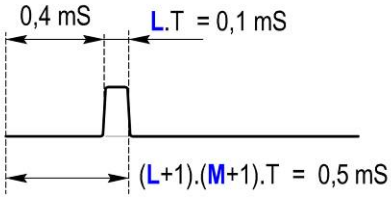


On souhaite obtenir une période de 1 mS
Donc 2 fois (N+1).T = 1 mS
l'horloge du 6809 à 1 Mhz → T = 1 µS
2 x (N + 1) x T = 1 ms
2 x (N + 1) x 1 µS = 1 000 µS

$$N = \frac{1\,000\ \mu S}{2 \times 1\ \mu S} - 1$$

N = $499_{(10)}$
soit **N = $01F3**

Temporisateur 2 (en 2 x 8 bits) _31b_



On souhaite obtenir une période de 0,5 mS et un rapport cyclique de 1/4.

L.T = 0,1 mS = 100 µS
l'horloge du 6809 à 1 Mhz → T = 1 µS
D'où **L** = $100_{(10)}$ = **LSB = $64**

(L+1).(M+1).T = 0,5 mS = 500 µS

$$M = \frac{500\ \mu S}{(L+1).T} - 1 \approx 4$$ **MSB = $04**

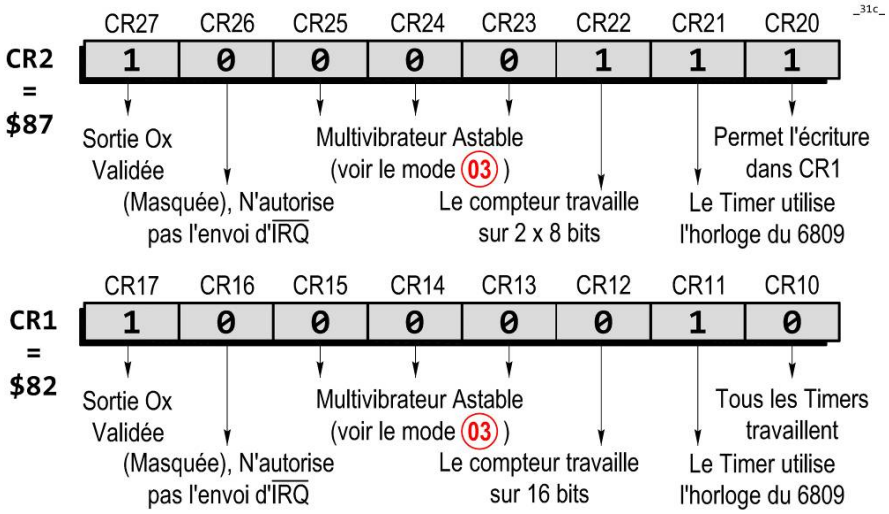
retour au Sommaire | Index | Liens Rapides

Exemple 01 : le programme est le suivant

```
;--------Timer 2 (à faire avant le Timer 1------
LDA     #$87        ;%1000 0111
STA     $F001       ;écriture dans CR2 avec CR20=1
                    ; pour autoriser l'écriture dans CR1
LDD     #$0464      ;
STD     $F004       ;écriture du registre tampon MSB 2 + LSB 2

;--------Timer 1--------------------------------
LDA     #$82        ;%1000 0010
STA     $F000       ;écriture dans CR1
LDD     #$01F3      ;
STD     $F002       ;écriture du registre tampon MSB 1 + LSB 1
```

retour au Sommaire | Index | Liens Rapides

Exemple 01 : explication de la programmation des registres CR2 et CR1



retour au Sommaire | Index | Liens Rapides

Exemple 01 : Remarques

- En fonctionnement sur 2 x 8 bits, si L = M = 0 on obtient un signal de sortie de fréquence moitié de celle de l'horloge.
- En fonctionnement sur 2 x 8 bits, si L = 0, le compteur revient au fonctionnement 16 bits avec apparition du Time Out (TO) au bout de M + 1 périodes d'horloge.
- **Toujours charger les registres MSB avant les registres LSB.**

6840 : Mode Monostable (Mode Monocoup)

retour au Sommaire | Index | Liens Rapides

(Mode 05, 06, 07 et 08 voir tableau ci-dessous)

Le mode est identique au mode astable précédent à trois exceptions près :

La première : la sortie est validée pour une seule impulsion jusqu'à une réinitialisation.

- Après le premier Time Out la sortie reste à l'état Bas jusqu'au prochain cycle d'initialisation.
- Comme en fonctionnement astable, l'on peut travailler sur 16 bits ou en 2 x 8 bits.
- Le fonctionnement du compteur interne reste cyclique dans le fonctionnement en monostable.
- Chaque Time Out du compteur positionne à 1 l'indicateur d'interruption indépendant du registre d'Etat ainsi que la réinitialisation du compteur.

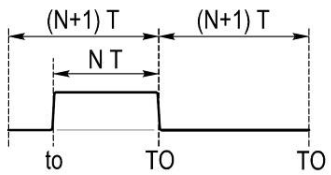
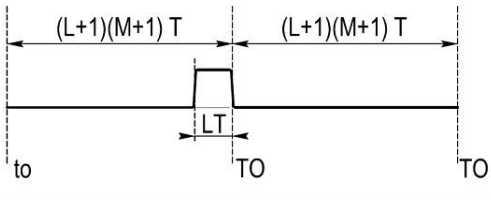
La seconde : la condition G| = 0 n'est pas prise en considération.

- Il suffit d'appliquer une transition sur la broche G| (G=Gate) ou déclencher le Monostable.

La troisième : si L = M = 0 ou N = 0, la sortie est inhibée.

- Lorsque L = M = 0, en 2 x 8 bits ou N = 0 en 16 bits, la sortie tombe à l'état bas sur cette première impulsion d'horloge reçue pendant ou après l'initialisation du compteur.
- La sortie reste à l'état bas jusqu'à ce que l'on change le mode de fonctionnement ou que l'on rentre des données différentes de zéro dans les registres tampons.
- On a toujours un Time Out à la fin de chaque période d'horloge.
- En fonctionnement normal sur 16 bits, le compteur sera décrémenté jusqu'à zéro au bout de (N+1) périodes d'horloge, N étant la donnée de 16 bits contenue dans le registre tampon.

Mode Monostable CRx7=1 CRx5=1 CRx3=0

	CRx4	CRx2	Initialisation compteur	Signal en broche de sortie Ox
Fonctionnement sur 16 bits	0	0	05 G| ou W ou R	
	1	0	06 G| ou R	
Fonctionnement sur 2 x 8 bits	0	1	07 G| ou W ou R	
	1	1	08 G| ou R	

_31a_

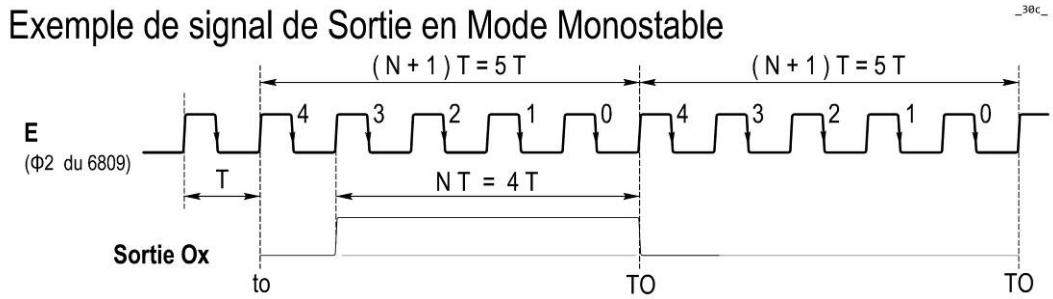
G↓	Font descendant sur la broche d'entrée Gx (GATE)
W	Commande d'Ecriture des registres tampon MSB puis LSB
R	Reset du Timer (CR10=1 ou broche RESET niv. Bas)
N	Donnée sur 16 bits contenue dans les registres tampons du compteur
L	Donnée sur 8 bits contenue dans le registre tampon LSB du compteur
M	Donnée sur 8 bits contenue dans le registre tampon MSB du compteur
T	Front descendant de l'horloge appliquée au compteur
to	Initialisation du compteur
TO	Time Out : Temps de décrémentation des compteurs

retour au Sommaire | Index | Liens Rapides

Exemple avec N=04

La sortie, si elle est validée, passe à l'état Haut après la première impulsion d'horloge qui se produit pendant ou après la durée de la mise à l'état initial et y reste pendant N impulsions d'horloge suivantes.

_30c_



En fonctionnement sur 2 x 8 bits, le Time Out dure (L + 1) (M + 1) T.
L représente la donnée dans le registre tampon LSB
M représente la donnée dans le registre tampon MSB
La sortie, si elle est validée est à l'état Bas pendant toute la phase d'initialisation et y reste jusqu'à ce que le MSB soit à zéro.

retour au Sommaire | Index | Liens Rapides

6840 : Mode Mesure d'Intervalle de Temps

Ce mode est sélectionné quand le bit CRx3 = 1. Le bit CRx4 permet ensuite de choisir entre :

- Le mode comparaison de fréquence CRx4 = 0
- Le mode comparaison de largeur d'impulsion CRx4 = 1

Le mode intervalle de temps est utilisé dans des applications nécessitant plus de souplesse dans la génération des interruptions et l'initialisation des compteurs.
Dans ce mode les indicateurs individuels d'interruption sont fonctions à la fin du comptage d'un compteur (TO Time Out des compteurs) ou sur transition active de l'entrée Gx| (G=Gate).
Le principe de ce mode est de comparer une fréquence ou une impulsion externe, présente sur une des entrées Gx|, avec le contenu d'un compteur associé Cx|
Dans chacun de ce mode le signal de sortie Ox n'est pas défini, mais malgré cela le compteur peut travailler soit en 16 bits soit en 2 x 8 bits.
Un front descendant sur l'entrée G| active le compteur et commence un cycle d'initialisation.
Le compteur est alors décrémenté à chaque coup d'horloge pendant ou après l'initialisation du compteur et jusqu'à ce qu'une interruption soit engendrée.

retour au Sommaire | Index | Liens Rapides

6840 : Mode Mesure d'Intervalle de Temps Comparaison de Fréquence CRx4.CRx3 = %01

(Mode 09 et 10 voir tableau ci-dessous) Ce mode présente deux aspects en fonction du contenu du bit CRx5

CRx5 = 0 (mode 09)

Le compteur Cx est initialisé par un front descendant sur Gx|, une interruption est générée si l'entrée Gx| revient à l'état Bas avant la fin du comptage.
Si la période de comptage de compteur s'est écoulée avant l'arrivée de front descendant sur Gx|, le compteur recommence son cycle de décomptage et ainsi de suite jusqu'au moment où un front descendant est appliqué sur Gx|.
Dans ce cas, un bit est positionné à l'extérieur du temporisateur à la fin de la première période de comptage, afin d'empêcher la génération d'une interruption, tant que le compteur n'a pas été réinitialisé.
Cette réinitialisation est obtenue par l'application d'un front descendant sur Gx| (la condition front descendant sur Gx| et Bit indicateur d'interruption et Time Out est satisfaite puisque la fin d'une période est apparue sans positionnement indicateur d'interruption individuel).

CRx5 = 1 (mode 10)

Le signal à mesurer est présent sur une des entrées Cx|.
Le compteur Cx associé est initialisé par un front descendant sur Gx|.

Si à la fin de comptage du compteur apparaît avant un nouveau front négatif sur Gx|, l'indicateur individuel d'interruption est positionné.
Le compteur est inhibé, tant que cet indicateur individuel reste à 1 et qu'un nouveau front descendant n'a pas été détecté.
Si au contraire le front négatif apparaît sur Gx| avant la fin du comptage, l'indicateur individuel d'interruption reste à 0 et ce front négatif sur Gx| ne fait que réinitialiser le compteur.
Ce fonctionnement continue jusqu'à la fin du comptage.

retour au Sommaire | Index | Liens Rapides

6840 : Mode Mesure d'Intervalle de Temps Comparaison de Largeur d'Impulsion CRx4.CRx3 = %11

(Mode 11 et 12 voir tableau ci-dessous) Ce mode est similaire au mode de comparaison de fréquence, l'application d'un front descendant sur Gx| lance le compteur mais cette fois, c'est une transition positive de Gx | qui arrête le comptage. Comme en comparaison de fréquence, ce mode présente deux aspects de fonctionnement, en fonction du contenue de CRx5

CRx5 = 0 (mode 11)

Gx | passe à l'état Bas, le compteur est initialisé.
Si Gx | revient à l'état Haut avant la fin de la période de comptage, l'indicateur d'interruption individuel est positionné et le compteur se bloque.

CRx5 = 1 (mode 12)

Si à la fin du comptage TO (Time Out) apparaît avant le retour à l'état Haut de Gx|, une interruption est générée.

_34_

	CRx5	CRx4	CRx3	Initialisation du Compteur	Activation du compteur	Désactivation du compteur	Positionnement de l'indicateur d'interruption SR (0,1 ou 2) à 1
Comparateur de Fréquences	0	0	1	$\overline{Gx}$↓ et bit SR 0,1,2 à 0 et Compteur inactif ou $\overline{Gx}$↓ et bit SR 0,1,2 à 0 et TO et Compteur actif ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas)	$\overline{Gx}$↓ et bit SR (0,1 ou 2) à 0 et Pas de commande d'écriture des Registres Tampons et Pas de Reset du Timer (CR10=0 et $\overline{RESET}$ niv. Haut)	Commande d'écriture des Registres Tampons ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas) ou bit SR (0,1 ou 2) à 1	$\overline{Gx}$↓ avant TO (Time Out)
	1			$\overline{Gx}$↓ et bit SR 0,1,2 à 0 ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas)	$\overline{Gx}$↓ et bit SR (0,1 ou 2) à 0 et Pas de commande d'écriture des Registres Tampons et Pas de Reset du Timer (CR10=0 et $\overline{RESET}$ niv. Haut)	Commande d'écriture des Registres Tampons ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas) ou bit SR (0,1 ou 2) à 1	TO (Time Out) avant $\overline{Gx}$↓
Comparateur de largeurs d'implusions	0	1		$\overline{Gx}$↓ et bit SR 0,1,2 à 0 ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas)	$\overline{Gx}$↓ et bit SR (0,1 ou 2) à 0 et Pas de commande d'écriture des Registres Tampons et Pas de Reset du Timer (CR10=0 et $\overline{RESET}$ niv. Haut)	Commande d'écriture des Registres Tampons ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas) ou bit SR (0,1 ou 2) à 0 ou $\overline{Gx}$ niv. Haut	$\overline{Gx}$↓ avant TO (Time Out)
	1			$\overline{Gx}$↓ et bit SR 0,1,2 à 0 ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas)	$\overline{Gx}$↓ et bit SR (0,1 ou 2) à 0 et Pas de commande d'écriture des Registres Tampons et Pas de Reset du Timer (CR10=0 et $\overline{RESET}$ niv. Haut)	Commande d'écriture des Registres Tampons ou Reset du Timer (CR10=1 ou $\overline{RESET}$ niv. Bas) ou bit SR (0,1 ou 2) à 0 ou $\overline{Gx}$ niv. Haut	TO (Time Out) avant $\overline{Gx}$↓

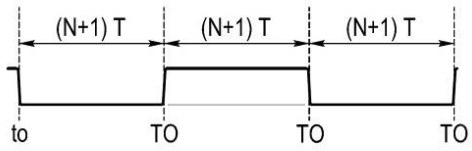
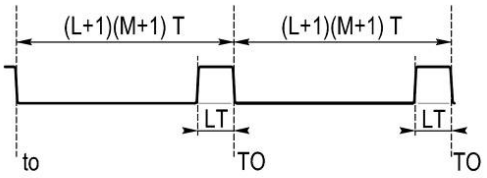
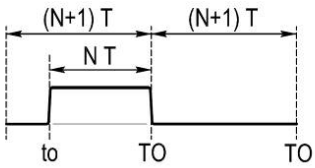
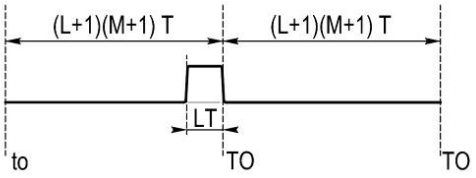
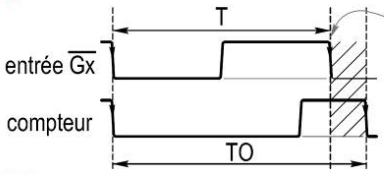
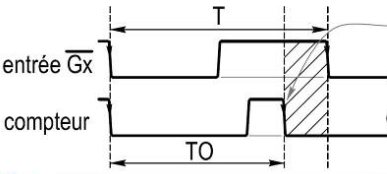
_31a_

$\overline{G}$↓	Font descendant sur la broche d'entrée $\overline{Gx}$ ($\overline{GATE}$)
W	Commande d'Ecriture des registres tampon MSB puis LSB
R	Reset du Timer (CR10=1 ou broche $\overline{RESET}$ niv. Bas)
N	Donnée sur 16 bits contenue dans les registres tampons du compteur
L	Donnée sur 8 bits contenue dans le registre tampon LSB du compteur
M	Donnée sur 8 bits contenue dans le registre tampon MSB du compteur
T	Front descendant de l'horloge appliquée au compteur
to	Initialisation du compteur
TO	Time Out : Temps de décrémentation des compteurs

6840 : Tableau regroupant tous les Modes de Fonctionnement

- Le mode Astable (ou mode continu) (mode 01, 02, 03 et 04)
- Le mode Monostable (ou mode monocoup) (mode 05, 06, 07 et 08)
- Le mode comparaison de fréquence (mode 09 et 10)
- Le mode comparaison de largeur d'impulsion (mode 11 et 12)

_33_

Mode de Fonctionnement	CRx5	CRx4	CRx3	CRx2	Format	Initialisation compteur	Signal en broche de sortie Ox
Astable (mode continu)	0	0	0	0	16 bits	01 $\overline{Gx}$ ↓ ou W ou R	
		1				02 $\overline{Gx}$ ↓ ou R	
		0		1	2 x 8 bits	03 $\overline{Gx}$ ↓ ou W ou R	
		1				04 $\overline{Gx}$ ↓ ou R	
Monostable (mode monocoup)	1	0		0	16 bits	05 $\overline{Gx}$ ↓ ou W ou R	
		1				06 $\overline{Gx}$ ↓ ou R	
		0		1	2 x 8 bits	07 $\overline{Gx}$ ↓ ou W ou R	
		1				08 $\overline{Gx}$ ↓ ou R	
Comparateur de Fréquences (Fréquencemètre)	0	0	1	0	16 bits	09 Mesure de durée plus PETITE que le Time Out	 Une interruption est engendrée si la période de la broche $\overline{Gx}$ est < au TO du compteur
				1	2 x 8 bits		
	1			0	16 bits	10 Mesure de durée plus GRANDE que le Time Out	 Une interruption est engendrée si la période de la broche $\overline{Gx}$ est > au TO du compteur
				1	2 x 8 bits		
Comparateur de largeurs d'implusions	0	1		0	16 bits	11 Dès que $\overline{Gx}$ ↓ le compteur est initialisé. Si $\overline{Gx}$ ↑ avant la fin de la période de comptage TO └alors le bit SR2, SR1 ou SR0 est positionné à 1 et le compteur se bloque.	Une interruption est générée si la durée de l'état Bas sur la broche $\overline{Gx}$ est < au TO (Time Out) du compteur
				1	2 x 8 bits		
	1			0	16 bits	12 Dès que $\overline{Gx}$ ↓ le compteur est initialisé. Si la fin du comptage TO apparaît avant ↑ de $\overline{Gx}$ └alors le bit SR2, SR1 ou SR0 est positionné à 1	Une interruption est générée si la durée de l'état Bas sur la broche $\overline{Gx}$ est > au TO (Time Out) du compteur
				1	2 x 8 bits		

_31a_

$\overline{G}\downarrow$	Font descendant sur la broche d'entrée $\overline{Gx}$ ($\overline{GATE}$)
W	Commande d'Ecriture des registres tampon MSB puis LSB
R	Reset du Timer (CR10=1 ou broche $\overline{RESET}$ niv. Bas)
N	Donnée sur 16 bits contenue dans les registres tampons du compteur
L	Donnée sur 8 bits contenue dans le registre tampon LSB du compteur
M	Donnée sur 8 bits contenue dans le registre tampon MSB du compteur
T	Front descendant de l'horloge appliquée au compteur
to	Initialisation du compteur
TO	Time Out : Temps de décrémentation des compteurs

6840 : Exemples de Programmation

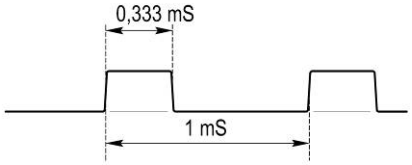
retour au Sommaire Liens Rapides

6840 : Exemple de Programmation Mode Astable

Exemple Multivibrateur Astable

_35a_

On souhaite obtenir un signal dont la forme est la suivante (avec une période de 1 mS)



Le 6840 travaille à 1 Mhz
On prend le temporisateur n° 3 afin d'utiliser le diviseur d'horloge.

(L + 1) (M + 1) x T = 1 mS
L x T = 333 µS

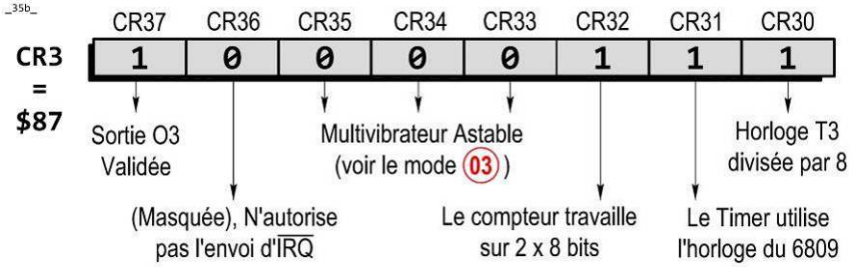
On utilise donc l'horloge interne divisée par 8 d'où T = 8 µS

$$L \times T = 333\ \mu S \quad \text{donc} \quad L = \frac{333\ \mu S}{8\ \mu S} = 41{,}625 \approx 42 \qquad \mathbf{L = 42 = \$2A}$$

$$\text{d'où } M = \frac{1\ mS}{(L+1) \times T} - 1 = \frac{1000\ \mu S}{(42+1) \times 8} - 1 = 1{,}90698 \approx 2 \qquad \mathbf{M = 2 = \$02}$$

La logique de décodage du 6840 donne comme première adresse $CFF8
La valeur du registre CR3 est de $87 = %1000 0111

_35b_

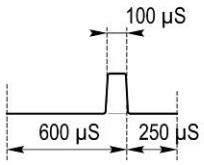


On obtient le programme suivant :

```
LDA    #$01     ;Accès à CR1
STA    ADRCR2   ;
CLRA            ;init logicielle de tous les Timers
STA    ADRCR1   ;
STA    ADRCR2   ;accès à CR3
LDA    #$87     ;init Timer 3
STA    ADRCR3   ;
LDX    #$022A   ;M=$02 L=$2A init registre tampon 3
STX    TEMP3    ;
```

6840 : Exemple de programmation Mode Monostable

Exemple En Monostable



Soit à programmer le temporisateur n°1 en mode Monostable afin d'obtenir une impulsion ci-dessous. _36a_

On utilise l'horloge interne du 6840 à 1 Mhz
Le compteur n°1 fonctionnera sur 2 x 8 bits .

(L + 1) (M + 1) x T = 600 µS
L x T = 100 µS

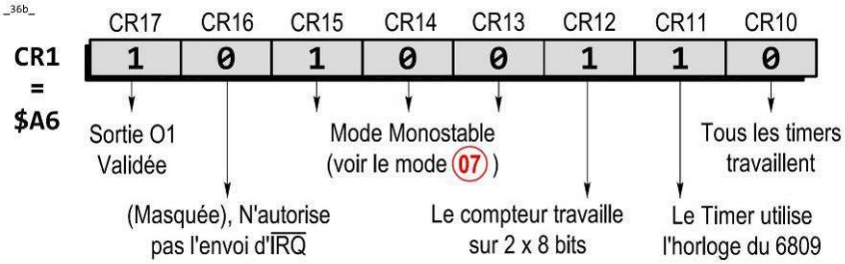
On utilise donc l'horloge interne d'où T = 1 µS

$$L \times T = 100\ \mu S \quad \text{donc} \quad L = \frac{100\ \mu S}{1\ \mu S} = 100 \qquad \mathbf{L = 100 = \$64}$$

$$\text{d'où } M = \frac{1\ mS}{(L+1) \times T} - 1 = \frac{600\ \mu S}{(100+1) \times 1} - 1 = 4{,}94059 \approx 5 \qquad \mathbf{M = 5 = \$05}$$

La logique de décodage du 6840 donne comme première adresse $8000
La valeur du registre CR1 est de $A6 = %1010 0110

_36b_



On obtient le programme suivant :

```
LDA     #$01    ;Accès à CR1
STA     ADRCR2  ;
LDA     #$A6    ;init Timer 1
STA     ADRCR1  ;
LDX     #$0564  ;M=$05 L=$64 init registre tampon 1
STX     TEMP1   ;
```

_37a_

Exemple En Comparaison de Fréquence

On désire mesurer une fréquence 60 kHZ < Fx < 200 kHz. On utilise le temporisateur n°1 activé avec l'horloge du 6840 à 1 MHz, une interruption sera envoyée au 6809.

On charge le registre tampon T1 de manière à ce que TO > T

Le compteur fonctionne sur 2 x 8 bits, on prend TO > $\frac{1}{Fx\ min}$ (TO = 20 µS)

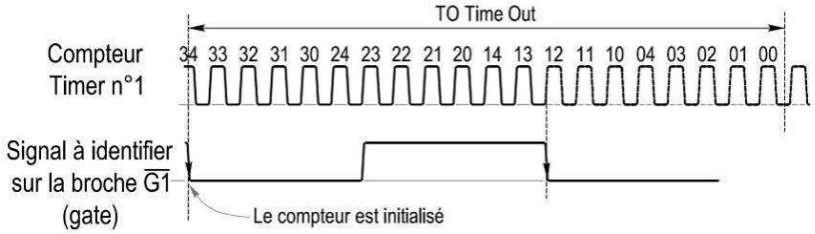
On a (M + 1) (L + 1) x T = 20 µS
Comme l'horloge est à 1MHz ⇒ T = 1 µS

On se donne L x T = 4 µS donc $L = \frac{4\ \mu S}{T} = \frac{4\ \mu S}{1\ \mu S} = 4$ **L = 4 = $04**

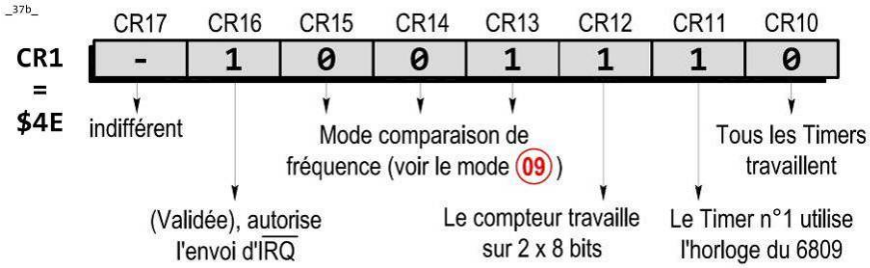
(M + 1) (L + 1) x T = 20 µS

$M = \frac{20\ \mu S}{(L+1) \times T} - 1 = \frac{20\ \mu S}{(4+1) \times 1} - 1 = 3$ **M = 3 = $03**

Un front descendant sur $\overline{G1}$ entraîne l'initialisation du compteur.
Le fonctionnement sera alors le suivant :



_37b_

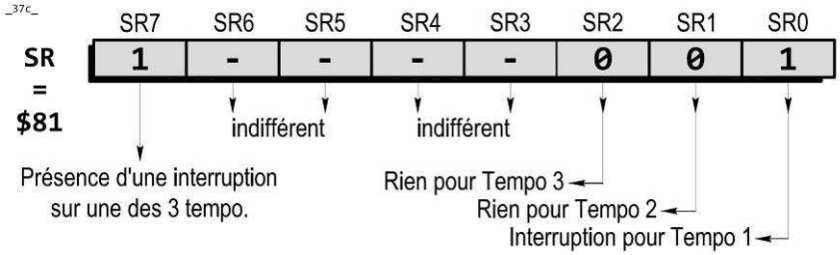


On obtient le Programme suivant :

```
LDA    #$01     ;Accès à CR1
STA    ADRCR2   ;
LDA    #$4E     ;init CR1
STA    ADRCR1   ;
LDX    #$0304   ;M=$03 L=$04 init registre tampon 1
STX    TEMP1    ;
```

Lorsque le compteur se bloque sur un second front descendant de G1| le registre d'état SR est positionné, son contenu est alors le suivant :

_37c_



Le 6809 est interrompu, le programme d'interruption qui suit consiste à lire le compteur et à calculer la fréquence Fx.

Au départ on avait M = 3 et L = 4. Le compteur donne M = 1 et L = 3

Ce qui fait **Tx = (M + 1) (L + 1) x T** = (1 + 1) (3 + 1) x 1µS = 8 µS

D'où la fréquence F **(Hz) = 1 / Période (S)** = 1 / 0,000 008 = 125 000 Hz Fx = 125 KHz

Le registre d'état est réinitialisé automatiquement par la lecture du compteur qui suit celle de ce même registre d'état.

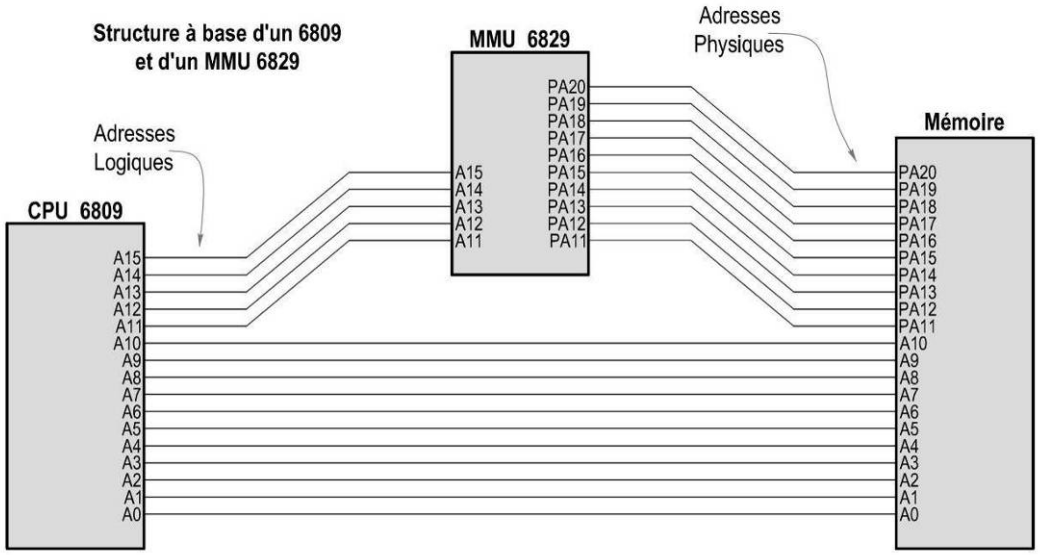
6829 : Généralités

MMU (Management Memory Unit) Interface d'Extension Mémoire

Ce circuit permet d'étendre l'espace mémoire du µp6809 de 64 Ko à 2 Mo.

Le principe est de connecter celui-ci entre le bus d'adresse du µp6809 et la mémoire.

_38a_



retour au Sommaire | Liens Rapides

6829 : Principe

Le principe de fonctionnement du 6829 est de traduire les adresses Logiques (issues du µp6809) en adresses Physiques commandant la mémoire.

Chaque MMU 6829 peut générer 4 tâches séparées de 64 Ko chacune.

La capacité d'adressage maximum est de 2 Mo obtenue en connectant 8 circuits MMU 6829 en parallèle.

Les broches A0 à A10 détermine l'accès à des pages de 2 Ko.

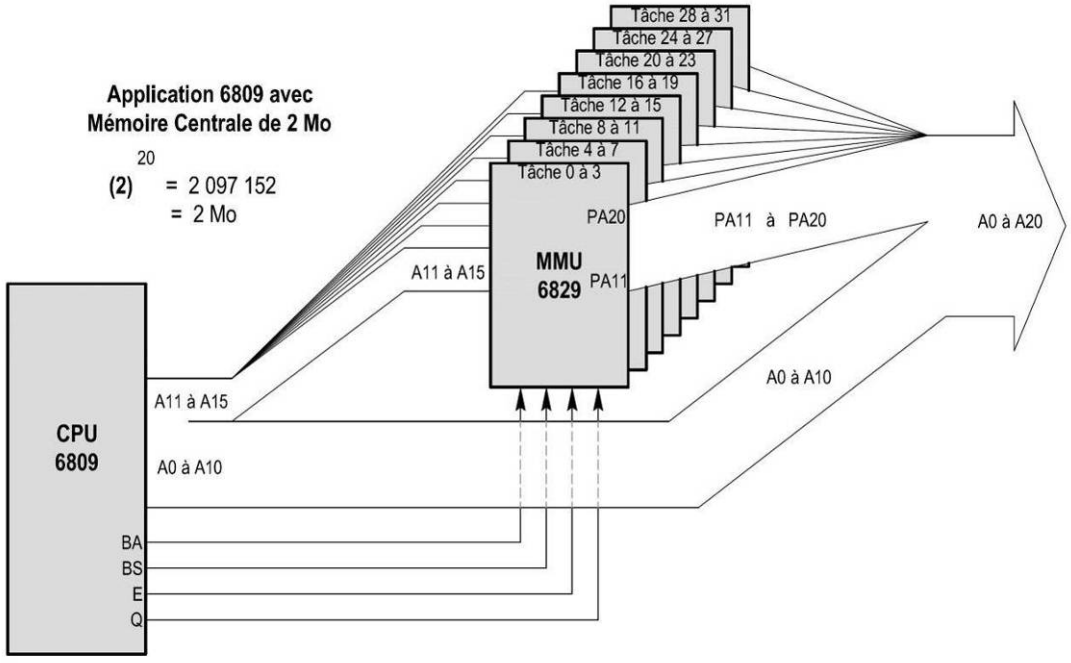
Les broches A11 à A15 permettent de sélectionner une des 32 pages d'une tâche sélectionnée au préalable.

La combinaison de la tâche choisie en fonction du numéro de la page permet d'obtenir les poids forts A11 à A20 de l'adresse Physique.

Chaque tâche peut être isolée et protégée en écriture.

L'espace mémoire total du µp6809 est donc partagé en 1024 pages de 2 Ko.

_38b_



retour au Sommaire | Liens Rapides

6829 : Utilisation

Dans le cas où l'utilisateur à besoin d'une mémoire centrale de 2 Mo, il est possible de connecter 8 circuit 6829 en parallèle.

La tâche 0 du MMU 6829 n°1 est réservée au système d'exploitation.

La tâche 1 du MMU 6829 n°1 est réservée aux accès directs à la mémoire.

Les tâches 2 à 31 sont disponibles pour l'utilisateur.

MauP : Généralités

Plusieurs étapes dans la création d'un programme :

- Poser le problème
- L'organigramme
- Ecriture du programme
- Programmation Structurée
- Mise au point (MauP)
- Economie de place mémoire, quelques conseils

MauP : Poser le problème

Savoir exactement quelles fonctions le programme devra réaliser, quelles Entrées-sorties on utilisera. Cette phase est peut être un peu astreignante mais elle oblige à avoir les idées claires et permet d'aborder plus sereinement la suite.

retour au Sommaire | Liens Rapides

MauP : L'organigramme

Quand on réalise un programme, surtout s'il est complexe, il est toujours bon de savoir comment va se dérouler son exécution avant de commencer à programmer. D'ailleurs, dans certains programmes, c'est essentiel, pour plusieurs raisons :

- La détection et le traitement des erreurs.
- Les structures de contrôle entraînant de longues conséquences sur votre programme.
- L'organisation des tâches pendant la création d'un programme en équipe.

MauP : Voici quelques règles utiles

- Diviser le programme en plusieurs parties, chacune d'elles réalisant une ou plusieurs fonctions élémentaires.
- Eviter les "astuces géniales" qui peuvent ne pas être comprises par d'autres ou par soi même dans plusieurs mois.
- Eviter de mettre des instructions de programmation.
- Concevoir son programme de façon structurée, séquence après séquence avec une entrée et une sorties par séquence.

Un organigramme est normalisé, c'est à dire que tout le monde s'est mis d'accord pour dessiner les mêmes symboles. Dans notre cas c'est la norme ISO 5807.

MauP : L'écriture du programme

retour au Sommaire | Liens Rapides

A ses débuts, le programmeur inexpérimenté dans le langage Assembleur a tendance à fixer son attention sur la fonctionnalité à produire, quelque soit la quantité de ligne de code, les procédures et les fonctions utilisées pour produire le résultat final. Et ceci sans comprendre parfois ce qu'il fait vraiment ou les spécificités de ce langage.

Le but est de faire de programmes **très lisible**. Ne pas oubliez que dans l'industrie on travaille en équipe. Quelqu'un d'autre doit être capable de reprendre le programme.

Ne pas être avare de commentaire détaillé dans le corps même du programme. Le fait d'écrire un maximum de commentaires, très utile lors de la mise au point ou lors de modifications ultérieures.

Enfin la rédaction de la documentation est primordiale, elle doit commencer à la genèse du programme, être mise à jour en fonction des évolutions majeure de la programmation, pour être finalisé et mise à jour dès que le programme a été complètement testé et débuguer.

MauP : Voici quelques règles d'or, pour éviter de faire trop d'erreurs

- **Etudiez au préalable le problème posé** et surtout rester simple.
  Réfléchir et imaginer un algorithme avant d'écrire la première ligne du programme. L'analyse du problème à résoudre doit rester simple. Cette analyse écrite doit être claire et facile à comprendre par d'autre programmeur. L'organigramme doit être lisible par des non-spécialistes

- **Début de la création d'une documentation** en fonction d'une analyse écrite. Déterminer les points principaux à traiter.

- **Ecriture d'un ordinogramme lisible par des non-spécialistes** (simple et surtout très structuré avec des modules une entrée une sortie), évitez les instructions du style GOTO, par exemple comme les mnémoniques JMP, BRA.... Les instructions JMP (jump) sont à proscrire, elles sont analogues au GOTO en BASIC. Concevoir son programme de façon structurée, séquence après séquence avec une entrée et une sorties par séquence.

- **Mettez un maximum de commentaires.** Commenter chaque morceau du programme et de l'ordinogramme de manière explicative et non descriptive.

- **Ne pas hésitez à faire des sous-programmes.** Diviser le programme en plusieurs parties, chacune d'elles réalisant une ou plusieurs fonctions élémentaires. Elles pourront être des sous-programmes indépendants, afin de pouvoir les tester séparément lors d'une mise au point.

- **Ne pas écrire de longues procédures**. Une procédure ne devrait pas avoir plus de 20 lignes de code. Chaque procédure doit avoir un objectif clair. Un bon programme doit avoir des procédures claires, sans cumul.

- **Evitez les étiquettes du style "DFGHTYG", soyer clair.**
  Lors du codage, choisir des noms parlants pour représenter les objets manipulés dans le programme, éviter l'abstrait et opter toujours pour le concret.
  Eviter le vocabulaire peu suggestif du type (TOTO, TATA, ESSAI, CHOSE, TRUC, XXX, ...)
  Faire attention à des ressemblances formelles du type : O et 0, 1 et I, Z et 2, B et 8.
  Lors d'un chiffrage, une bonne habitude consiste à utiliser deux chiffres (00, 01, 02, ...) ou trois chiffres (000, 001, 002, ...) si l'on sait que la plage pourrait aller au-delà de 100.

- **Eviter les "astuces géniales"** qui peuvent ne pas être comprises par d'autres ou par soi même dans plusieurs mois. Ne pas utiliser des astuces de programmation qui rendrait les programmes illisibles. Les programmeurs ne doivent pas utiliser les fonctions fantaisistes du langage. L'utilisation des fonctions simples oblige le programmeur à réfléchir à ce qu'il écrit. Ne jamais utiliser les fonctionnalités du langage dont vous n'êtes pas sûr(e) du résultat ou du rôle.

- **Tester chaque module, procédure, fonction séparément.** En test, prévoyez tous les cas de figure possibles, faire des simulations de tous les cas.

- **Finalisation de la documentation,** elle se voudra claire et concise. Faire la mise au propre et en final éditer une version papier (ne pas oublier d'indexer les pages et de mettre le numéro de version). Faire la relecture de cette documentation en corrélation avec les commentaires laissés tout au long du programme et de l'ordinogramme.

- **Dernière règle** On applique ces règles ci-dessus chaque jour pendant au moins six mois.

La pratique de la programmation en suivant ces règles d'or peut s'avérer très gênant. Mais c'est un excellent moyen d'apprendre l'assembleur du µp6809 et surtout la pratique d'une programmation structurée.

MauP : Programmation Structurée

En programmation structurée il existe 3 formes extrêmement employées (avec leur équivalent en assembleur) :

MauP : IF……THEN……ELSE……END IF (en assembleur 6809)

```
        LDA     VAR     ; charge VAR
        CMPA    $00     ;
        BLT     TEST1   ; si négatif
        JSR     PROG2   ;
        BRA     TEST2   ; si positif
TEST1   JSR     PROG1   ;
TEST2   SWI             ;
```


MauP : DO…WHILE (en assembleur 6809)

```
        LDA     VAR     ;
TEST1   CMPA    #$00    ;
        BLE     TEST2   ;
        .               ;
        . séquence d'instructions dans la boucle
        .               ;
        BRA     TEST1   ;
TEST2   SWI             ;
```


MauP : REPEAT…UNTIL (en assembleur 6809)

```
        LDA     VAR     ;
TEST1   .               ;
        .               ;
        . séquence d'instructions dans la boucle
        .               ;
        CMPA    $00     ;
        BGR     TEST1   ;
        .               ;
        .               ;
```


MauP : Mise au point

Mise au point ou déverminage (Debugging).

Utilisation de point d'arrêt, utilisation des instructions SWI. Elles permettent d'arrêter un programme là ou on le désire. On peut ensuite visualiser le contenu des registres internes et de la mémoire.

Erreurs classiques

- Erreurs de branchement des sauts conditionnels.
- Ordres des opérandes.
- Modes d'adressages.
- Ne pas oublier d'initialiser les compteurs de boucle ou les pointeurs de pile.
- Attention aux modifications que peuvent apporter des sous-programmes sur les bits du registre CC

MauP : Economie de place mémoire, quelques conseils

- Utilisation au maximum de sous-programme pour effectuer des tâches répétitives.
- Utilisation d'instruction utilisant peu d'octets et notamment l'adressage direct pour les données fréquemment utilisées.
- Utilisation de la pile Utilisateur pour le passage de paramètres entre les diverses parties du programme.
- Utilisation d'instruction opérant directement sur les registres ou les cases mémoires.

6809 Prog 01 : Création d'une table de données

retour au Sommaire | Index | Liens Rapides | Sommaire Principal

Prog 01 : Sujet

Une table de données consiste en une liste de données quelconques logées en mémoire à des adresses successives. L'adresse de la première donnée est qualifiée d'adresse de base de la table.

Prog 01 : Question A

Proposer un programme permettant de ranger en mémoire dans l'ordre croissant l'ensemble des données 8 bits non signées à partir de l'adresse de base $0100.
Commentaires

La plage des nombres non signés s'étend de $00 à $FF. Il faudra donc charger la mémoire avec ces 256 valeurs.

Prog 01 : Programme A

Création d'une table de données en bits non signés

```
        ORG     $0000     ; Début du programme
        LDX     #$0100    ; Début de table
        LDA     #$00      ; 1ere données $00
Boucle  STA     ,X+       ; Chargement et incrémentation du pointeur
        CMPA    #$FF      ; Dernière donnée = $FF alors fin de programme
        BEQ     Fin       ;
        INCA              ; Incrémentation de la donnée
        BRA     Boucle    ;
Fin     SWI               ;
```

Etat de la mémoire après exécution du programme

```
0100 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0110 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
0120 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
0130 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
0140 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
0150 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
0160 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
0170 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
0180 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
0190 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
01A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
01B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
01C0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
01D0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
01E0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
01F0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

Prog 01 : Question B

Faire la même chose pour l'ensemble des données 8 bits signées à partir de l'adresse de base $0200.

Prog 01 : Commentaires B

Il faudra en premier lieu charger la mémoire avec les nombres négatifs en décrémentant de $FF à$80, puis charger les nombres positifs en incrémentant de $00 à$7F.

retour au Sommaire | Index | Liens Rapides

Prog 01 : Programme B

Création d'une table de données en bits signés

```
        ORG     $0000     ;  Début du programme
        LDX     #$0200    ;  Début 1ere donnée négative
        LDY     #$0280    ;  Début 1ere donnée positive
        LDA     #$FF      ;  1ere donnée négative $FF
BOUCLE  STA     ,X+       ;  Chargement et incrémentation du pointeur X
        CMPA    #$80      ;  Si donnée = $80 fin des données négatives
        BEQ     POSITIF   ;
        DECA              ;  Décrémentation de la donnée
        BRA     BOUCLE    ;
POSITIF LDA     #$00      ;  1ere donnée positive
BOUCLE1 STA     ,Y+       ;  Chargement et incrémentation du pointeur
        YCMPA   #$7F      ;  Si donnée = $7F fin des données positives
```

```
        BEQ     FIN         ;
        INCA                ;  Incrémentation de la donnée
        BRA     BOUCLE1     ;
FIN     SWI                 ;
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

```
0200 FF FE FD FC FB FA F9 F8 F7 F6 F5 F4 F3 F2 F1 F0
0210 EF EE ED EC EB EA E9 E8 E7 E6 E5 E4 E3 E2 E1 E0
0220 DF DE DD DC DB DA D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
0230 CF CE CD CC CB CA C9 C8 C7 C6 C5 C4 C3 C2 C1 C0
0240 BF BE BD BC BB BA B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
0250 AF AE AD AC AB AA A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
0260 9F 9E 9D 9C 9B 9A 99 98 97 96 95 94 93 92 91 90
0270 8F 8E 8D 8C 8B 8A 89 88 87 86 85 84 83 82 81 80
0280 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0290 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
02A0 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
02B0 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
02C0 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
02D0 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
02E0 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
02F0 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
```

6809 Prog 02 : Dénombrement de données spécifiques dans une table

retour au Sommaire | Index | Liens Rapides

Prog 02 : Sujet

On souhaite, dans ce problème, évaluer le nombre de données d'une table qui répondent à une même caractéristique.

Prog 02 : Question A

Proposer un programme permettant d'effectuer le comptage des données positives, négatives et nulles d'une table de nombres signés de 8 bits. Le programme devra permettre de stocker ces résultats aux adresses $0050, $0051,$0052 par exemple.

Prog 02 : Commentaires A

Après avoir chargé la valeur dans le registre A, qui automatiquement positionne les bits N et Z, on peut utiliser les instructions de branchements qui en découlent.

retour au Sommaire | Index | Liens Rapides

Prog 02 : Programme A

Tri de données positives, négatives ou nulle

```
TABLE   EQU     $1000       ;  Déclaration du début de table
FIN_TAB EQU     $1009       ;  Déclaration du pointeur de fin de table
        ORG     $0000       ;  Début du programme
        LDX     #TABLE      ;  Chargement du pointeur
Boucle  CMPX    #FIN_TAB+1  ;  Si le pointeur dépasse la fin de la table
        BEQ     FIN         ;  alors FIN
        LDA     ,X+         ;  Chargement et incrémentation du pointeur
        BMI     Negatif     ;  Si l'opération est négative -> Négatif
        BEQ     Nul         ;  Si A = 0 -> Nul
        LDB     >$0050      ;  Sinon la données est positive
        INCB                ;  Incrémente le compteur situé en $0050
        STB     >$0050      ;  On mémorise la valeur
        BRA     Boucle      ;
Negatif LDB     >$0051      ;  La données est négative
        INCB                ;  Incrémente le compteur situé en $0051
        STB     >$0051      ;  On mémorise la valeur
        BRA     Boucle      ;
Nul     LDB     >$0052      ;  La données est nulle
        INCB                ;  Incrémente le compteur situé en $0052
        STB     >$0052      ;  On mémorise la valeur
        BRA     Boucle      ;
FIN     SWI                 ;
                            ;
        ORG     $1000       ;  Début de la TABLE
        FCB     -1,-1,0,5   ;
```

```
        FCB     8,-7,0,-9    ;
        FCB     2,6          ;
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

Résultats du dénombrement

```
0050 04 04 02 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Table des données

```
1000 FF FF 00 05 08 F9 00 F7 02 06 00 00 00 00 00 00
```

Prog 02 : Question B

Proposer un programme permettant d'effectuer le comptage du nombre de données paires et impaires d'une table.

Prog 02 : Commentaires B

Pour connaître la parité d'un mot de 8 bit, il suffit de faire un ET logique entre le mot et $11. Si le résultat est zéro alors le nombre est pair, sinon il est impair.

retour au Sommaire | Index | Liens Rapides

Prog 02 : Programme B

Tri de données paires ou impaires

```
TABLE   EQU     $1000        ;  Déclaration du début de table
FIN_TAB EQU     $1009        ;  Déclaration du pointeur de fin de table
        ORG     $0000        ;  Début du programme
        LDX     #TABLE       ;  Chargement du pointeur
Boucle  CMPX    #FIN_TAB+1   ;  Si le pointeur dépasse la fin de la table
        BEQ     FIN          ;  alors FIN
        LDA     ,X+          ;  Chargement et incrémentation du pointeur
        ANDA    #$11         ;  ET logique avec $11 pour connaître la parité
        CMPA    #$00         ;  Si A = 0 la donnée est paire -> Pair
        BEQ     Pair         ;
        LDB     >$0050       ;  Sinon la donnée est impaire
        INCB                 ;  Incrémentation du compteur
        STB     >$0050       ;  Mémorisation du compteur
        BRA     Boucle       ;
Pair    LDB     >$0051       ;  La donnée est paire
        INCB                 ;  Incrémentation du compteur
        STB     >$0051       ;  Mémorisation du compteur
        BRA     Boucle       ;
FIN     SWI                  ;
                             ;
        ORG     $1000        ;  Début de la TABLE
        FCB     1,2,3,4,5,6,7,8,9,0
```

Etat de la mémoire après exécution du programme

Résultat du dénombrement

```
0050 05 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Table des données

```
1000 01 02 03 04 05 06 07 08 09 00 00 00 00 00 00 00
```

retour au Sommaire | Index | Liens Rapides

Prog 02 : Question C

Proposer un programme permettant de compter le nombre de données d'une table dont le bit b3 est égal à 1.

Prog 02 : Commentaires C

Pour connaître l'état du bit 3 d'un nombre de 8 bit, il suffit de faire un ET logique entre ce mot et $08, si le résultat est égal à 0, le bit 3 est à 0, sinon le bit 3 est à 1.

Prog 02 : Programme C

Tri de données suivant la valeur du bit 3 de la donnée

```
TABLE   EQU     $1000        ;  Déclaration du début de table
FIN_TAB EQU     $1009        ;  Déclaration du pointeur de fin de table
        ORG     $0000        ;  Début du programme
        LDX     #TABLE       ;  Chargement du pointeur Boucle
        CMPX    #FIN_TAB+1   ;  Si le pointeur dépasse la fin de la table
        BEQ     FIN          ;  alors FIN
        LDA     ,X+          ;  Chargement et incrémentation du pointeur
        ANDA    #$08         ;  ET logique avec $08 pour savoir si bit3=1
        CMPA    #$00         ;  Si A = 0 bit3=0 -> Boucle
        BEQ     Boucle       ;
        LDB     >$0050       ;  Sinon bit3=1
        INCB                 ;  Incrémentation du compteur
        STB     >$0050       ;  Mémorisation du compteur
        BRA     Boucle       ;
```

```
FIN     SWI                 ;
                            ;
        ORG     $1000       ;  Début de la TABLE
        FCB     1,2,3,4,5,6,7,8,9,0
```

Etat de la mémoire après exécution du programme

Résultat du dénombrement

```
0050 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Table des données

```
1000 01 02 03 04 05 06 07 08 09 00 00 00 00 00 00 00
```

6809 Prog 03 : Multiplication

retour au Sommaire | Index | Liens Rapides

Prog 03 : Question

Soit le nombre hexadécimal X1=$23.
Mettre au point un programme permettant de trouver le nombre X2 tel que le produit X1*X2 soit strictement inférieur à $0299.

Prog 03 : Commentaire

Pour connaître X2, on incrémente un mot de 8 bit que l'on multiplie à $23, puis on teste le résultat à pour savoir s'il est supérieur ou égal à la valeur que l'on recherche. Si c'est le cas, la valeur de X2 est donc le mot de 8 bit –1, puisque l'on désire obtenir un résultat strictement inférieur.

Prog 03 : Programme

Recherche du résultat − 1 d'une division

```
        ORG     $0000       ;  Début du programme
        LDA     #$23        ;  Chargement de la valeur à multiplier X1
        LDB     #$01        ;  Chargement de la 1ere valeur
BOUCLE  STB     $1000       ;  Mise en mémoire de l'accumulateur B
        MUL                 ;  Multiplication de A par B
        CMPD    #$0299      ;  Si A.B est inférieur ou égal à $0299
        BHS     RESULT      ;  alors RESULT
        LDB     >$1000      ;  Recharge de l'accumulateur B
        INCB                ;  Incrémentation de l'accumulateur B
        LDA     #$23        ;  Recharge de l'accumulateur A
        BRA     BOUCLE      ;
RESULT  LDB     >$1000      ;  Recharge de l'accumulateur B
        DECB                ;  Décrémentation de l'accumulateur B
        SWI                 ;
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

Résultat en $1000

```
1000 13 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Etat du registre B après exécution du programme

B = 12 C'est le résultat que l'on cherchait !

6809 Prog 04 : Détermination du maximum ou du minimum d'une table

retour au Sommaire | Index | Liens Rapides

Prog 04 : Question A

On dispose d'une table de 10 données de 8 bits choisies arbitrairement. Proposer un programme de recherche de la donnée maximale et de la donnée minimale de la liste, les nombres considérés étant non signés.

Prog 04 : Commentaire A

Pour connaître le MIN et le MAX d'une table, on enregistre d'abord la 1ère donnée dans MIN et dans MAX, puis on vient les comparer avec la valeur suivante. Si la nouvelle donnée est plus grande que la valeur contenue dans MAX, on met da nouvelle valeur dans MAX, on procède de manière identique pour la valeur MIN.
Dans le cas ou la valeur n'est ni un MIN, ni un MAX, on pointe sur la valeur suivante de la table.

Prog 04 : Programme A

Tri de données MAX et MIN en non signé

```
MIN     EQU     $0050       ;  Déclaration de l'adresse du MAX
MAX     EQU     $0060       ;  Déclaration de l'adresse du MIN
TABLE   EQU     $1200       ;  Déclaration du pointeur de fin de table
```

```
        ORG     $0000           ;  Début du programme
        LDX     #TABLE          ;  Chargement du pointeur
        LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MAX            ;  Mémorise la 1ere valeur dans MAX
        STA     >MIN            ;  Mémorise la 1ere valeur dans MIN
Boucle  CMPX    #TABLE+10       ;  Si le pointeur dépasse la fin de la table
        BEQ     FIN             ;  alors FIN
        LDA     ,X              ;  Chargement et incrémentation du pointeur
        CMPA    >MAX            ;  Si A > MAX -> HightBHI Hight
        LDA     ,X              ;  Chargement et incrémentation du pointeur
        CMPA    >MIN            ;  Si A < MIN -> Low
        BLO     Low             ;
        LDA     ,X+             ;  Chargement et incrémentation du pointeur
        BRA     Boucle          ;
Hight   LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MAX            ;  Mémorise la valeur dans MAX
        BRA     Boucle          ;
Low     LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MIN            ;  Mémorise la valeur dans MIN
        BRA     Boucle          ;
FIN     SWI                     ;
                                ;
        ORG     $1200           ;  Début de la TABLE
        FCB     2,2,3,4,5,0,7,7,7,7
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Valeur MIN = 0

```
0060 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Valeur MAX = 7 Table de données

```
1200 02 02 03 04 05 00 07 07 07 07 00 00 00 00 00 00
```

Prog 04 : Question B

Compléter ce programme de sorte qu'il soit capable de déterminer également le maximum et le minimum lorsque les données sont signées.

Prog 04 : Commentaire B

La méthode générale est la même que l'exercice précédent, seules les instructions de branchement BGT et BLT sont modifiées pour travailler sur des données signées.

retour au Sommaire | Index | Liens Rapides

Prog 04 : Programme B

Tri de données MAX et MIN en signé

```
MIN     EQU     $0050           ;  Déclaration de l'adresse du MAX
MAX     EQU     $0060           ;  Déclaration de l'adresse du MIN
TABLE   EQU     $1200           ;  Déclaration du pointeur de fin de table
                                ;
        ORG     $0000           ;  Début du programme
        LDX     #TABLE          ;  Chargement du pointeur
        LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MAX            ;  Mémorise la 1ere valeur dans MAX
        STA     >MIN            ;  Mémorise la 1ere valeur dans MIN
Boucle  CMPX    #TABLE+10       ;  Si le pointeur dépasse la fin de la table
        BEQ     FIN             ;  alors FIN
        LDA     ,X              ;  Chargement et incrémentation du pointeur
        CMPA    >MAX            ;  Si A > MAX -> Hight
        BGT     Hight           ;
        LDA     ,X              ;  Chargement et incrémentation du pointeur
        CMPA    >MIN            ;  Si A < MIN -> Low
        BLT     Low             ;
        LDA     ,X+             ;  Chargement et incrémentation du pointeur
        BRA     Boucle          ;
Hight   LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MAX            ;  Mémorise la valeur dans MAX
        BRA     Boucle          ;
Low     LDA     ,X+             ;  Chargement et incrémentation du pointeur
        STA     >MIN            ;  Mémorise la valeur dans MIN
        BRA     Boucle          ;
FIN     SWI                     ;
                                ;
        ORG     $1200           ;  Début de la TABLE
        FCB     -2,2,3,-4,5,0,7,7,7,7
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

```
0050 FC 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Valeur MIN = FC soit 4

```
0060 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Valeur MAX = 07 soit 7 Table de données

```
1200 FE 02 03 FC 05 00 07 07 07 07 00 00 00 00 00
```

6809 Prog 05 : Transfert d'une table de données d'une zone mémoire vers une autre

retour au Sommaire | Index | Liens Rapides

Prog 05 : Question A

On dispose d'une table de 10 données de 8 bits, choisies arbitrairement, dont l'adresse de base est ADR1. Proposer un programme permettant de transférer cette table à l'adresse de base ADR2.

Prog 05 : Commentaire A

La méthode utilisée ici consiste à charger une valeur dans le registre A en ce servant du pointeur X (identifiant de la table source), et de stoker cette valeur à l'adresse désignée par le pointeur Y (identifiant la table de destination).

Prog 05 : Programme A

Transfert d'une table de 10 données de ADR1 -> ADR2

```
ADR1    EQU     $0050       ;  Déclaration de l'adresse ADR1
ADR2    EQU     $0060       ;  Déclaration de l'adresse ADR2
                            ;
        ORG     $0000       ;
        LDX     #ADR1       ;  Chargement du pointeur X
        LDY     #ADR2       ;  Chargement du pointeur Y
Boucle  LDA     ,X+         ;  Chargement et incrémentation du pointeur X
        STA     ,Y+         ;  Chargement et incrémentation du pointeur Y
        CMPX    #ADR1+10    ;  Si le pointeur dépasse la fin de la table
        BNE     Boucle      ;  alors FIN
        SWI                 ;
                            ;
        ORG     $0050       ;
        FCB     0,1,9,3,4,5,2,7,5,9
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

Table à l'adresse ADR1

```
0050 00 01 09 03 04 05 02 07 05 09 00 00 00 00 00 00
```

Table à l'adresse ADR2

```
0060 00 01 09 03 04 05 02 07 05 09 00 00 00 00 00 00
```

Prog 05 : Question B

Proposer un programme permettant de ranger les nombres hexadécimaux $00 à $09 aux adresses $0100 à $0109 et leur complément à 1 aux adresses $0200 à $0209.
Sur le même principe, proposer un programme qui n'utilise qu'un seul pointeur.

Prog 05 : Commentaire B

Ce programme comporte une petite difficulté car l'utilisation d'un seul pointeur implique que pour pointer sur la table2 d'adresse de base $0200, il faut rajouter au pointeur un déplacement égal à $FF ($100−1).
Car l'adresse 2 et décalée de $0100 dans la mémoire par rapport à l'adresse de base de la table 1, la soustraction du 1 vient du fait de l'auto incrémentation de 1 est induite par le dernier chargement du registre A.

retour au Sommaire | Index | Liens Rapides

Prog 05 : Programme B

Transfert d'une table de 10 données de ADR1 -> ADR2, 1 pointeur

```
TABLE   EQU     $0100       ;  Déclaration de l'adresse de TABLE
                            ;
        ORG     $0000       ;
        LDX     #TABLE      ;  Chargement du pointeur X
        LDA     #$00        ;  Initialisation de l'accumulateur A
Boucle  STA     ,X+         ;  Mémorisation de A à l'adresse pointée par X
        COMA                ;  Complément à 1 de A
        STA     $FF,X       ;  Mémorisation de A à l'adresse X+$FF
        COMA                ;  Complément à 1 de A -> valeur initiale
```

```
        INCA            ;  Incrémentation de la valeur de A
        CMPA    #$0A    ;  Si A = $0A
        BEQ     FIN     ;  alors FIN
        BRA     Boucle  ;
FIN     SWI             ;
```

Etat de la mémoire après exécution du programme

Table de données en ADR1

```
0100 00 01 02 03 04 05 06 07 08 09 00 00 00 00 00 00
```

Complément à 1 en ADR2

```
0200 FF FE FD FC FB FA F9 F8 F7 F6 00 00 00 00 00 00
```

retour au Sommaire | Index | Liens Rapides

Prog 05 : Question C

On dispose maintenant de deux tables de 10 données de 16 bits choisies arbitrairement. ADR1 et ADR2 sont les adresses de base de ces tables. On souhaite construire une troisième table, d'adresse de base ADR3, dont chaque élément résulte de l'addition des éléments de même rang des deux premières tables. Proposer le programme correspondant.

Prog 05 : Commentaire C

Le fait d'avoir trois tables de données, impose d'utiliser le pointeur X, pour pointer à la fois la table 1 et la table 2. Le déplacement rajouté à X, sera calculé de la manière suivante (ADR2 ADR1 2), la soustraction de 2 est ici due au fait que nous travaillons sur des données de 16 bits, de même les auto-incrémentations sont aussi sur 16 bits. Le pointeur Y quand a lui sert à pointer sur l'adresse de la table 3.

Prog 05 : Programme C

Addition de ADR1 + ADR2 -> ADR3, données de 16 bits

```
ADR1    EQU     $0050       ;  Déclaration de l'adresse ADR1
ADR3    EQU     $0090       ;  Déclaration de l'adresse ADR3
                            ;
        ORG     $0000       ;
        LDX     #ADR1       ;  Chargement du pointeur X
        LDY     #ADR3       ;  Chargement du pointeur Y
Boucle  LDD     ,X++        ;  Chargement de D et incrémentation de 2
        ADDD    $1E,X       ;  Addition de D avec le contenue de X+$1E
        STD     ,Y++        ;  Mémorisation de D et incrémentation de 2
        CMPX    #ADR1+20    ;  Si X = ADR1+20 alors fin de la table
        BNE     Boucle      ;  et du programme
        SWI                 ;
                            ;
        ORG     $0050       ;  Début de la ADR1
        FCB     $00,$00,$00,$01,$10,$13,$52,$30,$56,$89
        FCB     $21,$54,$14,$25,$01,$25,$87,$28,$45,$78

        ORG     $0070       ;  Début de la ADR2
        FCB     $01,$01,$01,$01,$01,$01,$01,$01,$01,$01
        FCB     $01,$01,$01,$01,$01,$01,$01,$01,$01,$01
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire après exécution du programme

Table 1 à l'adresse ADR1

```
0050 00 00 00 01 10 13 52 30 56 89 21 54 14 25 01 25
0060 87 28 45 78 00 00 00 00 00 00 00 00 00 00 00 00
```

Table 2 à l'adresse ADR2

```
0070 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
0080 01 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00
```

ADR1+ADR2 à l'adresse ADR3

```
0090 01 01 01 02 11 14 53 31 57 8A 22 55 15 26 02 26
00A0 88 29 46 79 00 00 00 00 00 00 00 00 00 00 00 00
```

6809 Prog 06 : Détermination logicielle de la parité croisée d'une table de données

retour au Sommaire | Index | Liens Rapides

Prog 06 : Question A

On dispose d'une table de 10 données correspondant à des caractères ASCII (codés sur 7 bits).Proposer un programme permettant de déterminer la clé de parité paire de chaque élément de la table et, le cas échéant, de rajouter cette clé devant la donnée.

Prog 06 : Exemple A

Supposons que le premier élément de la table soit le caractère ASCII " a ", qui se traduit par la combinaison 110 0001(soit $61 en hexadécimal). La clé de parité paire de ce caractère étant égale à 1, le programme devra permettre de modifier la donnée $61 et de la remplacer par $E1.

Prog 06 : Commentaire A

Pour connaître la clé de parité paire d'un nombre sur 7 bit, il faut compter le nombre de bit à 1 qui le compose, pour cela nous avons utilisé le décalage logique à gauche qui à la particularité de faire rentrer le bit de poids fort dans le bit C (CARRY), et lors du décalage à gauche d'insérer un zéro dans le bit de poids faible.

Il suffira ensuite d'incrémenter un compteur de " 1 ", l'opération s'arrêtera quand le nombre traité dans le registre A sera égal à $00.

Une fois ceci fait il faudra déterminer si le nombre de 1 est pair ou impair cf. ED1−P2−Q2, enfin dans le cas ou celui−ci serait impair pour mettre à 1 le bit 8 du nombre il suffira de faire un OU logique entre le nombre et $80. Et pour finir de stocker le nouveau nombre en remplacement de l'ancien.

retour au Sommaire | Index | Liens Rapides

Prog 06 : Programme A

Détermination logicielle de la parité d'un mot de 8 bits

```
COMPTEUR EQU     $0080        ;  Déclaration de la variable COMPTEUR
TABLE    EQU     $00A0        ;  Déclaration de la table TABLE
                              ;
         ORG     $0000        ;  Début du programme
         LDX     #TABLE       ;  Chargement du pointeur X
         CLRB                 ;  RAZ de B
         CLR     COMPTEUR     ;  RAZ de COMPTEUR
DATA     LDA     ,X+          ;  Chargement et incrémentation de X
         CMPX    #TABLE+11    ;  Si on a atteint la fin de la table
         BEQ     FIN          ;  alors FIN
BOUCLE   LSLA                 ;  Décalage logique à gauche de A
         BCS     INCREM       ;  Branchement si Carry = 1
RETOUR   CMPA    #$00         ;  Comparaison de A avec $00
         BEQ     PARITE       ;  Si A = $00 -> PARITE
         BRA     BOUCLE       ;  sinon -> BOUCLE
INCREM   INC     COMPTEUR     ;  Incrémentation de COMPTEUR
         BRA     RETOUR       ;  Aller à RETOUR
PARITE   LDA     COMPTEUR     ;  Chargement de A avec COMPTEUR
         ANDA    #$11         ;  ET logique entre A et $11 pour déterminer la parité
         CMPA    #$00         ;  Comparaison de A avec $00
         BNE     IMPAIR       ;  Si A est différent de $00 -> IMPAIR
         BRA     DATA         ;  Sinon -> DATA
IMPAIR   LDA     ,-X          ;  Chargement et incrémentation de X
         ORA     #$80         ;  OU logique entre A et $08
         STA     ,X+          ;  Mémorisation de A et incrémentation pointeur
         CLR     COMPTEUR     ;  RAZ COMPTEUR
         BRA     DATA         ;  Retour à DATA
FIN      SWI                  ;  Fin du programme
                              ;
         ORG     $00A0        ;  Début de la table TABLE
         FCB   'a','b','c','a','b','c','a','b','c','d'
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire avant exécution du programme

Table de données

```
00A0 61 62 63 61 62 63 61 62 63 64 00 00 00 00 00 00
```

Etat de la mémoire après exécution du programme

Données + clé de parité

```
00A0 E1 E2 63 E1 E2 63 E1 E2 63 E4 00 00 00 00 00 00
```

Prog 06 : Question B

Le programme précédent permettant d'adjoindre aux données un contrôle transversal de la parité, le modifier de sorte qu'il soit également susceptible de déterminer puis d'ajouter à la fin de la table un octet de vérification de la parité longitudinale.

Prog 06 : Commentaire B

Le programme qui suit est quasiment similaire au précédant, il à juste fallu rajouter un compteur transversal, c'est à dire qui s'incrémente à chaque fois que l'on remplace une valeur dans la table d'origine en ajoutant une clé de parité. Il suffit ensuite de tester la parité de se compteur et de rajouter en fin de table un 1 si il est impair, un zéro si il est pair.

retour au Sommaire | Index | Liens Rapides

Prog 06 : Programme B

Détermination logicielle de la parité croisée d'une table de données

```
 CPTEUR  EQU     $0080       ;  Déclaration de la variable CPTEUR
CPTEURT EQU     $0081       ;  Déclaration de la variable CPTEURT
TABLE   EQU     $00A0       ;  Déclaration de la table TABLE
                            ;
        ORG     $0000       ;  Début du programme
        LDX     #TABLE      ;  Chargement du pointeur X
        CLRB                ;  RAZ de B
        CLR     CPTEUR      ;  RAZ de CPTEUR
DATA    LDA     ,X+         ;  Chargement et incrémentation de X
        CMPX    #TABLE+11   ;  Si on a atteint la fin de la table
        BEQ     TRANS       ;  alors -> TRANS
BOUCLE  LSLA                ;  Décalage logique à gauche de A
        BCS     INCREM      ;  Branchement si Carry = 1
RETOUR  CMPA    #$00        ;  Comparaison de A avec $00
        BEQ     PARITE      ;  Si A = $00 -> PARITE
        BRA     BOUCLE      ;  Sinon retour à BOUCLE
INCREM  INC     CPTEUR      ;  Incrémentation de CPTEUR
        BRA     RETOUR      ;  Retour à BOUCLE
PARITE  LDA     CPTEUR      ;  Chargement de A avec le contenue de VALEUR
        ANDA    #$11        ;  ET logique entre A et $11
                            ;          pour déterminer la parité
        CMPA    #$00        ;  Comparaison de A avec $00
        BNE     IMPAIR      ;  Si A est différent de 0 -> IMPAIR
        BRA     DATA        ;  Sinon retour à DATA
IMPAIR  LDA     ,-X         ;  Décrémentation de X et chargement de A
        ORA     #$80        ;  OU logique entre A et $08
        STA     ,X+         ;  Mémorisation de A
        INC     CPTEURT     ;  Incrémentation de CPTEURT
        CLR     COMPTEUR    ;  RAZ de COMPTEUR
        BRA     DATA        ;  Retour à DATA
TRANS   LDA     CPTEURT     ;  Chargement de A avec CPTEUR
        ANDA    #$11        ;  ET logique entre A et $11
        CMPA    #$00        ;  Comparaison entre A et $00
        BNE     IMPAIRT     ;  Si A est différent de $00 -> IMPAIR
        LDA     #$00        ;  Chargement de A avec $00
        STA     TABLE+10    ;  Mémorisation de A en TABLE+10
        BRA     FIN         ;  -> Fin du programme
IMPAIRT LDA     #$01        ;  Chargement de la valeur $01 dans A
        STA     TABLE+10    ;  Mémorisation de A en TABLE+10
FIN     SWI                 ;  Fin du programme
                            ;
        ORG     $00A0       ;  Début de la table TABLE
        FCB     'a','c','a','c','a','a','a','a','a','a'
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire avant exécution du programme

Table de données

```
00A0 61 63 61 63 61 61 61 61 61 61 00 00 00 00 00 00
```

Etat de la mémoire après exécution du programme

Données + clés de parité

```
00A0 E1 63 E1 63 E1 E1 E1 E1 E1 E1 01 00 00 00 00 00
```

NB: Clé de parité longitudinale

6809 Prog 07 : Tri des données d'une table

retour au Sommaire | Index | Liens Rapides

Prog 07 : Question

On dispose d'une table de 10 données de 8 bits rangées initialement dans un ordre quelconque. Mettre au point un programme effectuant un tri des données et permettant après exécution de les ranger dans l'ordre croissant à partir de la même adresse de base.

Prog 07 : Commentaire

En tout premier lieu, on fait une copie de la table de données, puis on charge la 1ère valeur de la table d'origine que l'on compare aux valeurs du tableau de recopie. A chaque fois que la valeur trouvée est plus grande on incrémente la valeur COMPTEUR (qui représente la position de la valeur dans le tableau final).

Quand la table de données est entièrement balayée, on vérifie dans la table de position (TABLED) si l'emplacement est libre (00 = libre ; 01 = occupé) ; cette méthode de table de position est utilisé pour résoudre le problème de valeurs identique, le COMPTEUR ayant dans ce cas la même valeur, on évite ainsi de réécrire la même valeur au même endroit, en incrémentant de 1 le COMPTEUR.

Si on remplace l'instruction de branchement BLO par BLT, ce programme est valable pour les nombres signés.

retour au Sommaire | Index | Liens Rapides

Prog 07 : Programme

Tri des données d'une table dans l'ordre croissant

```
TABLE    EQU    $0080        ;  Déclaration de la table TABLE
TABLEC   EQU    $00A0        ;  Déclaration de la table TABLEC
TABLED   EQU    $00C0        ;  Déclaration de la table TABLED
COMPTEUR EQU    $00D0        ;  Déclaration de la variable COMPTEUR
VALEUR   EQU    $00E0        ;  Déclaration de la variable VALEUR
                             ;
         ORG    $0000        ;  Début du programme
         LDX    #TABLE       ;  Chargement du pointeur X
         LDY    #TABLEC      ;  Chargement du pointeur Y
Copy     LDA    ,X+          ;  Chargement et incrémentation du pointeur X
         STA    ,Y+          ;  Chargement et incrémentation du pointeur Y
         CMPX   #TABLE+10    ;  Si le pointeur dépasse la fin de la table
         BNE    Copy         ;  alors Copy
         CLRB                ;
Boucle   LDB    COMPTEUR     ;  Chargement dans B du contenue de COMPTEUR
         LDX    #TABLEC      ;  Chargement du pointeur X
         LDA    B,X          ;  Chargement de A avec le contenue de X+B
         STA    VALEUR       ;  Mémorisation de A à l'adresse VALEUR
         CMPB   #$0A         ;  Si B = nombre de donnée de la table
         BEQ    FIN          ;  alors FIN
         CLRB                ;  RAZ de l'accumulateur B
Data     CMPX   #TABLEC+10   ; Si pointeur X est égal à fin de table
         BEQ    Mem          ;  alors -> Mem, mémorisation de la donnée
         LDA    ,X+          ;  Chargement et incrémentation du pointeur X
         CMPA   VALEUR       ;  Si A est strictement plus petit que VALEUR
         BLO    Compt        ;  alors -> Compt
         BRA    Data         ;  Sinon -> Data
Compt    INCB                ;  Incrémentation de B
         BRA    Data         ;  Retour à Data
Mem      LDX    #TABLED      ;  Chargement du pointeur X
         LDA    B,X          ;  Chargement de A avec le contenue de X+B
         CMPA   #$01         ;  Si A = $01
         BEQ    Egal         ;  alors pointeur déjà utilisé -> Egal
         LDA    VALEUR       ;  Chargement de A avec le contenue de VALEUR
         LDX    #TABLE       ;  Chargement du pointeur X
         STA    B,X          ;  Mémorisation de A à l'adresse X+B
         LDX    #TABLED      ;  Chargement du pointeur X
         LDA    #$01         ;  Chargement de A avec $01
         STA    B,X          ;  Case mémoire utilisée -> $01
         INC    COMPTEUR     ;  Incrémentation de COMPTEUR
         BRA    Boucle       ;  Retour à Boucle
Egal     INCB                ;  Incrémentation de B
         LDA    B,X          ;  Chargement de A avec le contenue de X+B
         CMPA   #$01         ;  Si A = $01
         BEQ    Egal         ;  alors pointeur déjà utilisé -> Egal
         LDA    VALEUR       ;  Chargement de A avec le contenue de VALEUR
         LDX    #TABLE       ;  Chargement du pointeur X
         STA    B,X          ;  Mémorisation de A à l'adresse X+B
         LDX    #TABLED      ;  Chargement du pointeur X
         LDA    #$01         ;  Chargement de A avec $01
         STA    B,X          ;  Case mémoire utilisée -> $01
         INC    COMPTEUR     ;  Incrémentation de COMPTEUR
         BRA    Boucle       ;  Retour à Boucle
FIN      SWI                 ;  Fin du programme
                             ;
         ORG    $0080        ;  Début de TABLE
         FCB    1,6,0,1,1,2,1,4,1,5
```

retour au Sommaire | Index | Liens Rapides

Etat de la mémoire avant exécution du programme

Table de données

```
0080 01 06 00 01 01 02 01 04 01 05 00 00 00 00 00 00
```

Etat de la mémoire après exécution du programme Table de données après le tri

```
0080 00 01 01 01 01 01 02 04 05 06 00 00 00 00 00 00
```

retour au Sommaire | Index | Liens Rapides

Prog 08 : Sujet

Proposer un programme permettant après addition de deux données de 8 bits de vérifier la validité du résultat obtenu et, le cas échéant, de le corriger.

Prog 08 : Commentaires

Si A et B dont des données non signées de 8 bits elles peuvent prendre des valeurs allant de 0 à 255 et leur somme peut aller de 0 à 510. Ainsi, si l'on exprime le résultat sur 8 bits on court le risque de provoquer des dépassements de capacité.

Néanmoins, il est possible de vérifier la validité du calcul en testant la valeur de la retenue finale dans le registre CCR

- si C = 0 le résultat non signé est correct sur 8 bits ;
- si C = 1 il y a dépassement de capacité sur 8 bits (résultat correct sur 9 bits avec C = 9ème bit du résultat).

Dans ce dernier cas, il existe plusieurs possibilités pour corriger l'erreur commise, la meilleure consistant à exprimer le résultat avec un octet supplémentaire en considérant C comme le 9ème bit. C'est cette méthode que nous retiendrons puis que la possibilité de travailler sur 16 bits par le biais de l'accumulateur D nous est offert sur le µp6809.

Si A et B sont des données signées de 8 bits elles peuvent prendre des valeurs allant de −128 à +127 en représentationCà2, et leur somme peut aller de −256 à +254. Dans ce cas, l'addition de deux nombres de même signe et de valeurs élevées provoquera des dépassements de capacité sur 8 bits qui seront cette fois indiqués par le bit de débordement V du registre CCR

- si V = 0 le résultat signé en Cà2 est correct sur 8 bits ;
- si V = 1 dépassement de capacité en Cà2 sur 8 bits (résultat correct sur 9 bits avec C = bit de signe).

Pour corriger l'erreur on choisira également d'exprimer le résultat sur 16 bits ce qui nécessite d'effectuer une extension de signe qui consiste à recopier le bit C sur l'ensemble de l'octet de poids fort. Considérons par exemple l'addition suivante

```
11100011          (−29)
10000001       +(−127)
----------------------
101100100         (−156)
```

retour au Sommaire | Index | Liens Rapides

Résultat faux sur 8 bits (+100), correct sur 9 bits avec C = bit de signe (−156) Pour exprimer ce résultat sur 16 bits il faut étendre le signe :
R = 11111111 01100100 = (−156).

Dans le programme proposé, les opérandes A et B à additionner sont placés aux adresse $0050 et $0051, le résultat de l'addition en non signé à l'adresse $0052 (et $0053 si la somme s'exprime sur l 6 bits) et le résultat de l'addition signée à l'adresse $0054 (et $0055 pour une somme sur 16 bits).

Par ailleurs, on choisit de placer le résultat de l'addition dans l'accumulateur B dans la mesure où celui−ci correspond à l'octet de poids faible de D (en cas d'erreur on rectifiera le contenu de A de manière à avoir le résultat correct dans D).

Pour tester la validité de la somme non signée on utilise un branchement conditionnel BCS qui aiguille le µP vers l'étiquette CORRUNS lorsque la retenue est égale à 1. La correction proprement dite consiste à faire pénétrer la retenue C dans l'accumulateur A par le biais de l'instruction ROLA.

Le résultat corrigé est ensuite placé en mémoire par stockage du contenu de D à l'adresse $0052.
Pour tester la validité de la somme signée on utilise un branchement BVS qui aiguille le µP vers l'étiquette CORRSIG lorsque le bit V est à 1.

retour au Sommaire | Index | Liens Rapides

Comme indiqué précédemment, la correction nécessite ici de faire une extension de signe par recopie du bit C. L'instruction SEX permet de réaliser une extension de signe en transformant un nombre signé enCà2 de 8 bits en un nombre signé en Cà2 de 16 bits par recopie du bit de signe de B dans l'accumulateur A.

Autrement dit, il faut au préalable faire pénétrer la retenue C à la place du bit de signe dans B avant d'utiliser l'instruction SEX : C'est le rôle de l'instruction RORB (C−>b7 et b0 −>C).

Après l'extension, on veillera à récupérer le bit de poids faible de B par une instruction ROLB (C−>b0).
Enfin, le résultat corrigé est stocké à l'adresse $0054.

Prog 08 : Programme

```
        CLRA
        LDB     $0050
        ADDB    $0051
        BCS     CORRUNS
        STB     $0052
SUITE   LDB     $0050
        ADDB    $0051
        BVS     CORRSIG
        STB     $0054
FIN     SWI
```

retour au Sommaire | Index | Liens Rapides

Correction de l'addition non signée

```
CORRUNS ROLA
        STD     $0052
        BRA     SUITE
```

Correction de l'addition signée

```
CORRSIG RORB
        SEX
        ROLB
        STD     $0054
        BRA     FIN

        ORG     $0050
DATA    FCB     $BC,$23
```

6809 Prog 09 : Table de correspondance hexadécimal décimal

retour au Sommaire | Index | Liens Rapides

Prog 09 : Question A

Analyser précisément le fonctionnement du programme proposé. Vous indiquerez ensuite la raison pour laquelle il ne fonctionne pas si, dans l'instruction de la ligne 3, on initialise le pointeur X par l'adresse $0150 par exemple.

Prog 09 : Programme A

```
        CLRA
        LDU     #$0050
        LDX     #$0100
        TFR     X,D
        EXG     A,B
BOUCLE  STA     ,X+
        CMPX    #$0164
        BEQ     FIN
        INCA
        PSHUA
        ANDA    #$0A
        CMPA    #$0A
        PULU    A
        BNE     BOUCLE

        DAA
        BRA     BOUCLE
        SWI
        END
```

retour au Sommaire | Index | Liens Rapides

Prog 09 : Commentaires A

Pour établir la table de correspondance, la méthode générale mise en œuvre dans le programme consiste à récupérer l'octet de poids faible de l'adresse de base de la table, puis de le placer dans l'accumulateur A et enfin à le ranger en mémoire.

Pour passer à la donnée suivante il suffit d'incrémenter de 1 le contenu de A puis de le ranger en mémoire.

Le premier problème se présente dès lors que le contenu de A est égal à $0A auquel cas il est nécessaire d'effectuer un ajustement décimal (instruction DAA) pour obtenir la traduction correcte. En effet, on ne dispose que de 10caractères (0 à 9) pour coder une donnée en décimal alors qu'il y en a 16 (0 à F) pour coder une donnée en hexadécimal.

Ainsi, pour traduire $0A, il faut lui ajouter $06 afin d'obtenir la correspondance adéquate $10. On peut ensuite continuer à incrémenter de 1 le contenu de A pour passer à la donnée suivante.

L'ajustement décimal devra être effectué chaque fois qu'apparaît le caractère "A" dans le nombre hexadécimal!

Ce programme ne répondra au fonctionnement souhaité que dans le cas où l'adresse de base de la table présente un octet de poids faible compris entre $00 et $09 dans la mesure où la première traduction correspond à une simple recopie de cet octet.

Prog 09 : Question B

Proposer une nouvelle version du programme assurant un fonctionnement correct quelle que soit l'adresse de base de la table.

retour au Sommaire | Index | Liens Rapides

Prog 09 : Commentaires B

Le principe du programme proposé pour respecter la contrainte imposée de même que précédemment, on commence par récupérer l'octet de poids faible de l'adresse de base de la table.

Pour traduire ce nombre en décimal il est nécessaire de savoir s'il correspond aux unités, aux dizaines, aux vingtaines, etc.

En effet, si on a affaire aux unités la traduction consiste en une simple recopie de l'octet de poids faible de l'adresse, si on a affaire aux dizaines il faut ajouter $06 à cet octet pour avoir son équivalent décimal, si on a affaire aux vingtaines il faut ajouter 2 * $06, etc.

Prog 09 : Programme B

```
ADRESSE EQU     $0126         ;  Adresse de base de la table
        LDX     #ADRESSE      ;
BOUCLE  TFR     X,D           ;
        CMPB    #$09          ;
        BHI     CORRIGE       ;  Si l'octet de poids faible est > $09 -> CORRIGE
;-----------------------------
NEXT    STB     ,X+           ;
        CMPX    #0164         ;
        BNE     BOUCLE        ;
        SWI                   ;
;------ Traitement des dizaines
CORRIGE CMPB    #$13          ;
        BHI     CORRIGE1      ;  Si l'octet de poids faible est > $13 ->CORRIGE1
        ADDB    #$06          ;  sinon ajustement décimal
        BRA     NEXT          ;
;------ Traitement des vingtaines
CORRIGE1 CMPB     #$1D        ;
         BHI      CORRIGE2    ; Si l'octet de poids faible est > $1D ->CORRIGE2
         ADDB     #$0C        ; sinon double ajustement décimal
         BRA      NEXT        ;
;------ Traitement des trentaines
CORRIGE2 CMPB     #$27        ;
         BHI      CORRIGE3    ; Si l'octet de poids faible est > $27 ->CORRIGE3
         ADDB     #$12        ; sinon triple ajustement décimal
         BRA      NEXT        ;
;-----------------------------
CORRIGE3 CMPB     #$31        ;
         BHI      CORRIGE4    ;
         ADDB     #$18        ;
         BRA      NEXT        ;
;-----------------------------
CORRIGE4 CMPB     #$3B        ;
         BHI      CORRIGE5    ;
         ADDB     #$1E        ;
         BRA      NEXT        ;
;-----------------------------
CORRIGE5 CMPB     #$45        ;
         BHI      CORRIGE6    ;
         ADDB     #$24        ;
         BRA      NEXT        ;
;-----------------------------
CORRIGE6 CMPB     #$4F        ;
         BHI      CORRIGE7    ;
         ADDB     #$2A        ;
         BRA      NEXT        ;
;-----------------------------
```

```
CORRIGE7 CMPB    #$59       ;
         BHI     CORRIGE8   ;
         ADDB    #$30       ;
         BRA     NEXT       ;
;--------------------------
CORRIGE8 ADDB    #$36       ;
         BRA     NEXT       ;
         END                ;
```

retour au Sommaire | Index | Liens Rapides

Prog 09 : Question C

Rédiger un programme de recherche de la traduction hexadécimale d'un nombre XX stocké en mémoire à l'adresse$0060.

Prog 09 : Commentaires C

Le programme de recherche de la traduction hexadécimale d'un nombre décimal XX quelconque (allant de 00 à 99) doit au préalable effectuer un rangement de la table de correspondance en mémoire.

Pour ce faire, nous utiliserons le programme proposé à la question 1 transformé ici en sous−programme.

La recherche proprement dite consistera alors en une lecture de la table jusqu'à ce que la donnée XX considérée soit décelée ; il suffit ensuite de récupérer l'octet de poids faible de son adresse dans la table pour avoir la traduction hexadécimale souhaitée.

Prog 09 : Programme C

```
ADR      EQU    $0100     ;  Définition de l'adresse de base de la table
         JSR    TABLE     ;  Stockage de la table en mémoire
         LDX    #TABLE    ;
LECTURE  LDA    ,X+       ;  Lecture de la table
         CMPA   $0060     ;
         BNE    LECTURE   ;
         LEAX   -1,X      ;
                          ;
         TFR    X,D       ;
         STB    $0061     ;
         SWI              ;
```

retour au Sommaire | Index | Liens Rapides

Sous programme de rangement de la table de correspondance en mémoire

```
TABLE    CLRA             ;
         LDU     #$0050   ;
         LDX     #ADR     ;
         TFR     X,D      ;
         EXG     A,B      ;
BOUCLE   STA     ,X+      ;
         CMPX    #$0164   ;
         BEQ     FIN      ;
         INCA             ;
         PSHUA            ;
         ANDA    #$0A     ;
         CMPA    #$0A     ;
         PULUA            ;
         BNE     BOUCLE   ;
         DAA              ;
         BRA     BOUCLE   ;
FIN      RTS              ;
                          ;
         ORG     $0060    ;
DATA     FCB     $23      ;
         END              ;
```

Prog 09 : Remarque C

L'adresse de la donnée $23 dans la table est $0117 ; donc suite à l'exécution de ce programme, c'est la traduction hexadécimale $17 qui sera placée à l'adresse $0061

6809 Prog 10 : Conversion DCB−binaire

retour au Sommaire | Index | Liens Rapides

Prog 10 : Question A

A l'aide de la méthode présentée en cours, établir un algorithme permettant de convertir un nombre entier codé en DCB sur 8 bits en binaire.

Prog 10 : Commentaires A

La conversion nécessite l'emploi de deux registres de 8 bits : l'un contenant initialement la donnée DCB à convertir, le second recueillant la donnée traduite en binaire.

Prog 10 : Algorithme A

```
A <- donnée DCB B <-0 Compteur <- 8
    Tant que (Compteur > 0)
        Décalage vers la droite des contenus
        combinés des registres A et B
        Si(a[3]= 1) alors
            A <- A - $03
        Fin si
        Compteur <- Compteur - 1
    Fin tant que
```

retour au Sommaire | Index | Liens Rapides

Prog 10 : Remarque A

Lorsqu'on parle de décalage à droite des contenus combinés des registres A et B cela signifie que le bit de poids faible de A doit être introduit comme bit de poids fort dans B.

Prog 10 : Question B

En vous appuyant sur l'algorithme précédent, proposer un programme réalisant la conversion d'un nombre DCB rangé en mémoire à l'adresse $0050 et stockant sa traduction binaire à l'adresse $0051.

Prog 10 : Programme B

```
        LDA     $0050       ;  Charge dans A la donnée à traduire
        CLRB                ;
        LDX     #$0008      ;  L'index X, utilisé comme pointeur, est
                            ;  initialisé à 8
DECALE  LSRA                ;  Décalage droite de A (a[0] -> C)
        RORB                ;  Rotation droite de B (C -> b[7])
        BITA    #$08        ;  Test du bit a[3]
        BEQ     SUITE       ;  Si a[3] = 0 on continue en SUITE
        SUBA    #$03        ;  sinon on retranche $03
SUITE   LEAX    -1,X        ;  Décrémentation du compteur
        BNE     DECALE      ;  S'il est non nul on poursuit les décalages
        STB     $0051       ;  sinon on stocke le résultat à l'adresse $0051
        SWI                 ;
                            ;
        ORG     $0050       ;
DATA    FCB     $28         ;  Donnée DCB à traduire
```

Prog 10 : Remarque B

Après exécution, on trouvera la donnée $1C à l'adresse $0051.

6809 Prog 11 : Multiplication

retour au Sommaire | Index | Liens Rapides

Prog 11 : Question A

Sur le modèle de l'algorithme proposé en cours, rédiger un programme réalisant la multiplication de deux données de8 bits non signées. Le multiplicande, le multiplicateur et le résultat obtenu seront placés aux adresses mémoire $0050, $0051 et $0052.

Prog 11 : Commentaires A

On rappelle que l'algorithme de multiplication de deux nombres non signés de 8 bits nécessite l'emploi de trois registres 8 bits. Le µp6809 n'en possédant que deux, on travaillera directement sur le contenu de la case mémoire renfermant le multiplicande.

Prog 11 : Algorithme A

```
A <- 0     B <- multiplicateur     $0050 <- multiplicande     Compteur <- 8
    Tant que (Compteur > 0)
        Si (b[0] = 1) alors
```

```
            A <- A + multiplicande
        Fin Si
        Décalage vers la droite des contenus
        combinés des registres A et B
        Compteur = Compteur - 1
    Fin Tant que
```

retour au Sommaire | Index | Liens Rapides

Prog 11 : Remarque A

En fin d'exécution, le résultat de la multiplication est situé dans D.

Prog 11 : Programme A

```
        LDX     #$0008      ;  Compteur
        CLRA                ;
        LDB     $0051       ;  Le multiplicateur est placé dans B
DEBUT   BITB    #$01        ;  Test b[0]
        BEQ     DECALE      ;  Si b[0] = 0 on passe en DECALE
        ADDA    $0050       ;  sinon on additionne le multiplicande
```

Décalage à droite de A-B dans le cas où le bit b [0] = 1

```
        RORA                ;  Rotation droite de A C -> a[7]
                            ;                     et a[0] -> C
        RORB                ;  Rotation droite de B C = a[0] -> b[7]
        LEAX    -1,X        ;
        BNE     DEBUT       ;
        STD     $0052       ;
        SWI                 ;  Décalage à droite de A-B dans le cas où b[0] =0
DECALE  LSRA                ;  Rotation droite de A 0 -> a[7]
                            ;                     et a[0] -> C
        RORB                ;  Rotation droite de B C = a[0] -> b[7]
        LEAX    -1,X        ;
        BNE     DEBUT       ;
        STD     $0052       ;
                            ;
        ORG     $0050       ;
DATA    FCB     $FF,$48     ;
        END                 ;
```

retour au Sommaire | Index | Liens Rapides

Prog 11 : Remarque A

On notera que le processus de décalage combiné des registres A et B diffère selon que b [0] = 1 ou que b[0] = 0.

En effet, dans un cas il faut introduire en a[7] la retenue issue de l'addition du multiplicande (RORA) et dans l'autre il faut introduire en a[7] un 0 (LSRA).

Pour l'exemple choisi, le résultat de la multiplication stocké à l'adresse $0052 (et $0053) sera $47B8.

Enfin, si l'on souhaite vérifier la validité du programme on peut ajouter la séquence suivante au début du programme

```
        LDA     $0050       ;
        LDB     $0051       ;
        MUL                 ;
        STD     $0054       ;
```

Prog 11 : Question B

Modifier le programme précédent de sorte qu'il puisse travailler sur des données de 16 bits.

Prog 11 : Commentaires B

L'algorithme utilisé reste identique au précédent mais la difficulté réside ici dans le fait que seuls les accumulateurs A et B peuvent subir des opérations de décalage ou de rotation.

Autrement dit, ces décalages doivent être effectués en deux temps : décalage de l'octet de poids fort de la donnée en premier lieu puis décalage de l'octet de poids faible en prenant soin de faire le report correctement.

Dans le programme présenté ci-dessous, le multiplicande est rangé aux adresses $0050 et $0051, le multiplicateur en$0052 et $0053 et le résultat de la multiplication en $0054, $0055, $0056 et $0057.

On travaillera donc sur ces quatre cases mémoire ; en particulier, le contenu de $0054 et $0055 est initialisé à 0 et le multiplicateur est placé en $0056 et $0057. Enfin, le compteur doit être initialisé par la valeur 16 soit $0010 en hexadécimal.

retour au Sommaire | Index | Liens Rapides

Prog 11 : Programme B

```
        LDX     #$0010      ;  Initialise le compteur
        CLR     $0054       ;
        CLR     $0055       ;
        LDD     $0052       ;
        STA     $0056       ;
        STB     $0057       ;  Place le multiplicateur en $0056 et $0057
DEBUT   BITB    #$01        ;  Teste le bit de poids faible du multiplicateur
        BEQ     DECALE      ;  S'il est égal à 0 on passe en DECALE
        LDD     $0054       ;
        ADDD    $0050       ;  S'il est à 1, on additionne le multiplicande
;------ au contenu des adresses $0054 et $0055
;   Décalage à droite du contenu combiné des adresses $0054
;   à $0057 dans le cas où le bit testé est égal à 1
        RORA                ;  Rotation à droite du contenu des adresses
                            ;  $0054 et $0055 de manière à prendre en
                            ;  compte la retenue issue de l'addition
        STA     $0054       ;
        RORB                ;
        STB     $0055       ;
        LDD     $0056       ;
        RORA                ;  Rotation droite du contenu adresses $0056 $0057
        STA     $0056       ;
        RORB                ;
        STB     $0057       ;
        LEAX    -1,X        ;
        BNE     DEBUT       ;
        SWI                 ;
;------ Décalage à droite du contenu combiné des adresses $0054 à $0057
;  dans le cas où le bit testé est égal à 0
DECALE  LDD     $0054       ;
        LSRA                ;
        STA     $0054       ;
        RORB                ;
        STB     $0055       ;
        LDD     $0056       ;
        RORA                ;
        STA     $0056       ;
        RORB                ;
        STB     $0057       ;
        LEAX    -1,X        ;
        BNE     DEBUT       ;
                            ;
        ORG     $0050       ;
DATA    FDB     $04FF,$0336 ;
        END                 ;
```

Prog 11 : Remarque B

A l'issue de l'exécution, le résultat de la multiplication placé de $0054 à $0057 est égal à $00100ACA.

6809 Prog 12 : Division

retour au Sommaire | Index | Liens Rapides

Prog 12 : Question A
Prog 12 : Question B
Prog 12 : Question C

Prog 12 : Question A

Proposer un programme effectuant la division de X par Y, où X et Y sont deux nombres de 8 bits non signés tels que X supérieur ou égal à Y et Y différent de 0. Le dividende, le diviseur, le quotient et le reste seront rangés en mémoire à des adresses successives.

La division de deux entiers binaires non signés de 8 bits nécessite l'emploi de trois registres 8 bits.

Prog 12 : Algorithme A

```
     A <- 0    B <- dividende      $0051 <- diviseur      Compteur <- 8
         Tant que (Compteur > 0)
             Décalage à gauche des registres
             combinés A et B

             Si (A < diviseur) alors
                 b[0] <- 0
             sinon
```

```
                A <- A - diviseur
                b[0] <- 1
            Fin si
            Compteur <- Compteur - 1
        Fin Tant que
```

En fin d'exécution, le quotient est situé dans B et le reste dans A.

retour au Sommaire | Index | Liens Rapides

Prog 12 : Programme A

```
        CLRA            ;
        LDB     $0050   ;  Place le dividende dans B
        LDX     #$0008  ;  Compteur
DEBUT   ASLB            ;  Décalage gauche des
                        ;  registres A et B
;------ attention ici il faut commencer par l'octet de poids faible
;  afin de faire le report correctement sur l'octet de poids fort
        ROLA            ;
        CMPA    $0051   ;
        BLO     SUITE   ;  Si A < diviseur -> SUITE
        SUBA    $0051   ;  sinon on retranche le diviseur
        ORB     #$01    ;  et on force b0 à 1
SUITE   LEAX    -1,X    ;
        BNE     DEBUT   ;
        STB     $0052   ;  Range le quotient en $0052
        STA     $0053   ;  Range le reste en $0053
        SWI             ;
                        ;
        ORG     $0050   ;
DATA    FCB     $FC,$26 ;
        END             ;
```

retour au Sommaire | Index | Liens Rapides

Prog 12 : Question B

Comme application, écrire un programme permettant de réaliser la conversion inverse de celle qui est traitée dans le problème P3.

Le nombre entier binaire de 8 bits à convertir sera stocké à l'adresse $0050 et sa traduction en DCB à l'adresse $0051. On utilisera un sous programme pour effectuer les divisions nécessaires.

Prog 12 : Commentaires B

Rappelons que la méthode de conversion binaire−DCB consiste à diviser le nombre binaire par 10 successivement jusqu'à obtenir un résultat nul ; les restes des divisions écrits sur 4 bits correspondent à chaque chiffre DCB, le premier reste traduisant le chiffre des unités.

Prog 12 : Programme B

```
        LDY     #$0051  ;
BOUCLE  JSR     DIVISE  ;
        STA     ,Y+     ;
        CMPB    #$00    ;
        BNE     BOUCLE  ;
        SWI             ;
                        ;
DIVISE  CLRA            ;
        LDB     $0050   ;
        LDX     #$0008  ;
DEBUT   ASLB            ;
        ROLA            ;
        CMPA    #$0A    ;
        BLO     SUITE   ;
        SUBA    #$0A    ;
        ORB     #$01    ;
SUITE   LEAX    -1,X    ;
        BNE     DEBUT   ;
        STB     $0050   ;
        RTS             ;
                        ;
        ORG     $0050   ;
DATA    FCB     $DC     ;
```

retour au Sommaire | Index | Liens Rapides

Prog 12 : Remarque B

La donnée binaire $DC = 1101 1100 doit être traduite en BCD par 0010 0010 000 (soit $0220 en hexadécimal). A l'issue de l'exécution de ce programme, on trouve $00, $02, $02 aux adresses $0051, $0052 et $0053 ce qui traduit le fait que les restes sont ici calculés sur 8 bits. Ainsi, le résultat souhaité est obtenu en examinant les 4 bits de poids faibles de ces 3 données.

Prog 12 : Question C

Proposer un programme de division travaillant sur des données de 8 bits signées. On rappelle que l'algorithme de division de nombres signés se présente sous la forme suivante

Prog 12 : Algorithme C

```
     M <- diviseur      R-Q <- dividende      Compteur <- n S <- R[n-1]
         Tant que (Compteur > 0)
             Décalage gauche des registres combinés R et Q
             T <- R
             Si ( R[n-1] = M[n-1]) alors
                 R <- R-M
             sinon
                 R <- R+M
             Finsi

             Si ( R[n-1] = T[n-1]) OU ( R = 0 ET Q = 0 ) alors
                 Q[0] <- 1
             sinon
                 Q[0] <- 0
                 R <- T (restaure le contenu de R)
             Finsi
             Compteur <- Compteur - 1
         Fin Tant que

         Si (S <> M[n-1]) alors
             Q <- Cà2 de Q
         Fin Si
```

retour au Sommaire | Index | Liens Rapides

Dans le programme propose ci-dessous, les cases mémoire utilisées se présentent comme suit :

```
     ADRESSE        CONTENU
     $0001          dividende
     $0002          signe du dividende (S)
     $0003          diviseur (M)
     $0004          quotient (Q)
     $0005          reste (R)
     $0006          registre temporaire (T)
```

Prog 12 : Programme C

```
          ORG     $0020       ;
          LDA     $0001       ;  Charge le dividende dans A
          BMI     SIGNE       ;  S'il est négatif on passe en SIGNE
          CLRA                ;  S'il est positif, on fixe S=$00
          STA     $0002       ;
NEXT      LDB     $0001       ;  Charge le dividende dans B
          SEX                 ;  Etend le signe du dividende dans l'accumulateur A
          STB     $0004       ;  Initialise Q avec le dividende
          STA     $0005       ;  Initialise R avec le signe du dividende
          LDB     #$08        ;  Initialise le compteur
          STB     $000A       ;  Compteur placé à l'adresse $000A
;------ Décalage gauche des registres combinés R-Q
DEBUT     LDA     $0004       ;
          ASLA                ;
          STA     $0004       ;
          LDA     $0005       ;
          ROLA                ;
          STA     $0005       ;
          STA     $0006       ;  Sauvegarde temporaire
          LDA     $0003       ;  Charge M dans A
          BMI     NEGATIF     ;  S'il est négatif on passe en NEGATIF
          LDA     $0005       ;  S'il est positif on charge le reste dans R
          BMI     ADDITION    ;  Si M > 0 et R < 0 an passe en ADDITION
          BRA     SOUSTRAI    ;  On passe en SOUSTRAIT car M > 0 et R > 0
```

retour au Sommaire | Index | Liens Rapides

```
;------ Vérification de la condition R[n-1] = T[n-1]
TEST2     LDA     $0006       ;
          BMI     NEGTIF2     ;  Si T < 0 on passe en NEGATIF2
          LDA     $0005       ;  Si T > 0 on charge R dans A
          BMI     RESTAUR     ;  Si T > 0 et R < 0 on passe en RESTAURE
QUN       LDA     $0004       ;  Cas T > 0 et R > 0
          ORA     #$01        ;  On force Q[0] à 1
          STA     $0004       ;
COMPT     DEC     $000A       ;  Décrémente le compteur
          BNE     DEBUT       ;  Tant que compteur > 0 on retourne en DEBUT
```

```
          LDA     $0003       ;
          BMI     NEG3        ;  Si M < 0 on passe en NEG3
          CLRA                ;  Cas M > 0 : on teste ta condition M[n-1] = S
          CMPA    $0002       ;
          BEQ     FIN         ;  Si M[n-1] = S on passe en FIN
COMPLEM   LDA     $0004       ;  Cas M[n-1] <> S, on prend le Cà2 du quotient
          NEGA                ;
          STA     $0004       ;
FIN       SWI                 ;
;------ Lorsque le dividende est négatif, on fixe S = $01
SIGNE     LDA     #$01        ;
          STA     $0002       ;
          BRA     NEXT        ;
          SWI                 ;
;------ Cas M < 0, étude du signe de R
NEGTIF    LDA     $0005       ;
          BMI     SOUSTRAI    ;  Si R < 0 on passe en SOUSTRAIT
          BRA     ADDITION    ;  Cas R > 0 on passe en ADDITION
          SWI                 ;
ADDITION  ADDA    $0003       ;
          STA     $0005       ;
          BEQ     TEST        ;  Si R = 0 on passe en TEST
          BRA     TEST2       ;  Cas R <> 0, on passe en TEST2
          SWI                 ;
SOUSTRAI  SUBA    $0003       ;
          STA     $0005       ;
          BEQ     TEST        ;  Si R = 0 on passe en TEST
          BRA     TEST2       ;  Cas R <> 0, on passe en TEST2
          SWI                 ;
```

retour au Sommaire | Index | Liens Rapides

```
;------ Cas R = 0, teste si Q = 0
TEST      LDA     $0004       ;
          BEQ     QUN         ;  Si Q = 0 on passe en QUN
          BRA     TEST2       ;  Cas Q <> 0, on passe en TEST2
          SWI                 ;
;------ Cas T < 0. étude du signe de R
NEGATIF2  LDA     $0005       ;
          BMI     QUN         ;  Si R < 0 on passe en QUN
          BRA     RESTAUR     ;  Cas R > 0, on passe en RESTAURE
          SWI                 ;
RESTAUR   LDA     $0006       ;
          STA     $0005       ;
          BRA     COMPT       ;
          SWI                 ;
;------ Cas M < 0, teste la condition M[n-1] = S
NEG3      LDA     #$01        ;
          CMPA    $0002       ;
          BEQ     FIN         ;  Si M[n-1] = S on passe en FIN
          BRA     COMPLEM     ;  Cas M[n-1] <> S, on passe en COMPLEM
          SWI                 ;
                              ;
          ORG     $0001       ;
DIVDEND   FCB     $8F         ;
                              ;
          ORG     $0003       ;
DIVSEUR   FCB     $FD         ;
          END                 ;
```

6809 Prog 13 : Kit MC09-B Sté DATA RD : Interface Parallèle

Voir également 6809 Prog 14, 15, 16 ci-après.

retour au Sommaire | Index | Liens Rapides

Prog 13 MC09-B : Introduction

L'étude du fonctionnement de l'interface parallèle peut être réalisée à l'aide d'un kit du type DATA RD ou MC09-B sur lesquels est implantés un microprocesseur µp6809 ainsi qu'un PIA. L'adresse de base de ce dernier est $7090 pour le 1" kit et $8000 pour le second.

Les divers registres du PIA ont donc pour adresse :

```
ORA/DDRA   $7090  (ou $8000)
CRA        $7091  (ou $8001)
ORB/DDRB   $7092  (ou $8002)
CRB        $7093  (ou $8003)
```

Les lignes d'interruption IRQA et IRQB des ports A et B du PIA sont reliées à la broche IRQ du microprocesseur. Le vecteur d'adresse de l'interruption IRQ est $3F40 pour le kit DATA RD et $3FF8 pour le MC09-B.
Un câble limande permet la liaison entre le PIA du kit et la platine d'étude schématisée ci-dessous :

```
CB2 CB1 CA2 CA1           Interrupteurs
 X   X   X   X             |       |       |       Masse Sortie     5V
 O   O   O   O            \       \...\               X            X
                           |       |       |
                          X       X       X

B7....B0        A7....A0
XXXXXXXX        XXXXXXXX
OOOOOOOO        OOOOOOOO
 Port B          Port A

                CB2          CB1
                 X            X             [Câble Limande]
                                             Vers le µ-P
```

NB: X > fiches femelles et O > leds

Les leds permettent de visualiser l'état des diverses lignes. Les interrupteurs, munis d'anti-rebond, sont reliés à des fiches femelles et permettent d'imposer un niveau logique 0ou 1 sur ces fiches. La masse du microprocesseur est ramenée sur la fiche « Masse » de la platine d'étude !

6809 Prog 14 : Kit MC09-B Sté DATA RD : Etude des Ports Entrée / Sortie

retour au Sommaire | Index | Liens Rapides

Prog 14 MC09-B : Sujet A

Port en entrée
Afficher, par le biais des interrupteurs, le mot $C4 sur les entrées A; du port A. Ecrire en langage assembleur un programme permettant de stocker ce mot à l'adresse $0100. Exécuter ce programme et vérifier son bon fonctionnement.

Prog 14 MC09-B : Commentaires A

$C4 = %11000100
On initialise le port A en entrée, puis on vient le lire et on mémorise la donnée lue en $0100

Prog 14 MC09-B : Programme A

```
DDRA    EQU    $8000    ;  |
ORA     EQU    $8000    ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU    $8001    ;  |
                        ;
        ORG    $0000    ;
        CLRA            ;  Effacement de A
        STA    CRA      ;  Stock A dans CRA pour demande d'accès à DDRA
        STA    DDRA     ;  Déclaration du port A en entrée
        LDA    #$04     ;   |
        STA    CRA      ;   | Demande d'accès à ORA
        LDA    ORA      ;  Chargement du registre ORA
        STA    $0100    ;  Stockage de la valeur à l'adresse $0100
        SWI             ;  Fin du programme.
```

retour au Sommaire | Index | Liens Rapides

Prog 14 MC09-B : Sujet B

Port en sortie On utilise le port B en sortie pour commander, par microprocesseur, un moteur pas à pas.
Ce moteur possède 4 entrées notées **I1** à **I4** l'activation de ces entrées suivant la séquence décrite ci-dessous fait tourner le moteur par pas de 7,5° (le fait d'inverser la séquence de l'étape 4 à l'étape 1 fait tourner le moteur en sens inverse).

PAS	I1	I2	I3	I4
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1
5 Répétition du pas I1				

Prog 14 MC09-B : Question B :

Ecrire un programme en langage assembleur permettant d'assurer une rotation de 360° par pas de 3,75° dans un sens puis dans l'autre (utiliser entre chaque pas une temporisation, correspondant au décomptage de $FFFF, dont

vous préciserez le rôle).

Pour vérifier le bon fonctionnement du programme, on prendra soin de connecter le moteur, par l'intermédiaire de ses entrées I1, à I4 au port B du PIA suivant le brochage : B0−I1, B1−I2, B2−I3 et B3−I4.

Remarque : la broche « 0 V » du moteur doit être reliée à la masse du microprocesseur.

Proposer une méthode permettant de déterminer la vitesse de rotation du moteur (on rappelle que le fonctionnement du µp6809 est rythmé par une horloge de Fréquence égale à 1 MHz).

retour au Sommaire | Index | Liens Rapides

Prog 14 MC09-B : Commentaires B

Il faut créer un table de quatre donnés, correspondant en ordre et en valeur à l'enchaînement de la séquence moteur, cette séquence comporte quatre phases qui permettent chacune un déplacement de 3,75° du moteur, pour réaliser un tour complet soit 360° il faut faire 24 fois la boucle de séquence des phases.

Soit 24 x 4 x 3,75 = 360°

Entre chaque phase est imposé une temporisation de $FFFF permettant au moteur d'effectuer l'enchaînement de ses transitions. Le temps peut être réduit et est fonction de l'accélération, et donc de ce fait du couple moteur.

Pour calculer la vitesse moteur on commence par négliger les cycles machines or temporisation auquel cas on pourrait dire que le moteur met 24 x 4 temporisations pour faire un tour.

Calcul de la vitesse à 1MHz

VT environ égal à (3 NC + 4 NC+ 3 NC) x $FFFF x 24 x 4/360 = 0.017 tr/s soit environ 1 tr/min.

Prog 14 MC09-B : Programme B

Sens Antihoraire

```
DDRB    EQU    $8002     ;  |
ORB     EQU    $8002     ;  | Définition des adresses de port DDRB, ORB, CRB
CRB     EQU    $8003     ;  |
                         ;
        ORG    $0000     ;  Début du programme à l'adresse $0000
;------ INITIALISATION DU PIA
        CLRA             ;  Effacement du registre A
        STA    CRB       ;  Stock A dans CRB pour demande d'accès à DDRB
        LDA    #$FF      ;   |
        STA    DDRB      ;   | Déclaration du port B en sortie
        LDA    #$04      ;   |
        STA    CRB       ;   | Demande d'accès à ORB
        CLRB             ;  RAZ du registre B qui va compter le NB de séquence
DEBUT   LDY    #DATA     ;  Chargement de l'adresse de début des données
L1      LDA    ,Y+       ;  Chargement de la donnée.
        STA    ORB       ;  Envoi du mot de commande
        JSR    TEMPO     ;  Appel du sous-programme TEMPO
        CMPY   #DATA+4   ;  Compare Y à l'adresse de Fin des données
        BNE    L1        ;  Si pas égal on boucle sur L1
        INCB             ;  Sinon, Incrémente B
        CMPB   #25       ;  Compare B à la valeur 25
        BNE    DEBUT     ;  Si pas égale on boucle sur DEBUT
        SWI              ;  Sinon, Fin du programme.
;------ DONNEES POUR UNE ROTATION AU PAS DE 3.75°
        ORG    $0200     ;
DATA    FCB    $03,$06,$0C,$09
;------ SOUS-PROGRAMME TEMPO
        ORG    $0250     ;
TEMPO   LDX    #$FFFF    ;  Chargement de X par $FFFF
T2      LEA    X,-X      ;  X=X-1
        BNE    T2        ;  Si X<>0 -> boucle sur T2
        RTS              ;  Sinon -> Retour au programme appelant.
```

Prog 14 MC09-B : Programme B

Sens horaire

Pour la rotation en sens inverse on changera seulement la séquence des données pour la rotation

```
DATA    FCB     $09, $0C, $06, $03
```

*

6809 Prog 15 : Kit MC09-B Sté DATA RD : Etude des Interruptions

Prog 15 MC09-B : Sujet

Programme chenillard, Un « chenillard » consiste à allumer une seule lampe à la fois parmi les huit et à la faire se déplacer dans l'ordre A0, A1, A2,....., A7, A0,..... À une vitesse donnée.

Prog 15 MC09-B : Question A

Proposer un programme en langage assembleur permettant de réaliser un tel chenillard sur le port A (on conservera la même temporisation que dans le problème précédent).

Prog 15 MC09-B : Commentaires A

On commence par établir une table de données correspondant en nombre et en valeurs à l'enchaînement du chenillard. Il suffit ensuite d'envoyer les données les unes après les autres sur le port A en intercalant la temporisation entre chaque séquences, lorsque l'on arrive à la fin de la table, on reboucle alors sur son début et ainsi de suite.

Prog 15 MC09-B : Programme A

```
DDRA    EQU     $8000       ;  |
ORA     EQU     $8000       ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU     $8001       ;  |
                            ;
        ORG     $0000       ;  Début du programme
;------ INITIALISATION DU PIA
        CLRA                ;  Effacement de A
        STA     CRA         ;  Stock A dans CRA pour accès à DDRA
        LDA     #$FF        ;   |
        STA     DDRA        ;   | Place le port A en sortie.
        LDA     #$04        ;   |
        STA     CRA         ;   | Demande d'accès à ORA
;------ PROGRAMME PRINCIPAL
DEBUT   LDY     #DATA       ;  Chargement de l'adresse de début des données
L1      LDA     ,Y+         ;  Chargement des données
        STA     ORA         ;  Stockage de A sur les sorties ORA
        JSR     TEMPO       ;  Temporisation
        CMPY    #DATA+8     ;  Compare Y a l'adresse de fin des données
        BNE     L1          ;  Si pas égale -> boucle sur L1
        BRA     DEBUT       ;  Sinon -> boucle sur DEBUT
;------ DONNEES pour un défilement de b0 à b7.
        ORG     $0200       ;
DATA    FCB     $01,$02,$04,$08,$10,$20,$40,$80
;------ Sous Programme TEMPO
        ORG     $0250       ;
TEMPO   LDX     #$FFFF      ;  Chargement de X par $FFFF
T2      LEA     X,-X        ;  X=X-1
        BNE     T2          ;  Si X<>0 -> boucle sur T2
        RTS                 ;  Sinon -> Retour au programme appelant.
```


Prog 15 MC09-B : Question B

Interruption par test d'état

Modifier le programme précédent de façon à ce qu'il soit susceptible d'être interrompu par test d'état après avoir allumé AN et qu'il fonctionne selon le principe suivant

- défilement normal du chenillard A0, A1,....., A7
- test d'état du registre CRA ;
- s'il n'y a pas eu de demande d'interruption, poursuite du défilement normal du chenillard avec nouveau test d'état après l'allumage de A7 ;
- s'il y a eu demande d'interruption, extinction pendant 5 s environ de toutes les lampes du port A (sous-programme d'interruption) puis reprise du défilement du chenillard.

La demande d'interruption sera réalisée à l'aide d'un front descendant envoyée sur CA1, par exemple. Pour tester le programme, on générera manuellement le front descendant à l'aide de l'un des interrupteurs préalablement relié àCA1.

Proposer un programme permettant d'effectuer le comptage du nombre de données paires et impaires d'une table.

Prog 15 MC09-B : Commentaires B

On exécute le chenillard pour toute la table, à la fin de celle ci on vient scruter le bit 7 de CRA correspondant à une interruption d'état du PIA si ce bit est à 1, une interruption a eu lieu, on éteint donc alors toute les leds pendant une valeur de huit tempo soit 5s environs puis on recommence.

Si aucune interruption n'est demandée, le chenillard recommence en début de table.

retour au Sommaire | Index | Liens Rapides

Prog 15 MC09-B : Programme B

```
DDRA    EQU     $8000       ;  |
ORA     EQU     $8000       ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU     $8001       ;  |
                            ;
        ORG     $0000       ;  Début du programme à l'adresse $0000
;------ INITIALISATION DU PIA
        CLRA                ;  Effacement de A
        STA     CRA         ;  Stock A dans CRA pour accès à DDRA
        LDA     #$FF        ;   |
        STA     DDRA        ;   | Place le port A en sortie.
        LDA     #$06        ;   | Demande d'accès à ORA et validation des
        STA     CRA         ;   | interruptions.
;------ PROGRAMME PRINCIPAL
DEBUT   LDY     #DATA       ;  Chargement de l'adresse de début des données
L1      LDA     ,Y+         ;  Chargement des données
        STA     ORA         ;  Stockage de A sur les sorties ORA
        JSR     TEMPO       ;  Temporisation
        CMPY    #DATA+8     ;  Compare Y a l'adresse de fin des données
        BNE     L1          ;  Si pas égale -> boucle sur L1
        LDA     CRA         ;  Chargement du registre d'état CRA
        ANDA    #$80        ;  Masque pour récupérer le bit b7 de CRA
        CMPA    #$00        ;  Compare à 0
        BEQ     DEBUT       ;  Si pas d'interruption, boucle sur DEBUT, sinon
;------ SOUS PROGRAMME D'INTERRUPTION.
        CLRA                ;   |
        STA     ORA         ;   | Extinction des sorties
        LDA     #$08        ;
L2      JSR     TEMPO       ;  Appel du sous programme TEMPO
        DECA                ;  Décrémente A de 1
        BNE     L2          ;  Si A n'est pas nul on boucle sur L2
        LDA     ORA         ;  Sinon, lecture de ORA pour RAZ du bit b7 de CRA
        BRA     DEBUT       ;  Sinon -> boucle sur DEBUT
;------ DONNEES
        ORG     $0200       ;
DATA    FCB     $01,$02,$04,$08,$10,$20,$40,$80
;------ Sous Programme TEMPO
        ORG     $0250       ;
TEMPO   LDX     #$FFFF      ;  Chargement de X par $FFFF
T2      LEA     X,-X        ;  X=X-1
        BNE     T2          ;  Si X<>0 -> boucle sur T2
        RTS                 ;  Sinon -> Retour au programme appelant.
```

retour au Sommaire | Index | Liens Rapides

Prog 15 MC09-B : Question C

Interruption vectorisée

Modifier le programme du chenillard de sorte qu'il soit susceptible d'être interrompu par une interruption vectorisée et de réaliser les séquences suivantes

- défilement normal du chenillard
- demande d'interruption vectorisée déclenchée par front montant sur CA2 réalisé manuellement comme précédemment
- exécution, le cas échéant, du programme d'interruption qui consiste à allumer toutes les lampes du port Adurant 5s environ, puis retour au programme principal.

Prog 15 MC09-B : Tester le programme et conclure C

Proposer un programme global, c'est à dire incluant les deux types d'interruption. Exécuter ce programme et lancer une interruption par test d'état ; pendant le déroulement de cette interruption, lorsque toutes les lampes sont éteintes, lancer une interruption vectorisée. Que se passe-t-il ? Recommencer la même expérience en lançant d'abord l'interruption vectorisée. Conclure.

Prog 15 MC09-B : Commentaires C

On exécute le chenillard normalement. L'interruption vectorisée sur front montant est déclarée sur CA2 par conséquent si pendant l'exécution du chenillard une interruption apparaît sur CA2 le programme se positionne automatiquement à l'adresse d'interruption du kit soit $3FF8 à laquelle est stocké l'adresse du programme d'interruption, on exécute alors ce programme qui consiste à allumer toutes les leds durant 5s, puis le chenillard reprends la ou il s'est arrêté.

Prog 15 MC09-B : Programme C

```
DDRA    EQU     $8000       ;  |
ORA     EQU     $8000       ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU     $8001       ;  |
                            ;
        ORG     $0000       ;  Début du programme à l'adresse $0000
;------ INITIALISATION DU PIA
        CLRA                ;  Effacement de A
        STA     CRA         ;  Stock A dans CRA pour accès à DDRA
        LDA     #$FF        ;   |
        STA     DDRA        ;   | Place le port A en sortie.
        LDA     #$1C        ;   | Demande d'accès à ORA et validation des
        STA     CRA         ;   | interruptions sur front montant.
        LDY     #INTVECT    ;   | Chargement de l'adresse du sous programme
        STY     $3FF8       ;   | d'interruption.
        ANDCC   #$EF        ;  Forçage du bit b4 du CCR à 0
;------ PROGRAMME PRINCIPAL
DEBUT   LDY     #DATA       ;  Chargement de l'adresse de début des données
L1      LDA     ,Y+         ;  Chargement des données
        STA     ORA         ;  Stockage de A sur les sorties ORA
        JSR     TEMPO       ;  Temporisation
        CMPY    #DATA+8     ;  Compare Y a l'adresse de fin des données
        BNE     L1          ;  Si pas égale -> boucle sur L1
        BRA     DEBUT       ;  Retour à DEBUT
;------ SOUS PROGRAMME D'INTERRUPTION.
        ORG     $0150       ;
INTVECT LDA     #$FF        ;   |
        STA     ORA         ;   | Allumage des sorties
        LDA     #$08        ;  Temporisation de 5S
L2      JSR     TEMPO       ;  Appel du sous programme TEMPO
        DECA                ;  Décrémente A de 1
        BNE     L2          ;  Si A n'est pas nul on boucle sur L2
        LDA     ORA         ;  Sinon, lecture de ORA pour RAZ du bit b7 de CRA
        RTI                 ;  Retour au programme principal
;------ DONNEES
        ORG     $0200       ;
DATA    FCB     $01,$02,$04,$08,$10,$20,$40,$80
;------ Sous Programme TEMPO
        ORG     $0250       ;
TEMPO   LDX     #$FFFF      ;  Chargement de X par $FFFF
T2      LEA     X,-X        ;  X=X-1
        BNE     T2          ;  Si X<>0 -> boucle sur T2
        RTS                 ;  Sinon -> Retour au programme appelant.
```


Prog 15 MC09-B : Programme Global C

Interruption vectorisée + Interruption d'état

```
DDRA    EQU     $8000       ;  |
ORA     EQU     $8000       ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU     $8001       ;  |
                            ;
        ORG     $0000       ;  Début du programme à l'adresse $0000
;------ INITIALISATION DU PIA
        CLRA                ;  Effacement de A
        STA     CRA         ;  Stock A dans CRA pour accès à DDRA
        LDA     #$FF        ;   |
        STA     DDRA        ;   | Place le port A en sortie.
        LDA     #$1E        ;   | Demande d'accès à ORA et validation des
        STA     CRA         ;   | interruptions.
        LDY     #INTVECT    ;   | Chargement de l'adresse du sous programme
        STY     $3FF8       ;   | d'interruption.
        ANDCC   #$EF        ;  Forçage du bit b4 du CCR à 0
;------ PROGRAMME PRINCIPAL
DEBUT   LDY     #DATA       ;  Chargement de l'adresse de début des données
L1      LDA     ,Y+         ;  Chargement des données
        STA     ORA         ;  Stockage de A sur les sorties ORA
        JSR     TEMPO       ;  Temporisation
        CMPY    #DATA+8     ;  Compare Y a l'adresse de fin des données
        BNE     L1          ;  Si pas égale -> boucle sur L1
        LDA     CRA         ;  Chargement du registre d'état CRA
        ANDA    #$80        ;  Masque pour récupérer le bit b7 de CRA
        CMPA    #$00        ;  Compare à 0
        BEQ     DEBUT       ;  Si pas d'interruption, boucle sur DEBUT, sinon
;------ SOUS PROGRAMME D'INTERRUPTION par TEST D'ETAT
```

```
        CLRA                ;   |
        STA     ORA         ;   | Extinction des sorties
        LDA     #$08        ;  Temporisation de 5s
L2      JSR     TEMPO       ;  Appel du sous programme TEMPO
        DECA                ;  Décrémente A de 1
        BNE     L2          ;  Si A n'est pas nul on boucle sur L2
        LDA     ORA         ;  Sinon, lecture de ORA pour RAZ du bit b7 de CRA
        BRA     DEBUT       ;  Sinon -> boucle sur DEBUT
;------ SOUS PROGRAMME D'INTERRUPTION VECTORISEE
        ORG     $0150       ;
INTVECT LDA     #$FF        ;   |
        STA     ORA         ;   | Allumage des sorties
        LDB     #$08        ;  Temporisation de 5S
L3      JSR     TEMPO       ;  Appel du sous programme TEMPO
        DECB                ;  Décrémente B de 1
        BNE     L3          ;  Si B n'est pas nul on boucle sur L3
        LDA     ORA         ;  Sinon, lecture de ORA pour RAZ du bit b7 de CRA
        RTI                 ;  Sinon -> Retour au programme principal.
;------ DONNEES
        ORG     $0200       ;
DATA    FCB     $01,$02,$04,$08,$10,$20,$40,$80
;------ Sous Programme TEMPO
        ORG     $0250       ;
TEMPO   LDX     #$FFFF      ;  Chargement de X par $FFFF
T2      LEA     X,-X        ;  X=X-1
        BNE     T2          ;  Si X<>0 -> boucle sur T2
        RTS                 ;  Sinon -> Retour au programme appelant.
```

Prog 15 MC09-B : Commentaires C

Par test il s'avère que l'interruption vectorisée est prioritaire devant l'interruption d'état c'est-à-dire que si une interruption d'état est demandé et si en même on à une requête d'interruption vectorisée, alors s'est l'interruption vectorisée qui est exécutée.

6809 Prog 16 : Kit MC09-B Sté DATA RD : Etude des Lignes de Dialogues

retour au Sommaire | Index | Liens Rapides

Prog 16 MC09-B : Question A

Mode programmé (Set-Reset)

On souhaite montrer l'action du bit b3 de CRA sur CA2, lorsque l'on travaille en mode programmé.

Comme application, proposer un programme permettant de faire clignoter la led associée à CA2 avec une période correspondant au décomptage de $FFFF.

Prog 16 MC09-B : Commentaire A

Il faut impérativement positionner le bit 4 de CRA à 1 pour utiliser le mode programmé, puis faire varier le bit 3 de 0à 1 entre chaque temporisation pour faire clignoter la led.

Prog 16 MC09-B : Programme A

```
DDRA    EQU     $8000       ;  |
ORA     EQU     $8000       ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU     $8001       ;  |
                            ;
        ORG     $0000       ;  Début du programme
;------ PROGRAMME PRINCIPAL
DEBUT   LDA     #$38        ;   |
        STA     CRA         ;   | Allumage de la LED
        JSR     TEMPO       ;  Temporisation
        LDA     #$30        ;   |
        STA     CRA         ;   | Extinction de la LED
        JSR     TEMPO       ;  Temporisation
        BRA     DEBUT       ;  Retour au DEBUT du programme.
;------ Sous Programme TEMPO
        ORG     $0250       ;
TEMPO   LDX     #$FFFF      ;  Chargement de X par $FFFF
T2      LEA     X,-X        ;  X=X-1
        BNE     T2          ;  Si X<>0 -> boucle sur T2
```

```
        RTS                 ;  Sinon -> Retour au programme appelant.
```

retour au Sommaire | Index | Liens Rapides

Prog 16 MC09-B : Question B

Mode impulsion (Pulse Strobe)

Ce mode de fonctionnement va être étudié pour effectuer une conversion analogique numérique commandée par le microprocesseur.

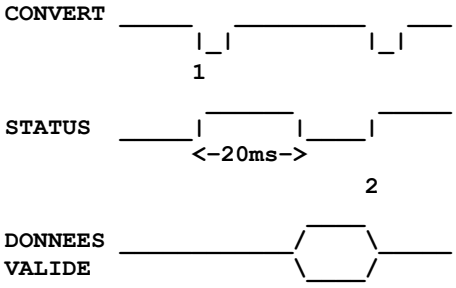
On utilisera pour cela un convertisseur (platine AD CONVERTER) dont les principales caractéristiques sont : tension analogique à convertir comprise entre 0 et 5 V, résolution égale à 20 mV et temps de conversion maximum égal à 30 ms.

L'entrée du signal analogique s'effectue sur la broche ANALOGUE I/P, la sortie numérique sur les broches D0, D1,....., D7.

L'entrée SELECT doit être positionnée à 1 afin de sélectionner le mode de conversion analogique −numérique.

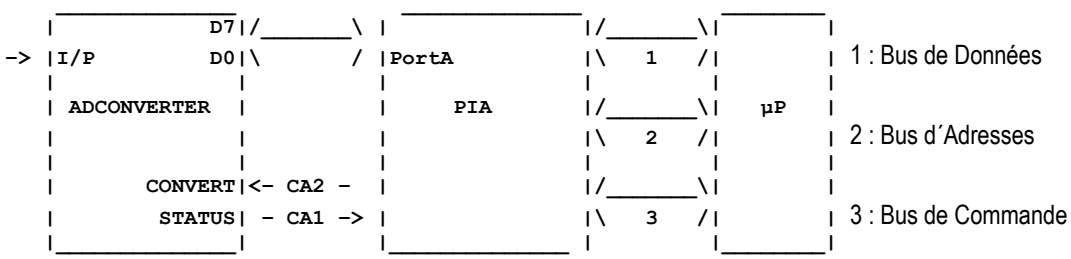
La masse de la platine de conversion doit être reliée à celle du microprocesseur via la borne MASSE de la platine PIA.

La séquence de conversion est la suivante :



retour au Sommaire | Index | Liens Rapides

On propose de travailler avec le port A du PIA en utilisant la ligne CA1, pour le signal STATUS et la ligne CA2 pour le signal CONVERT suivant le schéma



Ecrire en langage assembleur un programme permettant d'effectuer l'acquisition numérique d'une donnée analogique en mode impulsion.

Effectuer ces conversions et ces acquisitions pour des tensions analogiques comprises entre 0 et 5 V par pas de 0,5 V. Analyser les résultats obtenus et conclure.

retour au Sommaire | Index | Liens Rapides

Prog 16 MC09-B : Commentaire B

On provoque une impulsion sur CA2, en utilisant, le mode impulsion, cette impulsion permettra d'activer le CAN, il faudra ensuite, tester le bit 7 d'interruption qui sera positionné à 1 par l'intermédiaire de CA1 lorsque la conversion sera terminée. Il ne reste plus qu'à lire la donnée sur le port A et enfin la mémoriser en $0100.

Prog 16 MC09-B : Programme B

```
DDRA    EQU    $8000        ;  |
ORA     EQU    $8000        ;  | Définition des adresses de port DDRA, ORA, CRA
CRA     EQU    $8001        ;  |
                            ;
        ORG    $0000        ;  Début du programme
;------ PROGRAMME PRINCIPAL
        CLRA                ;    |
        STA    CRA          ;    | Demande d'accès à DDRA
        STA    DDRA         ;    | Configuration du port A en entrée
CONVERT LDA    #$3C         ;    |
        STA    CRA          ;    | Chargement du mot de commande dans CRA.
ETAT    LDA    CRA          ;  Chargement du registre d'état CRA
        ANDA   #$80         ;  Masquage du bit b7
        BEQ    ETAT         ;  Si pas d'interruption -> boucle sur ETAT
```

```
LDA     ORA         ;  sinon -> Conversion terminée => Lecture conversion
STA     >$0100      ;  Stockage de la conversion à l'adresse $0100
SWI                 ;
```

6809 Prog 17 : Ouvrage 06 : Mouvements de données 8 et 16 bits par LOAD et STORE

retour au Sommaire | Index | Liens Rapides

L'exercice suivant permet de voir les modes d'adressages associés aux instructions LD... et ST....
(Description incomplète → voir page IV.06 de l'ouvrage n°06)

```
                    OPT     LLE=80,G,ABS    ligne option doit être placéeen entête de programme, avant la directive ORG
                  ; ---Mouvement de données par Load et Store
8040                ORG     $8040           positionne l'adresse de chargement des directives FCB et FDB
8040      38        FCB     56,78           Charge 2 positions mémoires ($8040) = $38  et  ($8041) = $4E
8041      4E                                .
8042      162E      FDB     5678            attire l'attention sur l'emploi de base décimal et hexa, l'équivalent de FCB 56,78 n'est pas FDB 5678

                  ; ---section programme
8000                ORG     $8000           si ce ORG est absent le programme débutera à $8044 au lieu de $8000
8000 86   0C        LDA     #12
8002 C6   E0        LDB     #224
8004 8E   8060      LDX     #$8060          adresce début zone stockage
8007 ED   81        STD     ,X++            mise en mémoire données fixes
8009 FC   8040      LDD     $8040           transfert donnée tableau
800C ED   81        STD     ,X++
800E AF   84        STX     ,X              stockage index final
8010 3F                     SWI
```

6809 Prog 18 : Programme capable de décrémenter le nombre $0E cinq fois et de stocker le résultat dans la case mémoire $0800

retour au Sommaire | Index | Liens Rapides

```
            ORG     $0000
            LDA     #$0E        pour décrémenter $0E on utilisera l'accu A
            CLRB                quant à l'accu B il servira de compteur
DEBUT       DECA
            INCB
            CMPB    #$05
            BCC     FIN
            BRA     DEBUT
FIN         STA     $0800
            END
```

Programme identique à celui du dessus mais en utilisant un autre algorithme.

```
            ORG     $0000
            LDA     #$0E        pour décrémenter $0E on utilisera l'accu A
            CLRB                quant à l'accu B il servira de compteur
DEBUT       DECA
            INCB                ; B=1,2,3,4 et 5
            CMPB    #$05
            BCS     DEBUT
            STA     $0800
            END
```

6809 Prog 19 : Programme capable de calculer la somme des 10 premiers entiers, le résultat doit être stocké à l'adresse $4000

retour au Sommaire | Index | Liens Rapides

```
            ORG     $0000
            LDB     #$0A        l'accu B il servira de compteur
            CLRA
            CLR     $0200
DEBUT       INC     $0200
            ADDA    $0200
            DECB
            BEQ     FIN
```

```
        BRA     DEBUT
FIN     STA     $4000
        END
```

6809 Prog 20 : Programme capable de stocker les 100 premiers nombres entiers dans le bloc mémoire dont la première adresse est $1200

retour au Sommaire | Index | Liens Rapides

```
        ORG     $0000
        LDX     #$1200      Le nombre à stocker est dans l'accu A, il sert aussi
                          de compteur
        CLRA
DEBUT   STA     ,X          ;
        INCA                ;
        CMPA    #100        ; 100 en décimal
        BNE     DEBUT       ;
        END
```

ANN : Circuits d'Interfaces de la famille 6800 et 6809

La plupart des circuits d'interface du 6800 sont compatibles avec le 6809

Origine	Référence		Désignations
6800	6810	RAM	128 Ko 8 bits
"	6830	ROM	1024 Ko 8 bits
"	6821	PIA	Interface parallèle programmable
"	6828	PIC	Contrôleur de priorité d'interruption
"	6840	PTM	3 temporisateurs programmable
"	6843	FDC	Contrôleur de disque souple simple densité
"	6844	DMAC	Contrôleur d'accès mémoire
"	6845	CRTC	Contrôleur de visualisation
"	6846	ROM	Mémoire ROM 2 Ko, port parallèle 8 bits
"		COMBO	Temporisateur 16 bits
"	6850	ACIA	Interface série asynchrone
"	6852	SSDA	Interface série synchrone
"	6854	ADLC	Contrôleur de transmission avec protocole
"	6855	DMA	Contrôleur d'accès mémoire (pas introduit sur le marché)
"	68488	GPIA	Interface IEEE-488
"	9364		Contrôleur d'écran graphique
"	9365		Contrôleur d'écran graphique
"	9366		Contrôleur d'écran graphique
6809	6829	MMU	Interface d'extension mémoire
"	6839	ROM	Mémoire ROM mathématique

ANN : Table ASCII Description étendue de l'usage des caractères de contrôle (caractères 0 à 31)

$00	000	**NUL** (NULL) : caractère nul Typiquement (et spécialement en PureBasic) utilisé pour indiquer la fin d'une chaîne. Originellement une NOP, un caractère à ignorer. Lui donner le code 0 permettait de prévoir des réserves sur les bandes perforées en laissant des zones sans perforation pour insérer de nouveaux caractères a posteriori. Avec le développement du langage C il a pris une importance particulière quand il a été utilisé comme indicateur de fin de chaîne de caractères.
$01	001	**SOH** (Start Of Heading) : début de titre ou début d'en-tête Indique le début d'un bloc de données, ou la zone d'en-tête d'un bloc de données. Il est aujourd'hui souvent utilisé dans les communications séries pour permettre la synchronisation après erreur14.
$02	002	**STX** (Start of TeXt) : début de texte Typiquement envoyé comme premier caractère dans un bloc de texte, pendant les communications.
$03	003	**ETX** (End of TeXt) : fin de texte Typiquement envoyé comme dernier caractère dans un bloc de texte, pendant les communications.
$04	004	**EOT** (End Of Transmission) : fin de transmission Utilisé pour indiquer la fin d'une transmission.
$05	005	**ENQ** (ENQuiry) : requête - invitation à la transmission Envoyé à un récepteur afin d'obtenir une réponse.
$06	006	**ACK** (ACKnowledge) : accusé de réception Envoyé par un récepteur pour indiquer qu'il a reçu et/ou compris la requête.
$07	007	**BEL** (BELL) : cloche Produit un signal sonore (provoque l'émission d'un 'bip' par le haut-parleur du PC)
$08	008	**BS** (BackSpace) : retour arrière Déplace le curseur d'une position vers la gauche (pourrait également effacer le caractère à gauche du curseur avant d'effectuer le mouvement)
$09	009	**HT** (Horizontal Tab) : tabulation horizontale Typiquement utilisé pour la mise en forme de tableaux dans un texte.
$0A	010	**LF** (LineFeed) : saut de ligne Le caractère utilisé pour représenter l'action de passer une ligne sur une machine à écrire ou une imprimante en mode texte. Typiquement utilisé comme, ou partie des, caractères de fin de ligne.
$0B	011	**VT** (Vertical Tab) : tabulation verticale Même chose que la tabulation (horizontale), mais le déplacement s'effectue d'une rangée vers le bas au lieu d'une colonne vers la droite.
$0C	012	**FF** (Form Feed) : saut de page Caractère typiquement utilisé pour indiquer à une imprimante (en mode texte) de passer à la page (feuille) suivante.
$0D	013	**CR** (Carriage Return) : retour chariot Le caractère qui représente l'action de ramener la tête d'une machine à écrire ou d'une imprimante au début de la ligne. Typiquement utilisé comme, ou partie des, caractères de fin de ligne.
$0E	014	**SO** (Shift Out) : mouvement sortant Début d'un bloc de caractères dont la signification dépend de l'implémentation.
$0F	015	**SI** (Shift In) : mouvement entrant Ferme la transmission du type de bloc ci-dessus.

$10	016	**DLE** (Data Link Escape) : échappement de lien de donnée Utilisé pour indiquer que le caractère de contrôle suivant devrait être interprété comme donnée et non comme caractère de contrôle.
$11	017	**DC1** (Device Control 1) : contrôle de périphérique 1 Typiquement utilisé pour activer une partie d'un équipement. L'usage le plus courant aujourd'hui est en tant que caractère XON dans les communications série à contrôle de flux logiciel.
$12	018	**DC2** (Device Control 2) : contrôle de périphérique 2 Un autre caractère de contrôle de périphérique. Son usage dépend du contexte.
$13	019	**DC3** (Device Control 3) : contrôle de périphérique 3 Typiquement utilisé pour désactiver une partie d'un équipement. L'usage le plus courant aujourd'hui est en tant que caractère XOFF dans les communications série à contrôle de flux logiciel.
$14	020	**DC4** (Device Control 4) : contrôle de périphérique 4 Un autre caractère de contrôle de périphérique.
$15	021	**NAK** (Negative AcKnowledge) : accusé de réception négatif Typiquement utilisé pour signaler des données non-reçues ou non-comprises (erronée).
$16	022	**SYN** (SYNchronous idle) : attente synchronisée Comme son nom l'indique, il s'agit d'un signal envoyé à intervalle régulier pour indiquer que le canal de communication est en attente, mais toujours actif.
$17	023	**ETB** (End of Transmission Block) : fin de transmission de bloc Utilisé pour contrôler la transmission de donnée en indiquant la fin de bloc. A ne pas confondre avec EOT.
$18	024	**CAN** (CANcel) : annulation Signifie généralement que la donnée envoyée précédemment devrait être ignorée, bien que les détails dépendent de l'application.
$19	025	**EM** (End of Medium) : fin de média Utilisé pour indiquer la fin d'un média, par exemple la fin d'un lecteur de bande
$1A	026	**SUB** (SUBstitute) : substitution Un caractère utilisé pour indiquer qu'un caractère a été substitué.
$1B	027	**ESC** (ESCape) : échappement Le caractère produit habituellement en appuyant sur la touche 'ECHAP' de votre clavier, utilisé dans les "séquences d'échappement" pour fournir des informations de formattage aux afficheurs de texte (consoles, imprimantes, etc..)
$1C	028	**FS** (File Separator) : séparateur de fichier
$1D	029	**GS** (Group Separator) : séparateur de groupe
$1E	030	**RS** (Record separator) : séparateur d'enregistrement
$1F	031	**US** (Unit separator) : séparateur d'unité
$7F	127	**DEL** (Delete) : effacement. Lui donner le code 127 (1111111 en binaire) permettait de supprimer a posteriori un caractère sur les bandes perforées qui codaient les informations sur 7 bits. N'importe quel caractère pouvait être transformé en DEL en complétant la perforation des 7 bits qui le composaient.

Table ASCII (0 - 127)

000	$00	NUL
001	$01	SOH
002	$02	STX
003	$03	ETX
004	$04	EOT
005	$05	ENQ
006	$06	ACK
007	$07	BEL
008	$08	BS
009	$09	HT
010	$0A	LF
011	$0B	VT
012	$0C	FF
013	$0D	CR
014	$0E	SO
015	$0F	SI
016	$10	DLE
017	$11	DC1
018	$12	DC2
019	$13	DC3
020	$14	DC4
021	$15	NAK
022	$16	SYN
023	$17	ETB
024	$18	CAN
025	$19	EM
026	$1A	SUB
027	$1B	ESC
028	$1C	FS
029	$1D	GS
030	$1E	RS
031	$1F	US

032	$20	SP
033	$21	!
034	$22	"
035	$23	#
036	$24	$
037	$25	%
038	$26	&
039	$27	'
040	$28	(
041	$29	)
042	$2A	*
043	$2B	+
044	$2C	,
045	$2D	-
046	$2E	.
047	$2F	/
048	$30	0
049	$31	1
050	$32	2
051	$33	3
052	$34	4
053	$35	5
054	$36	6
055	$37	7
056	$38	8
057	$39	9
058	$3A	:
059	$3B	;
060	$3C	<
061	$3D	=
062	$3E	>
063	$3F	?

064	$40	@
065	$41	A
066	$42	B
067	$43	C
068	$44	D
069	$45	E
070	$46	F
071	$47	G
072	$48	H
073	$49	I
074	$4A	J
075	$4B	K
076	$4C	L
077	$4D	M
078	$4E	N
079	$4F	O
080	$50	P
081	$51	Q
082	$52	R
083	$53	S
084	$54	T
085	$55	U
086	$56	V
087	$57	W
088	$58	X
089	$59	Y
090	$5A	Z
091	$5B	[
092	$5C	\
093	$5D	]
094	$5E	^
095	$5F	_

096	$60	`
097	$61	a
098	$62	b
099	$63	c
100	$64	d
101	$65	e
102	$66	f
103	$67	g
104	$68	h
105	$69	i
106	$6A	j
107	$6B	k
108	$6C	l
109	$6D	m
110	$6E	n
111	$6F	o
112	$70	p
113	$71	q
114	$72	r
115	$73	s
116	$74	t
117	$75	u
118	$76	v
119	$77	w
120	$78	x
121	$79	y
122	$7A	z
123	$7B	{
124	$7C	|
125	$7D	}
126	$7E	~
127	$7F	DEL

Table ASCII (128 - 255)

128	$80	Ç
129	$81	ü
130	$82	é
131	$83	â
132	$84	ä
133	$85	à
134	$86	å
135	$87	ç
136	$88	ê
137	$89	ë
138	$8A	è
139	$8B	ï
140	$8C	î
141	$8D	ì
142	$8E	Ä
143	$8F	Å
144	$90	É
145	$91	æ
146	$92	Æ
147	$93	ô
148	$94	ö
149	$95	ò
150	$96	û
151	$97	ù
152	$98	ÿ
153	$99	Ö
154	$9A	Ü
155	$9B	¢
156	$9C	£
157	$9D	¥
158	$9E	Pts
159	$9F	ƒ

160	$A0	á
161	$A1	í
162	$A2	ó
163	$A3	ú
164	$A4	ñ
165	$A5	Ñ
166	$A6	ª
167	$A7	º
168	$A8	¿
169	$A9	⌐
170	$AA	¬
171	$AB	½
172	$AC	¼
173	$AD	¡
174	$AE	«
175	$AF	»
176	$B0	░
177	$B1	▒
178	$B2	▓
179	$B3	│
180	$B4	┤
181	$B5	╡
182	$B6	╢
183	$B7	╖
184	$B8	╕
185	$B9	╣
186	$BA	║
187	$BB	╗
188	$BC	╝
189	$BD	╜
190	$BE	╛
191	$BF	┐

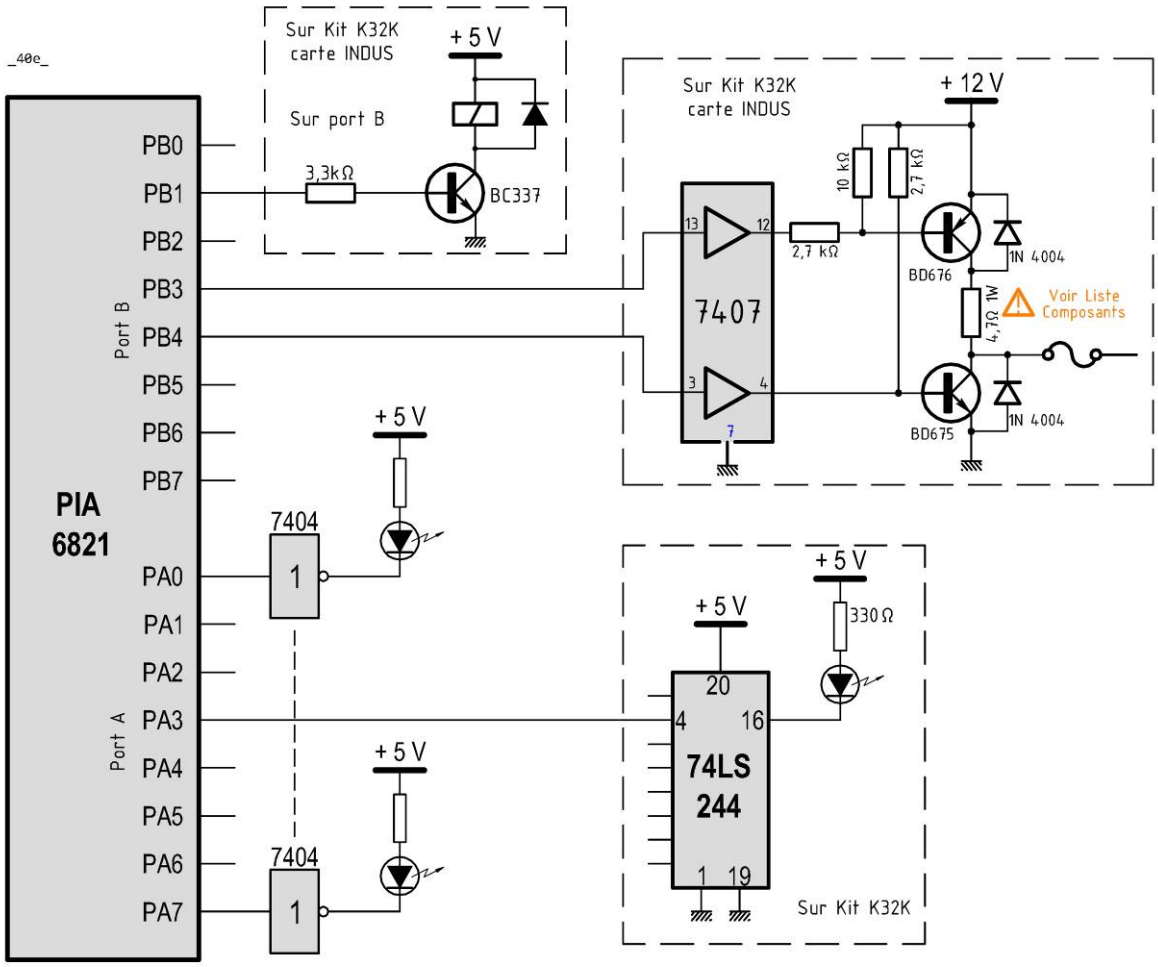
192	$C0	└
193	$C1	┴
194	$C2	┬
195	$C3	├
196	$C4	─
197	$C5	┼
198	$C6	╞
199	$C7	╟
200	$C8	╚
201	$C9	╔
202	$CA	╩
203	$CB	╦
204	$CC	╠
205	$CD	═
206	$CE	╬
207	$CF	╧
208	$D0	╨
209	$D1	╤
210	$D2	╥
211	$D3	╙
212	$D4	Ô
213	$D5	╒
214	$D6	╓
215	$D7	╫
216	$D8	╪
217	$D9	┘
218	$DA	┌
219	$DB	▉
220	$DC	▄
221	$DD	▌
222	$DE	▐
223	$DF	▀

224	$E0	α
225	$E1	ß
226	$E2	Γ
227	$E3	π
228	$E4	Σ
229	$E5	σ
230	$E6	µ
231	$E7	τ
232	$E8	Φ
233	$E9	Θ
234	$EA	Ω
235	$EB	δ
236	$EC	∞
237	$ED	φ
238	$EE	ε
239	$EF	∩
240	$F0	≡
241	$F1	±
242	$F2	≥
243	$F3	≤
244	$F4	⌠
245	$F5	⌡
246	$F6	÷
247	$F7	≈
248	$F8	≈
249	$F9	·
250	$FA	·
251	$FB	√
252	$FC	ⁿ
253	$FD	²
254	$FE	■
255	$FF	

PVM : Interfaçage des Afficheurs

PVM : AFF : Diode LED

retour au Sommaire | Index



La commande d'une diode LED se fait par un port en sortie d'un PIA.

Un inverseur du type 7404 est nécessaire pour absorber le courant de la LED, limité par une résistance.

La LED est allumée si Pax = 1

LR : LIENS RAPIDES

retour au Sommaire | Index | Sommaire Principal

retour au Sommaire | Index | Sommaire Principal

retour au Sommaire | Index | Sommaire Principal