

* E7C3 (PRC): Registre de données:

 bit0 (sortie): commutation mémoire écran caractères et
 mémoire écran couleur.
 bit1 (entrée): interrupteur crayon optique.
 bit2 (sortie): T07-70 : couleur du tour, pastel.
 bit3 (sortie): led clavier.
 bit4 (sortie): couleur du tour, rouge.
 bit5 (sortie): couleur du tour, vert.
 bit6 (sortie): couleur du tour, bleu.
 bit7 (entrée): lecture cassette.

* E7C5 (TCR): Registre contrôle timer

* E7C6-E7C7 (TMSB-TLSB): Valeur timer

*** PIA SYSTEME 6821 ***

* E7C8 (PRA): Registre de données, port A.

 bit0-7 (entrée): lecture matrice clavier

* E7C9 (PRB): Registre de données, port B:

 bit 0-7 (sortie): écriture matrice clavier
 T07-70: bit 0-2 (sortie): multiplexage clavier
 T07-70: bit 3-7 (sortie): sélection banques mémoire

* E7CA (CRA): Registre de contrôle, port A:
 CA1 (entrée): T07-70 : présence carte incrustation
 CA2 (sortie): moteur du L.E.P.

* E7CB (CRB): Registre de contrôle, port B:
 CB1 (entrée): T07-70 : interruption light-pen
 CB2 (sortie): T07-70 : commande d'incrustation.

15.4.2 PIA jeux

*** PIA JEUX 6821 ***

* E7CC (PRA1): Registre de données, port A:

 bits 0-7 (entrée): lecture des manettes de jeu.

* E7CD (PRB1): Registre de données, port B:

 bits 0-5 (entrée): convertisseur digital/analogique.
 bit6 (entrée): action manette de jeu 0.
 bit7 (entrée): action manette de jeu 1.

* E7CE (CRA1): Registre de contrôle, port A:
 CA1 (entrée): action manette de jeu 0.

* E7CF (CRB1): Registre de contrôle, port B:
 CB1 (entrée): action manette de jeu 1.

15.4.3 Interface de communication

*** INTERFACE DE COMMUNICATION 6821 ***

* E7E0 (PRA2): Registre de données, port A:

 bit0 (sortie): receive data
 bit1 (sortie): clear to send
 bit5 (entrée): request to send
 bit6 (entrée): data terminal ready
 bit7 (entrée): transmit data

* E7E1 (PRB2): Registre de données, port B:

 bits 0-7 (sortie): données en parallèle.

* E7E2 (CRA2): Registre de contrôle, port A:
 CA1 (entrée): request to send (demande d'émission).

* E7E3 (CRB2): Registre de contrôle, port B:
 CB1 (entrée): acknowledge.
 CB2 (sortie): strobe.

ANNEXE B

MICROPROCESSEUR 6809

INSTRUCTIONS ET ADRESSAGE

Extrait de *Microprocesseurs et périphériques*, édité par Thomson Semiconducteurs, 45, avenue de l'Europe, 78140 Velizy, téléphone (1) 946.97.19, télex 698 866 F .

Le circuit 6809 est un microprocesseur 8 bits de conception révolutionnaire, utilisant les techniques de programmation moderne telles que banalisation de l'implantation en mémoire, réentrance et programmation modulaire.

Cet apport de 3^e génération à la famille 6800 offre des améliorations d'architecture qui incluent des registres, des instructions et des modes d'adressage supplémentaires.

Les instructions de base de tout ordinateur sont particulièrement améliorées par la présence de modes d'adressage puissants. Le jeu de modes d'adressage disponible du 6809 est actuellement le plus complet des microprocesseurs existants. Les caractéristiques logicielles et matérielles du circuit, en font un processeur idéal pour l'exécution de programmes en langages évolués ou pour la réalisation d'applications standards.

- 10 modes d'adressage

Compatibilité ascendante des modes d'adressage avec la famille 6800.

Adressage direct dans tout l'espace mémoire.

Branchements relatifs longs.

Compteur programme relatif.

Indirection.

Adressage indexé étendu.

déplacements constants 0, 5, 8, 16 bits

déplacements accumulateur 8, 16 bits

auto-incrémentation/décrémentation

- Manipulation de pile améliorée

- 1464 instructions

- Multiplication non signée 8 × 8 bits

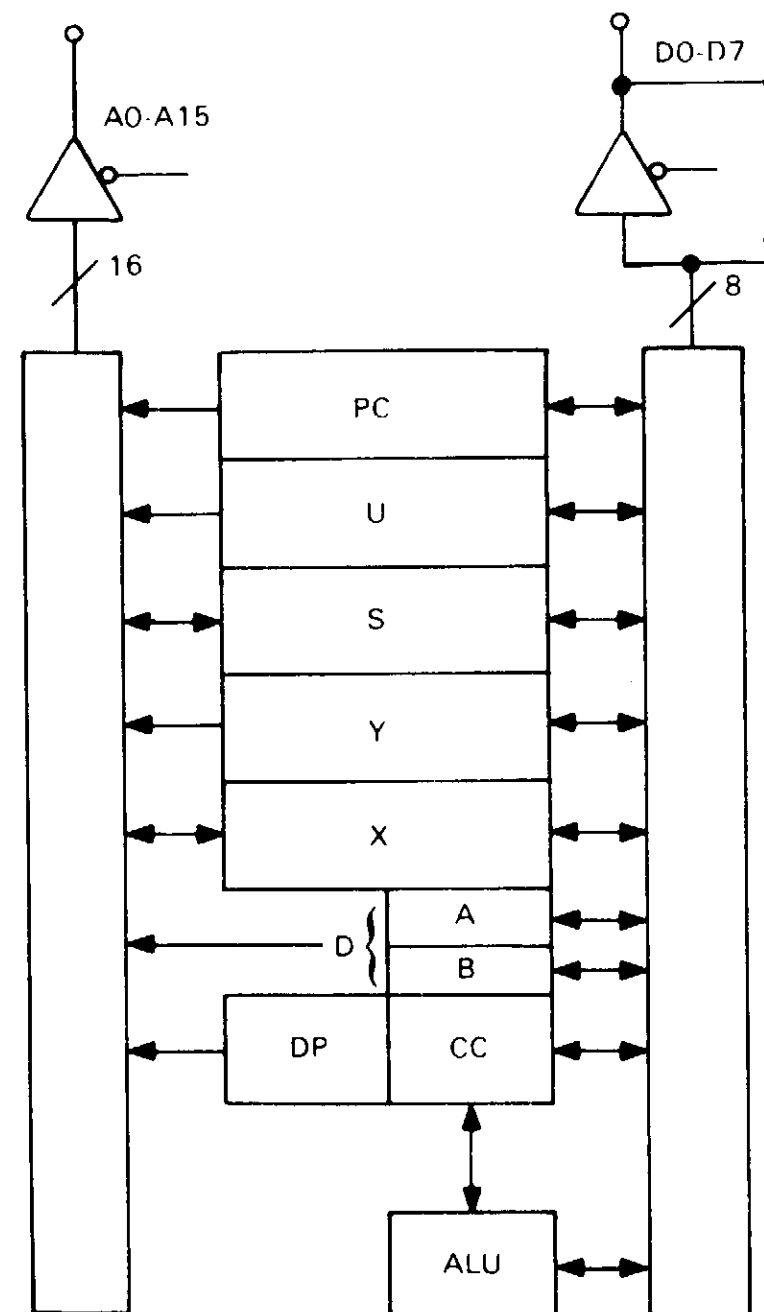
- Arithmétique 16 bits

- Transfert/échange tous registres

- Empilement/dépilage de chacun ou de l'ensemble des registres

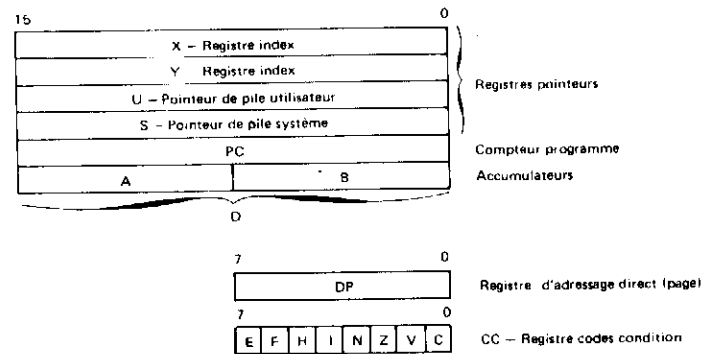
- Adresse effective de chargement

SCHÉMA FONCTIONNEL DU 6809



REGISTRES PROGRAMMABLES DU MICROPROCESSEUR

FIGURE 5 - REGISTRES PROGRAMMABLES DU MICROPROCESSEUR



REGISTRES PROGRAMMABLES

Comme indiqué ci-dessus, le microprocesseur 6809 comporte trois registres supplémentaires par rapport au 6800. Ces registres sont les suivants : un registre de page directe (DP), un registre pointeur de pile utilisateur (U) et un second registre index (Y).

ACCUMULATEURS (A, B, D)

Les registres A et B sont des accumulateurs universels utilisés pour les calculs arithmétiques et les manipulations de données. Certaines instructions concatènent les registres A et B pour former un seul accumulateur 16 bits. Le registre A constitue l'octet de poids fort de cet accumulateur référencé registre D.

REGISTRE PAGE DIRECTE (DP)

Le registre de page directe du circuit 6809 est utilisé pour étendre les possibilités d'adressage en mode direct. Le contenu de ce registre apparaît aux sorties d'adresse de poids forts (A8-A15) pendant l'exécution d'une instruction d'adressage direct. Ce registre permet d'utiliser le mode d'adressage direct, sous le contrôle du programme, dans tout l'espace d'adressage. Pour permettre la compatibilité avec la famille 6800, tous les bits de ce registre sont mis à zéro à l'initialisation du processeur.

REGISTRES INDEX (X, Y)

Les registres d'index sont utilisés pour les modes d'adressage indexé. Lors des calculs d'adresse effective, les 16 bits de ce registre sont utilisés. Les adresses contenues dans ces registres peuvent servir comme

pointeur de données et être modifiées par une constante optionnelle ou par une valeur de déplacement. Lors de rangement de données sous forme de table, dans certains modes d'adressage indexé, le contenu des registres d'index est incrémenté ou décrémenté pour pointer sur l'élément suivant. Les quatre registres (X, Y, U, S) peuvent être utilisés comme des registres d'index.

POINTEURS DE PILE (U,S)

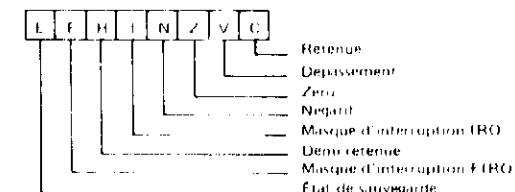
Le pointeur de pile (S) est utilisé automatiquement par le processeur pour mémoriser les états de la machine pendant l'exécution de sous-programmes et interruptions. Les pointeurs du 6809 pointent le haut de la pile, à l'opposé du pointeur du 6800, qui pointait l'emplacement libre suivant sur la pile. Le pointeur de pile utilisateur (U) est commandé par le programmeur exclusivement, permettant ainsi le passage de paramètres de et vers des sous-programmes avec facilité. Les pointeurs de pile U et S ont les mêmes possibilités que les registres X et Y pour les modes d'adressage indexé et pour les instructions d'empilement/dépilement. Le micro processeur 6809 peut être utilisé comme processeur avec gestion de pile, autorisant ainsi l'utilisation de langage de haut niveau et des méthodes de programmation modulaire.

COMPTEUR PROGRAMME (PC)

Le compteur programme est utilisé par le processeur pour pointer l'adresse de l'instruction suivante devant être exécutée. L'adressage relatif permet au compteur programme d'être utilisé comme un registre index dans certains cas.

REGISTRE DE CODES CONDITION (CC)

Le registre de codes condition définit l'état du processeur à tout instant.



DESCRIPTION DU REGISTRE CODES CONDITION (CC)

BIT 0 (C)

Le bit 0 est l'indicateur de retenue, il indique généralement la retenue lors d'une opération de l'unité arithmétique et logique. C est aussi utilisé pour représenter la retenue lors d'instructions correspondant à une soustraction (CMP, NEG, SUB, SBC). Dans ce cas cet indicateur est le complément de la retenue lors d'une opération de l'unité logique et arithmétique.

BIT 1 (V)

Le bit 1 est l'indicateur de débordement ; il est mis à un s'il y a débordement, en complément à deux signé après une opération arithmétique. Le débordement est détecté lors d'une opération dans l'unité logique et arithmétique quand la retenue du MSB ne correspond pas à la retenue du MSB-1.

BIT 2 (Z)

Le bit 2 est le bit indicateur de zéro, il est mis à un si le résultat de l'opération précédente est nul.

BIT 3 (N)

Le bit 3 indique un résultat négatif, il contient exactement la valeur du bit de poids fort de l'octet résultant de l'opération précédente. Un résultat négatif, en complément à deux, positionne N à 1.

BIT 4 (I)

Le bit 4 est le bit masque des interruptions IRQ. Ce bit mis à un, le processeur ne prendra pas en compte les interruptions arrivant sur la ligne IRQ. NMI, FIRQ, IRQ, RESET et SWI positionnent toutes 1 à un. SWI 2 et SWI 3 n'affectent pas 1.

BIT 5 (H)

Le bit 5 est le bit de demi-retenue ; il est utilisé pour indiquer une retenue du bit 3 dans l'ALU comme résultat d'une addition 8 bits seulement (ADC ou ADD). Ce bit est utilisé dans une instruction DAA pour réaliser une opération d'ajustement décimal. L'état de cet indicateur est indéfini dans toutes les instructions de soustraction ou équivalentes.

BIT 6 (F)

Le bit 6 est le bit masque des interruptions rapides FIRQ. Le processeur ne prendra pas en compte les interruptions de la ligne FIRQ, lorsque ce bit est à un. NMI, FIRQ, SWI, et RESET positionnent toutes F à un. F n'est pas affecté par IRQ, SWI 2 et SWI 3.

BIT 7 (E)

Le bit E est le bit indicateur de l'état de sauvegarde ; mis à un, il indique que l'état complet de la machine (tous les registres) est empilé, à la place de l'état précédent (PC et CC). Le bit E du registre CC empilé est utilisé sur un retour d'interruption (RTI) pour déterminer l'étendue du dépilement. Par conséquent, le bit E courant laissé dans le registre CC représente l'action précédente.

MODES D'ADRESSAGE

Les instructions de base de tout ordinateur sont particulièrement améliorées par la présence de modes d'adressage puissants. Le jeu de modes d'adressage disponible du 6809 est actuellement le plus puissant des microprocesseurs existants.

Par exemple, le 6809 possède 59 instructions de base, mais il admet 1464 possibilités différentes d'instructions et de modes d'adressage. Les nouveaux modes d'adressage permettent les techniques de programmation modernes. Les modes d'adressage suivants sont disponibles dans le 6809 :

Inhérent ou implicite (Inclut les accumulateurs)

Immédiat

Etendu

Etendu indirect

Direct

Registre

Indexé

Déplacement nul

Déplacement constant

Déplacement accumulateur

Auto incrémentation/décrémentation

Indexé indirect

Relatif

Branchement relatif long/court

Adressage relatif compteur programme

INHÉRENT (INCLUT LES ACCUMULATEURS)

Dans ce mode d'adressage, le code opération de l'instruction contient toute l'information adresse nécessaire. Des exemples d'adressage inhérent sont : ABX, DAA, SWI, ASRA, CLRB etc.

ADRESSAGE IMMÉDIAT

En adressage immédiat, l'adresse effective des données se trouve à l'emplacement suivant immédiatement le code opération ; les données à utiliser comme adresse dans l'instruction du 6809 utilisent deux valeurs immédiates 8 et 16 bits en fonction de la taille de l'opérande spécifié par le code opération. Des exemples d'instructions utilisant l'adressage immédiat sont :

```
LDA    #$20
LDX    #$F000
LDY    #ASTER
```

Note : # signifie adressage immédiat, \$ signifie valeur hexadécimale.

ADRESSAGE ÉTENDU

En adressage étendu, le contenu des deux octets suivant immédiatement le code opération spécifie complètement l'adresse 16 bits effective utilisée par l'instruction.

Il est à noter que l'adresse générée par une instruction étendue définit une adresse absolue et n'est pas translatable. Les exemples d'adressage étendu incluent :

```
LDA    ASTER
STX    OBEL
LDD    $2000
```

ÉTENDU INDIRECT

Comme cas spécial d'adressage indexé (exposé ci-dessous), un niveau d'indirection peut être ajouté à l'adressage étendu. En mode étendu indirect, les deux octets suivant le post octet d'une instruction indexée contiennent l'adresse de l'adresse des données.

```
LDA    [ASTER]
LDX    [$FFFE]
STU    [OBEL]
```

ADRESSAGE DIRECT

L'adressage direct est similaire à l'adressage étendu excepté qu'un octet d'adresse seulement suit le code opération. Cet octet spécifie les 8 bits de poids faible de l'adresse à utiliser. Les 8 bits d'adresse de poids fort sont fournis par le registre page directe. Un octet d'adresse étant seulement nécessaire en adressage direct, ce mode nécessite moins de mémoire et s'exécute plus rapidement qu'en adressage étendu. Bien entendu, seuls 256 emplacements (une page) peuvent être définis sans avoir à repositionner le contenu du registre DP. Le registre DP étant mis à \$ 00 à l'initialisation, l'adressage direct sur le 6809 est compatible avec l'adressage direct du 6800. L'indirection n'est pas permise en adressage direct.

Voici quelques exemples d'adressage direct :

```
LDA    $30
SETDP  $10 (directive d'assemblage)
LDB    $1030
LDD    <ASTER
```

Note : < est une directive d'assemblage qui force l'adressage direct.

ADRESSAGE PAR REGISTRE

Certains codes opération sont suivis d'un octet qui définit un registre ou un jeu de registres devant être utilisés par l'instruction, cet octet est appelé POST OCTET.

Quelques exemples d'adressage registre sont :

```
TFR    X,Y      Transfert de X dans Y
EXG    A,B,      Échange A et B
PSHS   A,B,X,Y   Transfert dans S Y, X, B puis A
PULU   X,Y,D     Transfert depuis U D, X, puis Y
```

ADRESSAGE INDEXÉ

Dans tout adressage indexé, un des registres pointeur (X, Y, U, S et parfois PC) est utilisé dans le calcul de l'adresse effective de l'opérande devant être utilisée par l'instruction. Cinq types d'indexation de base sont disponibles et sont exposés ci-dessous. Le post octet d'une instruction indexée spécifie le type de base et le choix du mode d'adressage ainsi que le registre pointeur devant être utilisé. Le tableau ci-dessous montre les formats autorisés pour le post octet. Le tableau "MODES d'Adressage INDEXÉ" donne la forme assembleur et le nombre de cycles et d'octets additionnés aux valeurs de base d'adressage indexé pour chaque variante.

Signification des bits du registre post-octet dans l'adressage indexé

Bit du registre post-octet								Mode d'adressage indexé
7	6	5	4	3	2	1	0	
0	R	R	X	X	X	X	X	EA = R ± 4 bits déplacement
1	R	R	0	0	0	0	0	,R+
1	R	R	1	0	0	0	1	,R++
1	R	R	0	0	0	1	0	,R-
1	R	R	1	0	0	1	1	,R
1	R	R	1	0	1	0	0	EA = R ± 0 déplacement
1	R	R	1	0	1	0	1	EA = R ± ACCB déplacement
1	R	R	1	0	1	1	0	EA = R ± ACCA déplacement
1	R	R	1	1	0	0	0	EA = R ± 7 bits déplacement
1	R	R	1	1	0	0	1	EA = R ± 15 bits déplacement
1	R	R	1	1	0	1	1	EA = R ± D déplacement
1	X	X	1	1	1	0	0	EA = PC ± 7 bits déplacement
1	X	X	1	1	1	0	1	EA = PC ± 15 bits déplacement
1	R	R	1	1	1	1	1	EA = , adresse

Champ du mode d'adressage
 Champ indirect
 bit de signe quand B7 = 0
 Champ du registre
 00: R = X
 01: R = Y
 10: R = U
 11: R = S
 X = indifférent

Indexé — Déplacement zéro. Dans ce mode, le registre pointeur sélectionné contient l'adresse effective des données devant être utilisées par l'instruction. Ce mode est le mode indexé le plus rapide.

Exemples :

```
LDD 0,X
LDA 0,S
```

Indexé — Déplacement constant. Dans ce mode, un déplacement en complément à deux et le contenu d'un des registres pointeurs sont additionnés pour former l'adresse effective de l'opérande. Le contenu initial du registre pointeur n'est pas changé par l'addition.

Trois valeurs de déplacement sont disponibles.

± 4-bit (− 16 à + 15)
 ± 7-bit (− 128 à + 127)
 ± 15-bit (− 32 768 à + 32 767)

Le déplacement 5 bits en complément à deux est compris dans le post octet et donc optimise l'utilisation des octets et des cycles. Le déplacement 8 bits en complément à deux est contenu dans un seul octet suivant le post octet. Le déplacement 16 bits en complément à

deux se trouve dans les deux octets suivant le post octet. Dans la plupart des cas, le programmeur n'a pas à connaître la valeur de ce déplacement puisque l'assembleur sélectionne automatiquement la valeur d'option.

Exemples d'indexation avec déplacement constant :

```
LDA 23,X
LDX -2,S
LDY 300,X
LDU ASTER,Y
```

Mode d'adressage indexé : non indirect

Type	Formes	Non Indirect			
		Syntaxe assembleur	Post-octet code OP	*	#
Déplacement constant à partir de R (signé)	pas de déplacement	R	1RR00100	0	0
	déplacement 5 bits	[n] R	0RRnnnnn	1	0
	déplacement 8 bits	[n] R	1RR01000	1	1
	déplacement 16 bits	[n] R	1RR01001	4	2
Accumulateur utilisé comme déplacement pour le Registre R (déplacement signé)	registre de déplac. A	A R	1RR00110	1	0
	registre de déplac. B	B R	1RR00101	1	0
	registre de déplac. D	D R	1RR01011	4	0
Auto incrémentation/décrémentation du registre R	incrémenté par 1	R+	1RR00000	2	0
	incrémenté par 2	R++	1RR00001	3	0
	décrémenté par 1	R-	1RR00010	2	0
	décrémenté par 2	R	1RR00011	3	0
Déplacement constant à partir de PC	déplacement 8 bits	[n] PCR	1XX01100	1	1
	déplacement 16 bits	[n] PCR	1XX01101	5	2
Indirect étendu	adresses 16 bits				

R = X, Y, U ou S
 X = indifférent
 X 00 Y 01
 U 10 S 11

* et # indiquent le nombre de cycles et d'octets additionnels pour un état particulier.

Mode d'adressage indexé : indirect

Type	Formes	Indirect			
		Syntaxe assembleur	Post-octet code OP	*	#
Déplacement constant à partir de R (signé)	pas de déplacement	[R]	1RR10100	3	0
	déplacement 5 bits	[n] R	1RR10101	4	0
	déplacement 8 bits	[n] R	1RR11000	4	1
	déplacement 16 bits	[n] R	1RR11001	7	2
Accumulateur utilisé comme déplacement pour le Registre R (déplacement signé)	registre de déplac. A	[A] R	1RR10110	4	0
	registre de déplac. B	[B] R	1RR10101	4	0
	registre de déplac. D	[D] R	1RR11011	7	0
	incrémenté par 1	impossible			
Auto incrémentation/décrémentation du registre R	incrémenté par 2	[R++]	1RR10001	6	0
	décrémenté par 1	impossible			
	décrémenté par 2	[R--]	1RR10011	6	0
	incrémenté par 1	[R+]	1RR10010	6	0
Déplacement constant à partir de PC	déplacement 8 bits	[n] PCR	1XX11100	4	1
	déplacement 16 bits	[n] PCR	1XX11101	8	2
Indirect étendu	adresses 16 bits	[n]	10011111	5	2

R = X, Y, U ou S
 X = indifférent
 X 00 Y 01
 U 10 S 11

* et # indiquent le nombre de cycles et d'octets additionnels pour un état particulier.

MODES D'ADRESSAGE INDEXÉ

Indexé — Déplacement accumulateur. Ce mode est semblable au mode indexé à déplacement constant, excepté que la valeur en complément à deux dans un des accumulateurs (A, B ou D) et le contenu de l'un des registres pointeurs sont ajoutés pour former l'adresse effective de l'opérande. Le contenu du registre pointeur et de l'accumulateur demeure inchangé par l'addition. Le post octet spécifie l'accumulateur à utiliser comme déplacement et aucun octet supplémentaire n'est nécessaire. L'avantage d'un déplacement accumulateur réside dans le fait que la valeur du déplacement peut être calculée par programme en cours d'exécution.

Exemples :

```
LDA    B, Y
LDX    D, Y
LEAX    B, X
```

Indexé — Auto Incrémentation/Décrémentation. En mode auto incrémentation, le registre pointeur contient l'adresse de l'opérande. Ainsi, après avoir été utilisé le registre pointeur est incrémenté de un ou deux. Ce mode d'adressage est très utile lors de l'utilisation de tables, de déplacement de données, ou pour la création de piles logicielles. En auto décrémentation le registre pointeur est décrémenté avant d'être utilisé comme adresse des données. L'utilisation en auto décrémentation est similaire à celle en auto incrémentation, mais les tables sont scrutées les adresses élevées vers les adresses faibles. La valeur d'incrément/décrément peut être égale à un ou deux pour permettre d'accéder à des tables de données 8 ou 16 bits, elle est sélectionnée par le programmeur. L'aspect pré-décrément, post-incrément permet à ces modes d'être utilisés pour créer des piles logicielles supplémentaires qui se comportent de manière identique aux piles U et S.

Voici quelques exemples de modes d'adressage auto incrément/décrément :

```
LDA    ,X+
STD    ,Y++
LDB    ,-Y
LDX    ,--X
```

INDEXE INDIRECT

Tous les modes indexé indirect sont inclus à l'exception d'incrément/décrément par un, ou déplacement de ± 4 bits et peu-

vent avoir un niveau d'indirection supplémentaire spécifié. En adressage indirect, l'adresse effective est contenue à l'emplacement spécifié par le contenu du registre index, additionné d'un quelconque déplacement. Dans l'exemple ci-dessous, l'accumulateur A est chargé indirectement en utilisant une adresse effective calculée à partir du registre index et d'un déplacement.

Avant exécution :

A = x x (indifférent)

X = \$ F000

\$0100 LDA [10,X] l'EA est alors \$F010

\$F010 \$F1 F150 est alors la nouvelle

\$F011 \$50 adresse effective

\$F150 \$AA

Après exécution :

A = \$AA Donnée chargée

Note : EA = adresse effective.

Tous les modes indexé indirect sont inclus à l'exception de ceux qui sont sans signification (exemple : auto incrément/décrément par 1 indirect). Quelques exemples de mode indexé indirect sont :

```
LDA    [,X]
LDD    [10,X]
LDA    [B,Y]
LDD    [,X+ +]
```

ADRESSAGE RELATIF

Le(s) octet(s) suivant(s) le code opération de branchement est (sont) traité(s) comme un déplacement signé qui est additionné au compteur programme.

Si la condition de branchement est vraie, alors l'adresse calculée (PC + déplacement signé) est chargée dans le compteur programme. L'exécution du programme se poursuit jusqu'au nouvel emplacement comme indiqué par le PC, les modes d'adressage relatif court (1 octet de déplacement) et long (déplacement de deux octets) sont disponibles. Tout emplacement mémoire peut être atteint en mode d'adressage relatif long, l'adresse effective étant interprétée modulo 2^{16} . Quelques exemples d'adressage relatif sont :

	BEQ	ASTER	(court)
	BGT	OBEL	(court)
ASTER	LBEQ	BAMBI	(long)
OBEL	LBGT	BUNNY	(long)

BAMBI	NOP
BUNNY	NOP

Le compteur programme peut être utilisé comme registre pointeur avec des déplacements signés de 8 ou 16 bits. Comme en adressage relatif le déplacement est additionné au PC en cours pour former l'adresse effective. L'adresse effective est alors utilisée comme adresse opérande ou adresse données. L'adressage relatif par compteur programme est utilisé pour écrire des programmes translatables. Les tables relatives à un programme particulier gardent la même liaison après translation du programme, si celles-ci sont référencées en relatif par rapport au compteur programme.

Exemples :

LDA	BUNNY, PCR
LEAX	TABLE, PCR

Le mode compteur programme relatif étant un type d'indexation, un niveau supplémentaire d'indirection est utilisable.

LDA	[ASTER, PCR]
LDU	[OBEL, PCR]

JEU D'INSTRUCTIONS DU 6809

Le jeu d'instruction du 6809 est comparable à celui du 6800 et compatible ascendant au niveau du code source. Le nombre de codes opération a été réduit de 72 à 59, mais grâce à son architecture améliorée et modes d'adressage supplémentaires, le nombre de codes opération disponibles (avec les différents modes d'adressage) est passé de 197 à 1 464.

Certaines instructions et certains modes d'adressage sont décrits en détail ci-dessous :

PSHU/PSHS

Ces instructions ont la propriété d'empiler tout(s) registre(s) du MPU soit sur la pile matériel (S) soit sur la pile utilisateur (U) en une seule instruction.

PULU/PULS

Les instructions de dépilement ont la même propriété que les instructions d'empilement, dans l'ordre inverse. L'octet immédiat suivant

le code opération des instructions d'empilement ou de dépilement détermine quel ou quels registres doivent être empilés ou dépilés. La séquence effective d'empilement/dépilage est fixée ; chaque bit détermine un registre unique à empiler/dépiler comme indiqué.

POST OCTET D'EMPILEMENT/DÉPILEMENT

← ordre d'empilement				ordre de dépilement →			
PC	U	Y	X	DP	B	A	CC PSHS/PULS
FFFF... ← adresse mémoire croissante... 0000							
PC	S	Y	X	OP	B	A	CC PSHU/PULU

TFR/EXG

Dans le 6809 chaque registre peut être transféré ou échangé avec un autre registre de même format, c'est-à-dire 8 bits à 8 bits ou 16 bits à 16 bits. Les bits 4-7 du post octet définissent le registre source, tandis que les bits 0-3 représentent le registre destination.

Ceci se représente comme suit :

0000 – D	0101 – PC
0001 – X	1000 – A
0010 – Y	1001 – B
0011 – U	1010 – CC
0100 – S	1011 – DP

Note : Toutes les autres combinaisons sont indéfinies et non valables.

Chargement d'adresse effective (LEA)

L'instruction LEA s'exécute en calculant l'adresse effective utilisée dans une instruction indexée et mémorise cette valeur d'adresse, au lieu des données de cette adresse, dans un registre pointeur. Ceci met l'ensemble des caractéristiques d'adressage interne matériel à la disposition du programmeur. Quelques-unes des implications de cette instruction sont illustrées à l'aide d'exemples.

L'instruction LEA permet aussi à l'utilisateur d'accéder à des données quel que soit l'emplacement. Par exemple :

	LEAX	MSG1, PCR
	LBSR	PDATA (programme d'impression message)
MSG1	FCC	/MESSAGE/

Cet ensemble de programme imprime "message". En écrivant MSG1, PCR, l'assembleur calcule la distance entre l'adresse présente et MSG1. Ce résultat est placé comme une constante dans l'instruction LEAX qui est indexée par la valeur du PC au moment de l'exécution. Peu importe la position du code pendant son exécution puisque le déplacement calculé depuis le PC mettra l'adresse absolue de MSG1 dans le registre pointeur X. Ce code est totalement translatable.

EXEMPLES D'UTILISATION DE L'INSTRUCTION LEA

Instruction	Opération	Commentaire
LEAX 10, X	X ← 10 → X	Addition constante sur 5 bits de 10 dans X
LEAX 500, X	X ← 500 → X	Addition constante sur 16 bits de 500 dans X
LEAY A, Y	Y ← A → Y	Addition de l'accumulateur sur 8 bits dans Y
LEAY D, Y	Y ← D → Y	Addition de l'accumulateur D sur 16 bits dans Y
LEAU -10, U	U ← 10 → U	Soustraction de 10 dans U
LEAS -10, S	S ← 10 → S	Réservation d'une zone dans la pile
LEAS 10, S	S ← 10 → S	Remise en ordre de la pile
LEAX 5, S	S ← 5 → X	Transfert aussi bien qu'addition

MUL

Multiple les nombres binaires non signés des accumulateurs A et B et place le résultat non signé dans l'accumulateur 16 bits D.

BRANCHEMENTS RELATIFS LONG ET COURT

Le 6809 a la possibilité de réaliser des branchements relatifs au compteur programme sur tout l'espace mémoire. Dans ce mode, en cas de branchement, le déplacement signé de 8 ou 16 bits est additionné à la valeur du compteur programme utilisé comme adresse effective. Ceci permet le branchement du programme n'importe où dans les 64 K d'espace mémoire. Le code translatable peut être facilement généré par l'utilisation du branchement relatif. Les deux branchements court (8 bits) et long (16 bits) sont disponibles.

TABLEAU DE JEU D'INSTRUCTIONS DU 6809

Les instructions du 6809 ont été séparées en cinq catégories différentes qui sont :

- Fonctionnement 8 bits
- Fonctionnement 16 bits
- Instructions portant sur le registre index/pointeur de pile
- Branchements relatifs (long et court)
- Instructions diverses
- Instructions sur valeur hexadécimale

INSTRUCTIONS SUR LES ACCUMULATEURS ET LA MÉMOIRE (8 BITS)

Mnémoniques	Opérations
ADCA, ADCB	Addition du contenu mémoire à l'accumulateur, avec retenue
ADDA, ADDB	Addition du contenu mémoire à l'accumulateur
ANDA, ANDB	ET logique entre mémoire et l'accumulateur
ASL, ASLA, ASLB	Décalage arithmétique à gauche du contenu mémoire ou accumulateur
ASR, ASRA, ASRB	Décalage arithmétique à droite du contenu mémoire ou accumulateur
BITA, BITB	Test de bit mémoire avec l'accumulateur
CLR, CLRA, CLRB	Mise à zéro du contenu de l'accumulateur ou de la mémoire
CMPA, CMPB	Comparaison du contenu mémoire avec l'accumulateur
COM, COMA, COMB	Complément à deux de l'accumulateur ou du contenu mémoire
DAA	Ajustement décimal de l'accumulateur A
DFC, DFCA, DECB	Décrémentation du contenu mémoire ou de l'accumulateur
EORA, EORB	« OU » exclusif du contenu mémoire avec l'accumulateur
EXG R1, R2	Echange de R1 avec R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Incrémement du contenu mémoire ou de l'accumulateur
LDA, LDB	Chargement de l'accumulateur avec le contenu mémoire
LSL, LSLA, LSLB	Décalage logique à gauche du contenu mémoire ou de l'accumulateur
LSR, LSRA, LSRB	Décalage logique à droite du contenu mémoire ou de l'accumulateur
MUL	Multiplication non signée (A × B → D)
NEG, NEGA, NEGB	Complément à un du contenu mémoire ou de l'accumulateur
ORA, ORB	« OU » logique mémoire et accumulateur
ROL, ROLA, ROLB	Décalage circulaire à gauche du contenu mémoire ou de l'accumulateur
ROR, RORA, RORB	Décalage circulaire à droite du contenu mémoire ou de l'accumulateur
SBCA, SBCB	Soustraction du contenu mémoire de l'accumulateur
STA, STB	Mise en mémoire du contenu de l'accumulateur
SUBA, SUBB	Soustraction du contenu mémoire de l'accumulateur
TS1, TSTA, TSTB	Test mémoire ou accumulateur
TFR, R1, R2	Transfert de R1 à R2 (R1, R2 = A, B, CC, DP)

NOTE : A, B, CC ou DP peuvent être remplis sur (ou dépiler) de chaque pile avec les instructions PSHS, PSHU, IPULS, PULU).

INSTRUCTIONS SUR LES ACCUMULATEURS ET LA MÉMOIRE (16 BITS)

Mnémoniques	Opérations
ADDD	Addition du contenu mémoire à l'accumulateur D
CMPD	Comparaison du contenu mémoire avec l'accumulateur D
EXG D, R	Echange de D avec X, Y, S, U ou PC
LDD	Chargement de l'accumulateur D avec le contenu mémoire
SEX	Extension de signe de l'accumulateur B à l'accumulateur A
STD	Mise en mémoire de l'accumulateur D
SUBD	Soustraction du contenu mémoire de l'accumulateur D
TFR D, R	Transfert de D vers X, Y, S, U ou PC
TFR R, D	Transfert de X, Y, S, U ou PC vers D

INSTRUCTIONS SUR LES REGISTRES INDEX ET LE POINTEUR DE PILE

Mnémoniques	Opérations
CMP _S CMP _U	Comparaison mémoire avec pointeur de pile
CMPX CMPY	Comparaison mémoire avec registre index
EXG R1 R2	Echange de D, X, Y, S, U ou PC avec D, X, Y, S, U, ou PC
LEAS LEAU	Chargement de l'adresse effective dans le pointeur de pile
LEAX LEAY	Chargement de l'adresse effective dans le registre index
LDS LDU	Chargement du pointeur de pile avec le contenu mémoire
LDX LDY	Chargement du registre index avec le contenu mémoire
PSHS	Empilement de tout (s) registre (s) (sauf S) sur la pile S
PSHU	Empilement de tout (s) registre (s) (sauf U) sur la pile U
PULS	Dépilement de tout (s) registre (s) (sauf S) de la pile S
PULU	Dépilement de tout (s) registre (s) (sauf U) de la pile U
STS STU	Mise en mémoire du pointeur de pile
STX STY	Mise en mémoire du registre index
TFR R1 R2	Transfert de D, X, Y, S, U ou PC vers D, X, Y, S, U ou PC
ABX	Addition de l'accumulateur B à X (non signé)

INSTRUCTIONS DE BRANCHEMENT

Mnémoniques	Opérations
BCC LBCC	Branchement si pas de retenue
BCS LBCS	Branchement si retenue
BEQ LBEQ	Branchement si égal
BGE LBGE	Branchement si supérieur ou égal (signé)
BGT LBGT	Branchement si supérieur (signé)
BHI LBHI	Branchement si supérieur (non signé)
BHS LBHS	Branchement si supérieur ou égal (non signé)
BLE LBLE	Branchement si inférieur ou égal (signé)
BLO LBLO	Branchement si inférieur (non signé)
BLS LBLS	Branchement si inférieur ou égal (non signé)
BLT LBLT	Branchement si inférieur (signé)
BMI LBMI	Branchement si négatif
BNE LBNE	Branchement si non égal
BPL LBPL	Branchement si positif
BRA LBRA	Branchement inconditionnel
BRN LB RN	Non branchement
BSR LB SR	Branchement à un sous-programme
BVC LBVC	Branchement si pas de débordement < 0
BVS LBVS	Branchement si débordement > 1

INSTRUCTIONS SPÉCIALES

Mnémoniques	Opérations
ANDCC	« E E » logique avec le registre codes condition
CWAI	« E E » logique avec le registre codes condition puis attente d'interruption
NOP	Non opération
ORCC	« OU » logique avec le registre codes condition
JMP	Saut inconditionnel
JSR	Saut à un sous-programme
RTI	Retour d'interruption
RTS	Retour de sous-programme
SWI CWI2 SWI1	Interruption programmée
SYNC	Synchronisation avec la ligne d'interruption

VALEURS HEXADÉCIMALES DU CODE MACHINE

Code OP Mnémonique	Mode	#	Code OP Mnémonique	Mode	#	Code OP Mnémonique	Mode	#
00 NEG	Direct	6 2	30 LEAX	Indexé	4+ 2+	60 NEG	Indexé	6+ 2+
01 *			31 LEAY	Indexé	4+ 2+	61 *		
02 *			32 LEAS	Indexé	4+ 2+	62 *		
03 COM		6 2	33 LEAU	Indexé	4+ 2+	63 COM		6+ 2+
04 LSR		6 2	34 PSHS	Indexé	5+ 2	64 LSR		6+ 2+
05 *			35 PULS	Indexé	5+ 2	65 *		
06 ROR		6 2	36 PSHU	Indexé	5+ 2	66 ROR		6+ 2+
07 ASR		6 2	37 PULU	Indexé	5+ 2	67 ASR		6+ 2+
08 ASL/LSL		6 2	38 *			68 ASL/LSL		6+ 2+
09 ROL		6 2	39 RTS		5 1	69 ROL		6+ 2+
0A DEC		6 2	3A ABX		3 1	6A DEC		6+ 2+
0B *			3B RTI		6 15 1	6B *		
0C INC		6 2	3C CWAI		20 2	6C INC		6+ 2+
0D TST		6 2	3D MUL		11 1	6D TST		6+ 2+
0E JMP		3 2	3E *			6E JMP		3+ 2+
0F CLR	Direct	6 2	3F SWI	Implicite	19 1	6F CLR	Indexé	6+ 2+
10 Page 2			40 NEGA	Implicite	2 1	70 NEG	Étendu	7 3
11 Page 3			41 *			71 *		
12 NOP	Implicite	2 1	42 *			72 *		
13 SYNC	Implicite	2 1	43 COMA		2 1	73 COM		7 3
14 *			44 LSRA		2 1	74 LSR		7 3
15 *			45 *			75 *		
16 LBRA	Relatif	5 3	46 RORA		2 1	76 ROR		7 3
17 LBSP	Relatif	9 3	47 ASRA		2 1	77 ASR		7 3
18 *			48 ASLA/LSLA		2 1	78 ASL/LSL		7 3
19 OAA	Implicite	2 1	49 ROLA		2 1	79 ROL		7 3
1A ORCC	Implicite	3 2	4A DECA		2 1	7A DEC		7 3
1B *			4B *			7B *		
1C ANDCC	Immédiat	3 2	4C INCA		2 1	7C INC		7 3
1D SFX	Implicite	2 1	4D TSTA		2 1	7D TST		7 3
1E EXG		8 2	4E *			7E JMP		4 3
1F TFR	Implicite	6 2	4F CLRA	Implicite	2 1	7F CLR	Étendu	7 3
20 BRA	Relatif	3 2	50 NEGB	Implicite	2 1	80 SUBA	Immédiat	2 2
21 BRN		3 2	51 *			81 CMPA		2 2
22 BHI		3 2	52 *			82 SBRA		2 2
23 BLS		3 2	53 COMB		2 1	83 SUBD		4 1
24 BHS/BCC		3 2	54 LSRB		2 1	84 ANDA		2 2
25 BLO/BCS		3 2	55 *			85 BITA		2 2
26 BNE		3 2	56 *			86 LDA		2 2
27 BEQ		3 2	56 RORB		2 1	87 *		
28 BVC		3 2	57 ASRA		2 1	88 FOHA		2 2
29 BVS		3 2	58 ASLB/LSLB		2 1	89 ADRA		2 2
2A BPL		3 2	59 ROLB		2 1	8A ORA		2 2
2B BMI		3 2	5A DECB		2 1	8B ADDA		2 2
2C BGE		3 2	5B *			8C CMPX	Immédiat	4 3
2D BLT		3 2	5C INCB		2 1	8D HSR	Relatif	7 3
2E BGT		3 2	5D TSTR		2 1	8E LUX	Immédiat	1 3
2F BLE	Relatif	3 2	5E *			8F *		
			5F CLRB	Implicite	2 1			

Légende
 + nombre de cycles MPU
 # nombre d'opérations
 * code OP non utilisé

Tableau des instructions (suite)

Instruction	Forms	Addressing Modes										Description	5 3 2 1 0										
		Immediate		Direct		Indexed ¹		Extended		Inherent			H	N	Z	V	C						
		Op	#	Op	#	Op	#	Op	#	Op	#												
LSL	LSLA LSLB LSL																						
LSR	LSRA LSRB LSR			04	6	2	04	6	2	04	6	2	04	6	2	04	6	2	04	6	2		
MUL				04	6	2	04	6	2	04	6	2	04	6	2	04	6	2	04	6	2		
NEG	NEGA NEGB NEG			00	6	2	00	6	2	00	6	2	00	6	2	00	6	2	00	6	2		
NOP																							
OR	ORA ORB ORCC			0A	2	2	0A	2	2	0A	2	2	0A	2	2	0A	2	2	0A	2	2		
PSH	PSHS PSHU			34	5	4	2																
PUL	PULS PULU			35	5	4	2																
ROL	ROLA ROLB ROL							09	6	2	09	6	2	09	6	2	09	6	2	09	6	2	
ROR	RORA RORB ROR							06	6	2	06	6	2	06	6	2	06	6	2	06	6	2	
RTI																							
RTS																							
SBC	SBCA SBCB			02	2	2	02	2	2	02	2	2	02	2	2	02	2	2	02	2	2		
SEX																							
ST	STA STB STD STS STU STX STY					02	2	2	02	2	2	02	2	2	02	2	2	02	2	2	02	2	2
SUB	SUBA SUBB SUBD			00	2	2	00	2	2	00	2	2	00	2	2	00	2	2	00	2	2		
SWI	SWI ⁶ SWI ¹² SWI ¹⁶																						
SYNC																							
TFR	R1 R2			1F	6	2																	
TST	TSTA TSTB TST							0D	6	2	0D	6	2	0D	6	2	0D	6	2	0D	6	2	

Notes

1. Cette colonne donne le nombre de cycles de base et le décompte d'octets. Afin d'obtenir le décompte total, ajouter les valeurs obtenues dans la table Mode d'adressage indexé.
2. R1 et R2 doivent être une paire de registres 8 bits ou une 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. 5 (6) signifie : cinq cycles si branchement non pris, six cycles si branchement pris (instructions de branchement).
6. SWI1 fixe les bits F et I. SWI11 et SWI12 n'affectent pas I et F.
7. L'état du registre code condition résulte directement de cette instruction.
8. Valeur du flag de demi-retenue non défini.
9. Cas particulier : retenue de bit fixé si B7 est SET.

Instruction	Forms	Addressing Mode			Description	5	3	2	1	0
		Relative				H	N	Z	V	C
		OP	-	#						
BCC	BCC LBCC	24 10 24	3 5(6) 4	2 4	Branch C = 0 Long Branch C = 0	*	*	*	*	*
BCS	BCS LBCS	25 10 25	3 5(6) 4	2 4	Branch C = 1 Long Branch C = 1	*	*	*	*	*
BEQ	BEQ LBEQ	27 10 27	3 5(6) 4	2 4	Branch Z = 1 Long Branch Z = 1	*	*	*	*	*
BGE	BGE LBGE	2C 10 2C	3 5(6) 4	2 4	Branch ≥ Zero Long Branch ≥ Zero	*	*	*	*	*
BGT	BGT LBGT	2E 10 2E	3 5(6) 4	2 4	Branch > Zero Long Branch > Zero	*	*	*	*	*
BHI	BHI LBHI	22 10 22	3 5(6) 4	2 4	Branch Higher Long Branch Higher	*	*	*	*	*
BHS	BHS LBHS	24 10 24	3 5(6) 4	2 4	Branch Higher or Same Long Branch Higher or Same	*	*	*	*	*
BLE	BLE LBLE	2F 10 2F	3 5(6) 4	2 4	Branch ≤ Zero Long Branch ≤ Zero	*	*	*	*	*
BLO	BLO LBLO	25 10 25	3 5(6) 4	2 4	Branch lower Long Branch Lower	*	*	*	*	*
BLS	BLS LBLS	23 10 23	3 5(6) 4	2 4	Branch Lower or Same Long Branch Lower or Same	*	*	*	*	*
BLT	BLT LBLT	2D 10 2D	3 5(6) 4	2 4	Branch < Zero Long Branch < Zero	*	*	*	*	*
BMI	BMI LBMI	2B 10 2B	3 5(6) 4	2 4	Branch Minus Long Branch Minus	*	*	*	*	*
BNE	BNE LBNE	26 10 26	3 5(6) 4	2 4	Branch Z = 0 Long Branch Z = 0	*	*	*	*	*
BPL	BPL LBPL	2A 10 2A	3 5(6) 4	2 4	Branch Plus Long Branch Plus	*	*	*	*	*
BRA	BRA LBRA	20 16 5	3 3	2 3	Branch Always Long Branch Always	*	*	*	*	*
BRN	BRN LBRN	21 10 21	3 5 4	2 4	Branch Never Long Branch Never	*	*	*	*	*
BSR	BSR LBSR	8D 17 9	7 3	2 3	Branch to Subroutine Long Branch to Subroutine	*	*	*	*	*
BVC	BVC LBVC	28 10 28	3 5(6) 4	2 4	Branch V = 0 Long Branch V = 0	*	*	*	*	*
BVS	BVS LBVS	29 10 29	3 5(6) 4	2 4	Branch V = 1 Long Branch V = 1	*	*	*	*	*

SIMPLE BRANCHES

	OP	~	I
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

SIMPLE CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
N = 1	BMI	2B	BPL	2A
Z = 1	BEQ	27	BNE	26
V = 1	BVS	29	BVC	28
C = 1	BCS	25	BCC	24

SIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BGT	2E	BLE	2F
r ≥ m	BGE	2C	BLT	2D
r = m	BEQ	27	BNE	26
r ≤ m	BLE	2F	BGT	2E
r < m	BLT	2D	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BHI	22	BLS	23
r ≥ m	BHS	24	BLO	25
r = m	BEQ	27	BNE	26
r ≤ m	BLS	23	BHI	22
r < m	BLO	25	BHS	24

Notes

1. Tous les branchements conditionnels ont des variantes longues et courtes.
2. Tous les branchements courts sont de deux octets et nécessitent trois cycles.
3. Tous les branchements conditionnels longs sont constitués en préfixant \$ 10 au code opération branchement court et utilisent un déplacement de destination de seize bits.
4. Tous les branchements conditionnels longs nécessitent quatre octets et six cycles si le branchement est pris, ou cinq cycles si le branchement n'est pas pris.

ANNEXE C CODE ASCII

ASCII Character Set

b7 b6 Bits		b5				0		0		0		0		1		1		1		1	
						0		0		1		0		1		0		1		0	
b4	b3	b2	b1	Row	Column Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	
0	0	0	0	0	0	NUL	DLE	SP	0	@	P	.	p								
0	0	0	1	1	1	SOH	DC1	!	1	A	Q	a	q								
0	0	1	0	2	2	STX	DC2	"	2	B	R	b	r								
0	0	1	1	3	3	ETX	DC3	#	3	C	S	c	s								
0	1	0	0	4	4	EOT	DC4	\$	4	D	T	d	t								
0	1	0	1	5	5	ENQ	NAK	%	5	E	U	e	u								
0	1	1	0	6	6	ACK	SYN	&	6	F	V	f	v								
0	1	1	1	7	7	BEL	ETB	'	7	G	W	w	w								
1	0	0	0	8	8	BS	CAN	(8	H	X	x	x								
1	0	0	1	9	9	HT	EM)	9	I	Y	y	y								
1	0	1	0	10	A	LF	SUB	*		J	Z	z	z								
1	0	1	1	11	B	VT	ESC	+		K	[k	k								
1	1	0	0	12	C	FF	FS	,	<	L	\	l	l								
1	1	0	1	13	D	CR	GS	-	=	M]	m	m								
1	1	1	0	14	E	SO	RS	.	>	N	^	n	n								
1	1	1	1	15	F	SI	US	/	?	O	_	o	o							DEL	

SI	Shift In
DLE	Data Link Escape
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
NAK	Negative Acknowledge
SYN	Synchronous Idle
ETB	End of Transmission Block
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator
DEL	Delete

ANNEXE D

MESSAGES D'ERREUR

Les erreurs d'entrée/sortie

- « **Bad Parameters** »

Le paramètre de la commande est incorrect.
(Erreur dans le numéro du disque).

- « **Disk not ready** »

Le disque n'est pas prêt.

- « **Bad file descriptor** »

Erreur de syntaxe dans le descripteur du fichier ou descripteur absent.

- « **I/O Error** »

Impossibilité physique de réaliser une entrée/sortie.

- « **File Format Error** »

Le fichier ne peut pas être chargé. Il y a deux causes possibles :

- erreur sur la disquette,
- le mode de chargement utilisé est incorrect.

- « **File not Found** »

Le fichier décrit par le descripteur n'est pas trouvé sur le périphérique indiqué.

- « **Verification Error** »

Le contenu de la mémoire est différent du contenu du fichier indiqué.

- « **Bad Memory** »

L'adresse indiquée pour l'implantation du code objet est incorrecte.

- « **File Already Exists** »

Le nom du fichier ne peut pas être remplacé par un nom déjà existant dans le catalogue (RENAME).

- « **File Already Open** »

Le fichier ne peut pas être copié sur lui-même.

- « **Disk Write Protected** »

Essai d'écriture sur une disquette protégée.

- « **Out Of Memory** »

Il n'y a pas suffisamment de RAM libre.

- « **Disk Full** »

Il n'y a plus de place sur la disquette.

- « **Unreadable Disk** »

La disquette utilisée n'a pas été formatée.

- « **Directory Full** »

Il n'y a plus de place dans le catalogue de la disquette.

Les erreurs d'assemblage

- « **Branch Out Of Range** »

L'adresse du branchement est trop éloignée de l'adresse de l'instruction courante.

- « **Bad Address** »

L'adresse indiquée pour l'implantation du code objet lors de l'assemblage en mémoire est incorrecte.

- « **Bad Operand** »

L'opérande n'est pas reconnu par l'assembleur.

- « **Bad Includ** »

Le fichier indiqué par **INCLUD** se trouve dans la cassette ou contient lui-même une autre directive **INCLUD**.

- « **Expression Error** »

Mauvaise expression utilisée dans le champ opérande.

- « **Bad Opcode** »

Mauvais code opération.

- « **Undefined symbol** »

Le symbole utilisé n'a pas été défini.

- « **Missing END Statement** »

Le programme source n'est pas terminé par la directive **END**.

- « **Multiply defined symbol** »

Le même symbole est utilisé plusieurs fois.

- « **Register Error** »

Le symbole utilisé ne correspond pas à un registre du microprocesseur.

- « **Bad Label** »

Mauvaise étiquette.

- « **Opérand Too Large** »

Opérande trop grand.

- « **DP Error** »

Erreur sur la page directe.

- « **Symbol Table Full** »

Il n'y a plus de place dans la table des symboles.

Les autres erreurs possibles sous éditeur-moniteur ou Entrées/Sorties

- « **Bad Command** »

La commande n'est pas reconnue par le système.

- « **Bad Parameter(s)** »

Le paramètre de la commande est incorrect ou absent.

- « **Missing operand** »

Il manque au moins un paramètre dans les commandes **SAVE** et **VERIFY** sous contrôle du moniteur.

- « **Undefined Symbol** »

Le symbole utilisé n'est pas reconnu par le moniteur.

- « **String not found** »

La chaîne de caractères n'a pas été trouvée.

- « **Internal Error** »

Erreur interne *

- « **Missing information** »

Il manque une information dans la commande.

- « **Can't Continue** »

Le programme en cours est arrêté et ne peut plus être repris par **C** (**CONTINUE**) car il est en ROM.

- « **Bad Breakpoint** »

Erreur dans les commandes des points d'arrêt.

***N.B. :** Cette erreur est généralement due à des effets incontrôlés du programme utilisateur. Une fois le programme sauvé, il peut être prudent de réinitialiser tout le système en coupant l'alimentation car des registres propres à l'Assembleur peuvent avoir été altérés.

ANNEXE E

PROGRAMMES TYPES ET UTILISATION DES BANQUES MÉMOIRES DU T07-70

Programme Type A

```

*
* Programme de balayage aleatoire de
* la memoire point.
*
      ORG      ENDMEM-$400 1K Reserve.
DIRECT EQU    *K-8      Page 0 ici.
      SETDP   DIRECT
*
      TITLE   Balayage Ecran.
*
      INCLUDE EQUATES Fichier contenant
* les principales adresses du Moniteur
* du T07.
*

```

Note : en version cassette, INCLUDE doit être remplacé par le contenu du fichier EQUATES.

```

CONST FDB     1          Constante de dep-
* lacement, initialisee a 1, ou les car-
* acteres entres au clavier vont rentrer
* par la droite.

ACC     RMB     2          Accumulateur de
* calcul.

TABIT   FCB     $80,$40,$20,$10,$08,$04
        FCB     $02,$01
* Table des bits qui vont etre inverses
* en memoire. C'est un bit a 1 qui occu-
* pe successivement les 8 positions d'un
* octet.

```

```

COUL0 SET      $44      Bleu.
COUL1 SET      $52      Fond vert.
COUL2 SET      $62      Tour vert.
COUL   FCB     ESC,COUL0,ESC,COUL1,ESC
        FCB     COUL2,FF,EOT
* Sequence de codes pour mettre l'ecran
* en BLEU sur fond VERT, avec tour VERT
* et effacement (FORM FEED). EOT indi-
* que la fin de la sequence.

      PAGE
START   PSHS     A,B,X,Y,U,DP Sauvegarde.
        JSR      INIT      Initialisation.
* (point d'entree defini dans EQUATES).

        LDA      #DIRECT Page 0.
        TFR      A,DP
NEW     LDX      #COUL      Codes qui mettent
* L'ecran en COUL et effacent tout.

        JSR      DISPL      Routine d'afficha-
* ge d'une sequence de codes.

        LDA      PORTC      Mise en memoire
* Points par mise a 1 du bit 0 du Port C
* (EQUATES).

        ORA      #1
        STA      PORTC
        CLR      ACC          Accumulateur sur
* 16 bits initialise a 0.

        CLR      ACC+1
LOOP    CLRA      Boucle generale de
* parcours d'un ecran complet. Comme A
* vaut 0 D = [A,B] vaut de 0 a 255.

        LDB      ACC+1      L'ensemble de 16
* bits [ACC,ACC+1] va servir de registre
* de calcul de l'adresse dans l'ecran :
* [ACC+1] donne la coordonnee X, et le
* OU exclusif entre ses deux octets don-
* ne la coordonnee Y. C'est l'algorithme
* qui sert a balayer l'ecran de maniere
* pseudo-aleatoire.

```


ADDD #32 Pour faire une
 * image centree : Y (ligne) va de 0 a
 * 199 et X (colonne) va de 32 a 287, c'
 * est un rectangle de 200 * 256.

TFR D,X X = colonne.
 LDB ACC OU exclusif.
 EORB ACC+1
 CMPD #200 Si la resultat est
 * superieur a 199 (ligne maximum), on
 * ignore ce tour et on continue la bou-
 * cle.

BHS SAUTE
 TFR D,Y Y = ligne.
 BSR CALCUL Routine qui trans-
 * forme les coordonnees logiques X et Y
 * en une adresse physique dans la memoir-
 * ecran du T07. Cette adresse est retou-
 * rnee par le registre X qui pointe sur
 * l'octet recherche. Le point dans l'oc-
 * tet est donne par les 3 derniers bits
 * du no de colonne : les 3 LSB de ACC+1
 * transformes en bit par TABIT.

LDY #TABIT Table de transfor-
 * mation de [0,7] en un bit a 1.
 LDA ACC+1 On transforme les
 * 3 bits de droite de l'adresse colonne.

ANDA #7
 LDA A,Y Offset par 3 LSB.
 EORA ,X Seul le bit a 1
 * dans A inversera le bit correspondant
 * de l'adresse [X,Y]. Le OU exclusif
 * avec les autres bits a 0 est sans ef-
 * et.

STA ,X
 SAUTE LDD ACC D = [ACC,ACC+1].
 ADDD CNST La constante CNST
 * fait passer a l'octet suivant : elle
 * s'ajoute a X et definit ainsi l'incra-
 * ment de parcours de l'ecran. Il y a
 * 65536 possibilites.

STD ACC
 LEAU 4,U Frequence de lect-
 * ure du clavier : tous les 65536 / 4
 * tours. Le clavier est lu et si une
 * touche est enfoncee elle modifie la
 * constante CNST, donc le pas de balay-
 * age et le resultat sur l'ecran.

CMPU #4
 BHS LOOP
 JSR KTST Test rapide d'une
 * eventuelle touche enfoncee (EQUATES).

BCC LOOP Si non, continue.
 JSR GETCH Si oui, lire la
 * touche enfoncee (EQUATES).
 TSTB
 BEQ LOOP Elle a ete rela-
 * chee entre temps.

CMPB #CR ENTREE ?
 BEQ OUT Oui, on sort.
 L2 ANDB #\$F Sinon, on ne garde
 * que les 4 bits de droite du code de la
 * touche, on decale CNST (16 bits) de 4
 * bits vers la gauche et on rentre les 4
 * bits nouveaux par la droite.

PSHS B Sauver les 4 bits.
 LDD CNST
 ASLB Decaler 4 fois.
 ROLA
 ASLB
 ROLA
 ASLB
 ROLA
 ASLB
 ROLA
 ASLB
 ROLA
 ADDB ,S+ Ajouter les 4 bits
 STD CNST
 JMP NEW On recommence un
 * nouvel ecran avec la nouvelle valeur
 * de CNST.

```

OUT      PULS    A,B,X,Y,U,DP Sortie.
        SWI      Retour au Moniteur
        PAGE

```

```

*
* Sous programme de conversion d'une ad-
* resse logique dans [X,Y] en une adres-
* se physique retourne par X.

```

```

CALCUL PSHS    A,B,Y
        TFR     Y,D      B = Ligne.
        LDA     #40
        MUL     D = 40 * Ligne.
        ADDD    #STADR
        EXG     D,X      X = Ligne * 40 +
* Adresse de debut de l'ecran, et D =
* Colonne.

```

```

        LSRA
        RORB
        LSRB
        LSRB      D = Colonne / 8.
        LEAX     D,X    + Ligne * 40.
        PULS     A,B,Y,PC Retour au pro-
* gramme appelant avec l'adresse dans X.

```

```

*
* Sous programme affichant une chaine de
* codes pointee par X, terminee par le
* code EOT (Defini dans EQUATES).

```

```

CONT    JSR     PUTC      Point d'entree de
* la routine d'affichage du moniteur T07
* defini dans EQUATES.

```

```

DISPL   LDB     ,X+      B = code pointe
* par X.
        CMPB    #EOT     Fin de la chaine ?
        BNE     CONT     Sinon, affichage.
        RTS
        END      START

```

Programme Type B

```

*
* Programme de balayage aleatoire de
* la memoire couleur.
*
        ORG      ENDMEM-$400 1K Reserve.
DIRECT EQU      *K-8      Page 0 ici.
        SETDP    DIRECT
*
        TITLE    Balayage Ecran.
*
        INCLUD    EQUATES Fichier contenant
* les principales adresses du Moniteur
* du T07.
*

```

Note : en version cassette, INCLUD doit être remplacé par le contenu du fichier EQUATES.

```

CONST   FDB      1      Constante de dep-
* lacement, initialisee a 1, ou les car-
* acteres entres au clavier vont rentrer
* par la droite.

```

```

ACC      RMB      2      Accumulateur de
* calcul.

```

```

TABIT    FCB      $80,$40,$20,$10,$08,$04
        FCB      $02,$01
* Table des bits qui vont etre inverses
* en memoire. C'est un bit a 1 qui occu-
* pe successivement les 8 positions d'un
* octet.

```

```

NOIR     FCB      ESC,$40,ESC,$50,ESC,$60
        FCB      FF,EOT
* Sequence de codes pour mettre l'ecran
* en noir sur fond noir, avec tour noir
* et effacement (FORM FEED). EOT indi-
* que la fin de la sequence.

```

```

        PAGE
START    PSHS     A,B,X,Y,U,DP Sauvegarde.
        JSR      INIT Initialisation.
* (point d'entree defini dans EQUATES).

```

```

        LDA    #DIRECT Page 0.
        TFR    A,DP
NEW      LDX    #NOIR    Codes qui mettent
* L'ecran en noir et effacent tout.

        JSR    DISPL    Routine d'afficha-
* ge d'une sequence de codes.

        LDA    PORTC    Mise en memoire
* couleur par mise a 0 du bit 0 du
* Port C. (EQUATES).

        ANDA    #$FE
        STA    PORTC

        LDX    #STADR    Adresse de debut
* de l'ecran (EQUATES).

        CLR    ACC+1
LOOP     CLRA            Boucle generale de
* parcourir d'un ecran complet. Comme A
* vaut 0 D = [A,B] vaut de 0 a 256.

        LDB    ACC+1    L'ensemble de 16
* bits [ACC,ACC+1] va servir de registre
* de calcul de l'adresse dans l'ecran :
* [ACC+1] donne la coordonnee X, et le
* OU exclusif entre ses deux octets don-
* ne la coordonnee Y. C'est l'algorithme
* qui sert a balayer l'ecran de maniere
* pseudo-aleatoire.

        ADDD    #32      Pour faire une
* image centree : Y (ligne) va de 0 a
* 199 et X (colonne) va de 32 a 287, c'
* est un rectangle de 200 * 256.

        TFR    D,X      X = colonne.
        LDB    ACC      OU exclusif.
        EORB    ACC+1
        CMPD    #200     Si la resultat est
* superieur a 199 (ligne maximum), on
* ignore ce tour et on continue la bou-
* cle.

```

```

        BHS     SAUTE
        TFR     D,Y      Y = ligne.
        BSR     CALCUL    Routine qui trans-
* forme les coordonnees logiques X et Y
* en une adresse physique dans la memoir-
* ecran du T07. Cette adresse est retou-
* rnee par le registre X qui pointe sur
* l'octet recherche. Le point dans l'oc-
* tet est donne par les 3 derniers bits
* du no de colonne : les 3 LSB de ACC+1
* transformes en bit par TABIT.

        LDY     #TABIT    Table de transfor-
* mation de [0,7] en un bit a 1.
        LDA     ACC+1    On transforme les
* 3 bits de droite de l'adresse colonne.

        ANDA    #7
        LDA     A,Y      Offset par 3 LSB.
        EORA    ,X      Seul le bit a 1
* dans A inversera le bit correspondant
* de l'adresse [X,Y]. Le OU exclusif
* avec les autres bits a 0 est sans ef-
* et.

        STA     ,X
SAUTE    LDD     ACC      D = [ACC,ACC+1].
        ADDD    CNST     La constante CNST
* fait passer a l'octet suivant : elle
* s'ajoute a X et definit ainsi l'incree-
* ment de parcours de l'ecran. Il y a
* 65536 possibilites.

        STD     ACC
        LEAU    4,U      Frequence de lect-
* ure du clavier : tous les 65536 / 4
* tours. Le clavier est lu et si une
* touche est enfoncee elle modifie la
* constante CNST, donc le pas de balay-
* age et le resultat sur l'ecran.

        CMPL    #4
        BHS     LOOP

```

```

        JSR     KTST      Test rapide d'une
* eventuelle touche enfoncee (EQUATES).

        BCC     LOOP      Si non, continue.
        JSR     GETCH      Si oui, lire la
* touche enfoncee (EQUATES).

        TSTB
        BEQ     LOOP      Elle a ete relas-
* chee entre temps.

        CMPB    #CR        ENTREE ?
        BEQ     OUT        Oui, on sort.

L2      ANDB    ##F        Sinon, on ne garde
* que les 4 bits de droite du code de la
* touche, on decale CNST (16 bits) de 4
* bits vers la gauche et on rentre les 4
* bits nouveaux par la droite.

        PSHS    B          Sauver les 4 bits.
        LDD     CNST
        ASLB
        ROLA
        ASLB
        ROLA
        ASLB
        ROLA
        ASLB
        ROLA
        ADDB    ,S+        Ajouter les 4 bits
        STD     CNST
        JMP     NEW        On recommence un
* nouvel ecran avec la nouvelle valeur
* de CNST.

OUT     PULS    A,B,X,Y,U,DP Sortie.
        SWI
        PAGE          Retour au Moniteur

```

```

*
* Sous programme de conversion d'une ad-
* resse logique dans [X,Y] en une adres-
* se physique retourne par X.

```

```

CALCUL PSHS    A,B,Y
        TFR     Y,D        E = Ligne.
        LDA     #40
        MUL
        D = 40 * Ligne.
        ADDD    #STADR
        EXG     D,X        X = Ligne * 40 +
* Adresse de debut de l'ecran, et D =
* Colonne.

        LSRA
        RORB
        LSRB
        LSRB        D = Colonne / 8.
        LEAX    D,X        + Ligne * 40.
        PULS    A,B,Y,PC Retour au pro-
* gramme appelant avec l'adresse dans X.

```

```

*
* Sous programme affichant une chaine de
* codes pointee par X, terminee par le
* code EOT (Defini dans EQUATES).

```

```

CONT    JSR     PUTCH      Point d'entree de
* la routine d'affichage du moniteur T07
* defini dans EQUATES.

```

```

DISPL   LDB     ,X+        B = code pointe
* par X.
        CMPB    #EOT        Fin de la chaine ?
        BNE     CONT        Sinon, affichage.
        RTS

        END     START

```

Utilisation des banques mémoires du TO7-70

Le TO7-70 comporte dans sa version de base 48 KO RAM destinée à l'utilisateur. Son champ d'adressage n'est que de 32 KO. Les 16 KO supplémentaires (de A000 à DFFF) sont positionnés en banque. On ne peut donc sélectionner qu'une de ces banques à la fois.

Dans le TO7-70 l'extension mémoire de 64 KO se compose de quatre banques de 16 KO. L'utilisateur dispose donc, en totalité (avec l'extension), de :

16 KO fixes de 6000 à 9FFF

et 6 fois 16 KO commutables de A000 à DFFF

COMMUT :

Permet de sélectionner une banque au choix parmi 6

Entrée : A (numéro de banque de 0 à 5)

Sortie : banque commutée

```
COMMUT EQU      *
        PSHS     D,X,U
        LDU      #$E7C0
        LDB      11,U
        ANDB     #$FB
        STB      11,U
        LDX      #TAB
        LDA      A,X
        STA      9,U
        ORB      #$04
        STB      11,U
        PULS     D,X,U,PC
TAB EQU      *
        FCB      $0F,$17,$E7,$57,$A7,$27
```

CONSEILS BIBLIOGRAPHIQUES

BUI Minh Duc

Programmation en assembleur 6809

Editions EYROLLES, Paris, 1983

DARDANNE Claude

Le microprocesseur 6809 - ses périphériques et le processeur graphique 9365-66

Editions EYROLLES, Paris, 1984

