

Les registres PC et CC sont seuls empilés, c'est à l'utilisateur d'empiler tous ceux qu'il va altérer.

L'adresse de branchement est le vecteur FIRQ situé en \$FFF6:FFF7, c'est-à-dire \$F640. A cet emplacement, se situe un branchement en RAM sur un vecteur situé en \$2067-\$2068. L'intérêt de ce passage par la RAM, est la possibilité de détourner l'interruption pour l'utiliser (voir plus loin).

Les interruptions FIRQ ne sont pas utilisées par le MO5, c'est donc à l'utilisateur de les générer par une extension.

■ *Interruptions normales IRQ*

Elles sont masquables suivant l'état du bit I de CC.

Tous les registres sont empilés.

L'adresse de branchement est le vecteur IRQ situé en \$FFFA-\$FFF9, c'est-à-dire \$F657.

A la fin du traitement de l'interruption se trouve un branchement situé sur un vecteur en \$2064-\$2065.

IRQ sert au clignotement du curseur et à la répétition des touches. La fréquence de IRQ est 50 Hz, c'est-à-dire une interruption toutes les 20 micro-secondes.

IRQ peut être amené à passer par un autre vecteur en RAM lorsqu'il y a répétition ou clignotement; celui-ci est situé en \$2061-2062.

■ *Interruptions non masquables NMI*

Elles sont inutilisées sur MO5 et de plus non vectorisées en RAM, ce qui en interdit l'accès.

■ *Interruptions logicielles SWI*

Elles sont d'une utilisation toute particulière sur MO5, puisqu'elles permettent d'accéder à toutes les routines moniteur et donc aux périphériques (crayon optique, clavier, joystick, écran, cassette, disque).

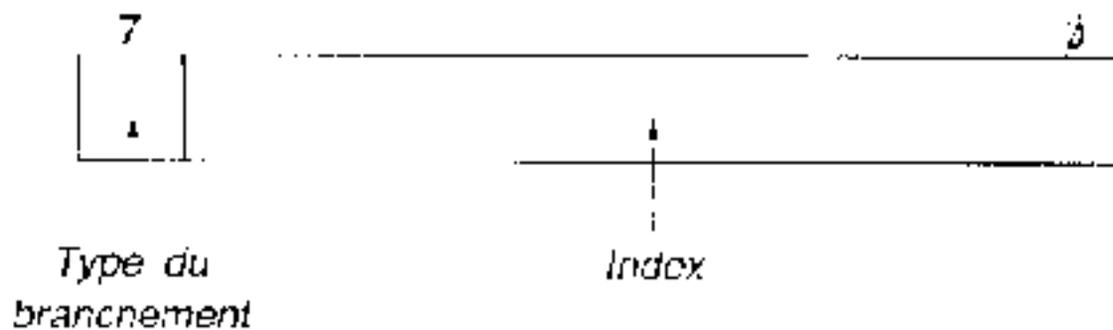
Lors de l'exécution d'un SWI, l'état complet des registres est sauvé dans la pile système. Les interruptions rapides et normales sont invalidées. PC est alors chargé avec l'adresse située en \$FFFA-\$FFFB, c'est-à-dire \$F630.

La routine SWI comporte un LDB [\$0A,S] ce qui signifie que l'on charge dans l'accumulateur B l'octet suivant le SWI. C'est cet octet qui est capital: en effet, sa valeur va permettre une indexation dans une table, ouvrant ainsi la possibilité d'accéder à diverses routines. C'est pour cette raison que sur MO5, SWI est considéré comme ayant un adressage immédiat: SWI #Sn, Sn étant l'index propre à chaque routine.

Les résultats d'une opération sont donnés, dans les divers registres du 6809 (souvent B et CC).

Nous vous indiquons dans la seconde partie, les paramètres d'appel et de retour de chaque routine moniteur.

Intéressons-nous maintenant à la structure de l'octet d'indexation.



Les bits 0 à 6 servent à déterminer l'index, tandis que le bit 7 sert à déterminer le type de branchement.

Si le bit 7 est 0, SWI se comporte comme un JSR (c'est-à-dire qu'il effectue la routine moniteur et continue le déroulement du programme).

Si le bit 7 est à 1, SWI se comporte comme un JMP (c'est-à-dire qu'il effectue la routine moniteur et rend la main au niveau du programme supérieur, c'est-à-dire au programme qui a appelé le sous-programme où se trouve le SWI).

Exemple : SWI #\$00

Instructions suivantes

RTS

Ceci effectue un BEEP puis l'exécution continue (le bit 7 est à 0).

SWI #\$88

Ceci effectue un BEEP puis rend la main au programme d'appel; en l'occurrence le BASIC (le bit 7 est à 1).

La routine swi passe par un vecteur en RAM situé en \$205E-205F. C'est à ce niveau qu'elle peut être détournée.

Attention : il faut remettre à sa valeur initiale le vecteur lors d'un retour en BASIC.

■ Interruptions logicielles SWI 2 et SWI 3

Elles sont inutilisées sur MOS et de plus non vectorisées en RAM, ce qui en interdit l'accès.

■ RESET

↳ A froid

Cette routine correspond à la mise sous tension de l'ordinateur : les variables système sont réinitialisées.

Elle se situe en \$F000.

1) A chaud

Cette routine correspond à un appui sur le bouton de réinitialisation programme. Seuls les pointeurs des variables sont remis à jour.

Elle se situe en \$F003.

Attention : si l'octet \$2200 est différent de \$A5, alors il s'effectue une réinitialisation à froid.

Remarque : la réinitialisation à chaud peut être aussi appelée par un SWI #500.

DÉTOURNEMENT D'UNE INTERRUPTION

Le détournement se fait par la modification du vecteur situé en RAM. Ce vecteur comporte trois octets : les deux premiers contiennent l'adresse de la routine d'interruption, le troisième est un indicateur. Cet indicateur doit être mis à 1 pour indiquer au système que vous avez détourné l'interrupteur ; il doit être mis à 0 lorsque vous remettez l'adresse d'origine.

Votre propre routine de détournement d'interruption devra se terminer par un RTI.

5

Le jeu d'instructions du 6809

Ce long chapitre détaille toutes les instructions que possède le 6809. Elles sont au nombre de 59 et nous avons cru bon de les classer par thèmes pour en faciliter l'assimilation.

Vous trouverez ainsi onze groupes :

- Chargements et stockages
- Echanges et transferts
- Opérations sur la pile
- Incrémentations-décrémentations
- Comparaisons et tests
- Opérateurs booléens
- Opérateurs arithmétiques
- Décalages et rotations
- Branchements inconditionnels et retour
- Branchements conditionnels
- Instructions de traitement des interruptions

Il est indispensable pour bien comprendre une instruction d'avoir assimilé parfaitement les modes d'adressage développés précédemment.

NOTATIONS

Les abréviations sont les suivantes :

- M** : opérande mémoire
- A** : accumulateur A
- B** : accumulateur B
- X** : registre d'index X
- Y** : registre d'index Y

- U** : pointeur de pile utilisateur
- S** : pointeur de pile système
- DP** : registre de page directe DP
- PC** : compteur ordinal
- CC** : registre d'état



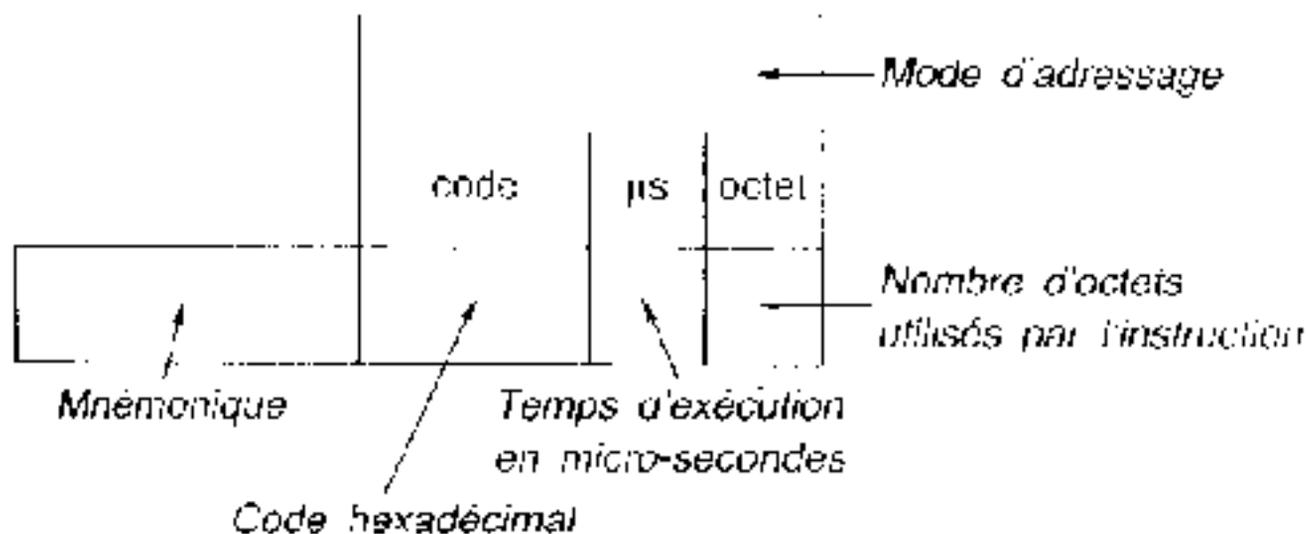
Pour chaque fonction étudiée, nous indiquons ses influences sur le registre d'état CC de la manière suivante :

- rien : pas d'affectation du bit
- le nom du bit : affectation du bit
- 1 : mise à 1 du bit
- 0 : mise à 0 du bit

Les opérations arithmétiques et logiques sont notées :

- I pour l'addition
- S pour la soustraction
- A pour le et logique
- V pour le ou logique
- V pour le ou exclusif

Chaque instruction est résumée en un tableau :



Le « + » pour le temps d'exécution signifie qu'il faut ajouter au temps de base, le temps spécifique au mode d'indexation (voir tableau).

Le « * » indique qu'il faut ajouter une micro-seconde au temps de base lorsque le branchement a lieu.

Le « + » pour le nombre d'octets d'exécution signifie qu'il faut ajouter au nombre de base, le nombre d'octets spécifiques au mode d'indexation (voir tableau).

MODES D'INDEXATION

Comme nous l'avons vu, le mode indexé donne accès à de nombreuses possibilités.

Le sous-mode d'adressage est défini par un post-octet qui se calcule à partir du tableau suivant.

TYPES	FORMES	DIRECT				INDIRECT			
		mnémo- nique	post-octet	μ S	octet	mnémo- nique	post-octet	μ S	octet
déplacement constant en complément à 2	sans déplacement	,H	1RR00100	0	0	[R]	1RR10100	3	0
	5 bits	,R	0RR00000	1	0	inexistent			
	8 bits	,R	1RR01000	1	1	[n,R]	1RR11000	4	1
	16 bits	,R	1RR01001	4	2	[n,R]	1RR11001	7	2
déplacement accumulateur en complément à 2	A	A,R	1RR00110	1	0	[A,R]	1RR10110	4	0
	B	B,R	1RR00101	1	0	[B,R]	1RR10101	4	0
	D	D,R	1RR01011	4	0	[D,R]	1RR11011	7	0
auto-incrémentat ou décrémentat en	+1	,R+	1RR00000	2	0	inexistent			
	+2	,R++	1RR00001	3	0	[R+1]	1RR10001	6	0
	-1	,R-	1RR00010	2	0	inexistent			
	-2	--R	1RR00011	3	0	[--R]	1RR10011	6	0
déplacement constant sur PC	8 bits	,PCR	1XX01100	1	1	[n,PCR]	1XX11100	4	1
	16 bits	,PCR	1XX01101	5	2	[n,PCR]	1XX11101	8	2
étendu indirect	16 bits	inexistent				[n]	10011111	5	2

RR représente le codage du registre :

X = 00

Y = 01

U = 10

S = 11

n représente le déplacement.

XX indique 2 bits indifférents : ils peuvent être quelconques.

Exemples : — déplacement 5 bits : \$8,X
post-octet codé 00000000 soit \$08

— déplacement accu A : A,U
post-octet codé 11000110 soit \$C6

auto-incréméntation : ,S I
 post-octet code 11100000 soit \$F0

— indirect étendu :
 post-octet code 10011111

Remarque : le mode indirect se distingue du mode direct par la présence du bit n° 4 à 1 (sauf pour le déplacement 5 bits).

CHARGEMENTS ET STOCKAGES

Ces fonctions servent à charger les registres du 6809 par des valeurs obtenues soit directement, soit par un adressage quelconque, ou à stocker ces registres selon un procédé analogue.

■ D : chargement d'un registre à partir de la mémoire

	IMMÉDIAT			DIRECT			ÉTENDU			INDEXÉ		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
LDA	8E	2	2	96	4	2	B6	5	3	A6	4+	2+
LDB	C6	2	2	D6	4	2	F6	5	3	E6	4+	2+
LDU	CC	3	3	DC	5	2	FC	6	3	EC	5+	2+
LDS	10CE	4	4	10DE	6	3	10FE	7	4	10EL	6+	3+
LDL	CE	3	3	DE	5	2	FE	6	3	EE	5+	2+
LDX	8F	3	3	9E	5	2	BE	6	3	AE	5+	2+
LDY	08E	4	4	109E	6	3	10BE	7	4	10AE	6+	3+

Registre d'état :

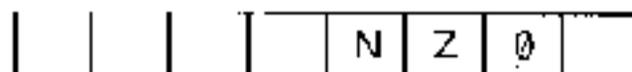
				N	Z	0	
--	--	--	--	---	---	---	--

Exemple : LDA #\$03 → A = 03
 LDX #\$C000 → X = \$C000

- *S* : stockage d'un registre en mémoire

	DIRECT			ETENDU			INDEXE		
	code	us	octet	code	us	octet	code	us	octet
STA	97	4	2	B7	5	3	A7	4	2+
STB	D7	4	2	F7	5	3	E7	4-	2+
STD	DD	5	2	FD	6	3	ED	5-	2+
STS	10DF	6	3	10FF	7	4	10EF	6+	3+
STU	DF	5	2	FF	6	3	EF	5+	2+
SIX	9F	5	2	BF	6	3	AF	5+	2+
STY	109F	6	3	10BF	7	4	10AF	6+	3+

Registre d'état :

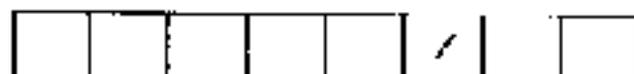


Exemple . A = \$03 STA \$6000 ; \$6000 = 03
 X = \$C000 STX \$7000 ; \$7000 = \$C000

- LEA : chargement d'une adresse effective

	INDEXE		
	code	us	octet
LEAS	32	4+	2+
LEAU	33	4+	2-
LEAX	30	4-	2-
LEAY	31	4-	2+

Registre d'état : — n'est pas affecté par LEAU et LEAS
 — affecté par LEAX et LEAY



Remarque : LEA est une fonction très pratique et complémentaire, car elle remplace les incréments, additions et soustractions inexistantes pour les registres d'index.

Exemple : LEAX 1,X → X = X + 1
 LEAY -1,Y → Y = Y - 1
 LEAX A,U → X = U + A

■ CLR : mise à 0 registre ou mémoire

	IMMÉDIAT			DIRECT			ÉTENDU			INDEXÉ		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
CLH				0F	6	2	7F	7	3	6F	6-	2+
CLRA	4F	2	1									
CLRB	5F	2	1									

Registre d'état

				0	1	0	0
--	--	--	--	---	---	---	---

Exemple : CLRA → A = 00
 CLR \$6000 → \$6000 = 00

ÉCHANGES ET TRANSFERTS

Ces instructions permettent l'échange ou le transfert de données entre registres. Elles peuvent servir par exemple à la conservation de données pour comparaisons postérieures.

Ces instructions n'affectent pas le registre d'état sauf lorsque DG est échangé ou transféré. Les échanges ou transferts ne peuvent se faire que sur des registres de même longueur (8 ou 16 bits).

■ EXG : échange de deux registres

	IMMÉDIAT		
	code	µs	octet
EXG	1E	8	2

Le deuxième octet, post-octet, détermine les deux registres de l'échange, chacun étant codé sur un quartet.

Post-octet

Registre 1	Registre 2
------------	------------

Voici la valeur du quartet pour chacun des dix registres :

Registre	Code
A	8
B	9
CC	A
D	0
DP	B
PC	5
S	4
U	3
X	1
Y	2

Exemple : EXG A R codé 1F 89
 EXG S U codé 1F 43
 EXG D,Y codé 1E 02

- TFR : transfert d'un registre vers un autre

	IMMEDIAT		
	code	ns	octet
TFR	1F	6	2

Le codage du post-octet est le même que la fonction EXG (précédente).

Exemple : TFR A,B codé 1F 89
 TFR B,A codé 1F 98

OPÉRATIONS SUR LA PILE

Ces instructions empilent ou depilent un, plusieurs ou la totalité des registres du CPU.

Le codage des registres se fait sur le post-octet.

7	6	5	4	3	2	1	0	N° du bit
PC	U/S	Y	X	DP	B	A	CC	Registre

CC,A,B,DP est codé 0F
 A,B,X,Y est codé 36
 CC,A,B,DP,X,Y,U,PC est codé FF

- L'index U peut être stocké dans la pile système, alors que l'index S peut être stocké dans la pile utilisateur.
- Le temps d'exécution se calcule en ajoutant à la valeur de base 5 μ s, un pour les registres huit bits et deux pour les seize bits.

■ PSH : empilement de registres

	IMMEDIAT		
	code	μ s	octet
PSHS	34	5+	2
PSHU	36	5+	2

L'ordre d'empilement est PC \rightarrow CC.

Registre d'état : non affecté.

Exemple : PSHS U,X,B,A \rightarrow 34 56

PSHU S,X,B,A \rightarrow 36 56

■ PUL : dépilement de registres

	IMMEDIAT		
	code	μ s	octet
PULS	35	5-	2
PULU	37	5-	2

L'ordre de dépilement est CC \leftarrow PC.

Registre d'état : non affecté, sauf quand CC est lui-même dépilé.

Exemple : PULS CC,A,B,DF \rightarrow 35 0F

PULU X,Y,S,PC \rightarrow 37 F0

INCRÉMENTATIONS — DÉCRÉMENTATIONS

Ces instructions permettent l'incréméntation ou la décrémentation de 1, d'un registre ou d'une mémoire. Elles peuvent servir à diverses tâches : compter des événements, faire des boucles...

- INC : *incrémentation d'un registre ou d'une mémoire*

	INHERENT			ETENDU			DIRECT			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
INC				7C	7	3	0C	6	2	6C	6+	2+
INCA	4C	2	1									
INCB	5C	2	1									

Registre d'état :

				N	Z	V	
--	--	--	--	---	---	---	--

Exemple : A = 02 INCA → A = 03
 \$6000 → 08 INC \$6000 → \$6000 = 09

- DEC : *décrémentation d'un registre ou d'une mémoire*

	INHERENT			ETENDU			DIRECT			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
DEC				7A	7	3	0A	6	2	6A	6+	2+
DECA	4A	2	1									
DECB	5A	2	1									

Registre d'état :

				N	Z	V	
--	--	--	--	---	---	---	--

Exemple : A = 02 DECA → A = 01
 \$6000 = 08 DEC \$ 6000 → \$6000 = 07

COMPARAISONS ET TESTS

Ces instruments permettent de positionner le registre d'état CC en fonction de la valeur d'un registre ou d'une position mémoire. Elles n'altèrent en aucun cas les registres ou positions mémoires testées.

■ BIT : test de bit

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
BITA	85	2	2	95	4	2	B5	5	3	A5	4+	2+
BITB	C5	2	2	D5	4	2	F5	5	3	E5	4+	2+

Remarque : un ET logique est effectué entre le contenu d'un accumulateur et le contenu de l'opérande mémoire spécifiée. Z = (R) \wedge (M)

Registre d'état :

				N	Z	\emptyset
--	--	--	--	---	---	-------------

Exemple : A = \$D0 \rightarrow BITA #541 : Z = 0
 B = \$F0 \rightarrow BITB #5F0 : Z = 1

■ CMP : comparaison entre registre et contenu mémoire

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
CMPA	81	2	2	91	4	2	B1	5	3	A1	4+	2+
CMPB	C1	2	2	D1	4	2	F1	5	3	E1	4+	2+
CMPD	1083	5	4	1093	7	3	10B3	8	4	10A3	7+	3+
CMP5	118C	5	4	119C	7	3	11BC	8	4	11AC	7+	3+
CMPU	1183	5	4	1193	7	3	11B3	8	4	11A3	7+	3+
CMPX	8C	4	3	9C	6	2	BC	7	3	AC	6+	2+
CMPY	108C	5	4	109C	7	3	10BC	8	4	10AC	7+	3+

Registre d'état :

				N	Z	V	C
--	--	--	--	---	---	---	---

■ TST *test d'un registre ou contenu mémoire*

	IMMÉDIAT			DIRECT			ÉTENDU			INDEXÉ		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
TST				0D	6	2	7D	7	3	6D	6+	2-
TSTA	4D	2	1									
TSTB	5D	2	1									

Registre d'état :

				N	Z	0
--	--	--	--	---	---	---

Remarque : TST positionne N et Z en fonction du registre ou du contenu mémoire testé.

Exemple : A = \$12 TSTA → N = 0 et Z = 0
 A = \$00 TSTA → N = 0 et Z = 1
 A = \$80 TSTA → N = 1 et Z = 0

OPÉRATEURS BOOLIENS

■ AND : *ET logique entre mémoire et registre*

	IMMÉDIAT			DIRECT			ÉTENDU			INDEXÉ		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
ANDA	94	2	2	94	4	2	94	5	3	A4	4+	2-
ANDB	C4	2	2	D4	4	2	F4	5	3	E4	4+	2-
ANDCC	1C	3	2									

Registre d'état : pour ANDA/ANDB

				N	Z	0
--	--	--	--	---	---	---

pour ANDCC

E	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

Table de vérité :

A	0	1
0	0	0
1	0	1

Exemple : A - \$FF ANDA #\$7F → A - \$7F
 B - \$EE ANDB #\$7F → B - \$6E

■ OR : ou logique entre mémoire et registre

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
ORA	8A	2	2	9A	4	2	DA	5	3	AA	4+	2+
ORR	CA	2	2	DA	4	2	FA	5	3	EA	4+	2+
ORCC	1A	3	2									

Registre d'état :

				N	Z	0	
--	--	--	--	---	---	---	--

pour ORCC

F	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

Table de vérité :

V	0	1
C	0	1
I		1

Exemple : A - \$63 ANDA # \$10 → A = \$73
 B - \$55 ANDB # \$AA → B - \$FF

■ EOR : ou exclusif entre mémoire et registre

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	µs	octet	code	µs	octet	code	µs	octet	code	µs	octet
EORA	88	2	2	98	4	2	B8	5	3	A8	4+	2+
EORB	C8	2	2	D8	4	2	F8	5	3	E8	4+	2+

Registre d'état :

				N	Z	0	
--	--	--	--	---	---	---	--

Table de vérité :

V	0	1
0	0	1
1	1	0

Exemple : $A = \$9C \text{ EORA } \#\$8A \rightarrow A = \$15$

■ COM : complémentation à 1

	INHERENT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
COM				03	6	2	73	7	3	63	6+	2+
COMA	43	2	1									
COMB	53	2	1									

Registre d'état :

				N	Z	0	1
--	--	--	--	---	---	---	---

Exemple : $A = \$10 \text{ COMA} \rightarrow A = \EF
 $B = \$00 \text{ COM} \rightarrow B = \FF

■ NEG : complémentation à 2

	INHERENT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
NEG				00	6	2	70	7	3	60	6+	2+
NEGA	40	2	1									
NFGR	50	2	1									

Registre d'état :

				N	Z	V	C
--	--	--	--	---	---	---	---

Exemple : $A = \$10 \text{ NEGA} \rightarrow A = \$F0$
 $A = \$00 \text{ NEGA} \rightarrow A = \00

OPÉRATEURS ARITHMÉTIQUES

Le 6809 est très puissant au point de vue arithmétique, notamment en ce qui concerne les opérations avec ou sans retenue et la multiplication 8 bits.

- **ABX** : addition non signée de B et X

$$(B) + (X) \rightarrow (X)$$

	INHERENT		
	code	μs	octet
ABX	3A	3	1

Registre d'état : non affecté.

Exemple : B = \$20 X = \$C000
 ABX \rightarrow X = \$C020

- **ADC** : addition de la mémoire à l'accumulateur avec retenue

$$(A \text{ ou } B) + (M) + (C) \rightarrow (A \text{ ou } B)$$

	IMMÉDIAI			DIRECT			ÉTENDU			INDEXÉ		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
ADCA	89	2	2	99	4	2	B9	5	3	A9	4+	2+
ADCB	C9	2	2	D9	4	2	F9	5	3	F9	4+	2+

Registre d'état :

			H		N	Z	V	C
--	--	--	---	--	---	---	---	---

Exemple : A = \$E4 ADCA # \$1F \rightarrow A = 03
 C = 0
 B = \$05 ADCB # \$20 \rightarrow B = \$26
 C = 1

- **ADD** : addition de la mémoire à l'accumulateur sans retenue

$$(A, B \text{ ou } D) + (M) \rightarrow (A, B \text{ ou } D)$$

	IMMÉDIAI			DIRECT			ÉTENDU			INDEXÉ		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
ADDA	8B	2	2	9B	4	2	BB	5	3	AB	4+	2+
ADDB	CB	2	2	DB	4	2	FB	5	3	EB	4+	2+
ADD	C3	4	3	D3	6	2	-3	7	3	E3	6+	2+

Registre d'état : pour ADDA et ADDB

	H		N	Z	V	C
--	---	--	---	---	---	---

pour ADDD

			N	Z	V	C
--	--	--	---	---	---	---

Exemple : A = \$E4 ADDA #\$1F → A = \$03
 B = \$05 ADDB #\$20 → B = \$25

■ DAA : ajustement décimal de l'accumulateur A

Cette instruction ne fonctionne qu'après une addition 8 bits.

Un facteur de correction est ajouté à chaque quartet pour rétablir le résultat d'une opération en DCB. 6 est ajouté au quartet de poids faible si la demi-retenue H est positionnée ou si le quartet a une valeur strictement supérieure à 9. 6 est ajouté au quartet de poids si la retenue C est positionnée ou si le quartet a une valeur strictement supérieure à 9.

INHERENT			
	code	µs	octet
DAA	19	?	-

Registre d'état :

				N	Z	Ø	C
--	--	--	--	---	---	---	---

Remarque : V n'a aucune signification ici, il en est de même pour H.

Exemple : A = \$06 en DCB

ADDA #\$64 → A = \$6A

DAA → A = \$70

A = \$70 en DCB

ADDA #\$44 → A = \$B4

DAA → A = \$14

■ MUL : multiplication de A par B (non signée)

(A) × (B) → (D)

INHERENT			
	code	μs	octet
MUL	3D	11	1

Registre d'état :

				0	Z	C
--	--	--	--	---	---	---

Exemple : A = \$0A R = \$20
 MUL D = \$0140
 A = \$03 B = \$06
 MUL → D = \$0012

- SBC : soustraction de la mémoire à l'accumulateur avec emprunt (retenue)

(A ou B) - (M) - (C) → (A ou B)

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
SBCA	82	2	2	92	4	2	B2	5	3	A2	4	2+
SBCB	C2	2	2	D2	4	2	F2	5	3	E2	4	2+

Registre d'état :

		?	N	Z	V	C
--	--	---	---	---	---	---

Exemple : A = \$FC SBCA #\$F0 → A = \$0B
 C = 1
 B = \$20 SBCB #\$10 → B = \$10
 C = 0

- SUB : soustraction de la mémoire à l'accumulateur sans emprunt (retenue)

(A, B ou D) - (M) → (A, B ou D)

	IMMEDIAT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
SUBA	80	2	2	90	4	2	B0	5	3	A0	4+	2+
SUBB	C0	2	2	D0	4	2	F0	5	3	E0	4+	2+
SUBD	83	4	3	93	5	2	B3	7	3	A3	6+	2+

Registre d'état : pour SUBA et SUBB



pour SUBD



Exemple : A = \$FC SUBA #\$F0 → A = \$0C
 B = \$20 SUBB #\$10 → B = \$10

■ **SEX** : extension de signe de B sur A

Cette instruction a pour but de transformer un nombre en complément à deux dans B, en un nombre en complément à deux dans D.

A est mis à \$FF si B est négatif (bit 7 à 1).

A est mis à \$00 si B est positif (bit 7 à 0).

	INHERENT		
	code	us	octet
SEX	1D	2	1

Registre d'état :



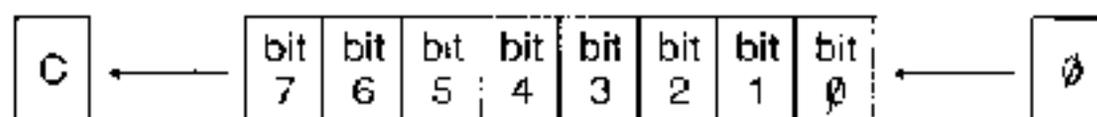
Exemple : B = \$10 SEX → D = \$0010
 B = \$84 SEX → D = \$FF84

DÉCALAGES ET ROTATIONS

Ces fonctions effectuent des déplacements de bits à l'intérieur d'une zone mémoire ou d'un registre. Leur utilisation est très courante dans de nombreux algorithmes : multiplications, divisions...

■ **ASL** : décalage arithmétique à gauche

Cette instruction remplace le bit 1 par un 0, le bit 2 par l'ex-bit 1... jusqu'à transférer l'ex-bit 8 dans le carry. Cette opération est donc en fait une multiplication par deux.



	INHERENT			DIRECT			ETENDU			INDEXL		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
ASI				08	5	2	78	1	3	68	6-	2-
ASLA	48	2	1									
ASLB	58	2	1									

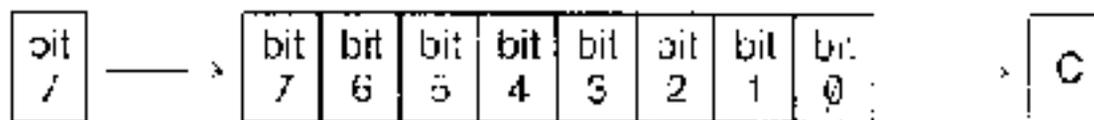
Registre d'état

		?		N	Z	V	C
--	--	---	--	---	---	---	---

Exemple : A = \$A9 A = %10101001
 ASLA A = %01010010 et C = 1
 A = \$52

■ ASR : *décalage arithmétique à droite*

Identique à ASL mais le décalage s'effectue vers la droite. C prend la valeur de l'ex-bit 0, et le bit 7 prend la valeur de l'ex-bit 7.



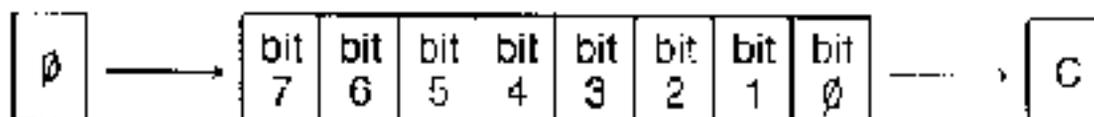
	INHERENT			DIRECT			ETENDU			INDEXL		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
ASR				7	6	2	11	1	3	67	6+	2+
ASRA	47	2	1									
ASRB	57	2	1									

Registre d'état :

		?		N	Z	V	C
--	--	---	--	---	---	---	---

■ LSR : *décalage logique à droite*

Identique à ASR, mais le bit 7 prend pour valeur 0.



	INHERENT			DIRECT			ETENDU			INDEXE		
	code	μ s	octet	code	μ s	octet	code	μ s	octet	code	μ s	octet
LSR				04	6	2	74	7	3	64	6+	2+
LSRA	44	2	1									
LSRB	54	2	1									

Registre d'état :



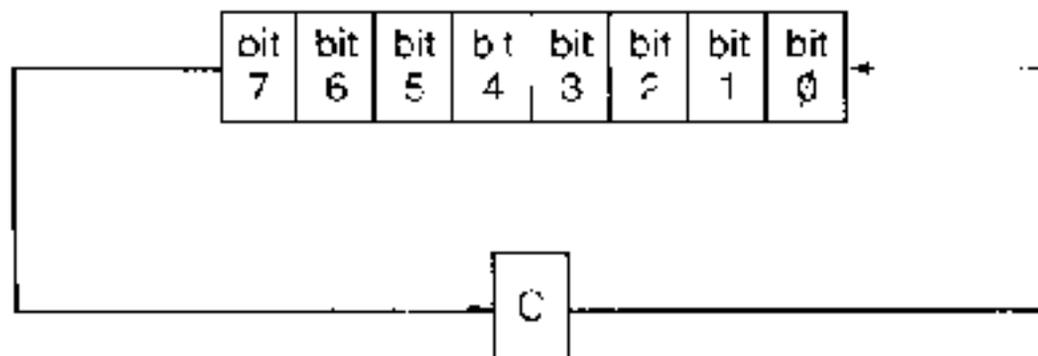
Exemple : A = \$A / A = % 10100111

LSRA A = % 01010011 et C = 1

A = \$53

■ ROL : rotation à gauche via la retenue

Le bit 0 prend la valeur de l'ex-bit 7, le bit 1 prend la valeur de l'ex-bit 0... et le bit 7 prend la valeur de l'ex-bit 6. C prend la valeur de l'ex-bit 7.



	INHERENT			DIRECT			ETENDU			INDEXE		
	code	μ s	octet	code	μ s	octet	code	μ s	octet	code	μ s	octet
ROL				09	6	2	79	7	3	69	6+	2+
ROLA	49	2	1									
ROLB	59	2	1									

Registre d'état :



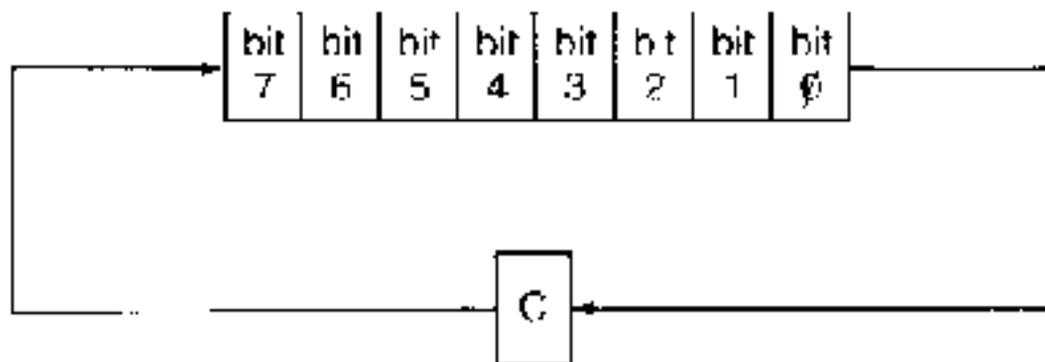
Exemple : A = \$83 = % 10000011 et C = 1

ROLA

A = % 00000111 = \$07 et C = 1

■ ROR : rotation à droite via la retenue

Identique à ROL, mais la rotation se fait à droite.



	INHERENT			DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet	code	μs	octet
ROR				06	6	2	76	7	3	66	6+	2-
RORA	46	2	1									
RORB	56	2	1									

Registre d'état :

				N	Z		C
--	--	--	--	---	---	--	---

Exemple : A = \$81 = %10000001 et C = 1

RORA

A = %11000100 = \$E4 et C = 1

BRANCHEMENTS INCONDITIONNELS ET RETOUR

- **BRA** : *branchement à une nouvelle adresse*

Cette instruction effectue un branchement relatif pour passer à une nouvelle adresse.

	BRANCHEMENT		
	code	μs	octet
BRA	20	3	2
LBRA	16	5	3

Registre d'état : non affecté.

- **JMP** : *saut à une nouvelle adresse*

Cette instruction effectue un branchement absolu pour passer à une nouvelle adresse.

	DIRECT			ETENDU			INDEXE		
	code	μs	octet	code	μs	octet	code	μs	octet
JMP	0F	3	2	7F	4	3	6E	3+	2+

Registre d'état : non affecté.

■ **BSR** : *branchement à un sous-programme*

Cette instruction effectue un saut en mémorisant en pile l'adresse de l'instruction qui suit le BSR (adresse de retour). C'est le pendant du GOSUB BASIC.

BRANCHEMENT			
	<i>code</i>	<i>µs</i>	<i>octet</i>
BSR	87	7	2
LBSR	17	9	3

Registre d'état : non affecté.

■ **JSR** : *saut à un sous-programme*

identique à BSR, mais le branchement est absolu

	DIRECT			EXTENDU			INDEXE		
	<i>code</i>	<i>µs</i>	<i>octet</i>	<i>code</i>	<i>µs</i>	<i>octet</i>	<i>code</i>	<i>µs</i>	<i>octet</i>
JSR	9D	7	2	BD	6	3	AD	7	2

Registre d'état : non affecté.

■ **BRN** : *aucun branchement*

Cette instruction n'effectue aucun branchement. BRN remplace 2 NOP, LBRN remplace 4 NOP

BRANCHEMENT			
	<i>code</i>	<i>µs</i>	<i>octet</i>
BRN	21	3	2
LBRN	1021	5	4

Registre d'état : non affecté.

■ **NOP** : *pas d'opération*

Comme son nom l'indique, cette instruction ne fait rien. Elle peut servir pour faire une temporisation ou pour remplacer des instructions devenues inutiles. Elle peut également servir pour la mise au point de programmes. Les instructions NOP seront enlevées une fois le programme débogué.

INSTRUCTION			
	code	μ s	octet
NOP	12	2	1

Registre d'état : non affecté

■ RTS : retour de sous programme

Cette instruction dépile les deux octets du sommet de la pile dans le compteur ordinal. Elle permet le retour à la suite du programme après l'exécution d'un JSR. Elle permet aussi de rendre la main au BASIC à la fin d'un programme.

INSTRUCTION			
	code	μ s	octet
RTS	99	5	1

Registre d'état : non affecté.

Remarque : lorsqu'un RTS est fait, il faut que la pile soit dans l'état où elle était lorsque l'adresse de retour a été empilée : n'oubliez donc pas de dupliquer tout ce qui a été empilé.

BRANCHEMENTS CONDITIONNELS

Ces instructions permettent d'effectuer des branchements selon l'état de chaque bit du registre CC. On utilise ces branchements après des instructions de tests ou comparaisons.

Note : l'étoile après le temps en micro-secondes signifie qu'il faut rajouter une micro-seconde quand le branchement s'effectue.

■ BCC : branchement si la retenue est à zéro

BRANCHEMENT			
	code	μ s	octet
BCC	24	3	2
LBCC	1024	5*	4

Condition : C = 0

Registre d'état : non affecté.

- BCS : *branchement si la retenue est à un*

BRANCHEMENT			
	code	μs	octet
BCS	25	3	2
lBCS	1025	5	4

Condition : C = 1

Registre d'état : non affecté.

- BEQ : *branchement si l'égalité se fait*

BRANCHEMENT			
	code	μs	octet
BEQ	27	3	2
lBEQ	1027	5	4

Condition : Z = 1

Registre d'état : non affecté.

- BNE : *branchement si différent*

BRANCHEMENT			
	code	μs	octet
BNE	26	3	2
lBNE	1026	5	4

Condition : Z = 0

Registre d'état : non affecté.

- BGE : *branchement si supérieur ou égal (signé)*

BRANCHEMENT			
	code	us	octet
BGF	2C	3	2
LBGE	102C	5*	4

Condition : $N \vee V = 0$

Registre d'état : non affecté.

- BLE : *branchement si inférieur ou égal (signé)*

BRANCHEMENT			
	code	us	octet
BLE	2F	3	2
LBLLE	102F	5*	4

Condition : $(N \vee V) + Z = 1$

Registre d'état : non affecté.

- BHS : *branchement si supérieur ou égal (non signé)*

BRANCHEMENT			
	code	us	octet
BHS	24	3	2
LBHS	1024	5*	4

Condition : $C = 0$

Registre d'état : non affecté.

Remarque : BHS est équivalent à BCC.

- **BLS** : *branchement si inférieur ou égal (non signé)*

	BRANCHEMENT		
	code	μs	octet
BLS	23	3	2
LBLS	1023	5'	4

Condition : $C = 1$

Registre d'état : non affecté

- **BGT** : *branchement si strictement supérieur (signé)*

	BRANCHEMENT		
	code	μs	octet
BGT	2E	3	2
LBGT	102E	5'	4

Condition : $(N \vee V) = Z = 0$

Registre d'état : non affecté

- **BLT** : *branchement si strictement inférieur (signé)*

	BRANCHEMENT		
	code	μs	octet
BLT	2D	3	2
LBLT	102D	5'	4

Condition : $N \vee V = 1$

Registre d'état : non affecté.

- BHI : *branchement si strictement supérieur (non signé)*

	BRANCHEMENT		
	code	μs	octet
BHI	22	3	2
LBHI	1022	5	4

Condition : $C = Z = 0$

Registre d'état : non affecté.

- BLO : *branchement si strictement inférieur (non signé)*

	BRANCHEMENT		
	code	μs	octet
BLO	25	3	2
LBLO	1025	5	4

Condition : $C = 1$

Registre d'état : non affecté.

Remarque : BLO est équivalent à BCS

- BMI : *branchement si négatif*

	BRANCHEMENT		
	code	μs	octet
BMI	2B	3	2
LBMI	102B	5	4

Condition : $N = 1$

Registre d'état : non affecté.

- BPL : *branchement si positif*

BRANCHEMENT			
	code	μs	octet
BPL	2A	3	2
LBPL	102A	5*	4

Condition : $N = \emptyset$

Registre d'état : non affecté.

- BVC : *branchement si pas de débordement*

BRANCHEMENT			
	code	μs	octet
BVC	28	3	2
LBVC	1028	5*	4

Condition : $V = \emptyset$

Registre d'état : non affecté.

- BVS : *branchement si débordement*

BRANCHEMENT			
	code	μs	octet
BVS	29	3	2
LBVS	1029	5*	4

Condition : $V = 1$

Registre d'état : non affecté.

INSTRUCTIONS DE TRAITEMENT DES INTERRUPTIONS

Ces instructions concernent toutes le traitement des interruptions. Pour plus de précisions, veuillez consulter le chapitre « *Interruptions* ».

- **CWAI** : *ET logique entre CC et l'opérande immédiat, puis attente d'interruption*

(CC) A (M) → (CC)

L'exécution du programme est suspendue jusqu'à la première interruption rencontrée (IRQ, FIRQ ou NMI).

IMMEDIAT			
	code	µs	octet
CWAI	3C	20	2

Registre d'état :

F	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

Remarque : CWAI empile tous les registres et met le bit F à 1 ; une interruption FIRQ peut donc être traitée sans sauvegarder aucun registre.

- **SYNC** : *synchronisation avec la ligne d'interruption*

L'exécution du programme est suspendue jusqu'à la première interruption rencontrée (IRQ, FIRQ ou NMI). Si le signal d'interruption dure plus de 3 µs, l'interruption est générée, sinon le programme normal continue.

INHERENT			
	code	µs	octet
SYNC	13	4	1

Registre d'état : non affecté.

- **SWI** : *interruption logicielle*

L'ensemble des registres est sauvegardé dans la pile système et un appel est fait à la routine de traitement.

IMMEDIAT (sur MO5)			
	code	μs	octet
SWI	3F	10	1
SWI 2	03F	20	2
SWI 3	13F	20	2

Registre d'état : non affecté.

Remarque : l'instruction SWI a une fonction toute particulière puisque sur MO5 elle est paramétrée et sert de point d'entrée aux routines moniteur. Consultez à ce effet le chapitre « Interruptions ».

■ RTI : retour d'interruption

Cette instruction dépile les registres qui ont été empilés lors d'une interruption.

CG est d'abord dépilé. Si le bit E est à 0, seul PC est dépilé, sinon tous les autres registres sont dépilés.

INHERENT			
	code	μs	octet
RTI	3B	$\frac{6}{15}$	1

Remarque : si E = 0 alors l'instruction dure 6 μs , sinon elle dure 15 μs .

Registre d'état :

E	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

ROUTINES ET
ADRESSES UTILES | **2**

6

L'écran et le crayon optique

LE SYSTÈME ÉCRAN

L'écran du MO5 est composé de 200×320 points, soit en tout 64000 points. Pour éviter de consacrer un octet à chacun de ces points, ils ont été assemblés par paquets de huit constituant alors un octet. La mémoire écran (les points) utilise donc 8000 octets soit \$1F40 en hexadécimal (et non pas \$1FFF comme le prétend la notice d'utilisation du MO5). Cependant, ceci ne suffit pas : en effet, avec seulement 8000 octets, on peut définir chaque point comme étant allumé ou éteint mais on ne peut définir sa couleur. C'est pourquoi 8000 autres octets sont alloués à la définition de la couleur d'encre et de fond de chaque octet. Il faut donc en tout 16000 octets pour l'écran. Utiliser les adresses de 0 à 16000 aurait pris trop de place et aurait limité la mémoire utilisateur. C'est pourquoi le fabricant a eu recours à une « ruse » : l'état des points de l'écran est logé de 0000 à \$1F40 et l'état des couleurs de 0000 à \$7F40.

Comment peut-on loger deux choses différentes aux mêmes adresses ?

Ceci est réalisé grâce au circuit d'entrée-sortie : le bit 0 (bit de poids faible) de l'adresse \$A7C0 sert à diriger les données sur les octets d'état de points ou sur les octets de couleur des points (ceci permet de n'occuper que 1F40 octets d'adresse sur le microprocesseur et d'avoir une image écran qui correspond au double de place).

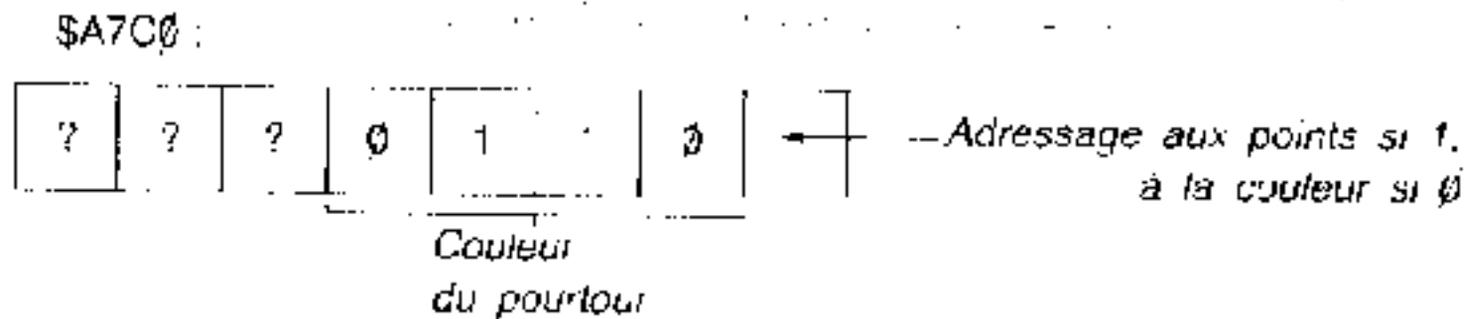
Pour vous en convaincre, tapez ce qui suit : appuyez tout d'abord sur le petit bouton blanc (Initial. prog.) puis tapez POKE 4000,1 : vous voyez alors un petit point s'allumer sur l'écran (le point est bleu foncé). A présent, tapez POKE &HA/C0,140 : POKE 4000,1 : le point est alors noir sur un fond rouge.

La règle est donc la suivante : si le bit de poids faible de \$A7C0 est à 1, on agit sur les points de l'écran. Si ce même bit est à 0, on agit sur la couleur selon la règle :

quartet de poids fort pour l'encre,
quartet de poids faible pour le fond

(dans notre exemple, le 1 correspond à 01, soit une encre noire (0) et un fond rouge (1)).

En fait, l'adresse \$A70 sert à définir ce qui vient d'être expliqué mais sert aussi à définir la couleur du pourtour de l'écran, l'organisation est la suivante :



(40 s'écrit 10001100, ce qui explique que la couleur du pourtour reste cyan (code 6))

LES COULEURS DE FOND ET D'ENCRE

Comme nous venons de le voir, les bits 1, 2, 3, 4, de la variable système située en \$A7C0, nous donnent la couleur du pourtour de l'écran.

Pour vous en convaincre, tapez POKE &HA7C0, 6 et vous verrez le pourtour se teinter en jaune ; en effet, 6 s'écrit 00000110 en binaire : les bits 1, 2, 3, 4 nous donnent 0011 soit 3 en décimal ce qui représente la couleur jaune.

Une autre adresse sert à définir la couleur de l'encre et du fond de ce qui s'imprime à l'écran ; il s'agit de l'adresse \$202B (le nombre des couleurs étant de 16, il suffit de 4 bits pour coder la couleur de l'encre et de 4 bits pour coder la couleur du fond, soit 1 octet au total). A l'adresse \$202B, le quartet de poids fort définit la couleur de l'encre et le quartet de poids faible celle du fond ; ainsi si vous faites POKE &H202B, 18, vous obtenez une encre rouge sur fond vert : 18 = 0001 0010

Rouge 1 2 Vert

(d'après les codes des couleurs que nous vous rappelons) :

0 : noir	8 : gris (= noir clair)
1 : rouge	9 : rouge clair
2 : vert	10 : vert clair
3 : jaune	11 : jaune clair
4 : bleu	12 : bleu clair
5 : magenta	13 : magenta clair
6 : cyan	14 : cyan clair
7 : blanc	15 : orange

A noter que pour une couleur donnée du type 0ppp (en binaire), la couleur claire correspondante est alors codée 1ppp (sauf pour le blanc pour lequel blanc d'air n'a pas de sens et que l'on remplace donc par orange).

Il est donc inutile de vouloir retenir les codes de toutes les couleurs ; il suffit de connaître les codes des couleurs foncées. On obtient alors les codes des couleurs claires en ajoutant 8 (mise à 1 du bit 3).

Vous pouvez cependant objecter que notre POKE &H202B,18 est équivalent à un COLOR 1,2 et par conséquent ne présente pas d'intérêt. Si cela est vrai en BASIC, cela devient faux en langage machine car au niveau assembleur, notre POKE &H202B,18 est équivalent à

```
LDA #$12
STA $202B
```

alors que l'accès à la fonction COLOR est difficile et de plus saugrenu car nécessitant beaucoup plus de temps.

LE CURSEUR

Sur l'écran, le curseur clignote sans cesse à un certain endroit ; cet endroit peut être défini par une coordonnée horizontale (comprise entre 1 et 40) et par une verticale (comprise entre 0 et 24). Deux adresses se chargent de cela : \$201B et \$201C

```
$201B : ligne courante du curseur,
$201C : colonne courante du curseur.
```

Ces deux adresses peuvent servir, lors de diverses applications à déplacer le curseur par exemple (avec autre chose que LOCATE inaccessible en assembleur)

LE CARACTÈRE D'ÉCHAPPEMENT (ESC)

Le caractère ESC (code ASCII 27) est un caractère de contrôle qui permet bien des manipulations quant à l'écran du MO5.

Il permet d'abord de manipuler les couleurs. Tapez par exemple (en mode direct) : ?CHR\$(27);CHR\$(&H62); le pourtour devient alors vert. On peut ainsi modifier grâce à la même méthode, l'encre et le fond.

Les codes sont les suivants :

4 bits forts

- 4 . . . → couleur encre
- 5 —→ couleur fond
- 6 —→ couleur purlouf

4 bits faibles

- 0 —→ noir
- 1 —→ rouge
- 2 —→ vert
- 3 —→ jaune
- 4 —→ bleu
- 5 . . . → magenta
- 6 —→ cyan
- 7 —→ blanc
- 8 —→ gris
- 9 —→ rouge clair
- A —→ vert clair
- H —→ jaune clair
- C . . . → bleu clair
- D —→ magenta clair
- E —→ cyan
- F —→ orange

Ainsi, si l'on veut passer en encre rouge, on tape ?CHR\$(27); CHR\$(&H41). Ceci peut paraître inintéressant en BASIC car accessible par la fonction COLOR mais est très intéressant en langage machine car le passage de deux paramètres est alors aisé tandis que l'accès à la fonction COLOR aurait été long et lourd.

Mais il existe d'autres attributs que ceux des couleurs : en effet, il existe aussi des tailles de caractères, des scrolls et de l'inversion vidéo.

Essayez l'exemple suivant : chargez en mémoire un petit programme BASIC permettant de faire un listing, tapez alors : LIST. Le listing s'effectue mais le scroll paraît saccadé. Tapez alors : ?CHR\$(27); CHR\$(&H75) puis faites LIST. L'impression de cascade a disparu. Mieux, tapez alors : ?CHR\$(27); CHR\$(&H79), puis faites LIST. Le listing se scroll e alors très lentement (utile pour une présentation!).

Ces attributs ont les codes suivants :

- \$70 —→ taille normale des caractères
- \$71 —→ double hauteur, largeur normale
- \$72 —→ double largeur, hauteur normale
- \$73 —→ double largeur, double hauteur
- \$74 . . . → caractères + couleurs (pour le scrolling)
- \$75 —→ caractères seuls
- \$76 —→ mode de non-incrustation
- \$77 —→ mode incrustation
- \$78 —→ scroll normal
- \$79 . . . → scroll lent
- \$7A . . . → rien
- \$7B —→ inversion vidéo

Ainsi, si vous voulez faire des scrolls lents avec les couleurs vous devez taper :

```
?CHR$(27), CHR$(&H79), CHR$(27), CHR$(&H74)
```

On dispose donc de 4 scrolls différents qui sont :

- scroll rapide sans couleur
- scroll rapide avec couleur
- scroll lent sans couleur
- scroll lent avec couleur

L'intérêt de pouvoir scroller sans les couleurs est important. On fixe les couleurs dans l'écran puis on fait défiler le texte par-dessus. Les caractères changent donc de couleur au fur et à mesure de leur déplacement vertical sur l'écran. Ceci peut être utile lors d'une présentation, par exemple.

Les précédentes pages vous ont offert des solutions lorsque l'on programme en BASIC, voire en BASIC associé à du langage machine. Mais on peut faire la même chose et ce très simplement en langage machine. Commençons par le plus simple : le système des attributs

Nous avons vu que pour envoyer un attribut, il faut d'abord envoyer un ESC (code 27). Les codes, qu'il s'agisse du ESC ou de l'attribut, sont envoyés par l'intermédiaire d'une routine dont l'octet d'indexation est 02 (on a bien-sûr accès à cette routine par un SWI, dans le cas présent, un SWI suivi d'un 02).

Pour faire un scroll lent (attribut 79) il faut donc réaliser :

```
LDB #$1B (code du ESC)
SWI #$02 (appel)
LDB #$79 (code du scroll lent)
SWI #$02 (appel)
```

Ceci prend au total 8 octets qui sont :

```
C6 1B LDB #$1B
3F 02 SWI #$02
C6 79 LDB #$79
3F 02 SWI #$02
```

Le procédé est le même qu'il s'agisse des attributs correspondants aux couleurs ou des autres.

Note : on a vu ici que l'on pouvait réaliser la fonction COLOR en langage machine, mais on peut aussi réaliser la fonction SCREEN qui à l'inverse fonctionne en plein écran. Pour ce faire, il suffit de faire la séquence suivante :

```

LDB # $1B (escape)
SWI # $02
LDB # $20 (plein écran)
SWI # $02
LDB # $?? (code encre ou fond)
SWI # $02
RTS

```

LES ROUTINES DU MONITEUR

Toutes les routines que nous indiquons dans ce sous-chapitre fonctionnent selon le même mode : on charge les paramètres d'entrée dans certains registres du 6809, puis on fait un SWI suivi du code (que nous donnons pour chaque routine) d'indexation de table.

■ Test de point

Cette routine sert à savoir ce qui se trouve en un endroit donné de l'écran : les paramètres d'entrée sont l'abscisse et l'ordonnée du point à tester qui doivent être stockées respectivement dans X et Y. Le code d'indexation de cette routine est \$14. Le paramètre de retour se trouve dans l'accumulateur B du 6809 : il varie de -16 à -1 si le point fait partie du fond et de 0 à 15 s'il fait partie de la forme, la valeur exacte étant déterminée par la couleur.

Le test du point de coordonnées 5,5 sera donc :

```

LDX # $5 (abscisse)
LDY # $5 (ordonnée)
SWI # $14 (appel)
RTS (retour)

```

Dans cette routine :

X varie entre 0 et 310
 Y entre 0 et 199
 B entre -16 et 15.

■ Test de caractère

Cette routine permet de tester s'il y a un caractère à un endroit de l'écran. Les paramètres d'entrée sont la colonne : de 1 à 40 placée dans le registre X, la ligne : de 0 à 24 placée dans l'accumulateur A. Le code d'indexation de cette routine est \$1A. Le paramètre de retour est le code ASCII du caractère trouvé à cette position et est placé dans B. S'il n'y a pas de caractère à la position demandée, le retour fournit 0. Si le caractère est un caractère accentué ou un « ç », la procédure est quelque peu différente, on doit faire trois appels à la routine :

le premier retourne \$16,

le second retourne le code de l'accent.

le troisième retourne le code ASCII du caractère.

Un test à la case de coordonnées 9; 4 se fait donc de la manière suivante :

```
LDX #S0009 (abscisse)
LDA #S04    (ordonnée)
SWI #S1A    (appel)
RTS         (retour)
```

Ce qui, codé, donne :

```
8F 00 09
86 04
3F 1A
39
```

■ Test du crayon optique

Cette routine sert à tester la position sur laquelle se trouve le crayon optique avec test des valeurs trouvées. Il n'y a pas de paramètre d'entrée. Le code d'indexation de cette routine est \$18. Les paramètres de retour sont :

l'abscisse placée dans X variant de 0 à 319,

l'ordonnée placée dans Y variant de 0 à 199.

Si la mesure est correcte, le bit de carry est mis à 0, sinon il est à 1. Le test du crayon optique se fait donc de la manière suivante : SWI #S18.

■ Test de l'interrupteur du crayon optique

Au bout du crayon optique, se trouve un petit interrupteur. C'est lui qui fait la différence entre les instructions INPEN et INPUTPEN du BASIC. La routine qui teste cet interrupteur, et a pour code d'indexation \$16, ne nécessite pas de paramètre d'entrée et retourne dans le bit de Carry de GC l'état de l'interrupteur du crayon optique : Carry :

1 si l'interrupteur est appuyé,
0 sinon.

Le test de cet interrupteur se fait de la manière suivante : SWI #S16.

■ Affichage d'un caractère à l'écran

Cette routine permet l'affichage d'un caractère à la position courante du curseur (que nous avons définie auparavant dans ce chapitre). Le paramètre d'entrée est le code ASCII du caractère à afficher qui doit se trouver dans B avant l'exécution. Le code d'indexation est 02, il n'y a pas de paramètre de retour.

Ainsi l'affichage d'un « A » à la position courante du curseur se fait comme suit :