programmation en assembleur du 68000

et des circuits d'interface

A travers cet ouvrage l'auteur nous invite à découvrir le 68000 en privilégiant les applications pratiques voire quotidiennes. Moniteur, assembleur, interruptions, programmation des circuits standards (temporisateur, interfaces série et parallèle) sont les chapitres-étapes par lesquels l'amateur est convié à parfaire ses connaissances.
De nombreux programmes, courts et commentés, fournissent les exemples qui sont nécessaires à tout apprentissage, tout en résolvant au passage des questions aussi diverses que peuvent être : la conversion numérique-analogique, la connexion à un réseau, le calcul en temps réel, l'écriture des EPROM, les problèmes numériques.
La connaissance d'un autre langage, sans être absolument nécessaire, peut rendre plus facile l'étude de cet ouvrage.

programmation en assembleur du 68000

par Frank Brown

DIFFUSE PAR

ISBN 2-86699-030-7 120 FF

IMPRIME EN FRANCE

programmation en assembleur du 68000

et des circuits d'interface

par Frank Brown

éditests

couverture AGRAPH

Achevé d'imprimer en février 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : février 1985

Numéro d'impression : 502002
N° d'édition : 86699-30-1
ISBN : 2-86699-030-7

DANS LA MEME COLLECTION

□ *Série «système d'exploitation»*

- **Le système UNIX** - utilisation des commandes par Violaine Prince
- **Le système CP/M pour Z-80** - adaptation du BIOS et compléments par Fabienne et Philippe Gysel
- **Le système CP/M pour 8080** - utilisation et programmation en version 2.2 par Jacques Pinto
- **Le système PASCAL UCSD** - 1/ organisation générale par Thierry Chamoret
- **Le système PASCAL UCSD** - 2/ structure interne par Thierry Chamoret
- **Le concept FORTH** - langage et système par Pascal Courtois
- **Le système MemDos** - conception de programmes par Pierre Clerc
- **Systèmes PC-DOS et MS-DOS - version 2** - introduction et utilisation avancée par Jacques Boyer, Jean-Pierre Lamoitier et Michel Treillet
- **Le système PROLOGUE - version 2.2** par Pierre Giraud

□ *Série «langage de programmation»*

- **Le langage ADA** par Daniel- Jean David
- **Le langage APL** par Daniel- Jean David
- **Le langage C** par Jean-Louis Fourtanier et Violaine Prince
- **Le langage FORTRAN** par Daniel-Jean David
- **Le langage D-PROLOG** - initiation au langage de la 5ème génération par Philippe Donz et Rosalie Hurtado
- **Le langage B.A.L.** - initiation et pratique par Didier Lévy
- **Le langage Pascal ISO** par Thierry Chamoret

□ *Série «communication»*

- **Les réseaux locaux d'entreprises** - marchés et technologies par Frédéric Hoste
- **Les techniques de la télématique** par Jérôme Toussaint et Philippe Masson

□ *Série «dialogue homme-machine»*

- **Synthèse, reconnaissance de la parole** par Marc Ferretti et François Cinare
- **La synthèse d'image** par Francis Martinez

□ *Série «informatique»*

- **Les opérations arithmétiques dans les ordinateurs** par Ioan Dancea

□ *Série «électronique numérique»*

- **Choisir un système de développement pour microprocesseurs** par Michel Blanchard, Jean-Claude Cavarroc et Michel Gay
- **Les systèmes à microprocesseurs** par Daniel-Jean David
- **Les circuits programmables** par Jean-Michel Bernard et Henri Breteuil
- **Microprocesseurs et circuits associés** par Roland Dubois
- **Mise en oeuvre du bus IEEE 488** - - utilisation et réalisation d'appareils par Gérard Bastide et Jean-René Vellas
- **Le filtrage numérique par microprocesseurs** par Robert Miquel

□ *Série «productique»*

- **Les systèmes industriels d'intelligence artificielle** - outils de productique par Lucas Pun
- **Introduction à la robotique - 1/ notions** de base, architecture, systèmes actionneur et sensoriel, modes de fonctionnement par Pierre Lopez et Jean-Numa Foulc
- **Introduction à la robotique** - 2/ communication homme-machine programmation et commande, transformations homogènes par Pierre Lopez et Jean-Numa Foulc

□ *Série «bandes dessinées»*

- **Les ordinateurs... Bonjour !** par Zévar

programmation en assembleur du 68000

et des circuits d'interface

par Frank Brown

Docteur es Sciences
Maître de Conférences à l'Université du Maine

éditests

1985

Les multiples tableaux et diagrammes de cet ouvrage concernant les composants 68000, 6821, 6840, 6850, sont extraits du catalogue MICROPROCESSEURS ET MEMOIRES, publication EFCIS-THOMSON-SEMICONDUCTEURS (1980).

Que la Compagnie THOMSON-SEMICONDUCTEURS trouve ici l'expression de nos profonds remerciements pour l'autorisation qu'elle nous a accordée d'en faire usage dans ce manuel.

Nos remerciements iront également à la Société YREL, (Z.I. de BUC, 78530) représentant FORCE-COMPUTER, qui nous a permis d'utiliser toute sa documentation concernant notre matériel d'essais.

Tous droits de traduction, d'adaptation et de reproduction par tous procédés réservés pour tous pays

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les «copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective» et, d'autre part, que les analyses et les courtes citations dans le but d'exemple et d'illustration, «toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause, est illicite» (alinéa 1er de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Editests - 5, place du Colonel Fabien - 75491 Paris cédex 10 - 1985
ISBN 2-86699-030-7
P.S.I. Diffusion - BP 86 - 77402 Lagny sur Marne
Tel. (6) 006 44 35

Imprimé en France

sommaire

		PAGES
CHAPITRE 1	DESCRIPTION DU 68000	7
	Les registres	7
	Description des signaux	9
	Le matériel d'essais utilisé	12
CHAPITRE 2	LE MONITEUR ET SES COMMANDES	17
	Visualisation d'un registre	17
	Déplacement d'une zone mémoire	17
	Chargement en mémoire	18
	Lancement d'un programme	18
	Affichage de la mémoire	19
	Modification de la mémoire	19
	Conversion de données	20
	Recherche d'une chaîne en mémoire	20
	Vidage mémoire	21
	Rechargement en mémoire	21
	Mode transparent	21
	Mise au point d'un programme	22
	Point d'arrêt	23
CHAPITRE 3	L'ASSEMBLEUR-EDITEUR	25
	Premiers pas vers l'écriture d'un programme	25
	Emulation d'autres consoles	26
	Conventions diverses	27
	Les codages binaire, BCD, hexadécimal, ASCII	28
	Codage numérique binaire	28
	Codage BCD	30

Codage hexadécimal 30
Codage ASCII 31

Les pseudo-instructions 31

CHAPITRE 4 GENERALITES SUR L'ADRESSAGE 35

Le mode d'adressage absolu 35
Commodités d'écriture dans l'adressage absolu 37
Le mode d'adressage registre indirect 38
Adressage registre indirect 38
Adressage indirect avec déplacement 39
Adressage indirect avec index 39
Adressage indirect avec index et déplacement sur 8 bits signés 39

Le mode d'adressage immédiat 40
Déplacement, addition et soustraction rapides 40

Le mode d'adressage indirect avec postincrémentation ou prédécrémentation 41

L'adresse courante : l'opérande '*' 42

Retour au moniteur : les routines 43
Les routines d'entrées/sorties 43
Les routines de retour au moniteur 45

CHAPITRE 5 LE JEU D'INSTRUCTIONS 47

Les instructions sur registres 48

Quelques programmes de mise en train 49
Ecriture de l'alphabet 49
Ecriture du contenu d'un registre en binaire 56
Conversion d'une touche en hexadécimal 56
Conversion avec table de traduction 59

Tests, comparaisons et codes condition 61

Les instructions de branchement 63

Les instructions logiques 65

Les instructions de décalage et de rotation 66

Les instructions de manipulation de bits 72

Les instructions de mouvement de données 72

Les instructions de contrôle du système 72

Les instructions arithmétiques 74

Les instructions en BCD 76

CHAPITRE 6 LES EXCEPTIONS 87

Nature des événements 87

Traitement de l'exception TRAPV 90

L'instruction CHK 91

CHAPITRE 7 LES INTERRUPTIONS 97

CHAPITRE 8 LE PIA 101

Programmation des lignes d'E/S 101

Programmation des lignes CA1 et CA2 103
Contrôle de CA2 105

Réalisation d'une conversion numérique/analogique 107

Conversion analogique/numérique 110

CHAPITRE 9 LE PTM 113

Organisation 113

Quelques applications du PTM 115
Fabrication d'une horloge 115
Obtention de très basses fréquences 115
Le mode monocoup 117

Amélioration des performances
Contrôle de la vitesse d'un moteur 120

CHAPITRE 10 L'ACIA 123

Description 123

Le mode interruption 125
Transmission et réception de données non ASCII 127

Le mode de transmission "S" 129

CHAPITRE 11 LES PILES 135

La pile système 135

Passage de paramètres 135

Stockage des registres dans la pile 137

CHAPITRE 12 AUTRES FONCTIONNALITES 139

Généralités sur les systèmes multi-utilisateurs 139

Améliorations des performances 140

Les communications assembleur-Basic 145

Communication par interruption 151

CHAPITRE 13 L'ECRITURE DES EPROM 155

CONCLUSION 161

ANNEXE 1 LE JEU DE CARACTERES ASCII 163

ANNEXE 2 MANIPULATION DE NOMBRES DESTINES A UNE TRANSMISSION SERIE 165

ANNEXE 3	EMULATION DE CONSOLES	167
ANNEXE 4	CONNEXION ENTRE LES ACIA	169
ANNEXE 5	REGARDS SUR LES PRODUITS NOUVEAUX	171
	Le 68008	171
	Le 68010	171
	Le 68020	172

CHAPITRE 1 description du 68000

LES REGISTRES

Réalisé en technologie VLSI, le microprocesseur 68000 se présente sous la forme d'un boîtier de 64 broches (**figure 1.1**). Il offre dix-sept registres de 32 bits, un registre d'état de 16 bits, et un registre compteur-programme de 24 bits. L'espace adressable peut atteindre 16 M octets (à comparer aux 64 K octets des processeurs 8 bits). Il possède 56 instructions pouvant opérer sur cinq principaux types de données, le bit, le demi-octet, l'octet, le mot et le mot-long, et l'utilisateur dispose de quatorze modes d'adressage.

Le 68000 comporte huit registres de données de 32 bits nommés D0, D1,...D7, huit registres d'adresse de 32 bits nommés A0, A1,...A7, (A7 étant le registre pointeur de pile système) et un autre registre pointeur de pile utilisateur USP. Tous peuvent servir de pointeur de pile logiciel ou de registre de base d'adressage.

Les adresses sont véhiculées par un bus de 23 bits, et le vaste espace mémoire adressable permet le développement de programmes très importants, rendant inutiles les techniques de pagination et de segmentation souvent génantes.

Le registre d'état de 16 bits contient le masque d'interruptions (huit niveaux) et les codes conditions sur 5 bits : X (extension), N (négatif), Z (zéro), V (débordement), C (retenue). Deux autres bits indiquent si le processeur est en mode Trace d'une part et en mode Superviseur ou Utilisateur, d'autre part (**figure 1.2**).

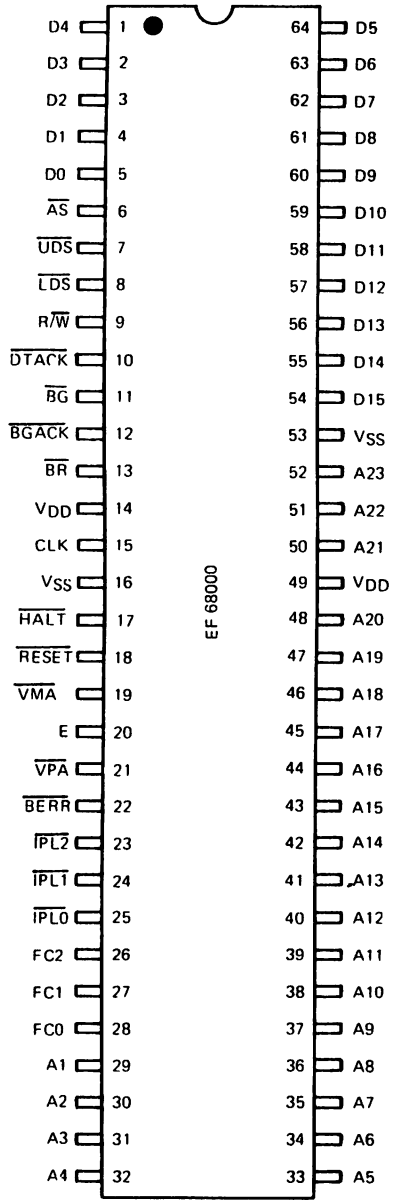


Fig. 1.1 - Le 68000 comporte 64 broches.

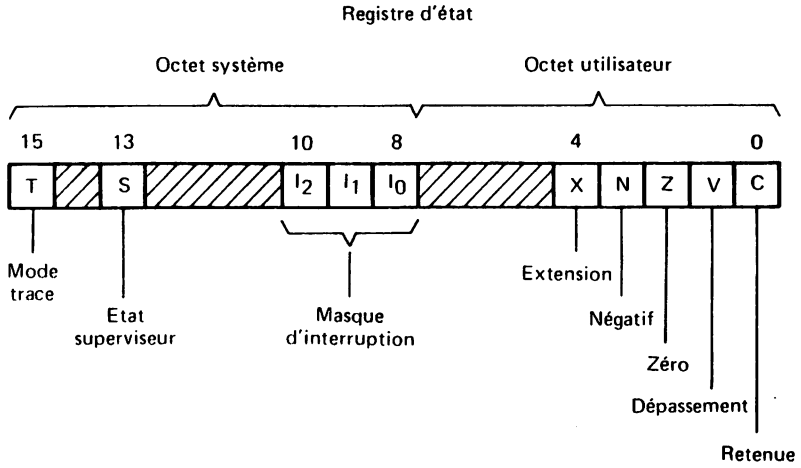


Fig. 1.2 - Le registre d'état contient, outre le masque d'interruption et les codes condition, des informations concernant le mode d'utilisation du processeur.

DESCRIPTION DES SIGNAUX

La **figure 1.3** indique le groupement des signaux selon leurs fonctions.

Le bus adresse (de A1 à A23) est unidirectionnel, à trois états, et peut adresser 8 M mots (de deux octets) ; dans le cas d'interruptions, les lignes A1, A2, A3 précisent le niveau d'interruption servi, les autres passant à l'état haut.

Le bus de données, sur 16 bits, est bidirectionnel, à trois états, et véhicule des données sous forme de mots ou d'octets. Au cours d'un cycle d'interruption, D0 - D7 reçoivent des circuits externes un numéro de vecteur.

Les divers signaux de **commande de bus asynchrone** sont : $\overline{AS}$ qui indique une adresse valide sur le bus adresse ; R/$\overline{W}$ qui définit le type de transfert (lecture ou écriture) sur le bus de données ; $\overline{UDS}$ et $\overline{LDS}$ qui permettent, en conjonction avec R/W, de contrôler les données selon le tableau de la **figure 1.4.**

$\overline{DTACK}$ indique que le transfert des données est réalisé.

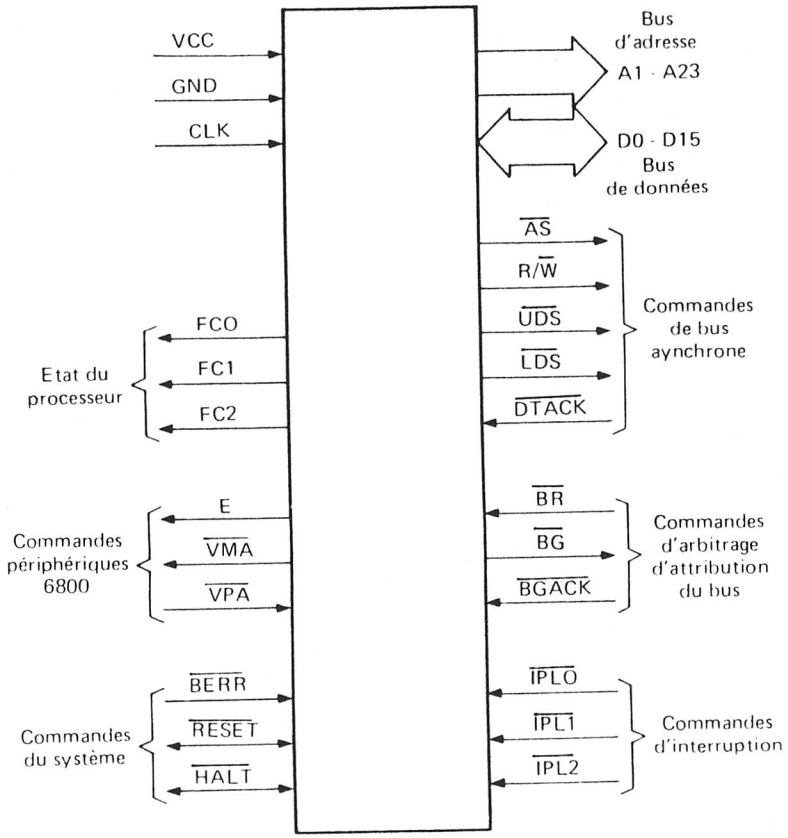


Fig. 1.3 - Les signaux d'entrée/sortie du 68000.

Les **commandes d'arbitrage d'attribution du bus** sont au nombre de trois : $\overline{BR}$ signale qu'un autre circuit désire être maître du bus ; $\overline{BG}$ (sortie) indique que le contrôle du bus va être libéré ; $\overline{BGACK}$ informe qu'un autre circuit a pris la commande du bus.

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7
haut	Haut	—	données non valides	données non valides
bas	bas	haut	bits données valides 8-15	bits données valides 0-7
haut	bas	haut	données non valides 8-15	bits données valides 0-7
bas	haut	haut	bits données valides 8-15	données non valides 0-7
bas	bas	bas	bits données valides 8-15	bits données valides 0-7
haut	bas	bas	bits données valides 0-7*	bits données valides 0-7
bas	haut	bas	bits données valides 8-15	bits données valides 8-15*

* Ces conditions résultent de l'implantation en cours et peuvent ne pas apparaître sur les circuits futurs.

Fig. 1.4 - Les signaux UDS, LDS et R/W assurent le contrôle des données.

$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$ indiquent **le niveau de priorité** du circuit demandant une interruption (sept est la plus haute priorité, et zéro indique qu'aucune interruption n'est demandée).

Les commandes systèmes sont également au nombre de trois : $\overline{BERR}$ signale une erreur dans le cycle en cours d'exécution ; $\overline{RESET}$ est une ligne bidirectionnelle qui initialise le processeur si un signal externe est appliqué (si l'initialisation est interne, les circuits externes sont mis à zéro) ; $\overline{HALT}$ (ligne bidirectionnelle), activée de l'extérieur, provoque l'arrêt du processeur à la fin du cycle bus en cours, le processeur indique aussi, dans l'autre sens, aux circuits extérieurs, qu'il est à l'arrêt.

Les commandes périphériques 6800 sont destinées à l'interfaçage avec les circuits périphériques très développés de la série 6800 : E est une validation standard à tous les périphériques de cette série, cette horloge est à une fréquence égale au dixième de celle du signal présent sur l'entrée CLK : $\overline{VPA}$ indique que le circuit adressé est de la famille 6800, avec pour conséquence l'utilisation du système de vectorisation automatique ; $\overline{VMA}$ informe que l'adresse sur le bus est valide en réponse à $\overline{VPA}$.

Les signaux d'état du processeur FC0, FC1, FC2 sont des codes fonction indiquant le mode de fonctionnement et le type de cycle en cours d'exécution.

CLK est une **entrée horloge TTL**, de fréquence fixe.

Le **tableau 1.5** est un résumé de tous les signaux présentés. Tous ces signaux sont utiles aux concepteurs de circuits et de matériels, mais n'ont pas d'influence sur la programmation du 68000 ; ils représentent l'état de l'art dans le développement d'une telle machine. Ils peuvent, quelquefois, expliquer le type de fonctionnement du processeur face à un signal extérieur.

LE MATERIEL D'ESSAIS UTILISE

Il existe maintenant de nombreux systèmes bâtis autour d'un 68000 et notre choix s'est porté sur le système monocarte PROFKIT 2 de Force Computer qui possède les caractéristiques intéressantes suivantes :

- ensemble monocarte (donc ni chassis, ni rack à prévoir) ;
- alimentation standard 0-5 volts (200 mA) et +/- 12 volts (50 mA) ;
- deux ports entrées/sorties "série" réglables de 110 bauds à 9600 bauds (RS232) ;
- une interface cassette 700 bauds permettant la sauvegarde et le chargement d'une partie quelconque de la mémoire ;

Nom du signal	Mnémonique	Entrée/Sortie	Niveau actif	Trois états
Bus d'adresse	A1-A23	sortie	haut	oui
Bus de données	D0-D15	entrée/sortie	haut	oui
Echantillonnage adresse	$\overline{AS}$	sortie	bas	oui
Lecture/écriture	R/$\overline{W}$	sortie	lecture/haut écriture/bas	oui
Echantillonnage données haut et bas	$\overline{UDS}$, $\overline{LDS}$	sortie	bas	oui
Reconnaissance de transfert de données	$\overline{DTACK}$	entrée	bas	non
Requête du bus	$\overline{BR}$	entrée	bas	non
Bus accordé	$\overline{BG}$	sortie	bas	non
Reconnaissance d'allocation de bus	$\overline{BGACK}$	entrée	bas	non
Niveau de priorité d'interruption	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	entrée	bas	non
Erreur bus	$\overline{BERR}$	entrée	bas	non
Initialisation	$\overline{RESET}$	entrée/sortie	bas	non *
Arrêt	$\overline{HALT}$	entrée/sortie	bas	non *
Validation (horloge)	E	sortie	haut	non
Validation adresse mémoire	$\overline{VMA}$	sortie	bas	oui
Validation adresse périphérique	$\overline{VPA}$	entrée	bas	non
Sortie code fonction	FC0, FC1, FC2	sortie	haut	oui
Horloge	CLK	entrée	haut	non
Entrée puissance	V_{CC}	entrée	-	-
Masse	GND	entrée	-	-

Fig. 1.5 - Résumé des signaux du 68000.

- une interface parallèle pouvant servir au branchement d'une imprimante de type "Centronics" et plus généralement aux entrées/sorties rapides (vingt lignes) ;
- un temporisateur triple, dont un avec prédiviseur par 8 ;
- 128 K octets de mémoire "dynamique" ;
- une fréquence d'horloge de 8 MHz ;
- possibilité de loger des programmes personnels jusqu'à 32 K octets sur Eprom ;
- Basic et assembleur-éditeur résidants sur Eprom.

Un tel système allie donc une vitesse rapide, un stockage important, et de nombreuses possibilités de connexion, soit à des dispositifs électroniques/électro-mécaniques, soit à des ensembles informatiques, grâce aux ports parallèle et série. Il est donc possible de développer des programmes de toute nature même sur la configuration minimale (**figure 1.6**).

Cette ensemble se présente pour l'utilisateur, aussi simplement qu'un ordinateur domestique (sauf pour l'instant, en ce qui concerne les disquettes), mais avec des possibilités et des performances très supérieures.

L'ensemble de la mémoire est structuré selon la **figure 1.7.** Les adresses sont notées en base 16. En cas de confusion, une valeur écrite en base 16 commence par le symbole monétaire $. Cet emploi est généralisé. Regrettons-le, car ce caractère typographique ne figure pas encore sur toutes les imprimantes ou machines à écrire.

La mise au point du programme se fait simplement, grâce à l'assembleur-éditeur, puis à l'aide du moniteur qui permet le lancement du programme et le contrôle de son déroulement (éventuellement instruction par instruction : c'est le mode 'TRACE'), et cela jusqu'à ce que la conformité avec les résultats désirés soit satisfaite. A ce moment, si cet ensemble n'est destiné qu'à couvrir cette seule application, le programme mis au point peut être enregistré sur Eprom (on "claque" les Eprom) et ces Eprom sont mises à la place de celles qui contiennent l'assembleur-éditeur, devenu désormais inutile.

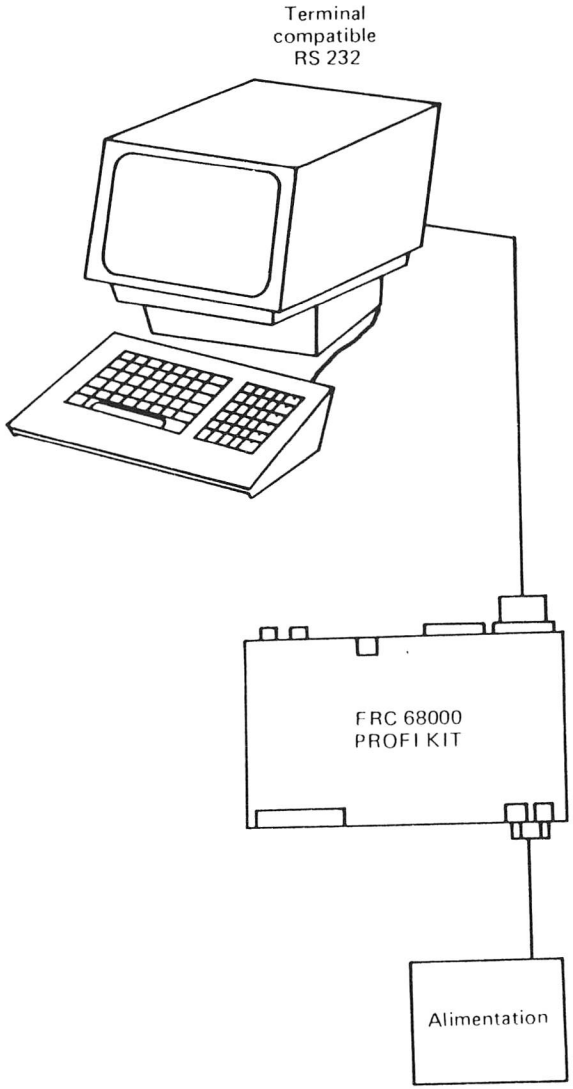


Fig. 1.6 - Configuration minimale du système PROFKIT-2.

000000 000007	initialisation du processeur
000008 0003FF	vecteurs d'exception
000400 0007FF	zone de travail du moniteur
000800 01FFFF	mémoire RAM utilisateur
020000 027FFF	EPROM SYSTEME
028000 02FFFF	EPROM UTILISATEUR (Assembleur, Basic, etc.)
04CF41 04CF4F	temporisateur PIM
050040 050042	interface série ACIA (TERMINAL)
050041 050043	interface série ACIA (HOTE)
05CEF1 05CEF7	interface parallèle PIA

On remarque que les positions adressables ne sont pas toujours consécutives et on notera le chevauchement des adresses des ACIA (Asynchronous Communications Interface Adaptor).

La signification des diverses zones mémoires consacrées aux circuits ACIA, PIA (Peripheral Interface Adaptor), PIM (Temporisateur Programmable) sera étudiée en même temps que les circuits eux-mêmes.

Fig. 1.7 - Organisation de la mémoire adressable.

CHAPITRE 2 | le moniteur et ses commandes

Le rôle du moniteur est de permettre la communication entre l'utilisateur et le 68000. Il veille à la bonne exécution du programme et peut agir sur son déroulement. Ses diverses commandes peuvent faciliter notre tâche.

VISUALISATION D'UN REGISTRE

La commande DF (Display Formated) assure l'affichage des registres.

DF [R]
([R] indiquera la touche RETURN : valeur hexadécimale 0D) visualise le contenu de tous les registres D0-D7, A0-A7, SR, PC, USP...

On peut agir sur un registre particulier, ainsi :

.A6 [R]
affichera la valeur du registre A6 ;

.A6 AB1145 [R]
placera dans A6 la valeur $AB1145 (cadrée à droite évidemment) ;

.A6 &456 [R]
placera dans A6 la valeur décimale 456 (donc en fait $1C8).

DEPLACEMENT D'UNE ZONE MEMOIRE

Déplacer une zone de mémoire par la commande BM (Block Move) est une opération fréquente par exemple pour :

- reloger un programme, c'est-à-dire l'implanter à un endroit où il sera mieux intégré ;

- préserver une partie de la mémoire que l'on ne veut pas détruire (on pourra, par exemple conserver en mémoire plusieurs programmes en assembleur, pour une mise au point générale, avec des assemblages totalement indépendants).

BM A000 B000 6000 [R]
déplace la zone comprise entre les adresses A000 et B000 pour la mettre à partir de l'adresse 6000. La zone initiale n'est pas affectée.

Par cette commande, on peut reloger l'assembleur lui-même :

BM 24000 28000 18000 [R]
le contenu de l'Eprom est recopié à partir de l'adresse 18000, dans la Ram. Il sera alors possible de modifier certains codes de l'assembleur afin de permettre l'utilisation d'une autre console, totalement différente, qui sera alors "émulée" ; l'intérêt de cette manipulation sera étudié plus loin.

CHARGEMENT EN MEMOIRE

BF (Block Fill) permet essentiellement de charger la mémoire, pour effacer les informations parasites :

BF A000 B000 0000 [R]
la zone mémoire comprise entre A000 et B000 va être chargée entièrement du mot 0000 (on ne peut utiliser l'octet ou le mot long). Avant d'introduire un autre programme, cette commande, qui détruit le "source", constituera une judicieuse précaution.

LANCEMENT D'UN PROGRAMME

G XXXX (Go) permet de lancer un programme dont l'adresse de début est XXXX (c'est, on le verra, l'adresse qui suit la pseudo-instruction ORG dans le cas général). Pour appeler le Basic, implanté sur Eprom à l'adresse début 28000 on fera :

G 28000 [R]

Comme pour toutes les autres commandes du moniteur, les paramètres, sauf convention contraire, sont des valeurs hexadécimales.

L'assembleur-éditeur se charge par :

G 24000 [R]
... ou à l'adresse où on l'a relogé (et modifié).

Lorsque l'exécution est achevée, le contrôle est repris par le moniteur qui affiche FORCEMON et attend la commande suivante... sauf si la fin du dit programme n'est jamais atteinte (mauvais branchement, boucle sans fin).

Les boutons poussoirs **Software abort** et **Taster abort** permettent la reprise en main, avec les différences suivantes :

- **Software abort** affiche le contenu des registres, au moment de l'interruption, et renvoie au moniteur sans rien modifier ;
- **Taster abort** ne visualise rien et procède à la réinitialisation complète.

AFFICHAGE DE LA MEMOIRE

MD (Memory Display) affiche sur une ligne le contenu de seize positions mémoires consécutives, ainsi que leur traduction ASCII, lorsqu'elle est possible. Cette commande est relancée par [R] et seize nouvelles lignes sont alors affichées, etc. Si la mémoire est isolée (cas des registres des interfaces, du temporisateur) une erreur surviendra lors de la commande, puisqu'une référence à une adresse illicite est demandée, mais la commande MM agissant, elle, au niveau de l'octet, évitera ce genre d'erreur.

Option désassemblage :

MD 1000 20;DI [R]
demande que vingt octets, à partir de l'adresse 1000, soient désassemblés. Il y aura donc reconstitution d'un source, assez fruste mais clair, correspondant à la partie du programme implantée en 1000. On peut donc reconstituer un listing à partir d'un binaire. Plus généralement, on utilise cette option pour contrôler un élément de programme, dont l'exactitude est sujette à caution, à la suite, par exemple, d'une manoeuvre malencontreuse. Le programme n° 2, donné plus loin, a été assemblé et désassemblé par cette commande. D'ores et déjà, il est facile de constater une assez grande similitude entre ces deux formes.

MODIFICATION DE LA MEMOIRE

MM (Memory Modify) permet de visualiser et de modifier éventuellement un octet, un mot ou un double mot :

MM 1000;L [R]
visualise le mot long d'adresse 1000. Cette commande permet

l'introduction manuelle d'un programme objet et plus généralement l'altération de toute zone. La touche [R] , tant qu'elle est enfoncée, permet le déroulement continu du processus. On utilisera aussi MM pour visualiser les registres de contrôle des circuits périphériques.

Enfin, on peut aussi obtenir un désassemblage en série avec l'option DI

MM 1000 10;DI [R]
... tant que [R] est enfoncée.

CONVERSION DE DONNEES

DC (Data Conversion) permet la conversion hexadécimale-décimale :

DC $10 [R]
affichera : $10 = &16

On peut aussi demander d'effectuer une addition (ou une soustraction) :
DC $10+$20 [R]
affichera : $30=&48

Les valeurs introduites sont toujours considérées comme positives, mais on peut fournir un signe si la zone est inférieure à 32 bits. A noter que la valeur $80000000, en principe négative en code binaire complémenté, et considérée comme telle par les programmes, est traduite par 0 par la commande DC (au lieu de -2147483648).

RECHERCHE D'UNE CHAINE EN MEMOIRE

BS (Block Search) assure la recherche d'un mot, d'un octet ou d'une chaîne en mémoire :

BS A000 B000 'RTS;' [R]
recherche de 'RTS;' entre les adresses A000 et B000, si cette chaîne existe l'adresse de début sera affichée (adresse du "R" ici) à chaque apparition :

00A14B 'RTS;'

00A151 'RTS;'

.....

Cette commande permet de connaître la fin d'un programme source, et doit être faite avant la sauvegarde sur un fichier, ou sur un support extérieur.

VIDAGE MEMOIRE

On peut vider la mémoire par la commande DUn (DUMP sur le périphérique n : 4=cassette ; 2=hôte)

DU4 A000 A260 [R]
va transférer en mode "S" le contenu de la mémoire entre les adresses A000 et A260. Il y a ensuite retour au moniteur, à la fin du transfert, qui se fait ici sur un magnétophone ordinaire à cassette, branché sur le port 4. Le mode "S" sera étudié en détail, par la suite.

RECHARGEMENT EN MEMOIRE

On peut ensuite recharger un tel programme par la commande symétrique LOn (LOad) :

LO4 [R]
(et on met la cassette en route, en lecture)

DU2 et LO2 permettent le même traitement avec l'ordinateur hôte.

Remarque : l'assembleur-éditeur possède une commande S (SAVE) qui joue le même rôle que DU2 A100 XXXX, mais avec une certaine simplification, car les adresses de début et de fin de programme n'ont pas à être précisées. Bien entendu, lorsque l'ordinateur hôte est sollicité pour recevoir des informations, un programme d'édition en mode insertion doit être actif ; il joue le rôle de la cassette, en enregistrement.

MODE TRANSPARENT

Une façon particulièrement commode de dialoguer avec un ordinateur hôte est l'utilisation du mode transparent TM (Transparent Mode).

TM x [R]
passe en mode transparent jusqu'à ce que le caractère x soit

envoyé (par défaut x = Control-A = $01). Tout se passe comme si la connexion était directement établie entre la console et la ligne de l'ordinateur ; il y a d'ailleurs un passage direct de l'information d'un port à l'autre, grâce à des portes NAND. Les ports doivent alors être réglés pour une même vitesse de transmission.

Une séquence classique pour décharger un programme est la suivante :

- TM / [R]
- se connecter (mot de passe etc.) ;
- appeler l'éditeur en mode insertion ;
- / fin du mode transparent, retour au moniteur ;
- G 24000 (appel de l'assembleur-éditeur, le source apparaît) ,
- S commande SAVE de l'assembleur-éditeur. Le transfert est lancé sur le port 2, dès la fin du transfert, il y a retour au moniteur, on peut alors : soit se remettre en mode transparent pour indiquer à l'ordinateur hôte d'inscrire le fichier sous un nom choisi, et revenir ensuite au mode normal ; soit, si les commandes sont simples, utiliser l'option *.

  * **message** [R]
  envoie **message** sur l'ordinateur hôte.

Pour récupérer un tel fichier, il faut procéder de façon analogue : se connecter et appeler l'éditeur en mode lecture puis envoyer une commande LOAD

LO2;X;=message [R]
le message est envoyé à l'éditeur, et celui-ci pourra décharger le fichier. De plus, on verra le transfert s'effectuer au fur et à mesure sur l'écran.

Ces modes de fonctionnement sont très sûrs et accroissent notablement les possibilités du microcalculateur. Ils seront détaillés dans le chapitre consacré à l'ACIA.

MISE AU POINT D'UN PROGRAMME

La commande de mise au point TR (Trace) permet au programme de se dérouler pas-à-pas et entre chaque instruction le contenu de tous les registres sera affiché :

.PC XXXX [R]
on charge l'adresse de début de programme dans le compteur-programme puis :

TR [R]
provoque l'arrêt dès la première instruction et l'affichage des registres. Chaque appui sur [R] continuera le processus. La trace ne permet pas toutefois d'explorer les routines du moniteur. Une commande illégale achève le mode trace et renvoie au moniteur.

POINT D'ARRET

Lorsque la partie d'un programme à explorer n'est activée qu'après le déroulement de multiples instructions, il est commode de placer un point d'arrêt, BR (Breakpoint) dont le rôle est de stopper le déroulement du programme à l'adresse indiquée :

BR 1000 [R]

G 800 [R]
le programme va se dérouler jusqu'à ce qu'il rencontre l'instruction d'adresse 1000 ; à ce moment le contrôle est rendu au moniteur, après visualisation des divers registres. Il est possible de continuer pas-à-pas cette fois, par :

TR [R]

Il est possible également de ne provoquer l'arrêt qu'après plusieurs passages sur une adresse ; ce peut être le cas pour une boucle au fonctionnement douteux :

BR 1000;10 [R]
le point d'arrêt aura lieu à la dixième rencontre avec l'adresse 1000.

Il existe d'autres commandes, mais celles que nous venons de voir permettent de développer tout type de programme. On verra qu'il existe également des routines spécialisées du moniteur appelables par programme. Selon les machines construites autour d'un 68000, on pourra trouver de légères différences de présentation, n'altérant en rien les fonctions et les commandes exposées.

CHAPITRE 3 l'assembleur-éditeur

PREMIERS PAS VERS L'ECRITURE D'UN PROGRAMME

L'assembleur-éditeur pleine page est un logiciel résidant implanté dans la zone utilisateur sur deux Eprom 2764, tout comme le moniteur dont nous venons de décrire les principales commandes. Il est possible, après une modification minime du circuit imprimé d'implanter le moniteur et l'assembleur sur deux 27128, ce qui libère deux emplacements Eprom pour l'application finale ou pour le programme Basic. Dans ce type de configuration l'assembleur est situé à l'adresse 24000, et le Basic à l'adresse 28000. Ainsi après G 24000 [R] il sera possible de choisir parmi les options suivantes :

- E	édition (donc introduction, modification de programme) ;
- A	assemblage ;
- G	Go ou lancement d'un programme assemblé ;
- N	numérotation des lignes du programme ;
- L	listing du programme après numérotation ;
- D xx jj	Delete, suppression de jj lignes après la ligne xx ;
- C xx yy zz	Copie des lignes xx à yy à placer derrière la ligne zz ;
- M xx yy zz	Mouvement des lignes xx à yy placées à la suite de la ligne zz ;
- R text1 text2 zz	Remplace text1 par text2 pendant zz lignes ;
- L texte	recherche et visualisation de "texte" éventuellement pour toutes les occurences ;

- + avancer le texte d'une page ;
- - reculer le texte d'une page ;
- LF la touche Line Feed permet le passage à la ligne suivante ;
- [R] la touche Return permet d'entrer une commande quelconque ;
- S Save, le texte introduit sera déchargé sur l'ordinateur hôte.

Enfin, quelques commandes supplémentaires permettent de traiter le corps du texte : suppression ou insertion d'une ou plusieurs lignes par appui sur une touche dédiée à ces usages, insertion ou suppression à l'intérieur d'une ligne ; en résumé, une vingtaine de commandes faciles à apprendre et qui assurent une mise au point très aisée. Il peut même être avantageux, dans le cas où un ordinateur hôte est connecté, de ne travailler qu'à partir de cet éditeur, pour introduire n'importe quel texte (du Fortran, par exemple), texte que l'on envoie ensuite par la commande SAVE.

L'assemblage est une opération en deux passages et, à notre fréquence d'horloge (8 MHz), 100 à 200 instructions par seconde peuvent être traitées. A la fin de l'assemblage, l'option G est automatiquement proposée, si aucune erreur n'a été détectée. Sinon, l'éditeur pointe sur la première instruction fautive ; ce système conversationnel permet une mise au point encore plus simplement qu'un micro-ordinateur utilisant le Basic.

EMULATION D'AUTRES CONSOLES

Ce système très efficace, est programmé en fonction de la console sur laquelle il est connecté et il lui revient toute la gestion de l'écran (effacement, saut en début ou fin de page, etc.). En contrepartie, il fonctionne mal, voire pas du tout, sur une console prise au hasard. En particulier, les commandes d'écran suivantes doivent exister obligatoirement :

- Curseur 'Home' (en haut à gauche) ;
- Curseur droite et gauche ;
- Curseur bas de page ;
- [R] Return ;
- LF Line Feed ;
- effacement d'écran ou effacement jusqu'au bas de l'écran.

Une solution, provisoire, va consister à : reloger l'assembleur-éditeur en Ram par BM 24000 28000 18000 [R] puis à modifier tous les codes de contrôle erronés par ceux correspondant à la

nouvelle console (les adresses de ces codes sont connues et un exemple complet figure en annexe). Par exemple, sur une VT52 de Digital Equipment "Curseur home" est traduit par quatre octets, 0000481B, sur une SFENA-25 par la séquence 2020591B ; or l'assembleur-éditeur attend ces codes en 1B5D6, dans la version relogée. Il suffira donc de modifier cette zone par :

MM 1B5D6;L [R]

00004B1B ? 2020591B [R]

et ainsi de suite pour les autres codes. On utilisera ensuite l'assembleur par G 18000 [R] . On pourrait écrire un petit programme chargé par LO2 ou LO4 qui assumerait toutes ces tâches, sans passer par le moniteur. Bien entendu, si une console précise est affectée à ce micro-ordinateur, il est alors plus commode de créer les Eprom adéquates.

CONVENTIONS DIVERSES

- Une instruction est écrite en majuscules et se termine par un point-virgule.
- Une ligne peut comporter plusieurs instructions.
- Les opérandes sont séparés par une virgule, ainsi :

  MOVE.B 4,D1;
  (mettre 4 dans le registre D1, considéré comme un seul octet). Cette opération travaille sur un octet (.B signifie 'BYTE'). D'une façon générale, les opérandes d'une instruction sont de même longueur, qui peut être : le 'byte' ou l'octet, suffixe .B ; le mot, suffixe .W (Word) ; le mot long, suffixe .L (Longword).

- La taille de l'opérande, lorsqu'elle est précisée dans le corps de l'instruction, annule, pour cette instruction toute autre convention. En effet, il est possible de préciser par la pseudo-instruction SIZE 1 (SIZE 2 ou SIZE 4) la dimension implicite retenue. Lorsque la taille de l'opérande est connue par l'assembleur, à la suite d'une réservation de cette zone, la longueur de l'opération n'a pas à être précisée. Sinon, en l'absence de toute indication, la taille .L sera assumée.

- Les constantes peuvent s'écrire sous forme **décimale**, **hexadécimale**, **ascii** ; ainsi les écritures suivantes sont équivalentes :

  MOVE 65,D1; MOVE $41,D1; MOVE 'A',D1; (SIZE 1 pour les trois écritures)

- Une adresse absolue doit être précédée du symbole 'arrobas' : @ ($40) :

  MOVE.B @$1000,D1; } mettre l'octet d'adresse réelle
  MOVE.B @4096,D1; } $1000 dans l'octet de droite de D1

- **Etiquette, nom de paragraphe** : c'est un nom quelconque suivi de ':' (deux points)

  RETOUR: MOVE 1,D4;
  (seules les quatorze premières lettres de l'étiquette sont significatives).

Il y a donc quelques différences entre cet assembleur Force-Computer et celui développé par Motorola dans lequel : une constante doit commencer par # (dièse) (exemple MOVE #65,D1) ; une adresse réelle est une simple valeur numérique ($2000 représente l'octet 2000) ; les instructions ne se terminent pas par ';' (une instruction par ligne seulement) ; une étiquette n'est pas suivie de ':', mais doit être dans la zone étiquette.

L'assembleur proposé ici est donc plus logique et plus souple ; les conventions d'écriture des constantes sont plus simples (20 au lieu de #20) car l'écriture d'adresses physiques réelles reste l'exception. Il n'y a donc pas lieu de privilégier ce type d'écriture rare au détriment des constantes d'usage systèmatique. Quant aux ponctuations différentes, elles ne modifient que la présentation.

Quoiqu'il en soit, il est très facile de passer de l'un à l'autre par des commandes très simples d'éditeur de texte ; il n'y a pas de différences de langages, mais de simples conventions d'écriture. Par ailleurs, le moniteur proposé étant celui développé par Motorola, le désassemblage sera fait selon la syntaxe de l'assembleur Motorola.

LES CODAGES BINAIRE, BCD, HEXADECIMAL, ASCII

Ce sont les principaux types de données qui vont être traités par un programme en assembleur. Il est donc primordial d'expliciter ces différents codes, qu'un programmeur en langage évolué n'a, en principe, pas à connaître.

Codage numérique binaire

Une donnée binaire sera structurée en 8, 16 ou 32 bits (octet, mot, mot-long).

Elle se présente, comme une donnée décimale, avec les forts en tête. Les poids sont ici les puissances de 2 : 2^0, 2 etc. Ainsi :

$00011001=25_{10}$ $01000001=65_{10}$

Les nombres négatifs commencent par 1 et sont codés "complément à 2". La méthode pour obtenir l'opposé d'un n est la suivante : changer les bits '1' par des bits '0' et inv ment ; ajouter 1 au résultat. Ainsi :

$00000001=1_{10}$; $11111110+1=11111111=-1_{10}$

$01111111=127_{10}$; $10000000+1=10000001=-127_{10}$

On passe d'un nombre négatif à son opposé par la mêm thode :

$10000001=-127_{10}$; $01111110+1=01111111=127_{10}$

$11111111=-1_{10}$; $00000000+1=00000001=+1_{10}$

Cette méthode s'étend naturellement à un nombre quelc de bits. Ce code est particulièrement efficace dans les calcu ce qui justifie son choix universel.

Une autre méthode pour obtenir le complément d'un nor X (soit -X) est de remarquer qu'un nombre négatif peut s'in préter comme une somme algébrique binaire ainsi : tous les sont affectés de leur poids binaire, sauf le premier, considér plus comme négatif.

11111111=-1 ; 10000000=-128 ; 01111111=127

on a bien -128+127=-1

Autre exemple :

10000001=10000000+00000001=-128+1=-127

Ces exemples supposent que ces nombres soient codés sur seul octet. Dans le cas d'un mot, on écrira que le poids fort vaut $-2^{15}=-32768$ et pour un mot long : $-2^{31}=-2147483648$. E on constate que la capacité de stockage des valeurs numériqu est selon la taille :

- $-128 \leqslant X \leqslant +127$ pour un octet ;
- $-32768 \leqslant X \leqslant +32767$ pour un mot de 16 bits ;
- $-2147483648 \leqslant X \leqslant 2147483647$ pour un mot long de 32

Transport du signe : puisque le signe d'une quantité déper de la façon dont on la considère (quant à sa taille) les opérat doivent se faire sur des zones de même longueur. Il sera donc

nécessaire de rallonger les opérandes trop courts. Pour un nombre positif, il suffit de rajouter autant de bits à 0 en tête, et pour un nombre négatif, on rajoutera le nombre de bits voulus, mais à 1. Ainsi les octets 01111111=127 et 1000001=-127 peuvent, s'ils doivent être manipulés avec des mots, être rallongés en mots : 0000000001111111=127 et 1111111110000001=-127 par la règle précédente. Une instruction (EXT) fera cette transformation sans problème. La représentation dans ce code des nombres décimaux est immédiate ; les poids après la "virgule" sont 2^{-1}, 2^{-2}, 2^{-3}... Ainsi 11111111,11111111 = -2^{-8} ; 00000000,00000001 = 2^{-8} et la somme de ces deux quantités est bien nulle. La règle précédente s'applique. Le programmeur a toute latitude pour placer la virgule là où il le désire, par de simples opérations de cadrage.

Codage BCD (décimal codé binaire)

Il est possible de considérer un octet comme étant formé de deux fois quatre bits. On peut alors placer un chiffre de 0 à 9 sur ces quatre bits, donc deux chiffres par octet, ainsi : 97_{10} =10010111 ; 22_{10} =00100010. Cette codification sera utilisée par l'assembleur pour traiter des nombres décimaux (addition et soustraction décimales). C'est le mode "packé", ou condensé, utilisé fréquemment dans d'autres langages de programmation.

Codage hexadécimal

C'est avant tout une présentation différente du codage binaire, ce dernier étant à la fois peu lisible et lourd à manipuler pour des valeurs numériques élevées. Chaque octet est scindé en deux fois quatre bits. Ces quatre bits peuvent prendre seize valeurs binaires de 0 à 15 que l'on note sur un caractère ainsi défini :

0000	= 0	0100	= 4	1000	= 8	1100	= C
0001	= 1	0101	= 5	1001	= 9	1101	= D
0010	= 2	0110	= 6	1010	= A	1110	= E
0011	= 3	0111	= 7	1011	= B	1111	= F

Ainsi un nombre hexadécimal de n digits est la représentation d'un nombre binaire de 4n bits :

$78F_{(16)}$ = 0111 1000 1111

Les entiers sur 32 bits des registres du 68000 seront manipulés sous forme hexadécimale et apparaîtront sur l'écran sous la forme de huit caractères de 0 à F. Bien que ce code ne se prête pas

facilement aux calculs (à cause de nos habitudes) on peut toutefois éviter le recours au code binaire, en écrivant directement les puissances de 16 correspondantes. Pour l'exemple précédent :

$78F = 7 \times 16^2 + 8 \times 16^1 + F \times 16^0 =$
$= 7 \times 256 + 8 \times 16 + 15 \times 1 = 1935$

Mais si le nombre hexadécimal est la représentation d'un nombre négatif, le mode de calcul précédent n'a plus d'intérêt. Le premier caractère rencontré pour un nombre négatif peut être 8,9,A,B,C,D,E,F qui tous contiennent '1' en tête. Le code ASCII qui va être étudié peut être considéré comme le codage hexadécimal d'un unique octet.

Codage ASCII

Purement conventionnel, il présente l'avantage d'être universel et de pouvoir être reconnu par toutes les machines. Les sept bits de poids faibles définissant 128 codes possibles sont seuls utilisés, le huitième bit servant de bit de contrôle (de parité) dans les transmissions. Les valeurs basses du code ne correspondent pas à des quantités représentables sur l'écran. L'assembleur, lui, ne considère que la valeur binaire de l'octet : on pourra donc traiter facilement les données alphabétiques. On a :

A<B<C<D, etc., ou 9<B ($39_{16} < 42_{16}$)

Le tableau des codes ASCII est donné en annexe.

LES PSEUDO-INSTRUCTIONS

Ce ne sont pas des ordres à proprement parler, mais des directives pour l'assembleur.

SIZE 1 (SIZE 2, SIZE 4) indique que, sauf indication contraire, dans le corps de l'instruction, la taille des opérandes est l'octet (respectivement le mot, le mot long) et cela jusqu'à la rencontre éventuelle du SIZE suivant.

LIST N précise la sortie du listing. Selon N, on peut avoir :

LIST 0 sortie des seules erreurs sur le terminal ;
LIST 1 listing d'assemblage sur le terminal ;
LIST 2 sortie en plus, sur l'imprimante ;
LIST 3 sortie de l'assemblage et de la table des symboles utilisés ;
LIST 4 sortie complète sur terminal et imprimante.

PAGE N selon N indique le nombre de lignes par page, pour l'imprimante (par défaut 60).

ORG adresse, demande que le binaire généré (ou code objet) soit implanté à partir de l'adresse précisée par ORG.

ORG $800; ORG 2048; représente la plus basse implantation possible sur notre plaquette, le début de la Ram (de $000 à $7FF) étant utilisé par le moniteur.

Il est possible d'avoir plusieurs ORG dans un programme, ainsi :

```
SOUPRO: ORG $3000;
.....
PRINCIPAL: ORG $4000;
```

Quelles que soient les modifications apportées dans SOUPRO, PRINCIPAL commencera toujours à l'adresse $4000. Il suffit de modifier l'adresse de ORG pour implanter le programme à un endroit plus judicieux, après nouvel assemblage.

EQU (équivalence) permet de donner une équivalence entre un symbole et une valeur explicite :

```
OUT:EQU @$21C14;
JSR OUT; (exécution du sous-programme OUT)
```

ces deux lignes pourraient être remplacées par : JSR@$21C14.
autre exemple :

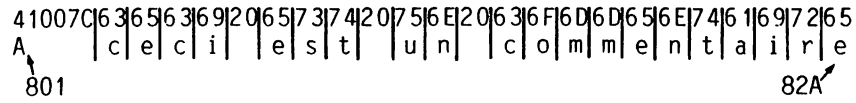
```
REGISTRE:EQU.L D0/D3/D5-D7/A4/A6;
.....
MOVEM REGISTRE,-(A7);
.....
MOVEM (A7)+,REGISTRE;
```

Ici l'intérêt dans la concision d'écriture est plus évident puisque REGISTRE représente une liste compliquée à écrire. On peut dire que EQU facilite avant tout l'écriture et la mise au point d'un programme.

DC (Define Constant) permet de définir une constante. Une constante est, soit une valeur numérique (pouvant tenir sur quatre octets au maximum), soit une chaîne de caractères ASCII. Les valeurs ainsi réservées sont toujours placées à des emplacements consécutifs.

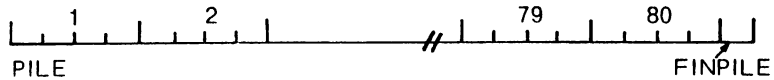
```
PIPO: DC.B 'A';
DC.W $7C;
DC 'ceci est un commentaire';
```


Il faudra toutefois vérifier que les mots et mots longs réservés commencent bien à des adresses paires, quitte à rajouter un remplissage d'un octet pour éviter le message de l'assembleur 'ALIGNMENT ERROR'. Nous allons trouver en mémoire, si PIPO est à l'adresse 801 :



La réservation de mémoire se fait par **DS** (Define Storage) :

```
PILE: DS.L 80 ; réservation de 80 mots longs ou 320 octets
FINPILE:EQU * ; * représente l'adresse courante, donc FINPILE
                est l'adresse de l'octet qui suit immédiatement
                la fin de PILE :
```



On verra, que selon les modes d'adressage utilisés, il peut être nécessaire de connaître l'adresse de fin d'une zone réservée ; c'est un des rôles de l'argument '*'. Il est préférable de placer les réservations de constantes et de données à la fin du programme, plutôt qu'au début ; on évite ainsi les défauts d'alignement et l'adresse de lancement du programme est alors celle spécifiée par ORG. C'est celle que prend la commande G de l'assembleur-éditeur à la suite d'un assemblage correct.

Enfin, le programme n'a pas à connaître les emplacements des mémoires sur lesquelles il va travailler, de façon absolue. Des instructions telles que LEA (Load Effective Address) lui fournissent directement les adresses des données qu'il doit traiter. Dans ces conditions, la relogeabilité d'un programme ne pose pas de problème.

CHAPITRE 4 généralités sur l'adressage

Six types d'adressage de base conduisent à quatorze modes d'adressage qui sont résumés dans le **tableau 4.1**.

Avec les significations suivantes :

EA = adresse effective
An = registre adresse ; Dn = registre de données
Xn = registre données ou adresse utilisé en registre d'index
SR = registre d'état
PC = compteur programme
() = contenu de
d_8 , d_{16} = déplacement sur huit ou seize bits
N = taille (1 : octet, 2 : mot, 4 : mot-long)
← = transféré dans

Les modes d'adressage sont donc nombreux sur le 68000 et ces diverses méthodes vont permettre l'accès à une partie de la mémoire (octet, mot, mot-long) de multiples façons ; il s'agira, pour le programmeur de choisir celle qui semble la plus adaptée au programme, les notions de relogeabilité, de répétitivité, de vitesse, d'encombrement pouvant être déterminantes.

LE MODE D'ADRESSAGE ABSOLU

L'instruction porte directement sur l'adresse.

CLR.B @$1000 met à zéro ($00) l'octet d'adresse réelle $1000.

CLR.W @$1000 met à zéro les octets d'adresses $1000 et $1001.

CLR.L @$1000 met à zéro les octets d'adresses $1000, $1001, $1002, $1003.

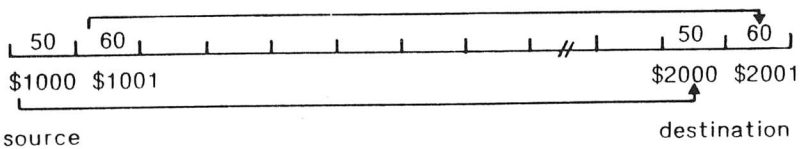
MODE	ADRESSE EFFECTIVE
Adressage direct registre	
Données registre direct	EA = Dn
Adresse registre direct	EA = An
Adressage données absolu	
Absolu court	EA = (Mot suivant)
Absolu long	EA = (2 mots suivants)
Adressage compteur programme relatif	
Relatif avec déplacement	EA = (PC) + d_{16}
Relatif avec index et déplacement	EA = (PC) + (Xn) + d_8
Adressage registre indirect	
Registre indirect	EA = (An)
Registre indirect post-incrémenté	EA = (An), An ← An + N
Registre indirect prédécrémenté	AN ← An - N, EA = (An)
Registre indirect avec déplacement	EA = (An) + d_{16}
Registre indirect indexé avec déplacement	EA = (An) + (Xn) + d_8
Adressage données immédiat	
Immédiat	Données = mots suivants
Immédiat rapide	Données implicites
Adressage implicite	
Registre implicite	EA = SR, USP, SP, PC

Fig. 4.1 - Le 68000 dispose de quatorze modes d'adressage.

Attention : lorsque l'opération porte sur un mot ou un double mot, l'adresse doit obligatoirement être **PAIRE.** L'adresse précisée peut être codée sur 16 bits (deux octets) ou, en mode 'long' sur 24 bits (trois octets), ce dernier cas permettant d'accéder au-delà de 64 K octets alors que le codage court permet d'adresser de 0 à $2^{16}-1$ (=64535) donc jusqu'à 64 K octets seulement.

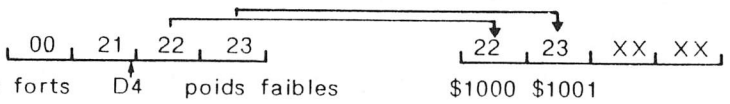
Les instructions à deux opérandes comportent un **opérande source** (le premier nommé) et un **opérande destination** (le second), **qui** en principe **est modifié** par l'instruction :

MOVE.W @$1000,@$2000;



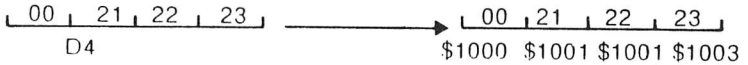
L'adressage registre direct : la seule différence avec l'adressage absolu est que l'opérande source est un registre.

MOVE.W D4, @$1000 transfère les 16 bits de plus faibles poids (LSB) de D4 en $1000 et $1001



alors que :

MOVE.L D4, @$1000; condult à :



On peut également avoir un registre comme destination :

MOVE.B @$1000,D7;

Mais si on utilise un registre d'adresse comme destination, il faut faire un MOVE-ADRESSE :

MOVEA.L @$1000,A5; } interdiction de transférer un octet
MOVEA.W @$1000,A4; } (MOVEA.B interdit)

Avec les valeurs placées précédemment en $1000, $1001, etc., on aura :

dans A5 : 00 21 22 23

dans A4 : 00 00 00 21

Dans le cas d'un transfert d'un mot, le registre adresse est étendu systématiquement à 32 bits. Cela est logique et fournit un code économique.

Commodités d'écriture dans l'adressage absolu

L'adressage des circuits périphériques (ACIA, PTM, etc.) se fait naturellement en fonction de leur implantation sur la carte. Nous trouvons, par exemple, sur la nôtre, le PIA aux adresses $05CEF1 (registre de données) et $05CEF43 (registre de contrôle). On peut alors écrire :

```
ORG $800;
PIA:EQU @$05CEF1;
.....
MOVE.B 0,PIA+2;
```

Cette instruction référence l'adresse $05CEF1 + 2 = $05CEF3. On a donc procédé à un adressage relatif (au niveau de l'écriture) par rapport à une position mémoire ; il est possible d'utiliser une expression complexe comprenant les symboles '+', '-', '*', '/', (dans ces deux derniers cas les opérandes ne doivent pas dépasser 32767).

L'intérêt de ce mode d'adressage est la minimisation de risque d'erreur (le PTM possède seize registres accessibles par huit adresses distinctes qu'il serait fastidieux d'adresser de façon absolue registre par registre). En contrepartie, il est peu économique quant au code généré.

LE MODE D'ADRESSAGE REGISTRE INDIRECT

C'est le mode le plus général, il comporte quatre variantes.

Adressage registre indirect

MOVE.W D1,(A3);

Le registre A3 contient l'adresse à laquelle s'effectuera l'instruction. Soit $00001000 le contenu de A3 et $40414243 celui de D1, après l'instruction précédente, on obtient :

42	43	XX	XX
$1000	$1001	$1002	$1003

$1002 et $1003 sont inchangés.

Tandis qu'à la suite de MOVE.L D1,(A3) on obtiendra :

40	41	42	43
$1000	$1001	$1002	$1003

Avec les mêmes données initiales et l'opérande (A3) en source :

MOVE.W (A3),D5; D5 = $XXXX4243 après l'instruction (XX signifie inchangé)

On se souviendra que l'on ne peut adresser un mot ou un mot-long que par adresses paires, et cela pour tous les modes d'adressage (sinon, à l'exécution on aura le message suivant : ADDRESS TRAP ERROR).

Ce mode d'adressage nécessite donc d'avoir placé dans le registre adresse, ici (A3), la valeur de l'adresse que l'on va utiliser, par exemple par une instruction du type suivant :

MOVEA.L $1000,A3;

MOVE.W (A3),D5;

Adressage indirect avec déplacement

L'adresse précisée par le registre sera augmentée du déplacement éventuellement signé, au plus sur 16 bits, ($-32768 \leqslant d \leqslant 32767$).

MOVE.W 50(A3),D5; soit A3 = $00001000

L'adresse source sera donc $1000 + 50_{10} soit $1032. Les octets d'adresse $1032 et $1033 seront transférés dans les poids faibles de D5.

7E	7F	$XXXX7E7F
$1032	$1033	D5

après l'opération.

Le déplacement, pouvant être signé, sera ajouté au contenu de A3, après avoir été étendu de 16 à 32 bits automatiquement. Il peut être écrit en base 16 ou 10. Naturellement, ce mode est permis pour les opérandes **source** et **destination**.

Adressage indirect avec index

Dans cette variante très utile, l'adresse finale s'obtient en ajoutant au contenu du registre d'adresse, celui d'un registre d'index.

MOVE.W (A3,D1),D5 soit A3 = $00001000 et D1 = $00000020

L'adresse réelle source est donc A3 + D1 = $00001020. Les octets d'adresse $1020 et $1021 vont aller dans les poids faibles de D5.

04	06	$XXXX0406
$1020	$1021	D5

après l'instruction.

Le registre d'index peut être soit un registre de données D0-D7 soit un registre d'adresse A0-A6. Il est possible d'utiliser un index sur 16 bits seulement :

MOVE.W (A3,D1.W),D5

On précise par D1.W que seuls les 16 bits de poids faible sont utilisés par l'index. Cette quantité sera étendue à 32 lors du calcul de l'adresse réelle. Si le poids fort (bit 16) est à **un** cette quantité sera évidemment négative.

Adressage indirect avec index et déplacement sur 8 bits signés

C'est la forme la plus complète de ces modes d'adressage.

L'adresse réelle (ou adresse effective notée AE) sera obtenue par deux additions :

```
MOVE.W 10(A3,D1.W),D5;   soit A3 = $00001000 et
                             D1 = $00000020
```

Le calcul de l'adresse effective donne AE = $1000 + $20 + 10_{10} = $102A. Le mot d'adresse $102A sera transféré dans les poids faibles de D5. L'index peut être court (D1.W) ou long (D1.L) et le déplacement signé sur un octet devra être compris entre -128_{10} et $+127_{10}$. Ce type d'adressage est universel et permet les traitements les plus divers.

LE MODE D'ADRESSAGE IMMEDIAT

C'est la possibilité de transférer directement une donnée, sur 8, 16 ou 32 bits vers un opérande destination quelconque.

```
MOVE.B $41,D0;

MOVE.B 'A',D0;

MOVE.B 65,D0;    représentent la même instruction immédiate.
```

Lorsque l'opérande de destination est un registre de données ou une position mémoire, la donnée immédiate peut être sous la forme octet, mot, ou mot-long. Lorsque la destination est un registre d'adresse, la donnée doit être obligatoirement sous la forme mot ou mot-long (transmettre un seul octet est interdit).

En outre, lors du transfert d'un mot, le registre adresse entier est concerné car le mot subit une extension sur 32 bits avant transfert :

```
MOVEA.W $1000,A2    mettra $00001000 dans A2
```

Déplacement, addition et soustraction rapides

Dans les variantes MOVEQ, ADDQ, SUBQ (Quick pour rapide), la destination est un registre de données (adressage mémoire possible sur FORCE). MOVEQ ne peut transférer qu'un seul octet... avec EXTENSION sur 32 bits !

```
MOVEQ -1,D5;      met $FFFFFFFF dans D5

MOVEQ 1,D5;       met $00000001   dans D5
```

Cette extension automatique n'a lieu sur aucune autre instruction, sur les registres Dn.

En ce qui concerne l'addition et la soustraction rapides, l'opérande doit être compris entre 1 et 8 et l'opération peut agir sur 8, 16 et 32 bits.

```
ADDQ 1,D4;

SUBQ.W 2,1(A2);  peut éventuellement être accepté
```

L'adressage immédiat mémoire peut toujours avoir comme destination une adresse mémoire (ADDI.W 267, $1000) et un opérande source sur 8, 16 ou 32 bits.

Toutes les instructions n'ont pas de variantes rapides ou immédiates : le tableau général des instructions donné plus loin apportera les précisions nécessaires. Du point de vue de l'efficacité, on choisira d'abord la variante rapide, puis la forme ordinaire, et en tout dernier lieu l'adressage mémoire immédiat.

```
ADDQ > ADD > ADDI
```

LE MODE D'ADRESSAGE INDIRECT AVEC POSTINCREMENTATION OU PREDECREMENTATION

Postincrémentation

Ce type d'adressage est particulièrement original :

```
MOVE.B $41,(A0)+;
```

va effectuer la séquence suivante (soit A0 = $00001000) :

- la valeur $41 est placée à l'adresse $1000 ;
- le registre A0 est incrémenté de 1 donc prend la valeur $1001.

On pressent que cette instruction va jouer un rôle important dans les boucles et cycles répétitifs. Si la taille de l'opération est le mot, l'incrément est alors de 2 et si la taille est le mot-long, l'incrément automatique est de 4. Comme toujours, le registre concerné doit pointer des adresses paires si la taille est le mot ou le mot-long.

Exception : le registre A7, réservé à la pile, ne peut jamais être incrémenté de 1, même si la taille précisée est l'octet.

Prédécrémentation

```
MOVE.B $41,-(A0);
```

la séquence est la suivante (soit A0 = $00001001) :

- A0 est décrémenté de 1 car la taille de l'instruction est l'octet : A0 = $00001000 ;
- l'opération est exécutée : l'octet d'adresse 1000 prend la valeur $41.

Même restriction que précédemment pour le registre A7 et les décréments s'ajustent à la taille de l'instruction.

L'ADRESSE COURANTE : L'OPERANDE '*'

Ecrivons un élément de programme qui va placer des 'A' dans les positions $901 à $951.

```
ORG $800;SIZE 1;        implantation du programme objet en $800 ;
                        taille implicite octet.
MOVE.L $51,D0;
MOVEA.L $901,A0;
PIP: MOVE 'A',(A0)+;    mettre 'A' en $901 ; A0 passe à la va-
                        leur A0 + 1.
SUBQ.L 1,D0;            soustraire 1 à D0.
BNE PIP                 si D0 n'est pas nul aller à PIP.
TRAP 14;                retour au moniteur en fin d'exécution.
```

L'assemblage, outre la liste précédente, peut fournir les adresses d'implantation des instructions. Nous trouvons :

```
80C PIP:MOVE 'A',(A0)+;
        SUBQ.L 1,D0;
812     BNE PIP
```

pour aller à PIP on va de l'adresse programme 812 à 80C ; on recule de 6. La séquence suivante est donc équivalente à la précédente :

```
MOVE 'A',(A0)+;
SUBQ.L 1,D0;
```

BNE *-6 (branchement à l'adresse courante moins 6). On peut écrire également BNE -6(*).

L'intérêt principal de l'utilisation de l'opérande adresse courante est de minimiser le nombre de symboles inutiles dans un programme, lorsque la séquence est parfaitement claire et surtout de permettre l'écriture en code indépendant de la position (le programme est alors relogeable). L'écriture des EPROM (voir chapitre 13) exige l'utilisation de l'adresse courante. La commande du

moniteur DC permet d'ajuster le déplacement relatif, par la simple différence des adresses d'implantation. Enfin, des instructions PEA et LEA permettront encore de gérer sans problème les adresses des données ou des paragraphes, indépendamment de leur implantation absolue.

En conclusion, les modes d'adressage du 68000 sont particulièrement riches et recouvrent toutes les applications possibles, mais pratiquement peuvent suffire : l'adressage absolu, l'adressage direct registre (opérandes : les registres), l'adressage indirect registre, le mode incrémental, l'adressage indirect registre avec index et déplacement.

L'addition décimale (ABCD) et la soustraction décimale (SBCD) utilisent le mode prédécrémenté.

RETOUR AU MONITEUR : LES ROUTINES

Les routines d'entrées/sorties

Les fonctions, ou routines, destinées aux entrées/sorties permettent de simplifier la programmation depuis et/ou vers la console, le port maître ou l'imprimante. Elles seront utilisées soit pour rentrer des données sous forme d'une chaîne ou d'un simple octet, soit pour restituer des résultats. Elles sont standard, bien que d'autres versions du moniteur M68000 puissent présenter quelques différences de détails, soit au niveau du fonctionnement, soit sur les adresses d'implantation.

La routine OUTCH (adresse $21E24) sort le caractère contenu dans l'octet de droite du registre D0 vers la console. Exemple d'utilisation :

```
OUTCH:EQU @$21E24
JSR OUTCH  (ou directement sans passer par EQU : JSR @$21E24)
TRAP 14;
```

A l'exécution, (supposons D0 = $4345FF41) sera affiché un 'A' correspondant à $41. JSR (Jump to Subroutine) provoque le déroulement de la routine spécifiée, avec retour à la suite du programme après achèvement de la routine.

La routine INCH (adresse $21FB0) place le caractère frappé sur le terminal dans le registre D0 (octet de droite). Exemple :

```
INCH:EQU @$21FB0;
```

```
JSR INCH;      (attente d'un caractère sur le terminal)
TRAP 14;
```

En réalité, les deux routines précédentes peuvent être utilisées sur les deux ports (console et hôte), il faut donc auparavant préciser sur quel port ira l'information par l'emploi d'une autre routine.

```
JSR @$21C7C;        entrées/sorties sur la console
JSR OUTCH           sortie sur la console
```

ou bien

```
JSR @$21CE8;        entrées/sorties sur le port hôte
JSR OUTCH           sortie sur le port hôte
```

Les routines $21C7C et $21C8E utilisent le registre A0.

La routine OUTPUT (adresse $21C14) écrit sur la console la chaîne dont l'adresse de début est contenue dans A5 et l'adresse de fin dans A6 (moins 1). Exemple : soit la chaîne 'ABCDEFGHIJKLM' implantée en $1000. 'M' est à l'adresse $100C

```
OUTPUT:EQU @$21C14;
MOVEA.L    $1000,A5;
MOVEA.L    $100D,A6; (A6 moins 1 = $100C)
JSR OUTPUT;
TRAP 14;
```

Ce programme provoquera la sortie sur le terminal de la chaîne ABCDEFGHIJKLM.

La routine PORTIN1 (adresse $21CA8) permet l'introduction d'une chaîne de caractères, (jusqu'à la rencontre de [R]), qui s'implante à l'adresse contenue dans A5. On doit avoir au début A5 = A6.

La routine PRCRLF (adresse $21D70) permet une sortie sur l'imprimante, avec le même fonctionnement qu'OUTPUT.

On a également des routines équivalentes sur le port hôte :

PORTIN20 (adresse $21FDE) permet la lecture d'une chaîne 'hôte'.

OUTPUT21 (adresse $21C34) sort une chaîne (comme OUTPUT) sur l'hôte.

Les routines de retour au moniteur

Elles sont au nombre de deux.

WARMSTRT (adresse $20232) retourne au moniteur. Elle est équivalente à TRAP 14.

COLDSTRT (adresse $20146) initialise le moniteur (avec donc restauration des zones modifiées : rôle semblable au bouton poussoir Abort).

Tous les noms de ces routines sont purement symboliques et ne sont pas connus, en tout cas, par l'éditeur-assembleur. On les utilisera dans un programme avec les noms que l'on veut (mais, il n'y a guère intérêt à les changer, sinon pour les raccourcir), mais sans oublier de faire une équivalence (EQU) pour associer le nom et l'adresse physique réelle.

Le **programme n° 1** donné à titre d'exemple affiche le caractère frappé à la console ; il s'achève à la rencontre du caractère '/'.

```
                        ORG $2000;LIST 3; /* EMPLOI DES ROUTINES */
                         /* SAISIE ET ECRITURE D UN CARACTERE VIA D0 */
                        REGIS:EQU D0;
                        OUT:EQU @$21E24;
                        IN:EQU @$21FB0;
                        ADR:EQU @$21C7C;
   2000    4EB900021C7C JSR ADR;
   2006    4EB900021FB0 RETOUR:JSR IN;
   200C    4EB900021E24 JSR OUT;
   2012    B0BC0000002F CMP $2F,REGIS; /* ARRET SUR  /  */
   2018    66EC         BNE RETOUR;
   201A    4E4E         TRAP 14;;

ADR                                 6               @$21C7C
IN                                  5               @$21FB0
OUT                                 4               @$21E24
REGIS                               3               D0
RETOUR                              8               $2006  FC-R.


   0 ERRORS
a77 G
FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
12345678?0
QWERTYUIOPZXCVBNM,./
FORCEMON 2.2 >
```

Programme 1 - Utilisation des routines du moniteur.

CHAPITRE 5 | le jeu d'instructions

Les 56 instructions de base du 68000 figurent dans le **tableau 5.1.** Certaines de ces instructions possèdent une ou plusieurs variantes. Une variante peut être une instruction spécialisée (par exemple MOVE et MOVEA, cette dernière étant utilisée uniquement pour les transferts vers un registre d'adresse). Ce peut être aussi une forme plus efficace - moins de place mémoire et plus grande rapidité - que l'on ne peut employer que dans des cas très précis (ainsi, les variantes rapides ne peuvent avoir qu'un opérande immédiat, compris entre 1 et 8), ou, au contraire un peu moins efficace, mais plus souple dans l'écriture (cas des variantes immédiates). Il reste aussi le cas où l'instruction elle-même est assez différente, agissant en particulier sur le bit X d'extension, qui devient donc une partie d'un opérande. Le **tableau 5.2** indique ces diverses variantes.

Mnémo.	Description	Mnémo.	Description	Mnémo.	Description
ABCD	addition décimale étendue	EOR	OU exclusif logique	PEA	empilement de l'adresse effective
ADD	addition	EXG	échange de registres	RESET	mise à zéro des circuits externes
AND	ET logique	EXT	extension de signe	ROL	décalage circulaire gauche non étendu
ASL	décalage arithmétique gauche	JMP	saut inconditionnel	ROR	décalage circulaire droite non étendu
ASR	décalage arithmétique droite	JSR	saut à un sous programme	ROXL	décalage circulaire gauche étendu
B_{CC}	branchement conditionnel	LEA	chargement d'adresse effective	ROXR	décalage circulaire droite étendu
BCHG	test de bit et inversion	LINK	chaînage de pile	RTE	retour d'exception
BCLR	test de bit et mise à zéro	LSL	décalage logique gauche	RTR	retour et restaure
BRA	branchement inconditionnel	LSR	décalage logique droite	RTS	retour de sous programme
BSET	test de bit et mise à un	MOVE	transfert	SBCD	soustraction décimale étendue
BSR	branchement à un sous programme	MOVEM	transfert de plusieurs registres	S_{CC}	mise à un conditionnelle
BTST	test de bit	MOVEP	transfert données périphériques	STOP	arrêt d'exécution du programme
CHK	test de registre aux limites	MULS	multiplication signée	SUB	soustraction
CLR	mise à zéro de l'opérande	MULU	multiplication non signée	SWAP	échanger les moitiés de registre
CMP	comparaison	NBCD	complément à 10 avec extension	TAS	test opérande et établis. d'1 indicateur
DB_{CC}	test cond. décrémentation et branchement	NEG	complément à 2	TRAP	trappe
DIVS	division signée	NOP	pas d'opération	TRAPV	trappe sur dépassement
DIVU	division non signée	NOT	complément logique	TST	test d'une opérande
		OR	OU logique	UNLK	rupture du chaînage de pile

Fig. 5.1 - Le jeu d'instructions du 68000.

Type d'instruction	Variante	Description	Type d'instruction	Variante	Description
ADD	ADD	addition	MOVE	MOVE	transfert
	ADDA	addition adresse		MOVEA	transfert adresse
	ADDQ	addition rapide		MOVEQ	transfert rapide
	ADDI	addition immédiate		MOVE depuis SR	transfert du registre d'état
	ADDX	addition étendue		MOVE vers SR	transfert vers registre d'état
				MOVE vers CCR	transfert vers registre codes condition
				MOVE USP	transfert pointeur de pile utilisateur
AND	AND	ET logique			
	ANDI	ET immédiat	NEG	NEG	complémentation
				NEGX	complémentation étendue
CMP	CMP	comparaison	OR	OR	OU logique
	CMPA	comparaison d'adresse		ORI	OU immédiat
	CMPM	comparaison mémoire			
	CMPI	comparaison immédiate	SUB	SUB	soustraction
				SUBA	soustraction adresse
				SUBI	soustraction immédiate
EOR	EOR	OU exclusif		SUBQ	soustraction rapide
	EORI	OU exclusif immédiat		SUBX	soustraction étendue

Fig. 5.2 - Certaines instructions comportent des variantes.

Le **tableau général 5.3** est plus complet, puisqu'il donne non seulement le fonctionnement de l'instruction, mais aussi l'état des codes condition après exécution d'une instruction. D'une façon générale, **source** désigne l'opérande le premier nommé, et **destination** le second, lorsque l'instruction porte sur deux opérandes. Dans le cas d'un opérande unique, seul est nommé **destination**. Pendant le déroulement d'une instruction, **destination** est modifié, et **source** est inchangé. On peut réunir en classes d'opérations ces diverses instructions pour en faciliter l'étude. C'est la démarche qui va être suivie par la suite, entrecoupée de nombreux exercices, qui tous ont été assemblés et lancés. Dans ce dernier cas, nous trouverons tout à la fois : l'adresse d'implantation, le code objet produit par l'assembleur et le source-assembleur qui vient d'être traduit. Il sera ainsi possible de comparer les codes de diverses instructions (voisines, ou à variantes...).

Comme dans l'étude de tout langage, la compréhension complète d'un programme n'est jamais immédiate, mais se fait peu à peu, par des retours en arrière, des comparaisons avec des programmes voisins et surtout par la simulation manuelle de la suite des instructions. Enfin, même si des zones obscures subsistent, il est préférable de poursuivre l'étude, tant qu'elle reste possible.

LES INSTRUCTIONS SUR REGISTRES

Les opérandes sont donc les registres données (D0 à D7) ou adresse (A0 à A7). Ces opérations sont rapides (pas de calcul d'adresse), si bien que d'une façon générale, lors des calculs ou des manipulations intermédiaires, il sera toujours préférable d'opérer le plus longtemps possible à partir des registres.

EXG (échange de registres), mot long uniquement.

EXG Ri,Rj;

permute les contenus des registres Ri et Rj (Ri, Rj registres données ou adresse).

SWAP Di (Di registre de données uniquement)

Cette instruction échange le mot de poids fort avec le mot de poids faible. Elle accroît le nombre de registres utiles, lorsque 16 bits sont suffisants, au prix d'un contrôle précis sur la partie du registre utilisée.

EXT (extend)

Cette instruction rallonge une zone de un à deux octets (EXT.W) ou de deux à quatre octets (EXT.L) tout en conservant la valeur signée du contenu. Elle ne concerne que les registres de données. Soit D1 = 000000FF D1.B = FF = -1 on a :

EXT.W D1 D1 devient 0000FFFF (D1.W = FFFF = -1)

EXT.L D1 D1 devient FFFFFFFF (D1.L = FFFFFFFF = -1)

L'utilisation de EXT.L immédiatement aurait conduit à un résultat erroné, avec D1 inchangé, D1.B = FF = -1 et D1.L = 000000FF = 255.

L'instruction **MOVEM** assure le stockage et la restauration des registres par l'intermédiaire d'une PILE (étudiée plus loin). Elle permet d'accroître à l'infini le nombre de registres de travail, d'où une programmation et une mise au point plus efficaces. Outre une plus grande vitesse de traitement, la visualisation du déroulement du programme grâce au mode TRACE, qui affiche le contenu de tous les registres après une opération, est un argument supplémentaire.

QUELQUES PROGRAMMES DE MISE EN TRAIN

Ecriture de l'alphabet

Le **Programme n° 2** écrit, à partir de l'adresse $2000, les 26 lettres de l'alphabet. On sait que dans le code utilisé (ASCII) le 'A' vaut $41, B vaut $42, etc.

Mnémonique	Description	Fonctionnement	Codes condition				
			X	N	Z	V	C
ABCD	Addition décimale étendue	(Destination)$_{10}$ + (Source)$_{10}$ → Destination	•	U	•	U	•
ADD	Addition binaire	(Destination) + (Source) → Destination	•	•	•	•	•
ADDA	Addition d'adresse	(Destination) + (Source) → Destination	—	—	—	—	—
ADDI	Addition immédiate	(Destination) + données immédiates → Destination	•	•	•	•	•
ADDQ	Addition rapide	(Destination) + données immédiates → Destination	•	•	•	•	•
ADDX	Addition étendue	(Destination) + (Source) + X → Destination	•	•	•	•	•
AND	ET logique	(Destination) .\ (Source) → Destination	—	•	•	0	0
ANDI	ET immédiat	(Destination) .\ données immédiates → Destination	—	•	•	0	0
ASL, ASR	Décalage arithmétique	(Destination) décalé par < compte > → Destination	•	•	•	•	•
BCC	Branchement conditionnel	Si CC alors PC + d → PC	—	—	—	—	—
BCHG	Test de bit et changement	~ (< numéro de bit >) de destination → Z ~ (< numéro de bit >) de destination ~ < numéro de bit > de destination	—	—	•	—	—
BCLR	Test de bit et mise à zéro	~ (< numéro de bit >) de destination → Z 0 → < numéro de bit > → de destination	—	—	•	—	—
BRA	Branchement inconditionnel	PC + d → PC	—	—	—	—	—
BSET	Test de bit et mise à un	~ (< numéro de bit >) de destination → Z 1 → < numéro de bit > de destination	—	—	•	—	—
BSR	Branchement à un sous-prog.	PC → SP@ – ; PC + d → PC	—	—	—	—	—
BTST	Test de bit	~ (< numéro de bit >) de destination → Z	—	—	•	—	—
CHK	Test registre aux limites	Si Dn < 0 ou Dn > (< ea >) alors TRAP	—	•	U	U	U
CLR	Mise à zéro de l'operande	0 → Destination	—	0	1	0	0

Mnémonique	Description	Fonctionnement	Codes condition				
			X	N	Z	V	C
CMP	Comparaison	(Destination) – (Source)	—	•	•	•	•
CMPA	Comparaison d'adresse	(Destination) – (Source)	—	•	•	•	•
CMPI	Comparaison immédiate	(Destination) – Données immédiates	—	•	•	•	•
CMPM	Comparaison mémoire	(Destination) – Source	—	•	•	•	•
DBCC	Test condition, décrémentation et branchement	Si ~CC alors Dn – 1 → Dn ; si Du = – 1 alors PC + d → PC	—	—	—	—	—
DIVS	Division signée	(Destination) / (Source) → Destination	—	•	•	•	0
DIVU	Division non signée	(Destination) / (Source) → Destination	—	•	•	•	0
EOR	OU exclusif logique	(Destination) ⊕ (Source) → Destination	—	•	•	0	0
EORI	OU exclusif immédiat	(Destination) ⊕ Données immédiates → Destination	—	•	•	0	0
EXG	Echange de registres	Rx ↔ Ry	—	—	—	—	—
EXT	Extension de signe	(Destination) signe étendu → Destination	—	•	•	0	0
JMP	Saut inconditionnel	Destination → PC	—	—	—	—	—
JSR	Saut à un sous-programme	PC → SP@ – ; Destination → PC	—	—	—	—	—
LEA	Chargement adresse effective	Destination → An	—	—	—	—	—
LINK	Chaînage de pile	An → SP@ – ; SP → An ; SP + d → SP	—	—	—	—	—
LSL LSR	Décal. logique gauche/droite	(Destination) décalé de < compte > → Destination	•	•	•	0	•
MOVE	Transfert de données de source à destination	(Source) → Destination	—	•	•	0	0
MOVE → CCR	Transfert vers CCR	(Source) → CCR	•	•	•	•	•
MOVE → SR	Transfert vers registre d'état	(Source) → SR	•	•	•	•	•

* affecté 0 mis à zéro U indéfini
– non affecté 1 mis à un

Mnémonique	Description	Fonctionnement	Codes condition				
			X	N	Z	V	C
MOVE depuis SR	Transfert du registre d'état	SR → Destination	—	—	—	—	—
MOVE USP	Transf. pointeur de pile utilisateur	USP → An ; An → USP	—	—	—	—	—
MOVEA	Transfert adresse	(Source) → Destination	—	—	—	—	—
MOVEM	Transfert registres multiples	Registres → Destination (Source) → Registres	—	—	—	—	—
MOVEP	Transfert données périphériques	(Source) → Destination	—	•	•	0	0
MOVEQ	Transfert rapide	Données immédiates → Destination	—	•	•	0	0
MULS	Multiplication signée	(Destination)* (Source) → Destination	—	•	•	0	0
MULU	Multiplication non signée	(Destination)* (Source) → Destination	•	U	•	U	•
NBCD	Prendre l'opposé décimal étendu	0 − (Destination)$_{10}$ − X → Destination	•	•	•	•	•
NEG	Complémentation à 2	0 − (Destination) → Destination	•	•	•	•	•
NEGX	Complémentation à 2 étendue	0 − (Destination) − X → Destination	—	—	—	—	—
NOP	Non opération		—	•	•	0	0
NOT	Complément logique	~ (Destination) → Destination	—	•	•	0	0
OR	OU logique inclusif	(Destination) v (Source) → Destination	—	•	•	0	0
ORI	OU logique immédiat	(Destination) v données immédiates → Destination	—	—	—	—	—
PEA	Empilement adresse effective	Destination → SP@−	—	—	—	—	—
RESET	Mise à zero des circuits externes		—	•	•	0	•
ROL, ROR	Décalage circulaire non étendu	(Destination) décalé de < compte > → Destination	•	•	•	0	•
ROXL, ROXR	Décalage circulaire étendu	(Destination) décalé de < compte > → Destination	•	•	•	•	•
RTE	Retour d'exception	SP@+ → SR ; SP@+ → PC					

Mnémonique	Description	Fonctionnement	Codes condition				
			X	N	Z	V	C
RTR	Retour et restaure codes condition	SP@+ → CC ; SP@+ → PC	•	•	•	•	•
RTS	Retour de sous-programme	SP@+ →	—	—	—	—	—
SBCD	Soustraction décimale étendue	(Destination)$_{10}$ − (Source)$_{10}$ − X → Destination	•	U	•	U	•
Scc	Mise à un conditionnelle	Si CC vrai alors 1 dans la destination Si CC faux alors 0 dans la destination	—	—	—	—	—
STOP	Chargement registre d'état et suspension d'exécution du prog.	Données immédiates SR ; STOP	•	•	•	•	•
SUB	Soustraction	(Destination) − (Source) → Destination	•	•	•	•	•
SUBA	Soustraction	(Destination) − (Source) → Destination	—	—	—	—	—
SUBI	Soustraction immédiate	(Destination) − (Données immédiates) → Destination	•	•	•	•	•
SUBQ	Soustraction rapide	(Destination) − Données immédiates → Destination	•	•	•	•	•
SUBX	Soustraction étendue	(Destination) − (Source) − X → Destination	•	•	•	•	•
SWAP	Echanger les moitiés de registre	Registre [31 : 16] ↔ [15 : 0]	—	•	•	0	0
TAS	Test opér. et établ. d'1 indicateur	(Destination) testée → CC ; 1 → [7] OF Destination	—	•	•	0	0
TRAP	Trappe	PC → SSP@− ; SR → SSP@− ; (Vecteur) → PC	—	—	—	—	—
TRAPV	Trappe sur dépassement	Si V alors TRAP	—	—	—	—	—
TST	Test d'un opérande	(Destination) testé → CC	—	•	•	0	0
UNLK	Rupture du lien	AN → SP ; SP@+ → An	—	—	—	—	—

[ ] = numéro du bit

Fig. 5.3 - Tableau détaillé des instructions (avec les codes condition).

```
                        ORG $1000;
                        SIZE 1;
                        LIST 3;
  1000    227C00002000  MOVEA.L $2000,A1;
  1006    303CFFE5      MOVE.W -27,D0;
  100A    123C0041      MOVE 'A',D1;
  100E    1281          PIPO:MOVE D1,(A1);
  1010    D23C0001      ADD 1,D1;
  1014    D3FC00000001  ADDA.L 1,A1;
  101A    D07C0001      ADD.W 1,D0;
  101E    4A40          TST.W D0;
  1020    66EC          BNE PIPO;
  1022    4E4E          TRAP 14;;

PIPO                                          $100E  PC-R.

   0 ERRORS

FORCEMON 2.2 > MD 2000
002000    41 42 43 44 45 46 47 48  49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP

FORCEMON 2.2 >
002010    51 52 53 54 55 56 57 58  59 5A 5B 7C 00 01 4A 40  QRSTUVWXYZ[:..J@
002020    66 EC 4E 4E FF FF FF FF  FF FF FF FF FF FF FF FF  fìNN............
FORCEMON 2.2 >
```

Programme 2 - Ecriture de l'alphabet à partir de l'adresse 2000. Visualisation des résultats par la commande MD.

Commentaires :

ORG $1000 : le programme assemblé sera implanté à partir de l'adresse 1000 ;

SIZE 1 : la dimension implicite des instructions est l'octet ;

LIST 3 : on veut le listing complet ;

ligne 1000 : on place la valeur $2000 dans le registre A1 ;

ligne 1006 : on place la valeur -27 dans le registre D0 (sur le mot de poids faible) ;

ligne 100A : on met 'A' (donc $41) dans le registre D1 (octet);

ligne 100E : étiquette PIPO, on place D1 à l'adresse contenue dans A1 ($2000 au début) ;

ligne 1010 : 1 est ajouté à D1 (qui prend la valeur $42) ;

ligne 1014 : 1 est ajouté au registre A1 ;

ligne 101A : 1 est ajouté à D0 qui prend la valeur -26 ;

ligne 101E : on teste (à zéro) le contenu de D0 ;


```
FORCEMON 2.2 > MD 1000 30;DI
001000    227C00002000          MOVE.L  #8192,A1
001006    303CFFE5              MOVE.W  #-27,D0
00100A    123C0041              MOVE.B  #65,D1
00100E    1281                  MOVE.B  D1,(A1)
001010    D23C0001              ADD.B   #1,D1
001014    D3FC00000001          ADD.L   #1,A1
00101A    D07C0001              ADD.W   #1,D0
00101E    4A40                  TST.W   D0
001020    66EC                  BNE.S   $00100E
001022    4E4E                  TRAP    #14
001024    FFFF                  DC.W    $FFFF
001026    FFFF                  DC.W    $FFFF
001028    FFFF                  DC.W    $FFFF
00102A    FFFF                  DC.W    $FFFF
00102C    FFFF                  DC.W    $FFFF
00102E    FFFF                  DC.W    $FFFF

FORCEMON 2.2 >
```

Programme 2 - Désassemblage du code objet par la commande MD ; DI.

ligne 1020 : si D0 n'est pas nul, on retourne à l'étiquette PIPO... sinon ;

ligne 1022 : retour au moniteur, fin normale.

variante : on pourrait en 100E écrire PIPO:MOVE D1,(A1)+; (postincrémentation de A1) et supprimer alors la ligne 1014 devenue inutile.

Il n'était pas utile de travailler sur D0.W, D0.B aurait suffit.

Pour vérifier le bon fonctionnement du programme, la commande du moniteur MD 2000 [R] sera suffisante. Si on désire voir automatiquement affiché le résultat, on insérera avant la ligne 1022 :

```
MOVEA.L $2000,A5;  (adresse début de la chaîne)
MOVEA.L $201B,A6;  (adresse fin de chaîne plus 1)
JSR @ $21C14;
TRAP 14;
```

Remarque : certains programmes seront terminés par l'instruction TRAP 10 (au lieu de TRAP 14). Cette instruction déroutera le programme sur une ERREUR TRAP, dont le seul but, ici, est d'afficher automatiquement les registres, sans autre manipulation.

Ecriture du contenu d'un registre en binaire

Le **programme n° 3** écrit le contenu du registre D2 en binaire. Par exemple, si D2 = $804242FF il sera affiché : 10000000010000100100001011111111.

Commentaires :

ligne 2004 : les sorties se feront sur la console ;

ligne 200A : on teste D2 ; si le bit de poids le plus fort est à 0, D2 est positif, on ira à ZERO (ligne 200C) ;

ligne 200E : si ce bit est à 1, D2 est négatif, on écrit alors 1 ($31) (lignes 200E et 2012);

ligne 2018 : branchement inconditionnel à SUITE, pour passer au bit suivant ;

lignes 201A et 201E : on écrit 0 sur la console (code $30) ;

lignes 2024, 2026 et 2028 : on regarde si les 32 bits ont été analysés, en incrémentant D7 de 1 à chaque fois et en le comparant (CMP) à 32_{10}, si D7 = 32 on va à FINI (ligne 15) sinon ;

ligne 202A : décalage vers la gauche de D2 d'un bit (arithmetic shift left) et l'on recommence ;

On utilise la propriété déjà vue du codage binaire complémenté ; le premier bit indique toujours le signe d'une quantité, il suffit de décaler la zone étudiée pour tester tous les bits.

Conversion d'une touche en hexadécimal

Le **programme n° 4** accomplit la conversion d'une touche de la console en son code hexadécimal. On sait qu'à chaque touche est associé un code hexadécimal qui est envoyé par le port au micro-ordinateur. On veut récupérer ce code et l'afficher (si on envoie directement le code hexadécimal à la console, celle-ci le traduira en un symbole, chiffre, lettre, etc., voire par un espace si ce code n'est pas imprimable, mais ne transmettra jamais le code hexadécimal).


```
                        ORG $2000;LIST 3;SIZE 1;

2000    1E3CFFE0        MOVE -32,D7; /* ECRITURE DE D2 EN BINAIRE */
2004    4EB900021C7C    JSR @$21C7C;
200A    4A82            RETOUR:TST.L D2;
200C    6A0C            BPL ZERO;
200E    103C0031        NEGA:MOVE $31,D0;
2012    4EB900021E24    JSR @$21E24;
2018    600A            BRA SUITE;
201A    103C0030        ZERO:MOVE $30,D0;
201E    4EB900021E24    JSR @$21E24;
2024    5207            SUITE:ADDQ 1,D7;
2026    6704            BEQ FINI;
2028    E39A            ROL.L 1,D2;
202A    60DE            BRA RETOUR;
202C    4E4E            FINI:TRAP 14; /* D2 INCHANGE */

FINI                            15              $202C  PC-R.
NEGA                             6              $200E  PC-R.
RETOUR                           4              $200A  PC-R.
SUITE                           11              $2024  PC-R.
ZERO                             9              $201A  PC-R.

   0 ERRORS
a?? G
FORCEMON 2.2 > .D2 F0F0F0F0

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
11110000111100001111000011110000
FORCEMON 2.2 > .D2 12345678

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
00010010001101000101011001111000
FORCEMON 2.2 > DF
PC=00002004 SR=2?08=.S?.N... US=FFFF77FF SS=0000077E
D0=00300430 D1=00000400 D2=12345678 D3=00000000
D4=F0F0F0F0 D5=00000000 D6=00000004 D7=123456E0
A0=00050040 A1=00000618 A2=00000414 A3=00000428
A4=00000434 A5=0000056B A6=00000572 A7=0000077E
--------------------002004    4EB900021C7C          JSR     $00021C7C

FORCEMON 2.2 >
```

Programme 3 - Ecriture du contenu d'un registre en binaire. Assemblage et exécution.

```
                        ORG $2000;LIST 3;SIZE 1;

  2000    4E71          NOP;
  2002    4EB900021C7C  JSR @$21C7C;
  2008    103C0020      RET:MOVE $20,D0;
  200C    4EB900021E24  JSR @$21E24;/* ECRITURE D UN BLANC */
  2012    4EB900021FB0  JSR @$21FB0;/* SAISIE DU CARACTERE */
  2018    1E00          MOVE D0,D7;
  201A    4204          CLR D4;
  201C    E808          LSR 4,D0;
  201E    C03C000F      RETOUR:AND $0F,D0;
  2022    B03C000A      CMP $0A,D0;
  2026    6A06          BPL LETTRE;
  2028    803C0030      CHIFFRE:OR $30,D0; /* TRANFORMATION EN CODE ASCII */
  202C    6004          BRA ECRIRE;
  202E    D03C0037      LETTRE:ADD 55,D0;/* TRANSFORMATION EN CODE ASCII */
  2032    4EB900021E24  ECRIRE:JSR @$21E24;
  2038    5204          ADDQ 1,D4;
  203A    B83C0002      CMP 2,D4; /* FAIRE DEUX FOIS LA SEQUENCE */
  203E    67C8          BEQ RET;
  2040    1007          MOVE D7,D0;
  2042    60DA          BRA RETOUR;/* CARACTERE SUIVANT */
  2044    4E4E          FINI:TRAP 14;;

CHIFFRE                          13              $2028  PC-R.
ECRIRE                           16              $2032  PC-R.
FINI                             22              $2044  PC-R.
LETTRE                           15              $202E  PC-R.
RET                               4              $2008  PC-R.
RETOUR                           10              $201E  PC-R.


   0 ERRORS
a77 G
FORCEMON 2.2 / G 2000
PHYSICAL ADDRESS=00002000
 30 31 32 33 34 35 36 37 38 39 30 41 42 43 44 45 46 47 48 49 4A 4B 4C
FORCEMON 2.2 > G
```

Programme 4 - Ecriture de la valeur hexadécimale d'une touche de la console.

Nous allons étudier séparément les deux demi-octets, d'abord celui de gauche (qui sera recadré à droite). Ensuite la valeur de ce demi-octet sera testée ; si elle est comprise entre 0 et 9 il faudra imprimer le chiffre correspondant, donc envoyer un code compris entre $30 et $39 ; sinon, pour les valeurs entre $0A et $0F il faudra imprimer une lettre de code compris entre $41 et $46 soit de 65 à 70_{10}. Cette transformation se fait simplement en ajoutant 55_{10} à la valeur de la lettre ($0A + 55 = $41 = 'A', $0B + 55 = $42 = 'B', etc.). Le passage d'un chiffre de 0 à 9 au code $30 à $39 se fait par un 'OU' binaire (les bits communs sont conservés) mais une addition de 48_{10} = $30 marcherait aussi bien.

Commentaires :

lignes 2000 à 2012 : écriture d'un espace sur la console et stockage du code en D0 ;

lignes 2018, 201A et 201C : duplication de D0 en D7 ; compteur D4 pour les deux demi-octets et décalage de l'octet de 4 bits à droite pour accéder au demi-octet de gauche ;

ligne 201E : intersection binaire 'ET', on ne prend que les bits communs aux deux opérandes, donc en fait, on isole le demi-octet de droite ;

lignes 2022 à 2032 : détermination de la valeur (chiffre ou lettre) et impression ;

lignes 2038 à 203E : a-t-on fait deux fois la séquence (pour les deux demi-octets) ? Si oui, on recommence tout pour un autre code, sinon ;

ligne 2040 : on poursuit pour l'octet sauvegardé dans D7 et la ligne 201E sélectionne immédiatement la partie droite, bien placée.

Conversion avec table de traduction

Le **programme n° 5** réalise la même fonction que précédemment, mais en utilisant une table de traduction et un adressage indirect avec index.

Commentaires :

ligne 2000 : CLR (Clear) met à zéro la zone considérée ;

ligne 2004 : l'adresse effective de TABLE est placée dans le registre A1, c'est l'adresse du premier octet de la table (donc l'adresse du '0') ;

lignes 2008 à 201E ; on inscrit un espace et le caractère étudié va en D0, puis en D4 et D5 (ligne 2020) ;

ligne 2022 : le demi-octet de gauche, recadré à droite (D4), est compris entre $00 et $0F ;

ligne 2024 : (A1,D4.W) est une adresse comprise entre la valeur de A1 (si D4=$00) et la valeur de A1 + $0F si D4 = $0F, adresse qui représente l'extrémité de la table (soit le 'F'). Cette adresse pointe sur le caractère qui va être placé en D0 ;

```
                        ORG $2000;LIST 3;SIZE 1;

2000    4280            CLR.L D0; /* ECRITURE D UNE TOUCHE EN HEXADECIMAL */
2002    4284            CLR.L D4;
2004    43FA0038        LEA TABLE,A1;
2008    4EB900021C7C    JSR @$21C7C;
200E    103C0020        SUITE:MOVE $20,D0; /* ECRITURE ESPACE */
2012    4EB900021E24    JSR @$21E24;
2018    4EB900021FB0    JSR @$21FB0;        /* SAISIE SUR CONSOLE */
201E    3800            MOVE.W D0,D4;
2020    3A04            MOVE.W D4,D5;
2022    E80C            LSR 4,D4;
2024    10314000        MOVE (A1,D4.W),D0;
2028    4EB900021E24    JSR @$21E24;        /* ECRITURE PARTIE GAUCHE */
202E    CA3C000F        AND $0F,D5;
2032    10315000        MOVE (A1,D5.W),D0;
2036    4EB900021E24    JSR @$21E24;        /* ECRITURE PARTIE DROITE */
203C    60D0            BRA SUITE;
203E    303132333435    TABLE:DC '0123456789ABCDEF';
        363738394142
        43444546

SUITE                               6               $200E  PC-R.
TABLE                              18               $203E  PC-R.


   0 ERRORS
a77 G
FORCEMON 2.2 : G 2000
PHYSICAL ADDRESS=00002000
 31 32 33 34 35 36 37 38 39 30 0A 0D 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 0D
FORCEMON 2.2
```

Programme 5 – Variante du programme 4 ; utilisation d'une table de consultation.

lignes 202E et 2032 : même traitement pour l'octet D5, demi-octet de droite ;

ligne 203C : retour continuel à l'étiquette SUITE pour un autre essai ;

ligne 203E : la table de traduction est réservée sous forme d'une chaîne ASCII.

Ce programme est moins acrobatique que le précédent, en particulier, aucun test numérique n'est nécessaire, et cela le rend beaucoup plus général. Ainsi, n'importe quel code peut être traduit en un autre par le seul changement de table. Une table de 256 positions permet donc la traduction de n'importe quel code (en octet) initial. Ce type de problème est assez fréquent en informatique où le parc de machines est très hétéroclite. Chez un même constructeur, on pouvait trouver plusieurs types de perforatrices (sur cartes) engendrant des codes différents à la lecture. Aussi, des langages comme le COBOL possèdent tous une instruction TRANSFORM qui assure la conversion de codes, à l'aide de tables de traduction selon le principe développé dans ce programme.

Naturellement, ici, pour reprendre la main, puisque le programme n'a pas de fin programmée, il faudra appuyer sur l'un des poussoirs Software Abort ou Taster Abort.

TESTS, COMPARAISONS ET CODES CONDITION

Le registre SR (Status Register) utilise 4 bits positionnés en fonction du résultat de l'opération qui vient d'être effectuée. Ces bits ont pour nom : C (carry ou retenue) V (dépassement), N (résultat négatif), Z (résultat nul). Après chaque opération, ces bits sont en général activés, et le tableau général des instructions (figure 5.3) précise lesquels de ces bits sont affectés ou inchangés. En règle générale, les opérations de branchement ne modifient pas les codes condition, et toutes les opérations arithmétiques ou de manipulation les affectent. On trouve aussi un bit supplémentaire X (extension) qui n'est pas testable directement et qui intervient sur certaines opérations, en tant que bit de liaison. L'intérêt provient de ce que ces bits sont testables de seize façons différentes. Ainsi :

ADD.W -1,D0 ; les tests suivants concerneront l'opérande **destination**, donc D0.

BEQ NUL; si D0 = 0 aller à NUL

BNE NONNUL; si D0 non = 0 aller à NONNUL

BPL POSIT; si D0 positif ou nul aller à POSIT

BMI NEGA; si D0 négatif aller à NEGA

BVS DEPASS; si dépassement aller à DEPASS

Cas du dépassement : il y a dépassement lorsque l'opération demandée conduit :

- à un changement de signe (dû au mode de codage binaire complémenté) ainsi 127 + 1 = 01111111 + 00000001 = 10000000 = -128, si on travaille sur un octet. L'opération est normalement faite, il revient au programmeur de tester sa validité.
- à une division impossible à faire à cause de la taille des opérandes (voir DIVS, DIVU).

Le dépassement peut aussi être testé par une véritable instruction TRAPV permettant le déroutement vers une séquence plus conforme, éventuellement. On peut aussi, avant de provoquer

un branchement tester une zone particulière, qui ne vient pas d'être modifiée immédiatement avant. C'est évidemment un cas fréquent :

TST.L D5; tester à zéro le contenu de D5, avec les tests possibles :

BEQ NUL;(si nul)

BPL POSI; (positif ou nul) ou BNE (non nul), BMI (négatif)

On peut également utiliser l'instruction de comparaison CMP qui procède de la manière suivante :

CMP AE,Di; (AE : adresse effective, Di : registre données) taille : .B,.W,.L

l'opérande **source** est soustrait virtuellement de celui de **destination** et les codes condition sont alors positionnés, comme précédemment.

Le **tableau 5.4** indique les conditions que l'on peut tester.

Mnémonique	Condition "cc"	Equation logique
T(1)	Toujours vrai	1
F(1)	Jamais vrai	0
HI	Supérieur	C + Z = 0
LS	Inférieur ou égal	C + Z = 1
CC	Retenue à zéro	C = 0
CS	Retenue à un	C = 1
NE	Différent	Z = 0
EQ	Egal	Z = 1
VC(2)	Pas de dépassement	V = 0
VS(2)	Dépassement	V = 1
PL	Positif ou nul	N = 0
MI	Négatif	N = 1
GE(2)	Supérieur ou égal	N ⊕ V = 0
LT(2)	inférieur	N ⊕ V = 1
GT(2)	Supérieur	Z + (N ⊕ V) = 0
LE(2)	Inférieur ou égal	Z + (N ⊕ V) = 1
(1) Non utilisé par l'instruction Bcc (2) Utilisé en mode complément à deux		

Fig. 5.4 - Les codes condition

LES INSTRUCTIONS DE BRANCHEMENT

Elles sont présentées dans la **figure 5.5**

Branchement inconditionnel :

BRA étiquette ; demande de poursuivre le programme à l'étiquette précisée.

Branchement conditionnel :

Bcc étiquette (adresse 16 bits) ; où cc représente l'une des seize conditions possibles.

AND.B 4,D0;

BNE LOIN; provoquera le branchement uniquement si le bit 2 (valeur $2^2 = 4$) est à un.

Primitive de boucle DBcc :

Exemple : DBRA Di, étiquette ; Di est décrémenté de 1 et branchement à l'étiquette. Si Di passe à -1, le programme passe en séquence. Il est donc possible de fabriquer des boucles avec cette instruction, mais cela est sans grand intérêt par rapport à

Fig. 5.5 - Les instructions de branchement.

Instructions	Fonctionnement
Conditionnelles	
B_{CC}	Branchement conditionnel (14 conditions) déplacement 8 et 16 bits
DB_{CC}	Test condition, décrémentation et branchement, déplacement 16 bits
S_{CC}	Etablir octet conditionnellement (16 conditions)
Inconditionnelles	
BRA	Branchement inconditionnel déplacement 8 et 16 bits
BSR	Branchement à un sous-programme déplacement 8 et 16 bits
JMP	Saut inconditionnel
JSR	Saut à un sous-programme
de retour	
RTR	Retour et restauration des codes condition
RTS	Retour de sous-programme

une boucle ordinaire car d'une part la décrémentation de Di est imposée jusqu'à -1, et d'autre part l'usage judicieux (et sans complication) des codes condition permet de minimiser les instructions. Soit à faire dix fois une séquence :

```
MOVE.B 10,D1;
BOUCLE:....
.....
SUBQ.B 1,D1;
BNE BOUCLE;
```

Ainsi, l'instruction DBcc gagne au plus une instruction.

L'intérêt de DBcc provient de ce que cette instruction ne modifie pas les codes condition, ce qui permet des opérations étendues, sans perte des bits X ou C.

Branchement à un sous-programme, BSR et JSR :

BSR étiquette ; est une instruction économique qui est codée sous forme de déplacement (Branch to Subroutine).

Si on fait appel à des routines situées à une distance supérieure à 32 K octets, il faut employer JSR adresse.

Les écritures suivantes sont équivalentes :

```
OUTPUT:EQU @ $21C14;
JSR OUTPUT;
.....
JSR @$21C14;
.....
MOVEA.L $21C14,A2;
JSR (A2);
```

La grande souplesse de JSR peut le faire préférer à BSR malgré un code généré plus long et une vitesse d'exécution moins bonne d'environ 10 %. La même remarque s'applique aux branchements directs (BRA étiquette, JMP adresse effective). Les sous-programmes sont donc appelés par BSR ou JSR et se terminent par l'instruction RTS (return from subroutine).

Les adresses de retour sont emmagasinées automatiquement dans la pile (pointée par le registre A7) lors du lancement des sous-programmes et sont récupérées par RTS. Un regret, on ne

dispose pas d'un branchement conditionnel à un sous-programme comme dans certains langages évolués :

```
IF A = B PERFORM SOUPRO    (en Cobol) ;
IF(A.EQ.B) CALL SOUPRO     (en Fortran) ;
nnn IF A=B THEN GOSUB mmm  (dans certains Basic).
```

Il faut donc passer par une écriture différée :

```
ADD 1,D0;
BEQ LOIN;
RETOUR:NOP;
.....
LOIN:JSR SOUPRO;
BRA RETOUR;
```

Moins élégante, cette écriture ne coûte guère qu'une instruction de plus. Enfin, toutes ces instructions de contrôle de programme laissent inchangés les codes condition ; on peut donc finir un sous-programme par un calcul qui sera analysé dans le programme principal.

LES INSTRUCTIONS LOGIQUES

AND permet l'intersection binaire (notation ∧) entre **source** et **destination**. Cette instruction peut servir à tester un bit quelconque :

```
AND.B 1,D0;
BNE SUITE;
```

Si le bit 0 (2^1) vaut 1, le programme continuera à SUITE. Mais le plus souvent on utilise AND pour sélectionner une partie d'un octet :

```
AND  $7F,D0;
```

supprime le bit 7 (bit de parité dans une transmission série 7 bits). OR réalise l'union (le OU noté V) entre source et destination.

```
MOVE.B 9,D0;
OR  $30,D0;
```

D0 contient alors $39, soit le caractère ASCII '9'

Ces deux instructions sont pratiquement toujours présentes dans un programme, dès qu'une remise en forme des caractères est nécessaire. En revanche, EOR, OU exclusif (si deux bits en correspondance sont tous les deux à "1", le bit sera mis à zéro dans destination) est d'un usage rare. Une autre instruction très utilisée est NOT qui donne l'opposé d'un nombre (en mode binaire complémenté :

```
MOVE.W  1,D0;     (D0.W = 11111111)
NOT.W D0;         (D0.W = 00000001 = 1)
```

Les diverses instructions logiques sont présentées dans le tableau de la **figure 5.6.**

LES INSTRUCTIONS DE DECALAGE ET DE ROTATION

Le décalage logique est réalisé par LSL (left = gauche) et LSR (right = droite) ; pour ces deux instructions le bit V des codes contition est toujours mis à zéro.

De la même façon, mais avec permutation circulaire, on pourra utiliser ROL et ROR. On peut faire également transiter le décalage par le bit X d'extension par ROXL et ROXR.

Les décalages arithmétiques ASL et ASR sont à préférer lorsque le décalage remplace une multiplication (ou une division) par 2^n.

Pour une valeur positive dans D0 :

```
ASL.L 3,D0;       le contenu de D0 est multiplié par 8
ASR.L 1,D0;       le contenu précédent est divisé par 2
```

Toutes ces instructions sont résumées dans le tableau de la **figure 5.7.**

On remarque que le bit de signe est conservé dans un décalage à droite par ASR. On sait également qu'un nombre négatif codé en binaire complémenté peut s'interpréter ainsi : le bit de poids fort vaut -2^n et les autres bits 2^{n-1}, 2^{n-2},... 2,1.

On peut donc écrire ce nombre en séparant ses parties négative et positive : $-2^n + X$. Lors d'un décalage à droite de un bit il passe de la valeur $-2^n + X$ à la valeur $-2^n + 2^{n-1} + X/2$, soit la moitié du nombre initial.

Instruction	Taille opérande	Fonctionnement
AND	8 16 32	Dn ∧ (EA) ►Dn (EA) ∧ Dn ►EA (EA) ∧ #xxx ►EA
OR	8 16 32	Dn v (EA) ►Dn (EA) v Dn ►EA (EA) v #xxx ►EA
EOR	8 16 32	(EA) ⊙ Dv ►EA (EA) ⊙ #xxx ►EA
NOT	8 16 32	~(EA) ►EA

Remarque ~ complément à 1.

Fig. 5.6 - Les instructions logiques.

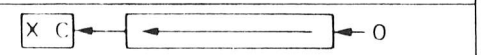
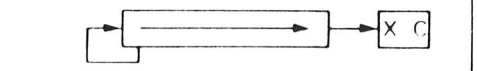
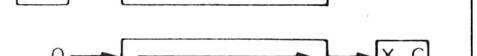
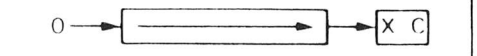
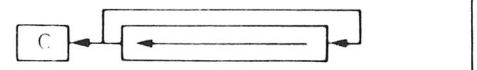
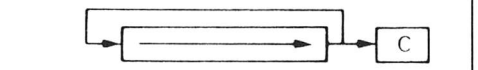
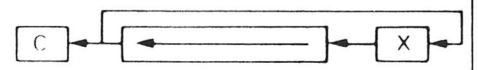
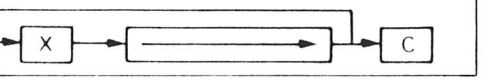
Instruction	Taille opérande	Fonctionnement
ASL	8 16 32	
ASR	8 16 32	
LSL	8 16 32	
LSR	8 16 32	
ROL	8 16 32	
ROR	8 16 32	
ROXL	8 16 32	
ROXR	8 16 32	

Fig. 5.7 - Les instructions de décalage et de rotation.

Exemple : soit -128 = 10000000

ASR.L1,D0; D0 = 11000000 = -64

ASR.L1,D0; D0 = 11100000 = -32

Il s'agit évidemment d'une division entière, ainsi avec D0 = -5 = 11111011

ASR 1,D0; D0 = 11111101 = -3

Ces instructions positionnent le bit V de débordement si le bit de poids fort change de valeur.

Elles sont en général très utiles et permettent, en particulier, d'étendre les opérations arithmétiques au-delà des opérandes de 16 bits.

Par exemple, la multiplication MULU à partir de deux opérandes d'au plus 16 bits fournit un résultat sur 32 bits :

MULU D0,D1; multiplie D1 par D0, D0 est inchangé, D0 et D1 sont au départ des mots de 16 bits.

Le **programme n° 6** très simple, effectue des produits, d'au plus 32 bits, entre deux opérandes quelconques D0 et D1. D0 est un mot et D1 un mot-long. On part de la règle de calcul : (D0.W = $XXXX, D1.L = $YYYYZZZZ)

$D0.W \times D1.L = 2^{16}.(YYYYXXXX) + ZZZZXXXX$

On est donc conduit à faire deux multiplications sur des opérandes de 16 bits puis à décaler le premier résultat de 16 bits à gauche et enfin d'additionner le second produit. Dans un autre programme (n° 9), une multiplication d'un grand nombre (Y) par 10 a été réalisée plus simplement en écrivant que 10.Y = 8.Y + 2.Y soit une somme de deux quantités obtenues en décalant Y de 3 et de 1 bit.

Une autre méthode peut être utilisée pour obtenir le même résultat. Il s'agit de l'algorithme "égyptien", cité par Pappus en... 300 après J.C. On écrit les deux nombres à multiplier face à face, puis on double le premier et on divise par 2 le second jusqu'à l'obtention de 1. Par exemple soit à effectuer 17 x 23 :

17	23
34	11
68	5
136	2
272	1


```
                        ORG $2000;LIST 3;
                        SIZE 4; /* MULTIPLICATION */
  2000    2601          MOVE D1,D3; /* EN D1 NOMBRE DE PLUS DE 16 BITS*/
  2002    C2C0          MULU D0,D1; /* EN D0 NOMBRE DE MOINS DE 16 BITS */
  2004    4843          SWAP D3;
  2006    C6C0          MULU D0,D3;
  2008    E183          ASL 8,D3;
  200A    E183          ASL 8,D3;
  200C    D283          ADD D3,D1;; /* RESULTAT EN D1 */
  200E    4E4A          TRAP 10;      /* D0 INCHANGE  */


   0 ERRORS
a77 G
FORCEMON 2.2 > .D0 64

FORCEMON 2.2 > .D1 654321

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

UT A TRAP ERROR
PC=00002010 SR=2700=.S7..... US=FFFF77FF SS=0000077A
D0=00000064 D1=278E38E4 D2=E1834E4A D3=27740000
D4=00004E4A D5=0000002C D6=00000002 D7=00000000
A0=00050040 A1=FFFF77FF A2=00000414 A3=00000554
A4=00002004 A5=00000540 A6=00000540 A7=0000077A
--------------------002010    13FC00000004CF4D     MOVE.B  #0,$0004CF4D

FORCEMON 2.2 > DC 64
$64=&100

FORCEMON 2.2 > DC 654321
$654321=&636321

FORCEMON 2.2 > DC 278E38E4
$278E38E4=&663632100

FORCEMON 2.2 >
```

Programme 6 - Multiplication avec résultat sur 32 bits avec un opérande supérieur à 32767.(D1 x D0 ⟶ D1)

On additionne ensuite les éléments de la colonne de gauche, mais seulement si les nombres qui leur font face (à droite) sont impairs. Dans notre exemple, on fera donc la somme de 17, 34, 68, 272 soit un total de 391 (17 x 23 = 391).

Il s'agit en fait d'une multiplication en base 2 par addition-décalage, dont la programmation est immédiate. Le **programme n° 7** le prouve suffisamment. La parité du registre D1 est connue

```
                        ORG $2000;LIST 3;
                        SIZE 4;
2000    4282            CLR D2;
2002    E289            LOP:LSR 1,D1;
2004    6402            BCC RIEN;
2006    D480            ADD D0,D2;
2008    E388            RIEN:LSL 1,D0;
200A    4A81            TST D1;
200C    66F4            BNE LOP;
200E    4E4A            FINI:TRAP 10;;

FINI                      10            $200E  PC-R.
LOP                        4            $2002  PC-R.
RIEN                       7            $2008  PC-R.

   0 ERRORS

FORCEMON 2.2 > .D0 100

FORCEMON 2.2 > .D1 FFFFFF

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

UT A TRAP ERROR
PC=00002010 SR=2714=.S7X.Z.. US=FFFF77FF SS=00000732
D0=00000000 D1=00000000 D2=FFFFFF00 D3=00000000
D4=FFFFFFFF D5=00000000 D6=00000047 D7=00000026
A0=00002000 A1=0000A1D8 A2=0001FE00 A3=000275AE
A4=0000A100 A5=0000A1D9 A6=0000A1D9 A7=00000732
--------------------002010    E38A                LSL.L   #1,D2

FORCEMON 2.2 >
```

Programme 7 - Variante du programme 6 avec utilisation de l'algorithme "égyptien". (D0 x D1 ⟶ D2).

(par le bit de retenue C qui prend la valeur du bit faible) après l'opération de décalage LSR 1,D1. Si C vaut zéro, on ignore l'addition cumulative qui se fait par ADD D0,D2. Lorsque D1 est nul, tous les bits ont été décalés et l'opération est terminée ; le produit de D0 par D1 se trouve alors en D2. Evidemment, on a tout intérêt à placer en D1 le plus petit des deux nombres.

Le **programme n° 8**, effectue la multiplication non signée des registres D0 et D2. Le résultat est placé en D4 et D5 sur 64 bits (poids forts en D4). La même méthode est utilisée. Pour additionner des nombres, qui à la suite de décalages, tiennent sur 64 bits, on effectue d'abord la somme de D2 sur D5, puis la somme de D1 sur D4 en demandant cette fois de tenir compte


```
                        ORG $2000;LIST 3;
                        SIZE 4;
2000    4281            CLR D1;
2002    4284            CLR D4;
2004    4285            CLR D5;
2006    E288            LOP:LSR 1,D0;
2008    6404            BCC RIEN;
200A    DA82            ADD D2,D5;
200C    D981            ADDX D1,D4;
200E    E389            RIEN:LSL 1,D1;
2010    E38A            LSL 1,D2;
2012    6404            BCC SUITE;
2014    823C0001        OR.B 1,D1;
2018    4A80            SUITE:TST D0;
201A    66EA            BNE LOP;
201C    4E4A            TRAP 10;;

LOP                        6            $2006  PC-R.
RIEN                      10            $200E  PC-R.
SUITE                     14            $2018  PC-R.

   0 ERRORS

FORCEMON 2.2 > .D0 7FFFFFFF

FORCEMON 2.2 > .D2 80000001

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

UT A TRAP ERROR
PC=0000201E SR=2704=.S7..Z.. US=FFFF77FF SS=0000072E
D0=00000000 D1=40000000 D2=80000000 D3=00000000
D4=3FFFFFFF D5=FFFFFFFF D6=00000047 D7=00000026
A0=00002000 A1=0000A1D8 A2=0001FE00 A3=000275AE
A4=0000A100 A5=0000A1D9 A6=0000A1D9 A7=0000072E
--------------------00201E    FFFF                DC.W    $FFFF

FORCEMON 2.2 >
```

Programme 8 - Variante du programme 7 avec des opérandes de 32 bits et résultat sur 64 bits. D0 x D2⟶D4 D5

de la retenue précédente éventuelle, grâce à l'instruction ADDX. Le décalage sur 64 bits se fait également en deux fois mais en cas de perte d'un bit, on le place dans le registre suivant par : OR.B 1,D1. A l'exécution du programme, D0 vaut 7FFFFFFF soit $2^{31} - 1$, et D1 vaut 80000001 soit $2^{31} + 1$. Le produit en D4 et D5 vaut alors 3FFFFFFF FFFFFFFF ou encore $2^{62} - 1$. Tous ces nombres sont considérés être en binaire pur non signé.

LES INSTRUCTIONS DE MANIPULATION DE BITS

Ces instructions (**figure 5.8**) permettent de tester un bit d'un opérande sur 8 ou 32 bits, en donnant son rang. Le bit Z des codes condition est sollicité, le bit en question pouvant également être positionné à 0 ou 1 selon l'instruction. Ces instructions complètent les instructions logiques, mais sont d'un usage plus restreint.

LES INSTRUCTIONS DE MOUVEMENT DE DONNEES

L'opération MOVEP (MOVE vers un périphérique) permet de transférer vers un périphérique un mot ou un mot-long. L'adresse spécifiée s'incrémente de 2 à chaque transfert ; il s'agit d'une succession de MOVE, plus efficace et adaptée à l'adressage particulier des registres de ces périphériques. Dans tous les exemples proposés, il a été préféré l'écriture de MOVE successifs, dont la mise en oeuvre, au niveau de l'écriture, est plus aisée, et dont la compréhension est très claire.

Nous avons volontairement extrait du tableau récapitulatif de la **figure 5.9**, les instructions EXG et SWAP, qui avec EXT forment une unité logique, comme cela a été vu au début de ce chapitre.

LES INSTRUCTIONS DE CONTROLE DU SYSTEME

On remarquera (**figure 5.10**) toutes les opérations portant sur le CCR (registre des codes condition), par exemple l'instruction :

```
AND $EF,CCR
```

mettra le bit d'extension X à zéro, les autres restant inchangés.

Pour le mettre à 1 on utilisera :

```
OR $10,CCR
```

Il est donc possible de modifier et de stocker les codes condition, spécialement avant une instruction étendue (ADDX, ABCD...) où la valeur de X est capitale.

Instruction	Taille opérande	Fonctionnement
BTST	8, 32	bit de (EA) ►Z
BSET	8, 32	bit de(EA) ►Z 1 ►bit de EA
BCLR	8 32	bit de (EA) ►Z 0 ►bit de EA
BCHG	8, 32	bit de (EA) ►Z bit de (EA) - bit de EA

Fig. 5.8 - Les instructions de manipulation de bits.

Instruction	Taille opérande	Fonctionnement
EXG	32	Rx◄►Ry
LEA	32	EA ►An
LINK	—	An ►SP@- SP ►An SP + d ►SP
MOVE	8, 16, 32	(EA)s ►EAd
MOVEM	16, 32	(EA) ►An, Dn An, Dn ►EA
MOVEP	16, 32	(EA) ►Dn Dn ►EA
MOVEQ	8	#xxx ►Dn
PEA	32	EA ►SP@-
SWAP	32	Dn [31 16]◄► Dn [15 0]
UNLK	—	An ►Sp SP @+ ►An

s = source
d = destination
[ ] = numéro de bit
@- = indirect avec prédécrémentation
@+ = indirect avec postincrémentation

Fig. 5.9 - Les instructions de mouvement de données.

Instructions	Fonctionnement
Privilégiées	
RESET	Mise à zéro des circuits extérieurs
RTE	Retour d'exception
STOP	Arrêt d'exécution du programme
ORI to SR	OU logique avec registre d'état
MOVE USP	Transfert pointeur de pile utilisateur
ANDI to SR	ET logique avec registre d'état
EORI to SR	OU exclusif logique avec registre d'état
MOVE EA to SR	Chargement nouveau registre d'état
Génératrices de trappes	
TRAP	Trappe
TRAPV	Trappe sur dépassement
CHK	Test registre aux limites
Registre d'état	
ANDI to CCR	ET logique avec codes condition
EORI to CCR	OU exclusif avec codes condition
MOVE EA to CCR	Chargement nouveaux codes condition
ORI to CCR	OU logique avec codes condition
MOVE SR to EA	Rangement du registre d'état.

Fig. 5.10 - Les instructions de contrôle du système.

LES INSTRUCTIONS ARITHMETIQUES

Toutes ces opérations (**figure 5.11**) sont d'un usage très courant, comme on a pu le constater au cours des divers programmes proposés. Une instruction originale, TAS, permet de gérer le partage d'une même ressource par plusieurs processeurs, de façon non conflictuelle.

Cette instruction ne peut donc être utilisée, ou testée, que sur des matériels bien plus complexes que ceux proposés couramment.

Instruction	Taille opérande	Fonctionnement
ADD	8, 16, 32	Dn + (EA) ►Dn
		(EA) + Dn ►EA
		(EA) + #xxx ►EA
	16, 32	An + (EA) ►An
ADDX	8, 16, 32	Dx + Dy + X ►Dx
	16, 32	Ax@- Ay@- + X ►Ax@
CLR	8, 16, 32	0 ►EA
CMP	8, 16, 32	Dn - (EA)
		(EA) - #xxx
		Ax@+ - Ay@+
	16, 32	An - (EA)
DIVS	32 : 16	Dn/(EA) ►Dn
DIVU	32 : 16	Dn/(EA) ►Dn
EXT	8 ►16	$(Dn)_8$ ►Dn_{16}
	16 ►32	$(Dn)_{16}$ ►Dn_{32}
MULS	16 * 16 ►32	Dn * (EA) ►Dn
MULU	16 * 16 ►32	Dn * (EA) ►Dn
NEG	8, 16, 32	0 - (EA) ►EA
NEGX	8, 16, 32	0 - (EA) - X - EA
SUB	8, 16, 32	Dn - (EA) ►Dn
		(EA) - Dn ►EA
		(EA) - #xxx ►EA
	16, 32	An - (EA) ►An
SUBX	8, 16, 32	Dx - Dy - X ►Dx
		Ax@- - Ay@- - X ►Ax@
TAS	8	(EA) - 0, 1 ►EA[7]
TST	8, 16, 32	(EA) - 0

Fig. 5.11 - Les instructions arithmétiques.

LES INSTRUCTIONS EN BCD

Ces instructions (**figure 5.12**) travaillent sur un octet et le bit d'extension X ; le mode le plus utile est le mode d'adressage prédécrémenté :

ABCD -(A1),-(A2);

Pour additionner deux nombres, il faudra écrire une boucle de calcul. Soit en $2000,2001,2002 la valeur : 658955 et en $3000, 3001,3002 la valeur 123488, l'instruction ABCD va considérer ces deux quantités comme décimales. Ensuite :

```
MOVEA.L $2003,A1;
MOVEA.L $3003,A2;
ABCD -(A1),-(A2);    (addition de 2002 sur 3002) } la retenue
ABCD -(A1),-(A2);    (addition de 2001 sur 3001) } se propage
ABCD -(A1),-(A2);    (addition de 2000 sur 3000) } par le bit X
```

on peut lire ensuite en $3000 le résultat 658955 + 123488 = 782443

L'utilisation de telles instructions permet l'impression de résultats décimaux.

Le **programme n° 9** assure la transformation décimale/hexadécimale d'un registre. La transformation, dans ce sens, est relativement simple. On applique une méthode dite "schéma de Horner" qui minimise le nombre d'opérations à effectuer (**figure 5.13**).

Fig. 5.12 - Les instructions en BCD.

Instruction	Taille opérande	Fonctionnement
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $Ax@_{-10} + Ay@_{-10} + X \rightarrow Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $Ax@_{-10} - Ay@_{-10} - X \rightarrow Ax@$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$


```
                    A
                                LIST 3;/* VALEUR DECIMALE A CONVERTIR EN D1 */
                                ORG $800;
                                SIZE 4;
   800     4280                 CLR D0;
   802     4282                 CLR D2;
   804     263CFFFFFFF7         MOVE -9,D3;
   80A     5283                 TEST:ADDQ 1,D3;
   80C     6714                 BEQ FIN;
   80E     E999                 ROL 4,D1;
   810     1001                 MOVE.B D1,D0;
   812     C03C000F             AND.B $0F,D0;
   816     2A02                 MOVE D2,D5;  /*DEBUT MULTIPLICATION DE D2 PAR 10 */
   818     E785                 ASL 3,D5;
   81A     E382                 ASL 1,D2;
   81C     D485                 ADD D5,D2;   /* FIN MULTIPLICATION DE D2 PAR 10 */
   81E     D480                 ADD D0,D2; /* RESULTAT HEXADECIMAL EN D2 */
   820     60E8                 BRA TEST;
   822     4E4A                 FIN:TRAP 10;; /* FIN AVEC TRAP ERROR */

FIN                                  18                  $822  PC-R.
TEST                                  7                  $80A  PC-R.


   0 ERRORS
a?? G
FORCEMON 2.2 > .D1 98765432

FORCEMON 2.2 > G 800
PHYSICAL ADDRESS=00000800

UT A TRAP ERROR
PC=00000824 SR=2715=.S7X.Z.C US=FFFF7FFF SS=0000076E
D0=00000002 D1=98765432 D2=05E30A78 D3=00000000
D4=FFFFFFFF D5=04B5A1F8 D6=00000045 D7=FFFFFFFF
A0=00050040 A1=00020348 A2=0000077E A3=0001B5AE
A4=0000A100 A5=0000A9F6 A6=0000A9F7 A7=0000076E
--------------------000824    662A                BNE.S   $000850

FORCEMON 2.2 > DC &98765432
$5E30A78=&98765432

FORCEMON 2.2 >
```

Programme 9 - Conversion décimale/hexadécimale non signée.

$D1_{10} \longrightarrow D2_{16}$

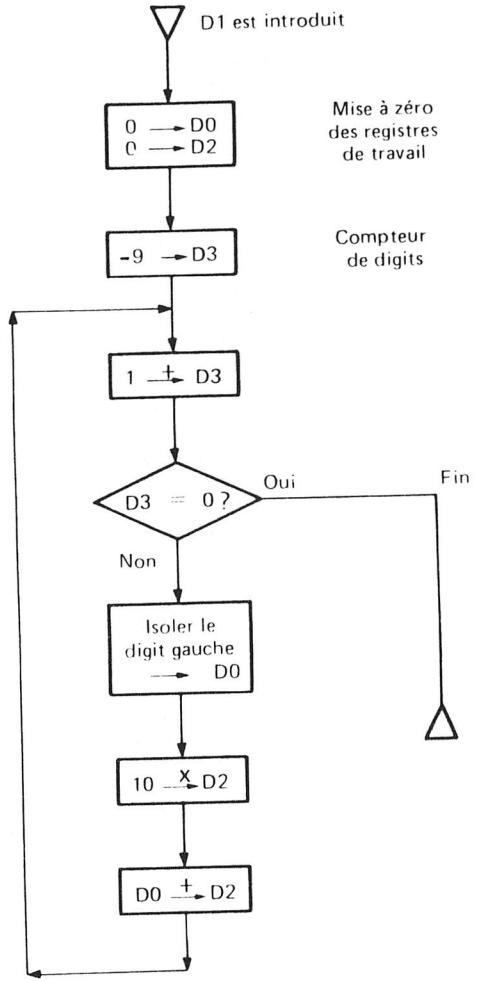


Fig. 5.13 - Organigramme de la conversion décimale/hexadécimale d'un registre.

Conventions d'écriture :

A → B A est mis dans B

A +→ B A est ajouté à B

A -→ B A est soustrait à B

A X→ B B est multiplié par A

Ces conventions sont très proches de l'écriture en assembleur.

Par exemple, pour transformer la quantité 3456 en hexadécimal, on évalue l'expression :

```
 3
[3x10
 3x10+4
[(3x10+4)x10
 (3x10+4)x10+5
[((3x10+4)x10+5)x10
 ((3x10+4)x10+5)x10+6=3000+400+50+6=3456
```

L'algorithme est simple, et répétitif.

Tout est basé sur le fait que les chiffres décimaux et binaires, entre 0 et 9, ont la même notation et la même valeur. En conséquence, toutes les opérations arithmétiques, sont des opérations binaires, ce qui assure la validité de la transformation. Mais dans l'autre sens, le transformation est plus délicate.

Le **programme n° 10** fournit un exemple de transformation décimale/hexadécimale. Première méthode (**figure 5.14**) : les 32 bits du registre vont être examinés un par un grâce à un décalage de un bit à gauche (si le bit vaut 1, la zone est considérée comme négative). Si le bit 32 vaut 1, on ajoute EN DECIMAL l'équivalent de 2^{31}, si le bit 31 vaut 1, on ajoute 2^{30}, etc.

Les différentes puissances de 2 sont mises en table, par ordre croissant, pour que le fonctionnement en mode prédécrémenté (les adresses décroissent) permettent l'addition des fortes valeurs, qui sont les premières testées. Le sous-programme ADDI réalise en mode prédécrémenté l'addition de quatre octets de la table, sur la zone TOTAL. Si le bit testé vaut 0, on se contente de décrémenter de 4 le registre A1, qui pointe sur la zone à ajouter. Le sous-programme ADDI effectue le cumul dans TOTAL.

On peut néanmoins se passer de l'addition décimale, en effectuant des soustractions des puissances décroissantes de 10. On soustrait d'abord 10^{10}, si possible, puis 10^{9} autant de fois qu'il le faut... etc. L'organigramme est décrit à la **figure 5.15** et le **programme n° 11** en est la traduction.

```
                        LIST 3;ORG $1000;SIZE 1;


1000    45FA0040        LEA TOTAL,A2;
1004    43FA00BC        LEA TABFIN,A1;
1008    42AAFFFC        CLR.L -4(A2);
100C    163C0020        MOVE 32,D3;
1010    4A81            TEST:TST.L D1; /* DONNEE HEXA EN D1 */
1012    6B0E            BMI LOIN;
1014    D3FCFFFFFFFC    ADDA.L -4,A1;
101A    5303            RETOUR:SUBQ 1,D3;
101C    670A            BEQ FIN;
101E    E399            ROL.L 1,D1;
1020    60EE            BRA TEST;
1022    4EBA000C        LOIN:JSR ADDI;
1026    60F2            BRA RETOUR;
1028    242AFFFC        FIN:MOVE.L -4(A2),D2; /* RESULTAT EN D2 */
102C    E399            ROL.L 1,D1; /* D1 INCHANGE */
102E    4E4A            TRAP 10;
1030    C509            ADDI:ABCD -(A1),-(A2); /* ADDITION 4 OCTETS */
1032    C509            ABCD -(A1),-(A2);  /* PAS DE BOUCLE */
1034    C509            ABCD -(A1),-(A2);
1036    C509            ABCD -(A1),-(A2);  /* FIN D ADDITION */
1038    45FA0008        LEA TOTAL,A2;
103C    4E75            RTS;
103E                    DS.L 1;
                        TOTAL:EQU *;
1042    00000001        DC.L 1;
1046    00000002        DC.L 2;
104A    00000004        DC.L 4;
104E    00000008        DC.L 8;
1052    00000016        DC.L $16;
1056    00000032        DC.L $32;
105A    00000064        DC.L $64;
105E    00000128        DC.L $128;
1062    00000256        DC.L $256;
1066    00000512        DC.L $512;
106A    00001024        DC.L $1024;
106E    00002048        DC.L $2048;
1072    00004096        DC.L $4096;
1076    00008192        DC.L $8192;
107A    00016384        DC.L $16384;
107E    00032768        DC.L $32768;
1082    00065536        DC.L $65536;
1086    00131072        DC.L $131072;
108A    00262144        DC.L $262144;
108E    00524288        DC.L $524288;
1092    01048576        DC.L $1048576;
1096    02097152        DC.L $2097152;
109A    04194304        DC.L $4194304;
109E    08388608        DC.L $8388608;
10A2    16777216        DC.L $16777216;
10A6    33554432        DC.L $33554432;
10AA    67108864        DC.L $67108864;
10AE                    DS.L 5;
                        TABFIN:EQU *;;
```



```
ADDI                    18              $1030  PC-R.
FIN                     15              $1028  PC-R.
LOIN                    13              $1022  PC-R.
RETOUR                   9              $101A  PC-R.
TABFIN                  54              $10C2  PC-R.
TEST                     6              $1010  PC-R.
TOTAL                   25              $1042  PC-R.


   0 ERRORS
a?? G


FORCEMON 2.2 > .D1 5E30A78

FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000

UT A TRAP ERROR
PC=00001030 SR=2700=.S7..... US=FFFF77FF SS=00000782
D0=00000029 D1=05E30A78 D2=98765432 D3=00000000
D4=FFFFFFFF D5=41424344 D6=00000017 D7=67100003
A0=00050040 A1=00001042 A2=00001042 A3=0001B5AE
A4=0000A100 A5=0001A1DC A6=0000B7AC A7=00000782
--------------------001030    C509                ABCD.B  -(A1),-(A2)

FORCEMON 2.2 > .D1 1111111

FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000

UT A TRAP ERROR
PC=00001030 SR=2701=.S7....C US=FFFF77FF SS=00000782
D0=00000029 D1=01111111 D2=17895697 D3=00000000
D4=FFFFFFFF D5=41424344 D6=00000017 D7=67100003
A0=00050040 A1=00001042 A2=00001042 A3=0001B5AE
A4=0000A100 A5=0001A1DC A6=0000B7AC A7=00000782
--------------------001030    C509                ABCD.B  -(A1),-(A2)

FORCEMON 2.2 > DC 1111111
$1111111=&17895697

FORCEMON 2.2 >
```

Programme 10 – Conversion hexadécimale/décimale non signée.
$D1_{16} \longrightarrow D2_{10}$

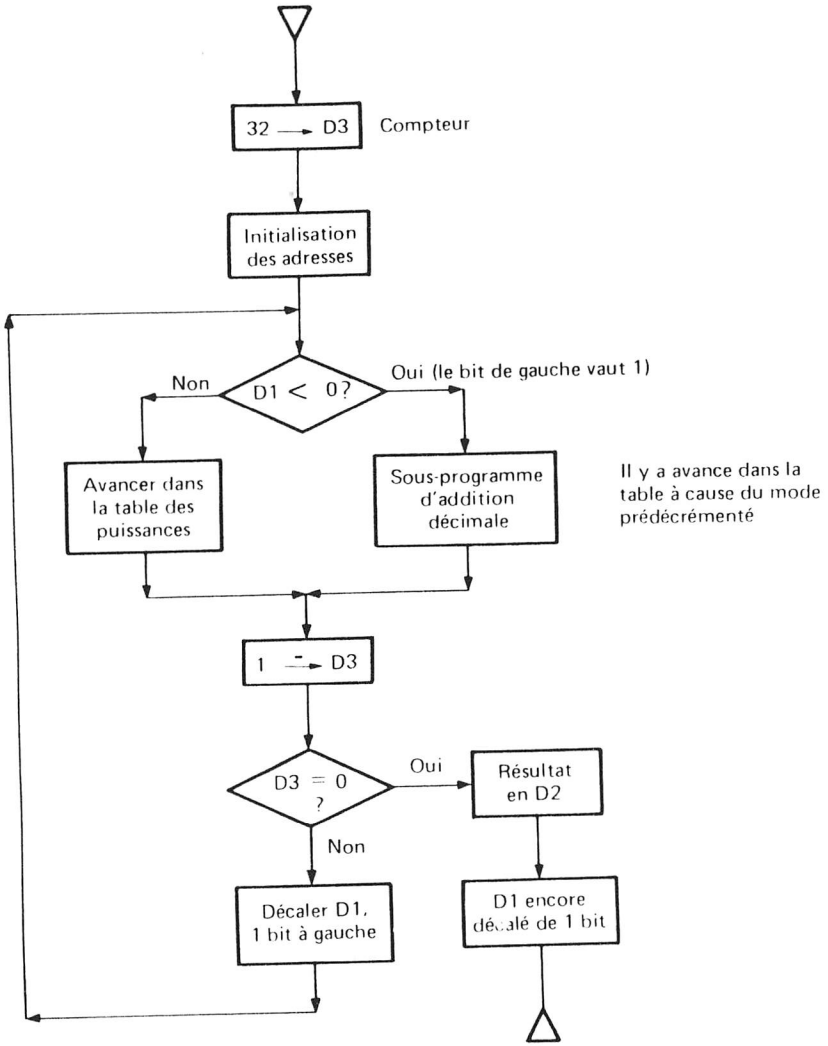


Fig. 5.14 - Organigramme de la conversion hexadécimale/décimale d'un registre.

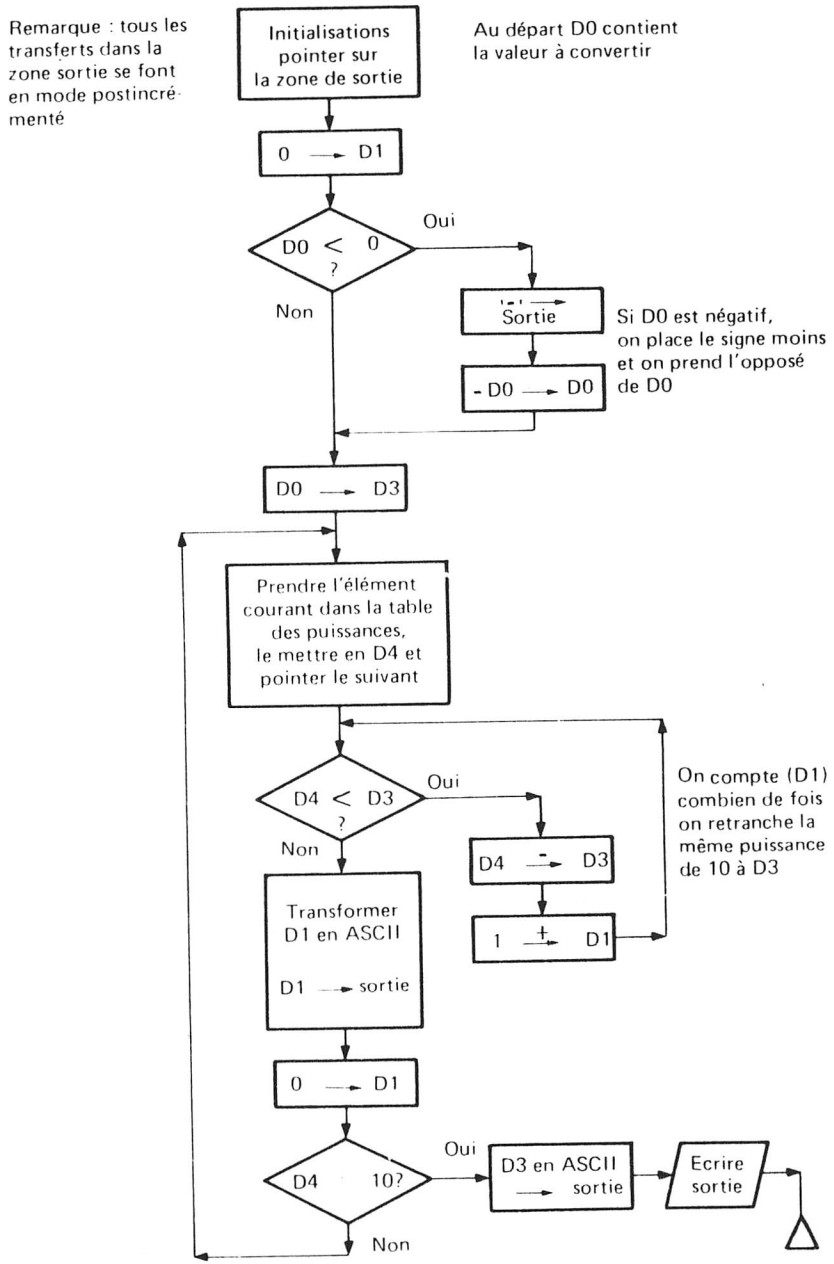


Fig. 5.15 - Organigramme de la conversion hexadécimale/décimale sans utilisation de l'addition décimale.

```
                        LIST 3;
                        ORG $1000;SIZE 1;

1000    4BF81500        LEA @$1500,A5;
1004    4DF81500        LEA @$1500,A6;
1008    4281            CLR.L D1;
100A    4A80            TST.L D0;
100C    6A06            BPL POSITIF;
100E    4480            NEG.L D0;
1010    1CFC002D        MOVE #'-',(A6)+;
1014    2600            POSITIF:MOVE.L D0,D3;
1016    43FA002C        LEA TABLE,A1;
101A    2819            RETOUR:MOVE.L (A1)+,D4;
101C    B684            RET:CMP.L D4,D3;
101E    6B06            BMI LOIN;
1020    9684            SUB.L D4,D3;
1022    5201            ADDQ 1,D1;
1024    60F6            BRA RET;
1026    823C0030        LOIN:OR $30,D1;
102A    1CC1            MOVE D1,(A6)+;
102C    4201            CLR D1;
102E    B8BC0000000A    CMP.L 10,D4;
1034    66E4            BNE RETOUR;
1036    863C0030        OR $30,D3;
103A    1CC3            MOVE D3,(A6)+;
103C    4EB900021C14    JSR @$21C14;
1042    4E4E            TRAP 14;
1044    3B9ACA00        TABLE: DC.L 1000000000;
1048    05F5E100        DC.L 100000000;
104C    00989680        DC.L 10000000;
1050    000F4240        DC.L 1000000;
1054    000186A0        DC.L 100000;
1058    00002710        DC.L 10000;
105C    000003E8        DC.L 1000;
1060    00000064        DC.L 100;
1064    0000000A        DC.L 10;

LOIN                    18              $1026  PC-R.
POSITIF                 10              $1014  PC-R.
RET                     13              $101C  PC-R.

FORCEMON 2.2 > .D0 5E30A78

FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000
0098765432
FORCEMON 2.2 > .D0 80000000
```



```
FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000
-2147483648
FORCEMON 2.2 > .D0 FFFFFFFF

FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000
-0000000001
FORCEMON 2.2 >
```

Programme 11 - Conversion hexadécimale/décimale signée. $D0_{16} \longrightarrow résultat_{10}$ sur la console

Un problème qui pourrait se poser est la baisse de performance de ce programme qui doit faire, éventuellement, un grand nombre de soustractions (conversion de 1999999999); le PTM permettra une véritable évaluation de la vitesse de traitement.

CHAPITRE 6 | les exceptions

Ce chapitre est typique de la programmation des micro-ordinateurs et n'a pas d'équivalent dans les langages évolués. Le fonctionnement est le suivant : à la rencontre d'un événement "anormal" interne ou externe le processeur interrompt le traitement en cours pour dérouler une séquence particulière (sous-programme d'exception) gérant l'événement survenu.

NATURE DES EVENEMENTS

Les exceptions peuvent être internes, elles sont alors de différentes sortes : erreur bus ; erreur d'adresse ; instruction illégale ; division par zéro ; instructions programmées (CHK et TRAP) ; violation de privilège ; trace.

Les événements externes sont essentiellement : le processus d'initialisation ; les interruptions périphériques.

C'est ce dernier point qui présente le plus grand intérêt pratique.

Toutes ces exceptions sont classées dans une table d'exception (**figure 6.1**) qui fournit, pour chacune un numéro de **vecteur** et son adresse d'implantation (adresse = quatre fois le numéro de vecteur).

A la rencontre d'un type d'exception, le processeur ira chercher dans la zone adresse qui lui correspond, l'adresse effective du début de la séquence à dérouler. Cette adresse effective peut être modifiée par le programmeur s'il entend gérer lui-même - c'est le cas ordinaire - l'exception pouvant survenir.

Exemple : le programmeur laisse faire par inadvertance une division par zéro ; il y a donc déroulement d'une séquence

Numéro vecteur(s)	Adresses			Affectation
	Dec	Hex	Espace	
0	0	000	SP	Initialisation de SSP
	4	004	SP	Initialisation de PC
2	8	008	SD	Erreur bus
3	12	00C	SD	Erreur adresse
4	16	010	SD	Instruction interdite
5	20	014	SD	Division par zéro
6	24	018	SD	Instruction CHK
7	28	01C	SD	Instruction TRAPV
8	32	020	SD	Violation de privilège
9	36	024	SD	Trace
10	40	028	SD	Emulateur ligne 1010
11	44	02C	SD	Emulateur ligne 1111
12*	48	030	SD	(non attribué, réservé)
13*	52	034	SD	(non attribué, réservé)
14*	56	038	SD	(non attribué, réservé)
15*	60	03C	SD	(non attribué, réservé)
16-23*	64	04C	SD	(non attribué, réservé)
	95	05F		–
24	96	060	SD	Interruption parasite
25	100	064	SD	Auto vecteur interruption niveau 1
26	104	068	SD	Auto vecteur interruption niveau 2
27	108	06C	SD	Auto vecteur interruption niveau 3
28	112	070	SD	Auto vecteur interruption niveau 4
29	116	074	SD	Auto vecteur interruption niveau 5
30	120	078	SD	Auto vecteur interruption niveau 6
31	124	07C	SD	Auto vecteur interruption niveau 7
32-47	128	080	SD	Vecteurs d'instruction TRAP
	191	0BF		–
48-63*	192	0C0	SD	(non attribué, réservé)
	255	0FF		–
64-255	256	100	SD	Vecteurs interruption utilisateur
	1023	3FF		–

Fig. 6.1 - Affectation des vecteurs d'exception.

d'exception systématique dont l'adresse de début est donnée dans les quatre positions $14,15,16,17. Ces quatre octets contiennent l'adresse 00 02 03 B0 ; le sous-programme implanté à cette adresse consiste uniquement à signaler ce type d'erreur et à rendre la main au moniteur. Cette méthode est modérément satisfaisante. Le programmeur peut placer en $14,15,16,17, par programme de préférence, l'adresse d'un sous-programme pouvant faire tout autre chose. Après déroulement de cette séquence, le programme principal poursuit son chemin. Le sous-programme d'exception diffère uniquement par sa dernière instruction qui au lieu de RTS sera RTE (return from exception).

Naturellement, le programmeur n'écrira que les sous-programmes d'exception qui lui sont utiles, en particulier, les exceptions dues à des erreurs de programmation (erreur d'adresse : tentative d'accéder à un mot ou un mot-long par adresse impaire) seront signalées par la séquence préprogrammée chargée à l'initialisation, ce qui suffira pour toute mise au point. Exemple d'un programme avec sous-programme d'exception :

```
VECT;EQU @$14;
LEA SOUPRO,A6;
MOVE.L A6,VECT;
.....
DIVU D0,D1;
.....
TRAP 14;
SOUPRO: LEA MES,A5;
LEA MES+20,A6;
JSR @$21C14;
RTE;
MES:DC 'DIVISION PAR ZERO !';
```

On note les points nouveaux : chargement de l'adresse du sous-programme dans la zone associée au vecteur 'DIVISION PAR ZERO', et la fin de SOUPRO qui est un RTE.

Le sous-programme écrit ici ne présente guère d'intérêt par rapport à la séquence standard (qui elle, en plus, visualise les registres) mais le déroulement du programme principal n'est pas modifié (ce qui n'est pas forcément judicieux).

TRAITEMENT DE L'EXCEPTION TRAPV

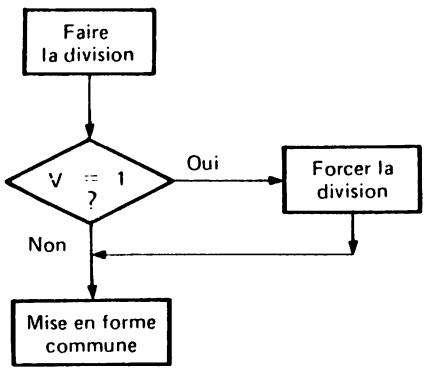
Cette instruction, sans opérande, provoque une exception si le bit V (débordement) des codes condition passe à 1. Outre les cas déjà vus, la division peut aussi conduire à V = 1. Voyons le mécanisme de cette opération :

DIVU source, Di; division non signée (la division signée DIVS peut être utile aussi).

source est le diviseur sur 16 bits et ce peut être : un registre Dj (poids faibles seulement), un entier inférieur à 65536, ou un mot dont on donne l'adresse. Dans cette instruction, Di est divisé par source (les opérandes sont inversés par rapport à nos usages ; les anglo-saxons emploient fréquemment DIVIDE B INTO A au lieu de DIVIDE A BY B). Le résultat de l'opération est placé dans le registre Di, ainsi le quotient occupe les 16 bits de poids faibles de Di, et le reste de la division, les 16 bits de poids forts de Di (récupérables par SWAP Di). **Mais**, si le quotient ne peut tenir sur 16 bits (supérieur à 65535), l'opération n'est pas faite, et le bit V passe à 1, les opérandes n'étant pas affectés.

Ainsi, il est impossible de savoir si une division se fera avant de l'effectuer, et cet handicap est très génant puisqu'il ne s'agit pas d'une erreur de programmation, mais d'une limitation du processeur. Nous nous proposons de dérouler une séquence d'exception déclenchée par TRAPV, qui réalisera quand même une telle division, par logiciel cette fois. Selon la présentation suivante (**figure 6.2**) :

Fig. 6.2 - Il peut être intéressant dans un programme de dérouler une séquence d'exception forçant les divisions.



V = 1 ? peut être testé directement par BVS ou indirectement par TRAPV (à l'aide d'un sous-programme d'exception dans ce cas)

La mise en forme des résultats doit avoir lieu systématiquement, puisque l'utilisateur n'a pas à se soucier de l'existence ou non d'un traitement d'exception, et que le quotient, quel qu'il soit, doit pouvoir occuper un registre à lui tout seul. En revanche, le diviseur ne doit pas excéder 65535 (valeur que le programme peut tester) puisque DIVU sera utilisé normalement, et obligatoirement.

Méthode proposée : le diviseur étant en D0 et le dividende en D1, si la division n'a pas été faite c'est que le dividende excède 16 bits, et se présente sous la forme :

A	B
16	16

ou sous forme numérique $2^{16}.A + B$

On va tout d'abord diviser A par D0, ce qui sera toujours possible. On écrit ensuite le résultat de la division de $2^{16}.A$ par D0 en multipliant le reste et le quotient par 2^{16}. Ce nouveau reste va être ajouté à B, devenant B'. On effectue ensuite la division de B' par D0, qui fournit le vrai reste final et un quotient partiel que l'on ajoute au premier quotient trouvé. Le cheminement est le suivant :

$(2^{16}.A + B)/D0 = 2^{16}.(A/D0) + B/D0$ avec $A/D0 = Q + R$;

donc $2^{16}.(A/D0) = 2^{16}.Q + 2^{16}.R$ ce qui conduit au résultat :

$D1/D0 = 2^{16}.Q + (2^{16}.R + B)/D0 = 2^{16}.Q + R'/D0; (R'/D0 = Q_p + R_v)$

$D1/D0 = 2^{16}.Q + Q_p$ avec le reste R_v

La programmation des multiplications et divisions par 2^{16} se réduit à des SWAP lorsqu'une partie du registre est nulle. Ainsi :

si D1 =	A | B	
CLR.W D1	A | 0	
SWAP D1	0 | A	division par 2^{16}

Le **programme n° 12** effectue la division de D1 par D0 dans tous les cas. Il place le quotient en D2 et le reste en D3, D0 et D1 restant inchangés.

L'INSTRUCTION CHK

CHK (check Register against bounds) examine le contenu du registre spécifié en destination (mot de poids faible uniquement),

```
                        ORG $2000;LIST 3;SIZE 1;

  2000    4282          CLR.L D2;
  2002    2601          MOVE.L D1,D3; /* DIVISEUR EN D0 */
  2004    86C0          DIVU D0,D3;  /* DIVIDENDE EN D1 */
  2006    6904          BVS ENCORE;
  2008    3403          MOVE.W D3,D2;
  200A    6010          BRA FINI;
  200C    4243          ENCORE:CLR.W D3;
  200E    4843          SWAP D3;
  2010    86C0          DIVU D0,D3;
  2012    3403          MOVE.W D3,D2;
  2014    3601          MOVE.W D1,D3;
  2016    86C0          DIVU D0,D3;
  2018    4842          SWAP D2;
  201A    3403          MOVE.W D3,D2; /* QUOTIENT EN D2 */
  201C    4243          FINI:CLR.W D3; /* RESTE EN D3  */
  201E    4843          SWAP D3;
  2020    4E4A          TRAP 10;

ENCORE                          8              $200C  PC-R.
FINI                           16              $201C  PC-R.


   0 ERRORS
a7? G
FORCEMON 2.2 > .D0 A

FORCEMON 2.2 > .D1 1FFFFF

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

UT A TRAP ERROR
PC=00002022 SR=2700=.67..... US=FFFFFFFF SS=00000786
D0=0000000A D1=001FFFFF D2=00033333 D3=00000001
D4=FFFD7FFF D5=FFFFFFFF D6=FFFF7FFF D7=FFFFFFFF
A0=FFDF7FFF A1=FFFFFFFF A2=FFFF5FFF A3=FFFFFFFF
A4=DFFF4FFF A5=FFFFFFFF A6=FFFF7FFD A7=00000786
--------------------002022    FFFF                DC.W    $FFFF

FORCEMON 2.2 > DC 1FFFFF
$1FFFFF=&2097151

FORCEMON 2.2 > DC 33333
$33333=&209715

FORCEMON 2.2 >
```

Programme 12 - Division logicielle.
D1/D0 ⟶ D2 et le reste D3

pour voir son appartenance à un intervalle donné :

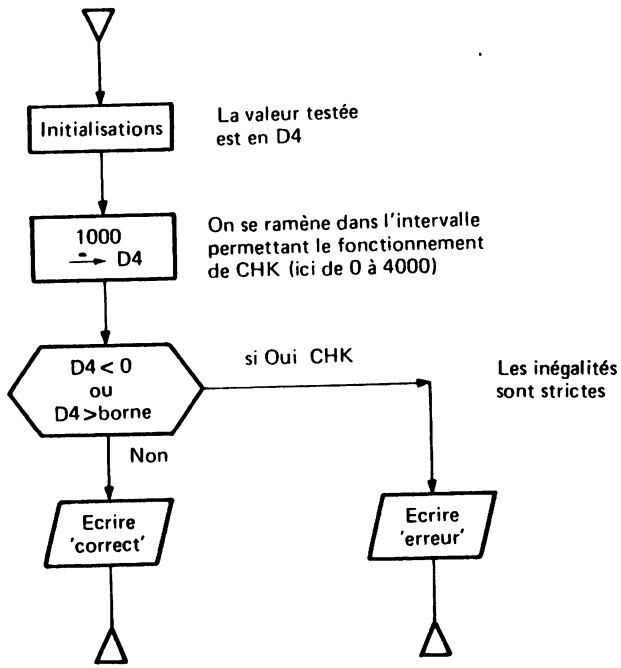
CHK source, Di

si Di est négatif ou supérieur à source, il y a déroulement d'une séquence d'exception (vecteur 6, adresse vecteur : $18).

Cette instruction est surtout intéressante par sa forme très structurée. D'une façon générale, l'utilisation systématique des traitements d'exception facilite la mise au point et la relecture des programmes.

Exemple d'application : on introduit un code matricule compris entre 1000 et 5000 (hexa) et l'on désire envoyer un message si le matricule tapé est hors de cet intervalle. L'organigramme est proposé en **figure 6.3** et le programme correspondant est donné par le **programme n° 13.**

Fig. 6.3 - Organigramme d'un programme indiquant si un code matricule frappé est en dehors d'un intervalle prédéfini (ici entre 1000_{16} et 5000_{16}).



```
A

                        ORG $2000;LIST 3;SIZE 1;

                        VECCHK:EQU @$18;
  2000    41FA0024      LEA CONTROLE,A0;
  2004    21C80018      MOVE.L A0,VECCHK;
  2008    987C1000      SUB.W $1000,D4;
  200C    49BA0012      CHK BORNE,D4;
  2010    4BFA0034      LEA OK,A5;
  2014    4DFA003F      LEA OK+15,A6;
  2018    4EB900021C14  JSR @$21C14;
  201E    4E4E          TRAP 14;
  2020    4000          BORNE:DC.W $4000;
  2022                  ENTRE:DS.L 1;;
  2026    4BFA000E      CONTROLE:LEA MESSAGE,A5;
  202A    4DFA0019      LEA MESSAGE+15,A6;
  202E    4EB900021C14  JSR @$21C14;/* ICI IL FAUDRAIT UN RTE MAIS :  */
  2034    4E4E          TRAP 14;; /* ARRET DU PROGRAMME DEFINITIF (ABUSIF) */
  2036    56414C455552  MESSAGE:DC 'VALEUR REJETEE..';
          2052454A4554
          45452E2E
  2046    56414C455552  OK:DC 'VALEUR CORRECTE..';
          20434F525245
          4354452E2E

BORNE                           11      WORD    $2020  PC-R.
CONTROLE                        13              $2026  PC-R.
ENTRE                           12      LONG    $2022  PC-R.
MESSAGE                         17              $2036  PC-R.
OK                              18              $2046  PC-R.
VECCHK                           2              @$18


   0 ERRORS
a7? G
FORCEMON 2.2 > .D4 5000

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
VALEUR CORRECTE
FORCEMON 2.2 > .D4 5001

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
VALEUR REJETEE.
FORCEMON 2.2 > .D4 1000

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
```



```
VALEUR CORRECTE
FORCEMON 2.2 > .D4 999

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
VALEUR REJETEE.
FORCEMON 2.2 > .D4 -100

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
VALEUR REJETEE.
FORCEMON 2.2 >
```

Programme 13 – Contrôle et rejet éventuel de valeurs hors bornes. ($D4 > 5000_{16}$ ou $< 1000_{16}$).

CHAPITRE 7 | les interruptions

Les interruptions sont des exceptions particulières, dont l'origine est externe (ou matérielle). Lors de la rencontre d'un signal d'interruption, le processeur, correctement programmé, pourra dérouler la séquence d'exception (donc d'interruption) qui le concerne. Les interruptions peuvent avoir pour origine le poussoir ABORT ou un circuit périphérique (signal changeant de transition sur une ligne du PIA, registre de réception plein pour l'ACIA, fin de comptage détecté pour le PTM, etc.).

Mais à la différence des exceptions internes, les interruptions peuvent être ignorées momentanément, en fonction des choix décidés à la programmation. Ainsi, on masque une interruption si le traitement afférent est moins prioritaire qu'un autre en cours. Les signaux d'interruption ne seront pas perdus pour autant, mais seulement mis en attente. Ces signaux sont classés et hiérarchisés en niveaux de priorité. La **figure 7.1** indique le classement fait sur notre processeur. Le niveau le plus élevé est le plus prioritaire.

C'est le registre SR (Status Register) qui permet de contrôler le flux des signaux ; SR.W contient, après initialisation par le moniteur, $2700. C'est la valeur des centaines (ici 7, qui autorise la prise en compte des interruptions), ainsi :

si $2700 aucune interruption reconnue, sauf le poussoir, non masquable ;

$2600 autorise une interruption de niveau 7 ;

$2500 autorise les interruptions de niveaux 7 et 6 ;

.....

$2100 autorise les interruptions de niveaux 7,6,5,4,3,2 ;

et d'une façon plus générale, il y a reconnaissance d'interruption si le numéro d'interruption est strictement supérieur au chiffre

NIVEAU	ADRESSE	SIGNAL	ORGANE PROVOQUANT L'INTERRUPTION
7	00007C	ABORT	POUSSOIR ABORT
6	000078	AC2IRQ	IRQ (HOTE)
5	000074	AC1IRQ	IRQ (TERMINAL)
4	000070	PTMIRQ	IRQ (PTM)
3	00006C	MIRQ1	IRQ (PIA partie A)
2	000068	MIRQ2	IRQ (PIA partie B)
1	000064	inutilisé	inutilisé

Fig. 7.1 - Hiérarchie des interruptions.
Le niveau le plus élevé (7) est le plus prioritaire.

des centaines de SR.W. Il est aussi possible qu'une séquence d'interruption soit à son tour interrompue par une demande provoquée par un niveau strictement supérieur (une séquence de niveau 3 peut être interrompue par une interruption de niveau 4, etc.), mais jamais dans l'autre sens. Dans ce dernier cas, l'interruption reste en attente, car en définitive les signaux d'interruption restent actifs tant que les registres (de données) des circuits périphériques les ayant provoqués, n'ont pas été lus.

Le niveau 7 est équivalent à une interruption non masquable.

Les interruptions sont transmises aux lignes $\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$ par un circuit encodeur (ici 74LS148) hiérarchisant les priorités.

Notons également que les interruptions en provenance du PIA peuvent être déroutées par INTERRUPTION PARASITE (niveau 2) si, par suite d'un choix décidé, le signal $\overline{VPA}$ n'est pas activé. C'est le cas sur notre plaquette d'essais.

L'instruction de lecture d'une interruption peut être une simple instruction MOVE source, destination; où source représente l'adresse du registre de données. Le bit d'interruption est alors mis à 0, et le processus peut recommencer.

La forme générale d'un sous-programme d'interruption peut être la suivante :

```
SOUPRO: MOVE.W $2700,SR;  (masquage, ici, de toutes les interruptions)

MOVE.B adresse-registre-données,Di; (récupération des données +
                                     bit interruption = 0)

.....
```



```
MOVE.W $2100,SR;          (autorisation de nouvelles interruptions niveaux 2 à 7)

RTE;                      (retour d'exception)
```

L'utilisation de l'instruction STOP permet une forme différente, dans le cas où le programme traite essentiellement des interruptions :

```
PRINCIPAL:.....

ICI: STOP $2100;

BRA ICI;
```

La valeur $2100 va dans SR.W et le processeur s'arrête et ne peut être relancé que par le mode TRACE, la REINITIALISATION, ou une interruption. Dans ce dernier cas, après exécution du sous-programme, le retour se fera sur BRA ICI, avec une nouvelle attente d'événement. Le sous-programme d'interruption commencera par l'instruction de masquage MOVE.W $2700,SR; (ou avec un niveau au moins égal à l'interruption traitée, et cela pour qu'une interruption de même nature ne perturbe pas le traitement en cours ; ce cas est en principe rare, puisque les arrivées d'interruption sont très lentes par rapport au traitement).

Ces modes de fonctionnement vont être détaillés en liaison avec les circuits spécialisés d'entrées/sorties susceptibles de travailler en mode interruption comme le PIA, le PTM, et les ACIA implantés sur notre micro-ordinateur.

CHAPITRE 8 | le PIA

PROGRAMMATION DES LIGNES D'E/S

Le PIA (6821) permet la programmation de seize lignes d'entrées/sorties (deux fois un octet) et de quatre lignes d'interruption dont deux peuvent être utilisées comme sorties seules pour le contrôle de la périphérie. Ce circuit est sensiblement symétrique et comporte une partie A, PIAA, et une partie B, PIAB, (dont la programmation est identique) chacune composée de huit lignes E/S (soit un octet) et deux lignes d'entrée d'interruption dont une utilisable en sortie (appelées CA1 et CA2).

Toutes ces lignes peuvent véhiculer des signaux de type TTL (0 ou 5 volts).

Tout ce qui sera expliqué par la suite concernera uniquement la partie A pour simplifier, et on pourra passer à la partie B en modifiant les adresses des registres associés.

Pour commander les dix lignes de la partie A on utilise trois registres :

- un registre précisant le sens de transfert des données d'adresse, ici, $05CEF1 (DDRA) ;
- un registre de données du périphérique de même adresse (de nom ORA) ;
- un registre de contrôle (permettant de choisir les options voulues) d'adresse $05CEF3 (CRA).

Pour la partie B du PIA, les adresses sont respectivement $05CEF5 et $05CEF7. Les registres DDRA et ORA ont la même adresse et c'est la valeur du bit 2 placé dans le registre de contrôle qui permet la sélection. S'il vaut 1 on accède à DDRA, sinon à ORA.

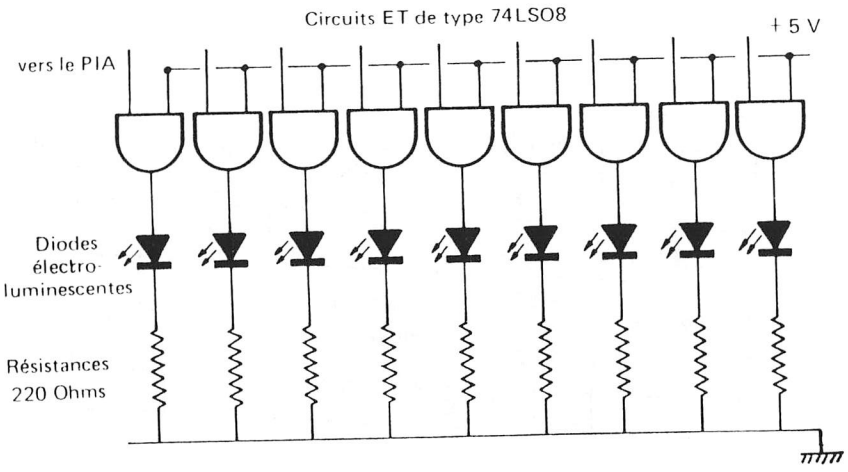
Le registre DDRA indique quelles sont les lignes, parmi les huit qui seront utilisées en entrée ou en sortie ; un 1 placé dans

le bit correspondant de DDRA indique que la ligne est une sortie et un 0 une entrée. Exemple :

```
ORG $800; SIZE 1;
PIAA:EQU @$05CEF1;
MOVE 0, PIAA+2;        (0 dans le registre de contrôle → bit 2 = 0 → accès DDRA)
MOVE $0F,PIAA;         (lignes 0,1,2,3 en sorties ; 4,5,6,7 en entrées)
MOVE 4,PIAA+2;         (bit 2 = 1 → accès à ORA donc en entrées-sorties sur le PIAA)
.....                  ici on introduit une valeur en D0, depuis le terminal...
MOVE D0,PIAA;          (envoi de tensions sur les lignes 0,1,2,3)
.....
MOVE PIAA,D1;          (réception de tensions sur les lignes 4,5,6,7, stockées en D1 ;
```

Il est donc relativement simple de programmer ce type d'entrées-sorties. Les autres bits du registre de contrôle permettent la programmation des lignes CA1 et CA2 comme sorties ou entrées d'interruption. Le montage de la **figure 8.1**, facilement réalisable avec des circuits TTL classiques, assure la visualisation des entrées-sorties.

Fig. 8.1 - Dispositif de visualisation des E/S en liaison avec le PIA.



Le programme affiche, en allumant une diode électroluminescente, le contenu du registre D0.B (un '1' allume la diode dans la position correspondante). Après programmation du PIAA, les routines spécialisées mettent le caractère frappé dans D0 et le renvoient sur le terminal (pour écriture éventuelle). Le code hexadécimal est alors affiché sur les diodes.

Note : sur notre plaquette, le PIAA est spécialisé en sortie (les lignes sont amplifiées et peuvent supporter dix charges TTL et le PIAB est normal).

PROGRAMMATION DES LIGNES CA1 ET CA2

Le format du mot de contrôle (devant être transmis à PIAA+2) est donné dans le tableau de la **figure 8.2** (le constructeur note les bits de 0 à 7 selon leur poids binaire de 2^0 à 2^7). La programmation de CA1 comme entrée d'interruption est résumée par le tableau de la **figure 8.3**. On constate qu'on a le choix entre une transition montante ou descendante (le signal passe de 0 à 5 V ou de 5 V à 0 V ce qui représente les niveaux de tension normalisés, en logique et circuiterie TTL - haut "1" et bas "0" -). On peut également vouloir que l'interruption soit inhibée, et dans ce cas, la transition attendue positionne seulement l'indicateur IRQA1 (bit 7) testable par programme. Lorsque la demande d'interruption n'est pas inhibée, outre l'indicateur qui passe à 1, la séquence d'exception est normalement activée. Cette interruption a pour numéro de vecteur 27 auquel est attribué l'auto-vecteur d'interruption de niveau 3.

Remarque : les circuits PTM, ACIA, PIA construits au départ pour s'intégrer à la série 6800 nécessitent une logique particulière lorsqu'ils sont employés dans un environnement 68000, celle de l'auto-vectorisation prise en charge par le 68000. Les circuits conçus directement pour le 68000 utilisent une autre logique en délivrant eux-mêmes un numéro de vecteur placé dans un de leurs registres programmables. Il n'est donc pas possible ici, sans modifier le câblage, de modifier le numéro de l'auto-vecteur. Les priorités d'interruption qui en découlent sont donc figées.

Le programme suivant permet de voir fonctionner une interruption :

```
ORG $800; LIST 3; SIZE 1;
PIAA:EQU @$5CEF1;
MOVE $07,PIAA+2;
```

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	Contrôle de CA2			Accès à DDRA	Contrôle de CA1	
CRB	IRQB1	IRQB2	Contrôle de CB2			Accès à DDRB	Contrôle de CB1	

Fig. 8.2 - Format des mots de contrôle CRA et CRB du PIA.

CRA-1 (CRB-1)	CRA-0 (CRB-0)	Transition active de l'entrée d'interruption CA1 (CB1)	Indicateur d'interruption CRA-7 (CRB-7)	Demande d'interruption du MPU $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	↓Active	Mis à un sur ↓de CA1 (CB1)	Inhibée – $\overline{IRQ}$ reste à l'état haut
0	1	↓Active	Mis à un sur ↓de CA1 (CB1)	Passe à l'état bas quand l'indicateur CRA-7 (CRB-7) passe à l'état haut
1	0	↑Active	Mis à un sur ↑de CA1 (CB1)	Inhibée – $\overline{IRQ}$ reste à l'état haut
1	1	↑Active	Mis à un sur ↑de CA1 (CB1)	Passe à l'état bas quand l'indicateur CRA-7 (CRB-7) passe à l'état haut

↑ Transition positive

↓ Transition négative

Fig. 8.3 - Contrôle des entrées d'interruption CA1 et CB1.

```
MOVE.W $2000,SR;
TRAP 14;
```

Le registre de contrôle contient $07, ce qui signifie : ($07 = 0000 0111) interruption sur transition montante de CA1 ; également accès à DDRA.

Un simple interrupteur permettant le passage de la tension de 0 à 5 V (via un TTL "OU") provoquera à l'enclanchement le déroulement de la séquence d'interruption, qui non écrite de façon explicite, sera entièrement prise en compte par le moniteur, avant l'arrêt du processus. A noter qu'il ne sera pas possible de lancer un programme quelconque à ce niveau, sans retomber sur le

traitement de l'exception, puisque d'une part, le bit indicateur n'est pas remis à zéro et, que, d'autre part, le contenu du masque (dans SR) est inchangé. Mais il suffit de réinitialiser, par le poussoir ABORT, pour que tout rentre dans l'ordre.

Contrôle de CA2

Le double rôle de CA2 (interruption ou sortie) impose l'utilisation d'un bit de contrôle de plus. Dans le cas des interruptions, le bit 5 du registre de contrôle est à zéro. Dans le cas où CA2 est une sortie, ce bit est à 1. Les tableaux des **figures 8.4** et **8.5** résument tous les cas possibles. On remarque que les différents niveaux de commande peuvent dépendre soit d'une transition active de CA1, soit, d'une toute autre façon, suivre la valeur du bit 3 du registre de contrôle. Le fonctionnement de CB2 (**figure 8.6**) est très peu différent. Si CA2 est programmé comme entrée d'interruption, celle-ci sera du même niveau que celle provoquée par CA1. Ce ne sera donc pas un auto-vecteur différent. Le programmeur, pour reconnaître l'origine exacte de l'interruption pourra tester les bits 6 et 7 de CRA de la façon suivante :

```
SOUPRO-INTER: MOVE.W $2700,SR;
MOVE.B PIAA+2,D0;
BMI IRQA1;
IRQA2:.....
```

cela dans le cas où l'on attend l'une ou l'autre exclusivement.

Fig. 8.4 - Contrôle de CA2 ou CB2 comme entrée d'interruption (CRA5 = 0).

CRA-5 (CRB-5)	CRA-4 (CRB-4)	CRA-3 (CRB-3)	Transition active de l'entrée d'interruption CA2 (CB2)	Indicateur d'interruption CRA-6 (CRB-6)	Demande d'interruption du MPU $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	0	↓Active	Mis à un sur ↓de CA2 (CB2)	Inhibée – $\overline{IRQ}$ reste à l'état haut
0	0	1	↓Active	Mis à un sur ↓de CA2 (CB2)	Passe à l'état bas quand l'indicateur CRA-6 (CRB-6) passe à un.
0	1	0	↑Active	Mis à un sur ↑de CA2 (CB2)	Inhibée – $\overline{IRQ}$ reste à l'état haut
0	1	1	↑Active	Mis à un sur ↑de CA2 (CB2)	Passe à l'état bas quand l'indicateur CRA-6 (CRB-6) passe à un.

Notes : 1 ↑ Transition positive

2 ↓ Transition négative

3 L'indicateur d'interruption CRA-6 (CRB-6) est mis à zéro par une lecture du registre données

4 Si CRA-3 (CRB-3) est à zéro (interruption inhibée) quand l'interruption se produit et est, par la suite, mis à un, $\overline{IRQA}$ ($\overline{IRQB}$) passe à l'état bas après que CRA-3 (CRB-3) ait été mis à un.

CRA-5	CRA-4	CRA-3	Mis à zéro CA2	Mis à un
1	0	0	Bas sur la transition négative de la première impulsion E qui suit une lecture du registre données de la périphérie A.	Haut quand l'indicateur d'interruption CRA-7 est mis à un par une transition active du signal CA1.
1	0	1	Bas sur la transition négative de la première impulsion E qui suit une lecture du registre données de la périphérie A.	Haut sur la transition négative de la première impulsion E qui se produit tandis que le circuit est désélecté.
1	1	0	Bas quand CRA-3 est mis à zéro par une écriture du registre de contrôle A.	Toujours bas tant que CRA-3 est à zéro. Passera à l'état haut quand CRA-3 sera mis à un par une écriture du registre de contrôle A.
1	1	1	Toujours haut tant que CRA-3 est à un. Passera à l'état bas quand CRA-3 sera mis à zéro par une écriture du registre de contrôle A.	Haut quand CRA-3 est mis à un par une écriture du registre de contrôle A.

Fig. 8.5 - Contrôle de CA2 comme sortie (CRA5 = 1).

CRB-5	CRB-4	CRB-3	Mis à zéro CB2	Mis à un
1	0	0	Bas sur la transition positive de la première impulsion E qui suit une écriture du registre données de la périphérie B.	Haut quand l'indicateur d'interruption CRB-7 est mis à un par une transition active du signal CB1.
1	0	1	Bas sur la transition positive de la première impulsion E qui suit une écriture du registre données de la périphérie B.	Haut sur la transition positive de la première impulsion E qui suit une impulsion E qui était arrivée tandis que le circuit était désélecté.
1	1	0	Bas quand CRB-3 est mis à zéro par une écriture du registre de contrôle B.	Toujours bas tant que CRB-3 est à zéro. Passera à l'état haut quand CRB-3 sera mis à un par une écriture du registre de contrôle B.
1	1	1	Toujours haut tant que CRB-3 est à un. Passera à l'état bas quand CRB-3 sera mis à zéro par une écriture du registre de contrôle B.	Haut quand CRB-3 est mis à un par une écriture du registre de contrôle B.

Fig. 8.6 - Contrôle de CB2 comme sortie (CRB5 = 1).

Tout ce qui vient d'être écrit pour la partie A du PIA reste valable pour la partie B. En général, le numéro d'auto-vecteur est 26, et l'adresse associée $68, pour des interruptions de niveau 2.

Mais, selon le câblage de la plaquette, certains signaux ne sont pas renvoyés vers le 68000. En particulier, face à un périphérique de la série 6800, il est nécessaire que le 68000 reçoive sur

sa broche $\overline{VPA}$ (validation adresse périphérique) un signal bas de ce périphérique. Faute de quoi, le processeur traite l'interruption reçue, comme une interruption parasite (vecteur 24, adresse $60), et de niveau 2.

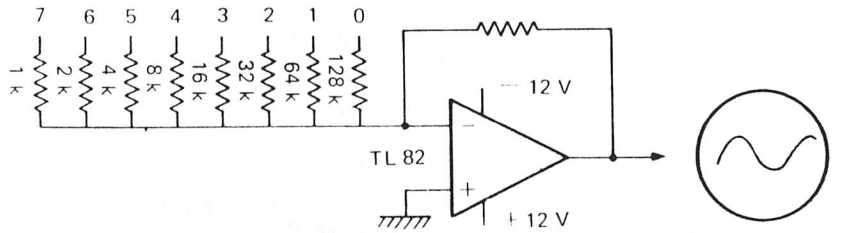
Si cela se généralise, la programmation devient moins claire puisqu'il revient au programmeur de tester tous les bits d'interruption pour déceler l'origine des interruptions, (cela ne pose aucun problème particulier mais occasionne une légère perte de temps).

REALISATION D'UNE CONVERSION NUMERIQUE-ANALOGIQUE

On réalise cette conversion via le PIA et un amplificateur opérationnel classique. Cette technique est maintenant très répandue et nous en rappelons les grandes lignes. Soit U une tension quelconque conventionnelle (0,01 V, 0,02 V, etc.) ; on associe aux différents bits d'un registre (ici le PIA) une tension proportionnelle à leur poids binaire : le bit 0 pèse U, le bit 1 pèse 2U puis 4U, 8U, 16U, 32U, 64U, 128U avec 8 bits. Il est donc possible d'exprimer de 0U (tous les bits à zéro) à 255U par pas de 1U. On remarque qu'augmenter le nombre de bits, pour une tension maximale donnée, revient à augmenter la précision, par diminution du pas U initial. Aussi, trouve-t-on des conversions jusqu'à 16, voire 18 bits. Il existe dans le commerce des convertisseurs numériques-analogiques (*) dont le rôle est la transformation d'une valeur binaire en un voltage proportionnel, avec maintien jusqu'à la réception d'une autre valeur binaire.

Mais on peut aussi réaliser cette transformation grâce à un amplificateur opérationnel utilisé en sommateur. Dans le montage de la **figure 8.7** on effectue une somme de tensions. Il suffit donc

Fig. 8.7 - Un amplificateur opérationnel monté en sommateur permet de réaliser une conversion numérique/analogique via le PIA.



(*) en anglais : Digital-to-Analog Converter, DAC.

d'ajuster huit résistances pour que les gains varient selon les puissances de 2, et qu'ainsi, il y ait correspondance entre la valeur numérique stockée dans le PIA et la valeur analogique issue de l'amplification.

Le **programme n° 14** permet de visualiser une sinusoïde préalablement stockée en mémoire sur 64 valeurs entre les adresses $2000 et $2040. Chaque valeur est expédiée vers le PIA, puis la routine WAIT permet d'attendre avant de passer à la valeur suivante. On règle de cette façon la fréquence de cette sinusoïde mais de manière coûteuse et sans grande précision ; le PTM, temporisateur programmable, sera parfaitement adapté à un tel découpage temporel et assurera une fréquence ajustable avec précision, comme on le verra plus loin.

Les valeurs expédiées sur le PIA représentent l'état d'une sinusoïde, calculée point par point, aux instants 0, T, 2T, 3T,... 63T ce qui correspond aux 64 valeurs de la fonction
Y = 120 + 120 x SIN(6.28/64)T pour T = 0, 1, 2, ...63 (pour T = 64 on revient au départ). On constate que Y est toujours positif et que sa valeur, variant de 0 à 240, utilise les 8 bits d'un octet pour être correctement transmise, à condition toutefois de considérer ce code comme du binaire pur (tous les bits ont un poids positif) et non en binaire complémenté. En effet, l'amplificateur opérationnel n'est pas apte à distinguer dans notre montage, si la quantité est positive ou négative, aussi seules des quantités positives sont envoyées. Si besoin est, le décalage de cette tension par rapport au zéro électrique pourra être supprimé par un autre amplificateur opérationnel de façon très simple.

Les valeurs stockées, arrondies à l'entier le plus proche, sont les suivantes :

78 84 8F 9B A5 B0 BB C4 CC D4 DC E2 E6 EA EE EF F0 EF
EE EA E6 E2 DC D4 CC C4 BA B0 B5 9B 8F 84 7C 60 56 4A
40 26 3C 24 1C 14 0E 0A 06 02 01 00 01 02 06 0A 0E 14 1C
24 3C 26 40 4A 56 60 70

(on verra plus loin comment le Basic résidant peut fournir simplement ces valeurs). Un oscilloscope branché directement à la sortie de l'amplificateur permet de voir la sinusoïde produite. Pour des fréquences situées dans le domaine audible (donc si WAIT retarde suffisamment le processus, et cela est réglable) l'écoute est possible avec un écouteur. S'il faut un temps t entre l'émission de deux valeurs successives, une période complète (64 points) est de 64t, et la fréquence de 1/64t. Deux problèmes sont toujours liés à la conversion numérique-analogique :

- pour reproduire une fréquence réelle F par conversion, il faut fournir au convertisseur au moins 2F points (théorème de Shannon). Par exemple, pour traiter un signal musical jusqu'à 20 000 Hz, il faut fournir, en une seconde, 40 000 valeurs numériques au moins ;
- le calcul par l'ordinateur se termine toujours par l'obtention d'une valeur entière, seule transmissible au moyen de bits. On est donc amené la plupart du temps à effectuer un arrondi (8,6 devient 9) ou une troncature (8,6 devient 8).


```
                        ORG $1000;LIST 3;
                        SIZE 1;
                        PIA:EQU @$5CEF1;
 1000    13FC00000005   MOVE 0,PIA+2;
         CEF3
 1008    13FC00FF0005   MOVE 255,PIA; /* TOUTES LIGNES EN SORTIE */
         CEF1
 1010    13FC00040005   MOVE 4,PIA+2; /* ACCES REGISTRE DONNEES */
         CEF3
 1018    247C00002041   MOVEA.L $2041,A2;
 101E    227C00002000   ENCORE:MOVEA.L $2000,A1;
 1024    13D90005CEF1   SUITE:MOVE (A1)+,PIA;/* EMISSION VERS LE PIA */
 102A    4EBA0008       JSR WAIT; /* TEMPORISATION  */
 102E    B5C9           CMPA.L A1,A2;
 1030    66F2           BNE SUITE;
 1032    60EA           BRA ENCORE;
 1034    203C00010000   WAIT:MOVE.L $10000,D0;
 103A    5380           SOUS:SUBQ.L 1,D0;
 103C    66F6           BNE WAIT;
 103E    4E75           RTS;;

ENCORE                          8               $101E  PC-R.
PIA                             3               @$5CEF1
SOUS                           15               $103A  PC-R.
SUITE                           9               $1024  PC-R.
WAIT                           14               $1034  PC-R.


   0 ERRORS
A?? G
```

```
FORCEMON 2.2 > MD 2000
002000    78 84 8F 9B A5 B0 BB C4  CC D4 DC E2 E6 EA EE EF  x...Z0;DLT\bfjno

FORCEMON 2.2 >
```

Programme 14 – Visualisation d'une sinusoïde stockée point par point à partir de l'adresse 2000_{16}.

Cela se répercute sur la qualité du signal analogique obtenu, altéré par un "bruit de quantification", qui est d'autant plus faible que le nombre de bits est grand (division par 2 de l'erreur d'arrondi par bit supplémentaire).

CONVERSION ANALOGIQUE-NUMERIQUE

Cette conversion (en anglais : Analog-to-Digital Converter, ADC) procède des mêmes principes, et permet d'introduire dans la mémoire, sous forme numérique, une valeur de tension fournie par un dispositif analogique. Des convertisseurs de ce type sont disponibles chez de nombreux constructeurs, mais ne sont guère réalisables de façon élémentaire.

La tension est lue puis est convertie après un certain nombre de cycles d'horloge. A la fin de la conversion, un signal d'interruption indique au processeur qu'il peut prendre en compte les valeurs binaires présentes sur les broches de sortie du convertisseur. Le modèle ADC 0804, très courant et simple à mettre en oeuvre, se programme selon ces principes.

La **figure 8.8** précise ses liaisons entre le PIA et une sortie du PTM. La tension à mesurer (ici entre 0 et 5 V) sera envoyée sur la broche 6. La masse analogique (borne 8) et la masse numérique (borne 10) seront réunies.

Fonctionnement : une impulsion sur la borne 3 déclenche la conversion. A la fin de la conversion un signal bas apparaît sur la borne 5 ; ce signal pourra être traîté comme interruption externe (transition descendante). La gestion de ce signal est assurée par la logique interne du convertisseur (il se remet seul au niveau haut, permettant des conversions en série).

Le **programme n° 15** prend en compte une tension fournie par un diviseur potentiométrique et affiche la valeur lue (mais en code ASCII). A 0 V correspondra la valeur binaire 00000000 et à 5 V, la valeur 11111111.

Si, par exemple, on passe graduellement d'une tension de 0 à 2,5 V, on voit apparaître les chiffres, les lettres majuscules, puis les minuscules. Au-delà de cette tension, la valeur binaire ne correspond plus à un code imprimable. Si on désire visualiser toutes les conversions de 00 à FF, on pourra intégrer le programme n° 4 qui affiche en hexadécimal le contenu d'un registre.

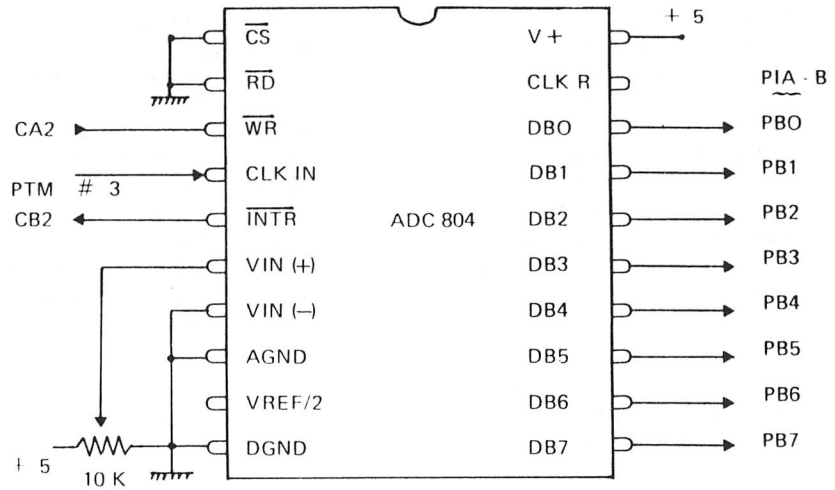


Fig. 8.8 - Conversion analogique-numérique.

Entre deux conversions, le programme n'affiche la dernière valeur lue que si elle diffère de la précédente. Enfin, le sous-programme WAIT permet d'ajuster l'intervalle entre deux conversions. On notera les points suivants :

ligne 15 : le PTM (#3) travaille en mode continu, sortie disponible... ;
ligne 21 : interruption sur transition descendante sur CB2, accès aux lignes de données ;
ligne 24 : l'interruption pourra relancer le programme, mis en sommeil.

Dans une véritable application, les tensions peuvent provenir de capteurs divers (température, vitesse, pression), d'un signal vocal... ; ces signaux feront l'objet de stockage avant traitement... plus complexe.

En définitive, le PIA permettant de transmettre vers (ou depuis) l'extérieur le contenu de la mémoire, à vitesse élevée, est le périphérique privilégié pour tous les transferts à haut débit (et à courte distance) limités seulement par la vitesse du processeur.

```
PHYSICAL ADDRESS=0000A100 0000A39?
     1   ORG $1000;
     2  LIST 3;
     3  SIZE 1;
     4  PTM:EQU @$4CF41;
     5  PIA:EQU @$5CEF1;
     6  PIB:EQU PIA+4;
     7  UN:EQU $3C;
     8  ZERO:EQU $34;
     9  VECT:EQU @$60;
    10  LEA SOUP,A6;
    11  MOVE.L A6,VECT;
    12  MOVE 0,PIB+2;
    13  MOVE 0,PIB;
    14  MOVE 0,PTM+2;
    15  MOVE $92,PTM;
    16  MOVE $00,PTM+12;
    17  MOVE $00,PTM+14;
    18  MOVE 1,PTM+2;
    19  MOVE 1,PTM;
    20  MOVE 0,PTM;
    21  MOVE $0C,PIB+2;
    22  DEPART:  MOVE ZERO,PIA+2;
    23  MOVE UN,PIA+2;
    24  STOP $2100;
    25  BSR WAIT;
    26  CMP D0,D7;
    27  BEQ DEPART;
    28  JSR @$21C7C;
    29  MOVE ' ',D0;
    30  JSR @$21E24;
    31  MOVE D7,D0;
    32  JSR @$21E24;
    33  BRA DEPART;
    34  WAIT:MOVE.L $1000,D3;
    35  BOU:SUB.L 1,D3;
    36  BNE BOU;
    37  RTS;
    38  SOUP:MOVE.W $2700,SR;
    39  MOVE PIB,D7;
    40  RTE;
    41  ;

STOP
to detach file10
r 14:48 1.572 6?
```

Programme 15 - Conversion analogique-numérique.

CHAPITRE 9 | le PTM

ORGANISATION

Le PTM (Temporisateur Programmable EF 6840) comporte trois compteurs indépendants (notés #1, #2, #3) qui peuvent être utilisés pour des mesures de fréquences et d'intervalles de temps, pour la fabrication d'horloges TTL (à rapports cycliques variables), pour la temporisation d'événements, pour des traitements en temps réel, pour générer des interruptions.

A chacun de ces compteurs sont associés trois registres de contrôle. Un seul registre d'état permet, principalement, de retrouver l'origine d'une interruption.

Chaque compteur est programmable, en durée, par deux registres tampons, dont le contenu exprimé en poids fort et poids faible, est d'autant plus grand que la période désirée est grande (à chaque cycle d'horloge, cette valeur binaire est décrémentée de 1, jusqu'à la valeur 0, ou jusqu'à ce qu'une autre condition préprogrammée soit rencontrée, et le cycle pourra soit recommencer, soit se terminer, ceci indépendamment du registre de contrôle associé).

L'horloge qui commande ce décomptage est soit l'horloge du système lui-même (ici F = 0,8 MHz : cette fréquence provient du signal E du 68000. C'est le dixième de la fréquence de fonctionnement du 68000, fixée ici à 8 MHz), soit une horloge extérieure TTL connectée sur la broche d'entrée C1 (ou C2 ou C3 selon le compteur) dont la fréquence peut aller du continu jusqu'à 1 MHz pour ce modèle de circuit. Deux autres broches sont aussi associées à chaque compteur : O1 (respectivement O2, O3) qui permet de récupérer le signal fabriqué par un compteur, pour une application extérieure, et l'entrée de déclenchement G1 (respectivement G2, G3) utilisable pour des mesures d'intervalle de temps, de largeur d'impulsion, ou pour relancer un cycle achevé.

La **figure 9.1** indique les contenus des trois registres de contrôle ; on remarquera, en particulier, le rôle spécial joué par le bit 0.

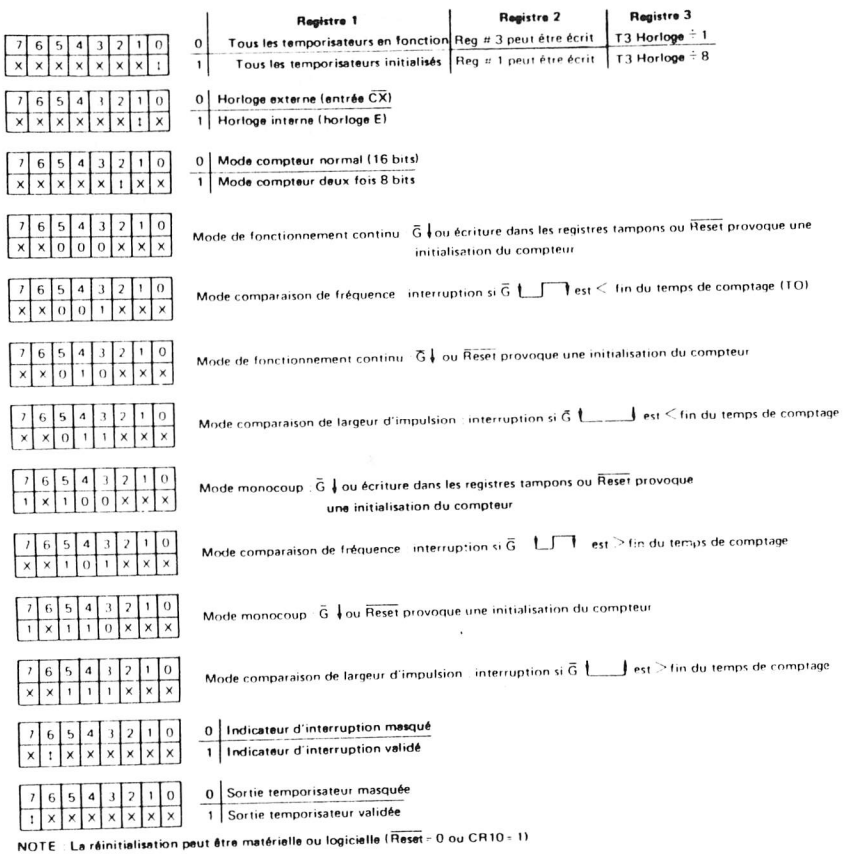


Fig. 9.1 - Programmation des registres de contrôle du temporisateur programmable.

Sur notre système les adresses du PTM sont les suivantes :

- registres de contrôle #1 et #3:$4CF41 ;
- registre de contrôle #2 (écriture seule) et registre d'état (lecture seule):$4CF43 ;
- registres tampons # 1:$4CF45 $4CF47 (PTM+4 et PTM+6) ;
- registres tampons # 2:$4CF49 $4CF4B (PTM+8 et PTM+10) ;
- registres tampons # 3:$4CF4D $4CF4F (PTM+12 et PTM+14).

dans l'ordre : poids forts (MSB) et poids faibles (LSB).

On prendra dans les exemples, comme ici :

PTM: EQU @$4CF41 ;

QUELQUES APPLICATIONS DU PTM

Fabrication d'une horloge

Nous désirons obtenir une horloge de fréquence 1000 Hz, avec sortie disponible sur # 2

```
ORG $1000;LIST 3; SIZE 1;
PTM:EQU @$04CEF1;
MOVE $83,PTM+2;          (accès reg. 1, horloge interne, mode
                          normal, mode continu, sortie validée)
MOVE $01, PTM+8;  }      chargement des registres tampons
MOVE $F8, PTM+10; }      pour la fréquence désirée dans cet ordre
MOVE 0, PTM;             (tous les temporisateurs en fonction)
MOVE 1, PTM;             (tous les temporisateurs initialisés ;
                          ceci déclanche le démarrage)
TRAP 14;
```

La période a en effet pour valeur 2x(N+1)T. N est le contenu des registres tampons, T vaut 1,25 μS. On en déduit que pour la fréquence donnée (période de 10^{-3} s), il faut prendre N = 399 soit $18F.

Un oscilloscope, ou un amplificateur, placé sur la broche 02 permettra de voir, ou d'entendre, un signal carré symétrique de fréquence 1000 Hz.

Remarque : il est souvent recommandé de commencer par charger les registres tampons, spécialement pour le temporisateur #1.

Obtention de très basses fréquences

Le temporisateur #3, grâce à son précompteur sur 3 bits peut produire des périodes huit fois plus longues que les autres temporisateurs. La plus grande période que l'on peut obtenir est alors de :

65535 x 8 x 1,25 μS = 0,65535 s en mode monocoup (le double en mode continu) donc de l'ordre de 0,6 seconde. Pour obtenir des temporisations bien plus longues, il suffit de combiner deux temporisateurs dont l'un fabrique l'horloge extérieure de l'autre. La durée de la plus grande période ainsi obtenue, peut alors s'exprimer par :

T max. = 2x(N+1)T (T durée entre deux transitions négatives de l'horloge, soit une période) soit environ : 2 x (65535) x (2 x 0,655) = 171 701 s soit 47 heures. Pour réaliser une temporisation de ce type, il faudra relier la sortie de #3 à l'entrée C2 de #2, et programmer pour le temporisateur # 2, l'horloge externe.

La **figure 9.2** décrit le mode deux fois 8 bits qui permet d'obtenir des rapports cycliques variables.

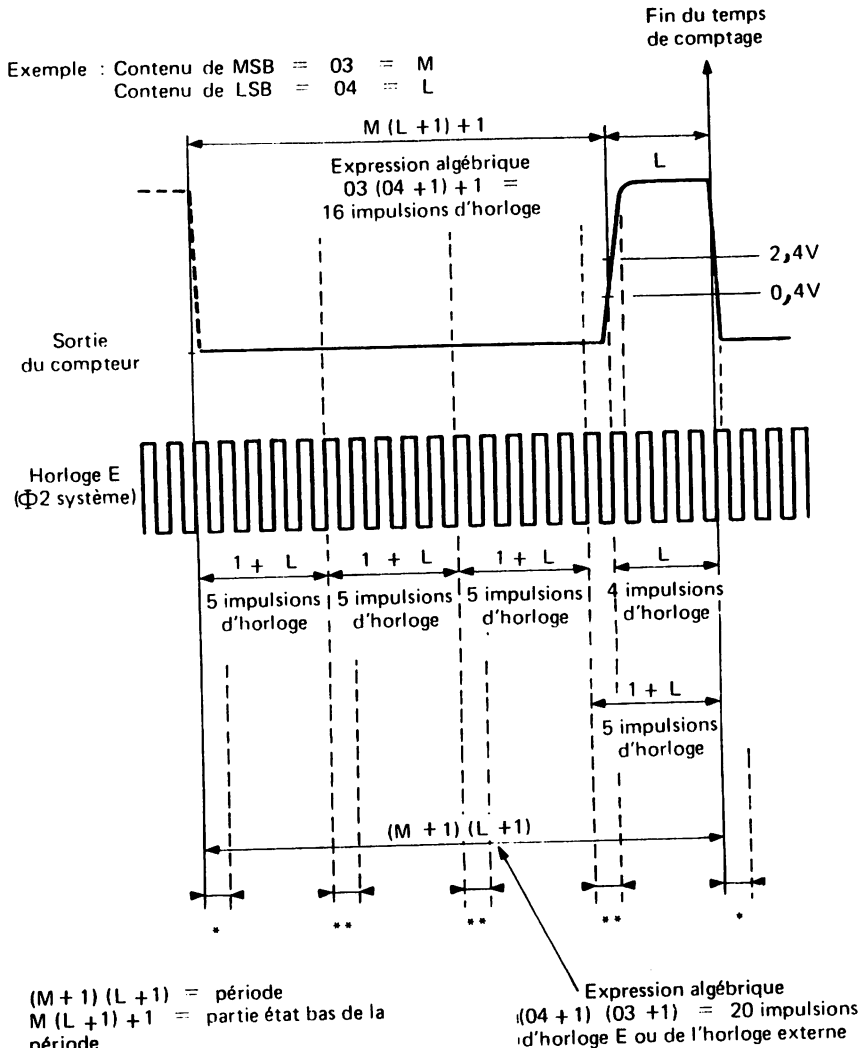


Fig. 9.2 - Mode de fonctionnement deux fois 8 bits du temporisateur programmable.

Le mode monocoup

Le mode monocoup crée des cycles de temporisation au bout desquels une interruption peut être éventuellement provoquée. C'est donc une méthode qui permet le déroulement d'un programme, et d'actions consécutives, selon un séquencement donné. Le registre d'état, à lecture seule, contient (bits 0, 1 et 2) les indicateurs d'interruption correspondant aux temporisateurs #1, #2, #3, alors que le bit 7 est l'indicateur commun qui lui ne passe à un que sous les conditions suivantes : un des bits 0, 1 ou 2 vaut "1", et le bit 6 du registre de contrôle correspondant est aussi à "1". Ainsi, on pourra toujours tester l'un des bits 0, 1 ou 2 mis à "1", à la fin d'un cycle monocoup (cas normal) ou encore, si l'on demande qu'une interruption soit provoquée (bit 6 du registre de contrôle), déterminer le temporisateur qui en est à l'origine, par l'étude des bits 0, 1 et 2.

Le **programme n° 16** permet la visualisation sur la console de tous les codes ASCII (imprimables ou non) l'un après l'autre, selon une vitesse programmable. Le temporisateur #3 est programmé en mode monocoup, et, dès qu'un caractère est expédié, le programme se branche sur la routine ATTEND qui se contente de tester le bit 2 du registre d'état. Lorsque ce bit passe à un, la boucle d'attente (poolling) est quittée, le mode monocoup a terminé son cycle. Il faut donc le réinitialiser avant de traiter le caractère suivant. L'intérêt du mode monocoup tient au fait que, quel que soit le traitement envisagé (différentes branches du programme), il y a resynchronisation du séquencement, bien sûr, tant qu'aucune branche du programme n'excède en temps d'exécution la durée d'un cycle de comptage.

AMELIORATION DES PERFORMANCES

Le temporisateur peut permettre l'amélioration des performances d'un programme donc de choisir les meilleures séquences d'instructions. Dans les systèmes actuels, la taille mémoire ne fait plus vraiment défaut autour d'un 68000. La vitesse de traitement, en revanche, dans une optique temps-réel peut être critique. Pour étudier globalement un programme, on peut programmer en mode d'interruption un des temporisateurs, selon la méthode suivante :

- choisir le mode monocoup, avec interruptions autorisées, de durée 10^{-5}s, 10^{-4}s, etc. ;
- compter, dans le sous-programme d'interruption, le nombre d'interruptions détectées jusqu'à la fin du processus (moment où l'on en interdit -SR.W=$2700- la prise en compte).

```
                        ORG $2000;LIST 3;SIZE 1;

                        PTM:EQU @$4CF41;
2000    13FC00000004    MOVE 0,PTM+2;
        CF43
2008    13FC00B30004    MOVE $B3,PTM;/* MODE MONOCOUP-DIVISION PAR 8 */
        CF41
2010    13FC00FF0004    MOVE $FF,PTM+12;
        CF4D
2018    13FC00FF0004    MOVE $FF,PTM+14; /* DUREE MAXIMALE */
        CF4F
2020    13FC00010004    MOVE 1,PTM+2;
        CF43
2028    13FC00000004    MOVE 0,PTM;
        CF41
2030    103C0030        MOVE #'0',D0;
2034    12390004CF43    PI:MOVE PTM+2,D1; /* BOUCLE D ATIENTE */
203A    C23C0004        AND 4,D1;
203E    67F4            BEQ PI;/* FIN DE CYCLE: ON PASSE EN SEQUENCE */
2040    5200            ADDQ 1,D0;
2042    4EB900021C7C    JSR @$21C7C;
2048    4EB900021E24    JSR @$21E24;
204E    13FC00010004    MOVE 1,PTM; /* TEMPORISATEUR INITIALISE */
        CF41
2056    13FC00000004    MOVE 0,PTM; /* TEMPORISATEUR ACTIF */
        CF41
205E    60D4            BRA PI;

PI                              10              $2034  PC-R.
PTM                              2              @$4CF41

   0 ERRORS
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\] _`abcdefghijklmnopqrstuvwxyz{|}
FORCEMON 2.2 >
```

Programme 16 – Sortie de caractères après temporisation sur la console. Le résultat est obtenu après une minute environ.

Il suffit alors de comparer les valeurs trouvées et d'en déduire les temps de traitement.

L'organigramme est simple (**figure 9.3**).

Une méthode plus rapide peut être employée pour analyser une courte séquence dont le déroulement complet peut tenir à l'intérieur du plus long cycle monocoup programmable (qui vaut sur #3, avec horloge ÷8 : 65535 μs).

Le temporisateur est mis en mode interruption, mais on n'écrit pas la séquence correspondante dans le programme pour simplifier. Il suffit ensuite d'activer le temporisateur et la séquence en test ; si la séquence prend plus de temps que le cycle de comptage, elle sera interrompue et les messages du moniteur apparaîtront sur l'écran. Dans l'autre cas rien ne se passera. Le seul travail consistera à modifier les contenus des registres tampons pour déterminer le plus petit cycle permettant le traitement complet.

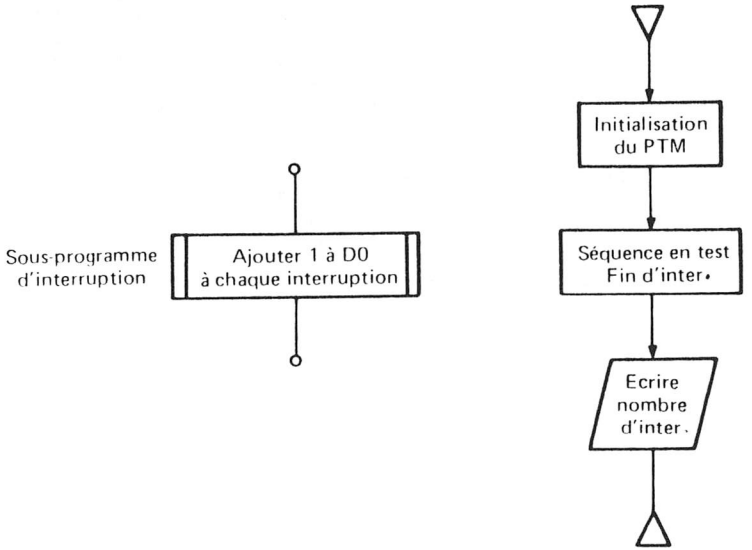


Fig. 9.3 – Le PTM peut mesurer la durée exacte d'un programme ou d'une séquence d'instructions.

On a donc une très bonne évaluation du temps mis, alors que dans le cas précédent, une partie - faible sans doute - du traitement global consistait à gérer les interruptions (masquage, sous-programme, retour, etc.). Un exemple complet sera donné dans le chapitre consacré aux performances. Le programme peut avoir la forme suivante :

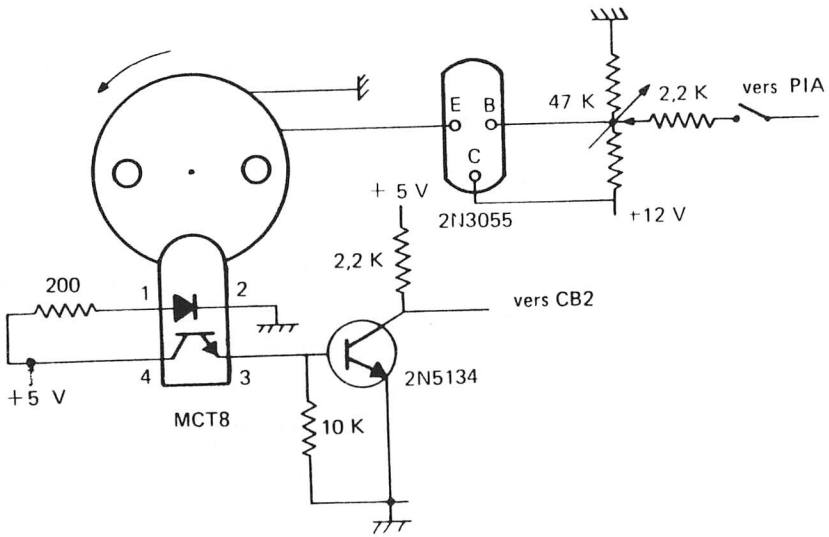
```
ORG $800;LIST 3;SIZE 1;
PTM:EQU @ $4CF41;
MOVE 0,PTM+2;              (écriture registre # 3)
MOVE $E3,PTM;              (horloge/8; monocoup + indicateur + sortie)
MOVE $xx,PTM+12;  }        valeurs à ajuster jusqu'à un
MOVE $yy,PTM+14;  }        déroulement complet
MOVE.W $2000,SR;           autorisation de toutes les interruptions
MOVE 1,PTM+2;              écriture registre 1
MOVE 0,PTM;                temporisateurs initialisés
séquence en test
MOVE.W $2700,SR;TRAP 14;
```

Contrôle de la vitesse d'un moteur

Le **programme n° 17** réalise le contrôle de la vitesse d'un moteur, tel qu'il est montré sur la **figure 9.4.** Ce moteur est commandé par un relai (ici transistor 2N3055) activé ou non par une tension de 5 V provenant du PIA. Si la tension reste à 5 V, le moteur atteint sa vitesse maximale rapidement. Si la tension passe à zéro, le moteur s'arrête. La cellule MCT8 (et son transistor d'amplification 2N5134) détecte les trous percés sur le disque solidaire de l'axe du moteur. La sortie OUT prend la valeur 0 V au passage d'un trou et reste à 5 V sinon. Cette sortie va être branchée sur la ligne CB2 du PIA, programmée en entrée d'interruption, transition descendante. Le fonctionnement du programme est le suivant : le PTM est programmé en mode monocoup ; à la fin du cycle, le nombre d'interruptions détectées comptabilisé dans le registre D6 est comparé avec la valeur voulue (qui est donc l'équivalent d'une vitesse) rentrée en D7 ; si ce nombre est plus petit (la vitesse est trop faible) les lignes de données du PIA sont mises à 5 V, activant le moteur et de plus un "1" est affiché sur l'écran ; dans le cas contraire, les lignes du PIA sont mises à zéro, et un "0" est affiché...

De très nombreuses améliorations sont possibles à partir de ces bases...

Fig. 9.4 - La vitesse d'un moteur peut être contrôlée par l'intermédiaire du temporisateur.



Remarque : les interruptions de notre PIA provoquent le déroulement de l'exception PARASITE (vecteur d'adresse $60). C'est une simple option de câblage des circuits, qu'il serait d'ailleur très facile de modifier.

```
                        ORG $2000;LIST 3;SIZE 1;

                        PTM:EQU @$4CF41;
                        PIA:EQU @$5CEF5;/* CONTROLE VITESSE MOTEUR */
                        VEC:EQU @$60;   /* NOMBRE DE TOURS PAR CYCLE EN D7 */
2000    4DFA00A4        LEA PRO,A6;
2004    21CE0060        MOVE.L A6,VEC;
2008    13FC00000005    MOVE 0,PIA+2;/* ACCES REGISTRE DDR */
        CEF7
2010    13FC00FF0005    MOVE 255,PIA;/* TOUTES LES LIGNES EN SORTIE */
        CEF5
2018    13FC000C0005    MOVE $C,PIA+2;/* ACCES DONNEES+INTERRUPTION SUR CB2 */
        CEF7
2020    13FC00000004    MOVE 0,PTM+2;
        CF43
2028    13FC00A30004    MOVE $A3,PTM;/* MODE MONOCOUP-DIVISION PAR 8 */
        CF41
2030    13FC00FF0004    MOVE $FF,PTM+12;
        CF4D
2038    13FC00FF0004    MOVE $FF,PTM+14; /* DUREE MAXIMALE */
        CF4F
2040    13FC00010004    MOVE 1,PTM+2;
        CF43
2048    13FC00000004    MOVE 0,PTM;
        CF41
2050    46FC2100        PA:MOVE.W $2100,SR;
2054    4206            CLR D6;
2056    10390004CF43    PI:MOVE PTM+2,D0;
205C    C03C0004        AND 4,D0;
2060    67F4            BEQ PI;/* ATTENTE FIN DE CYCLE */
2062    13FC00010004    MOVE 1,PTM;
        CF41
206A    13FC00000004    MOVE 0,PTM;
        CF41
2072    E206            ASR 1,D6;
2074    4EB900021C7C    JSR @$21C7C;
207A    BE06            CMP D6,D7;
207C    6B14            BMI ZER;
207E    13FC00FF0005    MOVE 255,PIA;/* ON ENVOIE LE COURANT */
        CEF5
2086    103C0031        MOVE '1',D0;
208A    4EB900021E24    JSR @$21E24;
2090    60BE            BRA PA;
2092    13FC00000005    ZER:MOVE 0,PIA;/* ON COUPE LE COURANT */
        CEF5
209A    103C0030        MOVE '0',D0;
209E    4EB900021E24    JSR @$21E24;
20A4    60AA            BRA PA;
20A6    46FC2700        PRO:MOVE.W $2700,SR;
20AA    12390005CEF5    MOVE PIA,D1; /* REMISE A ZERO BIT INTERRUPTION */
20B0    5206            ADDQ 1,D6;
20B2    4E73            RTE;;
```

suite page suivante

```
PA                              16              $2050  PC-R.
PI                              18              $2056  PC-R.
PIA                              3              @$5CEF5
PRO                             35              $20A6  PC-R.
PTM                              2              @$4CF41
VEC                              4              @$60
ZER                             31              $2092  PC-R.


   0 ERRORS
a?? G
FORCEMON 2.2 > .D? 19

FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000


11110001111000111000111100011000111110000
FORCEMON 2.2 >
```

Programme 17 - Contrôle de la vitesse d'un moteur.

CHAPITRE 10 | l'ACIA

L'ACIA (adaptateur pour communications asynchrones) 6850 réalise la mise en format des données et le contrôle de la transmission, entre un microprocesseur et un périphérique travaillant en mode série asynchrone. On va donc le trouver entre le processeur et la console ou entre le processeur et un autre ordinateur (appelé "hôte") qui considérera notre processeur comme un simple périphérique. Ce circuit, ou son équivalent, se trouve donc sur toutes les machines, ce qui n'est pas le cas du PTM ou du PIA.

DESCRIPTION

L'ACIA possède quatre registres accessibles par programme, en deux adresses seulement, situées sur notre système pour l'ACIA "Terminal" aux emplacements suivants :

- $050040 contrôle (en écriture) et état (en lecture) ;
- $050042 réception (en lecture) et émission (en écriture).

Le registre de contrôle permet de fixer le type de transmission désiré (bits 0 à 4) et la possibilité de travailler en mode interruption, soit en émission (bit 5 et 6) par exemple en cas de registre de transmission vide, soit en réception (bit 7) dans le cas d'un registre de réception plein. Les **figures 10.1** et **10.2** donnent les options possibles.

Le registre d'état, à lecture seule, permet de connaître l'avancement de la transmission. Le bit 0 passe à "1" lorsqu'une donnée est reçue (registre plein) et se remet automatiquement à zéro après une lecture du registre de réception, par le programme. Dans l'autre sens, le bit 1 passe à "1" lorsque le registre de transmission est libéré et qu'une autre donnée peut y être inscrite.

CR1	CR0	Fonction
0	0	÷ 1
0	1	÷ 16
1	0	÷ 64
1	1	Initialisation programmée

CR4	CR3	CR2	Fonction
0	0	0	7 bits+ parité paire+ 2 bits d'arrêt
0	0	1	7 bits+ parité impaire+ 2 bits d'arrêt
0	1	0	7 bits+ parité paire+ 1 bit d'arrêt
0	1	1	7 bits+ parité impaire+ 1 bit d'arrêt
1	0	0	8 bits+ 2 bits d'arrêt
1	0	1	8 bits+ 1 bit d'arrêt
1	1	0	8 bits+ parité paire+ 1 bit d'arrêt
1	1	1	8 bits+ parité impaire+ 1 bit d'arrêt

CR6	CR5	Fonction
0	0	$\overline{RTS}$ = bas, interruptions de transmission inhibées
0	1	$\overline{RTS}$ = bas, interruptions de transmission autorisées
1	0	$\overline{RTS}$ = haut, interruptions de transmission inhibées
1	1	$\overline{RTS}$ = bas, émission d'un "Break", sur la ligne de transmission interruptions de transmission inhibées

Fig. 10.1 - Le registre de contrôle de l'ACIA.

Une façon simple de travailler sera de tester ces bits pendant une transmission, et d'attendre éventuellement, qu'un caractère arrive ou qu'un autre soit bien parti. Exemple : programme permettant d'entrer un caractère depuis la console et de l'afficher.

```
ORG $800;LIST 3;SIZE 1;
ACIA:EQU @ $50040;      ACIA TERMINAL
MOVE 1,ACIA;            (division par 16, 7 bits, parité paire,
                        2 bits d'arrêt)
RECEPTION: MOVE ACIA+2,D3;
AND 1,D3;
```


Numéro du bit (sur le bus données)	Registre sélecté			
	RS • R/$\overline{W}$ Registre de transmission (Ecriture seule)	RS • R/$\overline{W}$ Registre de réception (Lecture seule)	RS • R/$\overline{W}$ Registre de contrôle (Ecriture seule)	RS • R/$\overline{W}$ Registre d'état (Lecture seule)
0	Bit donnée 0*	Bit donnée 0*	Sélection du rapport de division 1 (CR0)	Registre de réception plein (RDRF)
1	Bit donnée 1	Bit donnée 1	Sélection du rapport de division 2 (CR1)	Registre de transmission vide (TDRE)
2	Bit donnée 2	Bit donnée 2	Sélection du format des mots 1 (CR2)	Perte de la porteuse de données ($\overline{DCD}$)
3	Bit donnée 3	Bit donnée 3	Sélection du format des mots 2 (CR3)	Inhibition de l'émetteur ($\overline{CTS}$)
4	Bit donnée 4	Bit donnée 4	Sélection du format des mots 3 (CR4)	Erreur de format (FE)
5	Bit donnée 5	Bit donnée 5	Contrôle de transmission 1 (CR5)	Récepteur en surcharge (OVRN)
6	Bit donnée 6	Bit donnée 6	Contrôle de transmission 2 (CR6)	Erreur de parité (PE)
7	Bit donnée 7***	Bit donnée 7**	Autorisation des interruptions en réception (CR7)	Demande d'interruption (IRQ)

Fig. 10.2 - Contenu des registres de l'ACIA.

```
BEQ RECEPTION;          tant que le bit 0 n'est pas à 1 on
                        boucle (pooling)
MOVE ACIA,D4;           le caractère est arrivé, on le met dans
                        le registre D4
MOVE D4,ACIA;           D4 est expédié sur la console
BRA RECEPTION;          le cycle continue. Inutile de tester la
                        bonne transmission.
```

LE MODE INTERRUPTION

On peut aussi travailler en mode interruption. Celle-ci peut être provoquée lorsque le registre de réception est plein. Il est alors possible de faire exécuter un programme (au lieu de tester éternellement le passage d'un bit à la bonne valeur), quitte à l'interrompre dès qu'une valeur est frappée au clavier. Le **programme n° 18** illustre cette philosophie. Ce programme place des "A" à partir de l'adresse $1000 (et il est ralenti dans cette besogne par l'appel de la routine WAIT). Une simple frappe au clavier l'interrompt et le sous-programme d'interruption place le caractère frappé, aussi à partir de l'adresse $1000, écrasant les "A" qui s'y trouvent.

A partir de ce modèle, il est possible d'utiliser le mode interruption pour reprendre la main dans un programme et faire afficher des valeurs intermédiaires, en rentrer d'autres, etc.

```
                        ORG $2000;LIST 3;SIZE 1;

                        ACIA:EQU @$50040;
                        VECT:EQU @$74;
2000    43FA0030        LEA INTER,A1;
2004    21C90074        MOVE.L A1,VECT;
2008    13FC00910005    HAUT:MOVE $91,ACIA; /* INTERRUPTION SI     */
        0040
2010    46FC2100        MOVE.W $2100,SR;    /* BUFFER ENTREE PLEIN */
2014    247C00003000    MOVEA.L $3000,A2;
201A    264A            MOVEA.L A2,A3;
201C    4EBA0008        RETOUR:JSR WAIT; /* BOUCLE D'ATTENTE */
2020    14FC0041        MOVE 65,(A2)+;
2024    60F6            BRA RETOUR;
2026    283C00004E20    WAIT: MOVE.L  20000,D4;
202C    5384            SOUS:SUBQ.L 1,D4;
202E    66FC            BNE SOUS;
2030    4E75            RTS;
2032    46FC2700        INTER:MOVE.W $2700,SR;
2036    103900050042    MOVE ACIA+2,D0;/* ECRITURE DU CARACTERE VIA D0 */
203C    16C0            MOVE D0,(A3)+;
203E    4EB900021C7C    JSR @$21C7C;
2044    4EB900021E24    JSR @$21E24;
204A    46FC2100        MOVE.W $2100,SR;
204E    4E73            RTE;;

ACIA                    2               @$50040
HAUT                    6               $2008  PC-R.
INTER                   17              $2032  PC-R.
RETOUR                  10              $201C  PC-R.
SOUS                    14              $202C  PC-R.
VECT                    3               @$74
WAIT                    13              $2026  PC-R.


   0 ERRORS
a?? G
FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000
CECI VA SE PLACER A PARTIR DE L ADRESSE 3000 ECRASANT LES  A  !
FORCEMON 2.2 > MD 3000
003000    43 45 43 49 20 56 41 20  53 45 20 50 4C 41 43 45  CECI VA SE PLACE

FORCEMON 2.2 >
003010    52 20 41 20 50 41 52 54  49 52 20 44 45 20 4C 27  R A PARTIR DE L
003020    41 44 52 45 53 53 45 20  33 30 30 30 20 45 43 52  ADRESSE 3000 ECR
003030    41 53 41 4E 54 20 4C 45  53 20 27 41 27 20 21 41  ASANT LES  A  !A
```

Programme 18 – Interruptions provoquées sur le terminal, par registre d'entrée plein.

Ce type de traitement est particulièrement indiqué en mode frontal face à un ordinateur hôte. Une façon agréable de travailler est la suivante :

- se mettre en mode interruption sur l'ACIA hôte, sur registre de réception plein ;
- saisir un caractère à la console et l'expédier directement sur l'ACIA hôte ;
- traiter l'interruption qui surviendra lorsque le caractère envoyé reviendra sur l'ACIA, expédié par l'ordinateur hôte (c'est le mode classique Full-Duplex). Ce traitement pourra consister en une simple visualisation ;
- puis on recommence, saisir un caractère, etc.

L'utilisateur est à même de juger si les transferts se font correctement et si le caractère revient lentement (ou pas du tout) ; de plus, tous les messages de service envoyés par l'ordinateur lui parviendront, en déclenchant la séquence d'interruption.

Il serait beaucoup plus complexe d'essayer de décoder les fins de messages. Le **programme n° 19** travaille sur ce principe. On remarque qu'il boucle sur l'entrée clavier, le reste du programme étant la gestion des interruptions.

Bien entendu, cette connexion va permettre des échanges entre l'ordinateur hôte et le microprocesseur, et certains problèmes risquent de se poser dans l'interprétation qu'il faut leur donner. Si les données sont des caractères ASCII, la compatibilité est totale, de même, pour d'autres types de caractères, une table de décodification peut suffire. Le mode de transmission le plus courant se faisant sur 7 bits utiles (et un bit de contrôle), il n'est pas possible de vider ou de charger de cette façon une partie de la mémoire (représentant, par exemple, des données numériques ou un programme objet). La méthode la plus générale va consister à transmettre à chaque fois deux caractères ASCII. Un codage permettant la visualisation, donc une forme de contrôle, peut être le suivant : l'octet abcdefgh à transmettre étant sous forme binaire, on l'envoie en 0100abcd et 0100efgh après quelques manipulations binaires ; ce qui, on le constatera, représente une partie du code ASCII formé de majuscules.

Transmission et réception de données non ASCII

Transmettre et recevoir des nombres est plus complexe. Ceux que l'on peut traiter dans tous les cas sont ici codés sur 16 bits (opérandes de MULU, DIVU, etc.). Cela peut correspondre directement à des valeurs FORTRAN (INTEGER*2), mais le plus

```
1  ORG $2000;
2  LIST 3;
3  SIZE 1;
4  AC:EQU @$050041;
5  VECT:EQU @$78;
6  MULTICS:NOP;
7  BSR ININTER;
8  ENCORE:CLR.L D3;
9  JSR @$21C7C;
10 JSR @$21FB0;
11 CMP $1B,D0;
12 BNE ENVOI;
13 JSR @$20232;
14 ENVOI:MOVE D0,AC+2;
15 BRA ENCORE;
16 ININTER:LEA INTER,A6;
17 MOVE.L A6,VECT;
18 MOVE $91,AC;
19 MOVE.W $2100,SR;
20 RTS;
21 INTER:MOVE AC+2,D1;
22 MOVE D1,D0;
23 CLR.L D3;
24 JSR @$21C7C;
25 JSR @$21E24;
26 RTE;
```

Programme 19 - Frontalisation et dialogue avec un ordinateur hôte, en mode interruption.

souvent, les entiers sont codés sur 32 ou 36 bits. Pour les récupérer, il faut auparavant les transformer en caractères ASCII, avant de les transmettre. Cela peut être fait par le programme FORTRAN, qui, dans notre exemple, a produit ces valeurs. Soit I le résultat (compris entre -32768 et +32767, nombre entier Fortran), après division par 256, on peut le représenter sur deux octets. On est donc ramené au problème précédent, à savoir transformer un octet en deux codes ASCII : le plus simple en Fortran, est d'utiliser une table de consultation de type CHARACTER (un modèle est donné en annexe).

Cette partie est plus délicate que la reconstruction qui devra avoir lieu ensuite, où quelques opérations binaires du 68000 (LSL, AND, OR...) suffiront.

Remarque : on peut avoir intérêt, en Fortran, à ne traiter que des nombres positifs (on ajoute 32768 à I avant les divisions

par 256, et le nombre tient sur 16 bits) et du résultat mis en forme, on soustrait cette valeur.

Une fois de plus, on ne peut que regretter la multiplicité des méthodes de codage et de transmission qui compliquent bien inutilement le travail des programmeurs.

Un autre problème, purement technique, sera aussi à résoudre, au niveau des connexions électriques sur la prise standard V24, car tous les branchements ne sont pas forcément identiques. L'utilisation du mode transparent, avant toute programmation, semble indispensable. Ce dernier (voir chapitre 2) permet un passage direct de l'information entre la console et le processeur ou entre le processeur et l'hôte.

LE MODE DE TRANSMISSION "S"

Pour bénéficier des facilités offertes par le moniteur pour le chargement et le déchargement des programmes, il faut quitter le mode transparent qui ne doit servir qu'à établir un début de dialogue :

TM / [R]

demande à l'éditeur de l'ordinateur hôte d'accéder au fichier en lecture

/ (pas de RETURN, retour au moniteur)

L02;X;=message-destiné-à-l'éditeur-hôte [R]

(commande de déchargement du fichier)

Cette commande LOAD permet le chargement du programme précédemment stocké par une commande DU (DUMP) ou S (SAVE) qui toutes deux travaillent en mode "S" (Stream). Ce mode de transmission est plus complet que celui que nous avons proposé pour transmettre des octets à 8 bits utiles. Outre que chaque octet est éclaté en deux parties de quatre bits, chacune étant représentée par un chiffre hexadécimal (par exemple l'octet 10001111 sera transmis par 8F, donc par les codes ASCII 38 et 46), ce codage donne en plus l'adresse d'implantation de ces données, immédiatement après l'en-tête S113. On y trouve également après seize caractères réels, un "caractère" de contrôle et des caractères de mise en forme pour édition à la console. Enfin d'autres lignes particulières (début, fin, ligne écourtée) complètent cette présentation complexe. Le **programme n° 20** est la traduction en mode "S" du programme n° 1 (routines); à la suite

```
 1,$P
1   PHYSICAL ADDRESS=0000A100 0000A1D4
2
3   S0030000FC
4
5   S113A100123456784544495457 4F524B202020204E
6
7   S113A11020202020202020200000000010000000139
8
9   S113A1200000A1D40000A13800000001000000000DC
10
11  S113A13000000000C0000A1000A4F52472024323 8CE
12
13  S113A14030303B0A074C49535420333B07075349EB
14
15  S113A1505A4520313B07104F55543A45515520403C
16
17  S113A160243231453234 3B100F494E3A4551552083
18
19  S113A17040243231464230 3B0F104144523A45515B
20
21  S113A18055204024323143374 33B10084A53522070
22
23  S113A1904144523B080E5245544F55523A4A535289
24
25  S113A1A020494E3B0E084A5352204F55543B080B4E
26
27  S113A1B0434D502024324620 44303B0B0B424E4539
28
29  S113A1C0205245544F55523B0B095452415020 31B3
30
31  S108A1D0343B3B095380
32
33  S9030000FC
34
35
```

Programme 20 - Ecriture en mode "S" pour transmission RS232.

de ce programme on trouvera un dialogue avec un ordinateur hôte (Multics) (**programme 20 bis**). Cette option offre de grands avantages : on stocke sous une forme identique le source assembleur (donc la version pouvant être modifiée normalement par l'éditeur après chargement en mémoire) et, de façon indirecte, le source "listing", comme on le ferait d'un langage évolué. Travaillant sur un ordinateur hôte, on peut en utiliser toutes les ressources, et se passer d'imprimante, par exemple, sur le micro-ordinateur, en


```
FORCEMON 2.2 > IN /

*TRANSPARENT* EXIT=$2F = /

*
INC

OK!

CONFIGURATION REDUITE POUR TEST CPU.
Multics MR10.2.0: INRIA, Rocquencourt, France (Channel b.h201)
Load = 40.0 out of 75.0 units: users = 40, 09/20/84  1212.7 hfe Thu
I Brown
Password:

You are protected from preemption until 1313.
Brown          logged in 09/20/84  1213.0 hfe Thu from ASCII_OFLOW terminal "none
\c".
Last login 09/20/84  1208.4 hfe Thu from ASCII_OFLOW terminal "none".


r 12:13 1.925 67

ted
r routines

FORCEMON 2.2 > LO2;X;=1,$p
1,$p
No 2nd delimiter.

TIMEOUT
FORCEMON 2.2 > LO2;X;=1,$p
1,$p
PHYSICAL ADDRESS=0000A100 0000A1D4

S0030000FC

S113A100123456784544495457 4F524B202020204E

S113A11020202020202020200000000010000000139

S113A1200000A1D40000A13800000001000000000DC

S113A13000000000C0000A1000A4F52472024323 8CE

S113A14030303B0A074C49535420333B07075349EB

S113A1505A4520313B07104F55543A45515520403C

S113A160243231453234 3B100F494E3A4551552083

S113A17040243231464230 3B0F104144523A45515B

S113A18055204024323143374 33B10084A53522070

S113A1904144523B080E5245544F55523A4A535289
```

suite page suivante

```
S113A1A020494E3B0E084A5352204F55543B080B4E

S113A1B0434D50202432462C44303B0B0B424E4539

S113A1C0205245544F55523B0B095452415020313B3

S108A1D0343B3B095380

S9030000FC
FORCEMON 2.2 >
```

Programme 20 bis - Dialogue engagé en mode transparent avec l'ordinateur hôte MULTICS.

```
FORCEMON 2.2 > MD A100
00A100    12 34 56 78 45 44 49 54  57 4F 52 4B 20 20 20 20  .4VxEDITWORK

FORCEMON 2.2 >
00A110    20 20 20 20 20 20 20 20  00 00 00 01 00 00 00 01          ........
00A120    00 00 A1 D4 00 00 A1 3B  00 00 00 01 00 00 00 00  ..¦T..¦8........
00A130    00 00 00 0C 00 00 A1 00  0A 4F 52 47 20 24 32 38  ......¦..ORG $28
00A140    30 30 3B 0A 07 4C 49 53  54 20 33 3B 07 07 53 49  00;..LIST 3;..SI
00A150    5A 45 20 31 3B 07 10 4F  55 54 3A 45 51 55 20 40  ZE 1;..OUT:EQU @
00A160    24 32 31 45 32 34 3B 10  0F 49 4E 3A 45 51 55 20  $21E24;..IN:EQU
00A170    40 24 32 31 46 42 30 3B  0F 10 41 44 52 3A 45 51  @$21FB0;..ADR:EQ
00A180    55 20 40 24 32 31 43 37  43 3B 10 08 4A 53 52 20  U @$21C7C;..JSR
00A190    41 44 52 3B 08 0E 52 45  54 4F 55 52 3A 4A 53 52  ADR;..RETOUR:JSR
00A1A0    20 49 4E 3B 0E 08 4A 53  52 20 4F 55 54 3B 08 0B   IN;..JSR OUT;..
00A1B0    43 4D 50 20 24 32 46 2C  44 30 3B 0B 0B 42 4E 45  CMP $2F,D0;..BNE
00A1C0    20 52 45 54 4F 55 52 3B  0B 09 54 52 41 50 20 31   RETOUR;..TRAP 1
00A1D0    34 3B 3B 09 53 20 00 00  00 00 00 00 00 00 00 00  4;;.S ..........
00A1E0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00A1F0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00A200    FF FF FF FF FF FF 5E FF  FF FF FF FF FF FF FF FF  ...... ........

FORCEMON 2.2 >
```

Programme 20 bis (suite) - Vérification de la correcte introduction du programme en mémoire avant assemblage éventuel.

décodant directement le mode "S" stocké. Un programme écrit en Fortran standard réalise facilement cette transcription après suppression des lignes d'en-tête. Le programme se contente de décoder ligne après ligne les caractères des rangs 9 à 40 par une table de consultation. Une partie du programme de décodage est donnée à titre documentaire ainsi que le source (de "routines") qu'il produit depuis le mode "S" (**programmes n° 21 et 21 bis**).

On libère par cette connexion le PIA, qui en général supporte une imprimante, pour d'autres tâches. Il est également possible d'obtenir des copies d'écran directement, grâce à l'entrée auxiliaire de la console à laquelle on peut connecter une imprimante série. Petit inconvénient, la console doit alors être programmée à la même vitesse que l'imprimante série, lente en général.


```
      CHARACTER*2 BUF(21),C2
       CHARACTER*1 LIGNE(42),Z1
      COMMON LIGNE Z1 BUF C2
       L=0
      I=0
      L5=17
      READ(10,800) LIGNE
      WRITE(6,801) LIGNE
      DO 34 LI=1,8
34    READ(10,800) LIGNE
801   FORMAT(1H ,42A1)
 10    READ(10,900,END=1000) BUF
 900   FORMAT (21A2)
      DO 100 K=L5,20
      C2=BUF(K)
      Z1='w'
      GO TO 555
556   CONTINUE
      IF(Z1.EQ.'w') GO TO 100
      IF((Z1.EQ.' ').AND.(BUF(K-1).EQ.'20')) GO TO 100
      L=L+1
      LIGNE(L)=Z1
      IF((L.EQ.1).AND.(Z1.EQ.' ')) L=0
      IF(L.EQ.42) GO TO 666
      IF(Z1.EQ.';') GO TO 666
667    CONTINUE
100   CONTINUE
      L5=5
      GO TO 10
1000   STOP
666   DO 4 J=L+1,42
      LIGNE(J)=
4     CONTINUE
      I=I+1
      WRITE(6,805) I,LIGNE
 805  FORMAT(1H ,I5,'  ',42A1)
800   FORMAT(1H ,42A1)
      L=0
      L5=5
      GO TO 667
 555   CONTINUE
      IF(C2.EQ.'20') Z1=' '
      IF(C2.EQ.'21') Z1='!'
      IF(C2.EQ.'22') Z1='"'
      IF(C2.EQ.'23') Z1='/'
```

suite page suivante

```
99,$P
   99          IF(C2.EQ.'59') Z1='Y'
  100          IF(C2.EQ.'5A') Z1='Z'
  101         GO TO 556
  102          END
```

Programme 21 - Extraction d'un listing à partir du mode "S".

```
ec trada routines
io attach file10 vfile_ routines
trad
PHYSICAL ADDRESS=0000A100 0000A1D4
     1  ORG $2800;
     2  LIST 3;
     3  SIZE 1;
     4  OUT:EQU @$21E24;
     5  IN:EQU @$21FB0;
     6  ADR:EQU @$21C7C;
     7  JSR ADR;
     8  RETOUR:JSR IN;
     9  JSR OUT;
    10  CMP $2F,D0;
    11  BNE RETOUR;
    12  TRAP 14;
    13  ;

STOP
io detach file10
r 12:18 1.320 97
```

Programme 21 bis - Listing obtenu par décodification du mode "S" par le programme 21.

CHAPITRE 11 les piles

LA PILE SYSTEME

La pile système est pointée par le registre A7. Elle se charge par adresses (paires) décroissantes, et en mode prédécrémenté. Le rôle principal de la pile est de conserver les adresses de retour des sous-programmes successifs appelés. La dernière adresse stockée dans la pile sera la première récupérée : pile LIFO (Last In, First Out). Toute cette gestion est faite automatiquement à la suite des instructions JSR, BSR... RTS, RTE...

L'adresse du haut de la pile est, dans notre système 77E. En cas de besoin (nombreux sous-programmes emboîtés) cette pile peut être déplacée et ajustée par programme :

```
LEA PILE A7;

.....

DS.L 100;       réservation de 100 mots-longs

PILE:EQU *;     adresse du sommet de la pile
```

PASSAGE DE PARAMETRES

On peut utiliser la pile pour transmettre des paramètres de longueur variable à un sous-programme (cette dimension pouvant être fournie comme première valeur, ou dans un registre), toutefois, ce sous-programme ne devra pas être interrompu de façon aléatoire, ce qui modifierait le registre A7.

L'instruction LINK permet de passer des paramètres, en préservant un bloc de la pile, ainsi que l'adresse de ce bloc :

```
LINK A6,-4;
```

cette instruction a diverses fonctionnalités : la valeur de A6 est conservée dans la pile ; quatre octets sont réservés dans la pile ; A6 pointe sur le début de cette zone réservée. Après le traitement désiré on revient à l'état initial par :

UNLK A6;

Dans notre optique (mono-utilisateur) le recours aux piles ne semble guère indispensable et l'avantage procuré dans le transfert des données est contrebalancé par l'extrême minutie dont doit faire preuve le programmeur, dans la gestion des pointeurs.

Exemple : soit les instructions suivantes :

MOVEA.L $2008,A7;

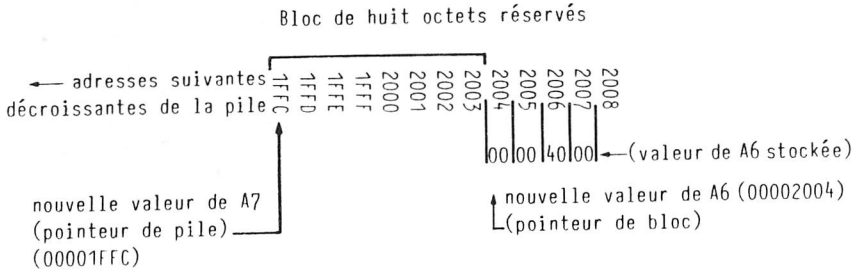
MOVEA.L $4000,A6;

LINK A6,-8;

Avant cette dernière instruction, les registres A6 et A7 contiennent

A6 [00004000] [00002008] A7

après LINK A6,-8, la gestion de la pile a été changée :



On constate que la valeur initiale de A6 ne joue aucun rôle à ce niveau, et doit être considérée comme un paramètre éventuellement utilisable, puisque transmissible. Tout revient à l'état initial après l'instruction :

UNLK A6;

déconnexion de la pile (A6 reprend la valeur 4000 et A7 pointe de nouveau sur 2008).

Le **programme n° 22** illustre la manière dont on peut utiliser une zone mémoire de la pile pour transmettre des données à un sous-programme, indépendamment des déplacements automatiques du pointeur de pile A7 (selon les appels, les interruptions, etc.).


```
                        ORG $1000;
                         LIST 3;
                         SIZE 4;
1000    4E51FFFC         LINK A1,-4;
1004    2341FFFC         MOVE.L D1,-4(A1);
1008    4EBA0006         JSR TIP;
100C    4E59             UNLK A1;
100E    4E4E             TRAP 14;
1010    4EB900021C7C     TIP:JSR @$21C7C;
1016    1029FFFE         MOVE.B -2(A1),D0;
101A    4EB900021E24     JSR @$21E24;
1020    1029FFFF         MOVE.B -1(A1),D0;
1024    4EB900021E24     JSR @$21E24;
102A    4E75             RTS;;

TIP                      9              $1010  PC-R.

   0 ERRORS

FORCEMON 2.2 > .01 4142

FORCEMON 2.2 > G 1000
PHYSICAL ADDRESS=00001000
AB
FORCEMON 2.2 >
```

Programme 22 – Utilisation de la pile à des fins de stockage et de transmission temporaires.

Dans les quatre octets réservés par l'instruction LINK, on place le contenu de D1 (préalablement chargé ici par une commande moniteur), puis on appelle le sous-programme TIP dont le rôle est d'afficher en ASCII, les octets de poids faibles, qui ont été rangés dans la pile et dont le pointeur de bloc A1 donne l'emplacement. D1 est écrit en -4(A1), -3(A1), -2(A1), -1(A1). La valeur initiale - et finale - de A1 ne joue aucun rôle.

STOCKAGE DES REGISTRES DANS LA PILE

L'instruction MOVEM (transferts multiples) qui fait appel en général à la pile est très utile. Elle permet la sauvegarde de tous les registres et leur restitution.

Son format est le suivant :

MOVEM.L liste-de-registres, -(A7) ; pour le stockage, et

MOVEM.L (A7)+, liste-de-registres ; (il s'agit de la même liste) pour la restitution.

La liste de registres doit être décrite dans un ordre imposé :

- les registres sont nommés en ordre croissant, les registres de données sont les premiers cités ;
- les séparateurs sont le tiret "-" et "/" dont le rôle est expliqué par l'exemple suivant

```
MOVEM.L D0-D3/D5/A0-A4/A6;
```

la liste des registres à sauvegarder comprend D0, D1, D2, D3, D5, A0, A1, A2, A3, A4, A6.

On a tout intérêt à mettre en équivalence une liste, même utilisée en un seul endroit :

```
SIZE 4;
LISTE:EQU D0/D2-D5/D7/A0-A6;
.....
MOVEM LISTE,-(A7);
.....
MOVEM (A7)+,LISTE;
```

Il est possible de transférer les 16 bits de poids faible de tous les registres concernés, ou les 32 bits (MOVEM.W ou MOVEM.L).

Le stockage des registres en mémoire est moins courant :

```
MOVEM.L LISTE,@$1000;  l'adresse $1000 sera automatiquement
                       incrémentée (ici de 4).
```


CHAPITRE 12 autres fonctionnalités

GENERALITES SUR LES SYSTEMES MULTI-UTILISATEURS

Notre carte de travail n'est pas configurée pour de tels développements, qui nécessitent l'adjonction d'un circuit périphérique (68451) destiné à éviter les empiètements des divers utilisateurs, les uns sur les autres. Sinon, les instructions du 68000 permettent de créer deux classes de programmes :

- en mode superviseur ;
- en mode utilisateur.

Le programme écrit en mode superviseur a accès à toutes les instructions, et son rôle va être de gérer les programmes utilisateurs, qui eux ont à leur disposition un jeu d'instructions plus réduit (les instructions privilégiées sont alors interdites).

Les instructions privilégiées (voir figure 5.10) sont susceptibles de modifier l'organisation elle-même du programme superviseur. Les employer déclenchera une exception de vecteur 8. Les piles sont maintenant séparées ; chaque utilisateur a la sienne propre et le programme superviseur, lui, travaille sur le pointeur A7.

On peut alors imaginer une structure fonctionnant sur de telles bases (**figure 12.1**).

Le programme superviseur reçoit une interruption du PTM. Il stocke alors les registres du programme utilisateur en cours et le compteur-programme. Il restaure ces registres pour le programme à qui il va donner la main, jusqu'à la prochaine interruption. Des priorités de temps peuvent être décidées par la simple modification des durées entre les interruptions. On conçoit donc que les instructions privilégiées pourraient, sans précaution, modifier ce déroulement. Il resterait également à protéger chaque utilisateur

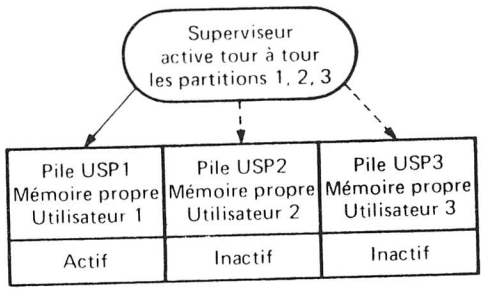


Fig. 12.1 - Structure multi-utilisateur.

contre les tentatives d'autrui d'accéder à sa mémoire propre. Dans notre cas, utilisateur unique, le mode superviseur est le mode normal.

Mais on peut se demander quel serait l'intérêt d'une telle organisation pour l'utilisateur final obligé de partager ses ressources avec autrui et de se contenter d'un jeu d'instructions plus réduit, omettant, en particulier, le traitement des interruptions tellement commode. Le faible coût des matériels ne justifie pas que d'énormes efforts soient consacrés au développement de logiciels multitâches très complexes.

La solution est au contraire l'indépendance vis à vis d'un système centralisateur de ce type gérant à grands frais une certaine pénurie. La connexion sur un réseau est autrement efficace : d'une part l'utilisateur a la pleine puissance de son propre matériel et il peut disposer en plus de toutes les ressources d'un gros système, parmi lesquelles : l'impression rapide, le stockage sur disque en ligne, l'accès à divers serveurs et l'assistance de haut niveau. Certains systèmes en projet, prévoient dès 1986 aux USA l'installation de réseaux de 8000 terminaux (Université Carnegie Mellon) ! Autant cette informatique distribuée semble cohérente et efficace, autant paraît périlleux le développement autour d'un seul micro-ordinateur d'un système multitâche.

AMELIORATION DES PERFORMANCES

Les vitesses de traitement des instructions peuvent influer sur les performances globales d'un programme, aussi il peut être intéressant de connaître les temps mis pour les instructions de base. Tous les temps donnés correspondent en fait à deux cycles

d'horloge d'entrée du 68000. Notre système partant d'une horloge de 8 MHz effectue donc 4 000 000 de cycles doubles par seconde. La **figure 12.2** indique l'influence du mode d'adressage choisi dans le calcul de l'adresse effective. On voit la supériorité indiscutable de l'adressage registre-direct, les bonnes performances de l'adressage indirect et les moins bonnes de l'adressage absolu long. Ces temps seront en général à rajouter au temps de l'instruction elle-même.

Les **figures 12.3** et **12.4** donnent tous les temps pour l'instruction MOVE. On sait que cette instruction représente, dans presque

Fig. 12.2 - Temps de calcul de l'adresse effective d'un opérande selon les modes d'adressage.

Mode d'adressage		TEMPS	
		octet, mot	mot long.
	Registre		
Dn	Registre données direct	0	0
An	Registre adresse direct	0	0
	Mémoire		
An@	Registre adresse indirect	2	4
An@+	Registre adresse indirect avec postincrément	2	4
An@-	Registre adresse indirect avec prédécrément	3	5
An@(d)	Registre adresse indirect avec déplacement	4	6
An@(d, ix)	Registre adresse indirect avec index	5	7
xxx W	Absolu court	4	6
xxx L	Absolu long	6	8
PC@(d)	Compteur programme avec déplacement	4	6
PC@(d, ix)	Programme compteur avec index	5	7
#xxx	Immédiat	2	4

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx W	xxx L
Dn	2	2	4	4	4	6	7	6	8
An	2	2	4	4	4	6	7	6	8
An@	4	4	6	6	6	8	9	8	10
An@+	4	4	6	6	6	8	9	8	10
An@-	5	5	7	7	7	9	10	9	11
An@(d)	6	6	8	8	8	10	11	10	12
An@(d, ix)	7	7	9	9	9	11	12	11	13
xxx W	6	6	8	8	8	10	11	10	12
xxx L	8	8	10	10	10	12	13	12	14
PC@(d)	6	6	8	8	8	10	11	10	12
PC@(d, ix)	7	7	9	9	9	11	12	11	13
#xxx	4	4	6	6	6	8	9	8	10

Les instructions de transfert (MOVE) sur octet et sur mot nécessitent chacune un accès mémoire pour la lecture et pour l'écriture de l'opérande.

Fig. 12.3 - Temps d'exécution d'une instruction MOVE portant sur un octet ou un mot selon l'adressage.

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx w	xxx L
Dn	2	2	6	6	6	8	9	8	10
An	2	2	6	6	6	8	9	8	10
An@	6	6	10	10	10	12	13	12	14
An@+	6	6	10	10	10	12	13	12	14
An@-	7	7	11	11	11	13	14	13	15
An@(d)	8	8	12	12	12	14	15	14	16
An@(d, ix)	9	9	13	13	13	15	16	15	17
xxx W	8	8	12	12	12	14	15	14	16
xxx L	10	10	14	14	14	16	17	16	18
PC@(d)	8	8	12	12	12	14	15	14	16
PC@(d, ix)	9	9	13	13	13	15	16	15	17
#xxx	6	6	10	10	10	12	13	12	14

Note : L'instruction de transfert (MOVE) sur mot long nécessite 2 accès mémoire pour la lecture de l'opérande et aussi pour l'écriture de l'opérande.

Fig. 12.4 - Temps d'exécution d'une instruction MOVE portant sur un mot long selon l'adressage.

tous les langages de programmation plus de 80 % des traitements. On constatera qu'il est plus économique de transférer un mot-long que deux mots.

Les **figures 12.5** à **12.14** complètent ce panorama.

Le **programme n° 23** mettant en oeuvre le PTM permet de connaître les vitesses réelles de traitement. Il s'agit d'effectuer douze instructions MOVE.L D1, D2 avant que le PTM ne déclenche une interruption en fin de cycle. Lorsque le registre PTM + 14 est chargé par la valeur 04, le programme n'a pas le temps de

Instruction	Taille	Registre *	Mémoire
ASR, ASL	octet, mot	3 + n	4+
	mot long	4 + n	—
LSR, LSL	octet, mot	3 + n	4+
	mot long	4 + n	—
ROR, ROL	octet, mot	3 + n	4+
	mot long	4 + n	—
ROXR, ROXL	octet, mot	3 + n	4+
	mot long	4 + n	—

\+ ajouter le temps de calcul de l'adresse effective

n nombre de décalages

Fig. 12.5 - Temps d'exécution des instructions de décalage et de rotation.

Instruction	Taille	Dynamique		Statique	
		registre	mémoire	registre	mémoire
BCHG	mot	—	4+	—	6+
	mot long	4*	—	6*	—
BCLR	mot	—	4+	—	6+
	mot long	5*	—	7*	—
BSET	mot	—	4+	—	6+
	mot long	4*	—	6*	—
BTST	mot	—	2+	—	4+
	mot long	3	—	5	—

\+ ajouter le temps de calcul de l'adresse effective

* indique la valeur maximale

Fig. 12.6 - Temps d'exécution des instructions de manipulation sur les bits.

s'achever correctement (interruption de niveau 4). Si on change (par la commande MM) la valeur 04 (qui est assemblée à l'adresse $201B) en 05, le programme se termine normalement. D'après nos tableaux, il faut pour chaque MOVE deux cycles doubles soit vingt-quatre pour les douze instructions, ou encore 6 μs. Or en quatre tops du PTM (1,25 x 4 = 5 μs), on ne peut exécuter que vingt cycles doubles. On ne peut donc aller jusqu'au bout du programme et l'instruction de masquage n'est pas prise en compte. En cinq tops (soit 6,25 μs) le programme a le temps de s'achever,

Instruction	Déplacement	(TRAP)	(No TRAP)
BCC	octet mot	5 5	4 6
BRA	octet mot	5 5	— —
BSR	octet mot	9 9	— —
DBCC	—	5	7
CHK	—	19*+	4+
TRAP	—	16	—
TRAPV	—	16	2

\+ ajouter le temps de calcul de l'adresse effective

* indique la valeur maximale

Fig. 12.7 - Temps d'exécution des instructions conditionnelles.

Instruction	Taille	Registre	Mémoire
CLR	octet, mot mot long	2 3	4+ 6+
NBCD	octet	3	4+
NEG	octet, mot mot long	2 3	4+ 6+
NEGX	octet, mot mot long	2 3	4+ 6+
NOT	octet, mot mot long	2 3	4+ 6+
SCC	octet, faux octet, vrai	2 3	4+ 4+
TAS	octet	2	5+
TST	octet, mot mot long	2 2	2+ 2+

\+ ajouter le temps de calcul de l'adresse effective.

Fig. 12.8 Temps d'exécution des instructions simple opérande.

Instruction	Taille	A op E →A	D op E →D	M op D →M
ADD	octet, mot mot long	4+ 3#+	2+ 3#+	4+ 6+
AND	octet, mot mot long	— —	2+ 3#+	4+ 6+
CMP	octet, mot mot long	3+ 3+	2+ 3+	— —
DIVS	—	—	79+*	—
DIVU	—	—	70+*	—
EOR	octet, mot mot long	— —	2 3#	4+ 5+
MULS	—	—	35+*	—
MULU	—	—	35+*	—
OR	octet, mot mot long	— —	2+ 3#+	4+ 6+
SUB	octet, mot mot long	4+ 3#+	2+ 3#+	4+ 6+

\+ ajouter le temps de calcul de l'adresse effective

\# 4 si l'adresse effective est registre direct

* indique la valeur maximale.

Fig. 12.9 - Temps d'exécution des instructions standard.

l'interruption du PTM est masquée et tout se termine normalement. On voit donc la parfaite concordance entre les vitesses de déroulement des instructions données par le constructeur et l'évaluation faite grâce au PTM. En général la complexité d'une vraie séquence ne permettra pas le calcul du temps qu'elle prend, instruction par instruction. Le PTM l'indiquera facilement.

LES COMMUNICATIONS ASSEMBLEUR-BASIC

Notre système de développement comporte, on l'a vu, un Basic sur EPROM. Il est donc possible de gérer au mieux des tâches très diverses, pouvant nécessiter des calculs complexes, pour

Instruction	Taille	D op I →D	M op I →M	SR op I →SR
ADDI	octet, mot	4	6+	—
	mot long	8	10+	—
ADDQ	octet, mot	2	4+	—
	mot long	4	6+	—
ANDI	octet, mot	4	6+	10
	mot long	8	10+	—
CMPI	octet, mot	4	4+	—
	mot long	7	6+	—
EORI	octet, mot	4	6+	10
	mot long	8	10+	—
MOVEQ	mot long	2	—	—
ORI	octet, mot	4	6+	10
	mot long	8	10+	—
SUBI	octet, mot	4	6+	—
	mot long	8	10+	—
SUBQ	octet, mot	2	4+	—
	mot long	4	6+	—

+ ajouter le temps de calcul de l'adresse effective.

Fig. 12.10 - Temps d'exécution des instructions immédiates.

Instruction	Taille	An@	An@+	An@-	An@(d)	An@(d, ix)	xxx W	xxx L	PC@(d)	PC@(d, ix)
JMP	—	4	—	—	5	7	5	6	5	7
JSR	—	8	—	—	9	11	9	10	9	11
LEA	—	2	—	—	4	6	4	6	4	6
MOVEM	mot	6 + 2n	6 + 2n	—	8 + 2n	9 + 2n	8 + 2n	10 + 2n	8 + 2n	9 + 2n
M →R	long	6 + 4n	6 + 4n	—	8 + 4n	9 + 4n	8 + 4n	10 + 4n	8 + 4n	9 + 4n
MOVEM	mot	4 + 2n	—	4 + 2n	6 + 2n	7 + 2n	6 + 2n	8 + 2n	—	—
R →M	long	4 + 4n	—	4 + 4n	6 + 4n	7 + 4n	6 + 4n	8 + 4n	—	—
PEA	—	6	—	—	8	9	8	10	8	9

n, nombre de registres à transférer

Fig. 12.11 - Temps d'exécution des instructions JMP, JSR, LEA, PEA, MOVEM.

Exception	Tps
Erreur adresse	24
Erreur bus	24
Interruption	21*
Instruction illégale	16
Instruction privilégiée	16
Trace	16

* on suppose que la reconnaissance d'interruption prend 2 cycles

Fig. 12.12 - Temps d'exécution des exceptions.

Instruction	Taille	D op D →D	M op M →M
ADDX	octet, mot	2	9
	mot long	4	15
CMPM	octet, mot	—	6
	mot long	—	10
SUBX	octet, mot	2	9
	mot long	4	15
ABCD	octet	3	9
SBCD	octet	3	9

Fig. 12.13 - Temps d'exécution d'instructions multi-précision.

lesquels l'assembleur est d'un emploi malaisé. Deux instructions Basic permettent une communication avec les programmes en assembleur, au niveau des données à transférer, PEEK et POKE. Leur fonctionnement est le suivant :

PEEK (K) fournit le contenu de la mémoire d'adresse K, sous forme d'un octet, K peut d'ailleurs être écrit en base 16 (comme toutes les constantes dans ce Basic)

```
J=PEEK(16#2000)    ; le contenu de la mémoire d'adresse $2000
                     va en J
```

Dans l'autre sens, l'instruction POKE (A,N) place N à l'adresse A. Un seul octet est transféré, celui de poids faible lorsque N est un nombre entier.

Instruction	Taille	Reg.	Mem.	Reg. ►Mem.	Mem. ►Reg.
MOVE de SR	—	3	4+	—	—
MOVE vers CCR	—	6	6+	—	—
MOVE vers SR	—	6	6+	—	—
MOVEP	mot	—	—	8	8
	long	—	—	12	12
EXG	—	3	—	—	—
EXT	mot	2	—	—	—
	long	2	—	—	—
LINK	—	8	—	—	—
MOVE de USP	—	2	—	—	—
MOVE vers USP	—	2	—	—	—
NOP	—	2	—	—	—
RESET	—	66	—	—	—
RTE	—	10	—	—	—
RTR	—	10	—	—	—
RTS	—	8	—	—	—
STOP	—	2	—	—	—
SWAP	—	2	—	—	—
UNLK	—	6	—	—	—

\+ ajouter le temps de calcul de l'adresse effective.

Fig. 12.14 - Temps d'exécution d'instructions spéciales.

Il est en outre possible de lancer, à partir du Basic, une routine en assembleur par CALL 16#JJJ où JJJ est l'adresse, en hexadécimal, de la routine.

Revenons au programme n° 14 qui permettait la visualisation d'une sinusoïde, il est maintenant simple de faire calculer en


```
                        ORG $2000;LIST 3;SIZE 1;

                        PTM:EQU @$4CF41;
2000    13FC00000004    MOVE 0,PTM+2;
        CF43
2008    13FC00E20004    MOVE $E2,PTM;
        CF41
2010    13FC00000004    MOVE 0,PTM+12;
        CF4D
2018    13FC00040004    MOVE 04,PTM+14;
        CF4F
2020    46FC2000        MOVE.W $2000,SR;
2024    13FC00010004    MOVE 1,PTM+2;
        CF43
202C    13FC00000004    MOVE 0,PTM;
        CF41
2034    2401            MOVE.L D1,D2;
2036    2401            MOVE.L D1,D2;
2038    2401            MOVE.L D1,D2;
203A    2401            MOVE.L D1,D2;
203C    2401            MOVE.L D1,D2;
203E    2401            MOVE.L D1,D2;
2040    2401            MOVE.L D1,D2;
2042    2401            MOVE.L D1,D2;
2044    2401            MOVE.L D1,D2;
2046    2401            MOVE.L D1,D2;
2048    2401            MOVE.L D1,D2;
204A    2401            MOVE.L D1,D2;
204C    46FC2700        MOVE.W $2700,SR;
2050    4E4E            TRAP 14;

PTM                             2               @$4CF41


   0 ERRORS
a7? G
FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

AV#4 TRAP ERROR
PC=0000204A SR=2004=.S0..Z.. US=FFFF
FORCEMON 2.2 > MM 201B
00201B    04 ?05.
```

suite page suivante

```
FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

FORCEMON 2.2 >
FORCEMON 2.2 > G 2000
PHYSICAL ADDRESS=00002000

FORCEMON 2.2 >
```

Programme 23 - Détermination de la vitesse d'exécution d'une séquence.

premier lieu les valeurs destinées au PIA, et ensuite de lancer le programme assembleur :

```
 5 REAL Y,T
10 K=16#2000
20 FOR I = 1 TO 64
30 T=REAL(I) -1.0
40 Y = 120.0 + 120.0*SIN(6.28*T/64.0)
50 J=INT(Y)
60 POKE(K,J)
70 K=K+1
80 NEXT I
90 CALL 16#1000
100 END
```

Remarque : ce Basic semi-compilé distingue, comme le Fortran, les quantités entières et réelles qui ne peuvent être mélangées.

On peut vouloir également passer des quantités plus importantes (mots, mots-longs). Une certaine reconstruction est à faire si l'on désire obtenir, dans l'autre sens, la récupération par le Basic d'une donnée sur quatre octets située en 2000, 2001, 2002, 2003_{16} (les poids forts commencent en 2000). On suppose que cette donnée est positive. La séquence Basic suivante permet de reconstruire un entier N sur 32 bits :

```
K=16#2000
N=PEEK(K+3) + 256*PEEK(K+2) + 256*256*PEEK(K+1)
            + 256*256*256*PEEK(K)
```


multiplier par 256 revient à décaler une quantité binaire de 8 bits à gauche.

Cette méthode est toutefois mise en défaut lorsque le nombre donné est négatif, car la multiplication de PEEK(K) (qui est compris entre $80 et $FF, mais qui est représenté par un nombre compris, lui, entre 00000080 et 000000FF **positif**) par 256^3 va le transformer en nombre négatif (en binaire complémenté) puisque le bit de poids fort sera 1. Le Basic détecte alors systématiquement un dépassement. Pour éviter les complications, il est préférable de ne transmettre que des nombres positifs, le signe étant précisé à part (ici, par exemple en 2004). Sur les mêmes bases, on peut écrire un nombre entier N à partir du Basic par des divisions successives par 256 :

```
10 K=16#2000
20 POKE(K+3,N)
30 N=N/256: POKE(K+2,N)
40 N=N/256: POKE(K+1,N)
50 N=N/256: POKE(K,N)
.....
```

COMMUNICATION PAR INTERRUPTION

Le **programme (Basic) n° 24** ne fait que calculer la suite des nombres entiers, mais, à chaque itération, il teste le contenu de la mémoire 2000. Si la valeur trouvée vaut 78 (ce qui en ASCII représente "N") le programme exécute la ligne 80 (impression de I). Si la valeur lue en 2000 est non nulle, le programme exécute la ligne 60 (impression de "?"). Dans les deux cas, il remet la zone 2000 à zéro. Enfin, si la valeur en 2000 est nulle, le programme n'affiche rien.

On remarque que ce programme commence par appeler une routine assembleur en $1000. Cette routine autorise les interruptions au clavier (tampon entrée-clavier plein).

Le **programme n° 25** détaille le fonctionnement de la séquence en assembleur : lors d'une frappe au clavier, la séquence d'interruption est activée ; on sauvegarde immédiatement les registres pouvant être utilisés par le Basic ; on lit la donnée que l'on place en D0 ; on l'affiche sur l'écran et on la place en $2000 ; enfin on restaure les registres stockés et on achève le traitement d'interruption par RTE.

```
list f,l ? >
1 CALL 16#1000
10 H=16#2000
20 I=I+1
30 T=PEEK(H)
40 IF T=78 GOTO 80
50 IF T=0 THEN GOTO 20
60 PRINT " ? ":POKE(H,0):
70 GOTO 20
80 PRINT " = ";I:POKE(H,0):
90 GOTO 20
100 END


execute


z ?
x ?
c ?
v ?
b ?
n ?
M ?
L ?
V ?
B ?
N = 344881
M ?
N = 363503
B ?
V ?
, ?
. ?
N = 407393
```

Programme 24 - Programme principal en Basic interrompu par une frappe au clavier.

Ce type de communication est très puissant ; rien n'empêche de provoquer des calculs intermédiaires, de lancer ou de stopper la "TRACE" (dans le cas d'une mise au point) et de commander en dynamique, un affichage, une impression partielle, un chargement de données.


```
                        ORG $1000;LIST 3;
                        SIZE 1;
                        ACIA:EQU @$50040;
                        VEC:EQU @$74;
1000    43FA0014        INIT:LEA INTER,A1;/* INITIALISATION ADRESSE $1000 */
1004    21C90074        MOVE.L A1,VEC;
1008    13FC00910005    MOVE $91,ACIA;
        0040
1010    46FC2100        MOVE.W $2100,SR;
1014    4E75            RTS;
1016    48E7FFFF        INTER:MOVEM.L D0-D7/A0-A7,-(A7);/* SAUVEGARDE */
101A    46FC2700        MOVE.W $2700,SR;
101E    103900050042    MOVE ACIA+2,D0;
1024    4EB900021C7C    JSR @$21C7C;
102A    4EB900021E24    JSR @$21E24; /* ECRITURE CONSOLE */
1030    11C02000        MOVE D0,@$2000;
1034    46FC2100        MOVE.W $2100,SR;
1038    4CDFFFFF        MOVEM.L (A7)+,D0-D7/A0-A7; /*RESTAURATION */
103C    4E73            RTE;;

ACIA                    3               @$50040
INIT                    5               $1000  PC-R.
INTER                   10              $1016  PC-R.
VEC                     4               @$74

   0 ERRORS
a7? G
```

Programme 25 - Séquence assembleur permettant l'interruption au clavier du programme 24.

CHAPITRE 13 | l'écriture des EPROM

Les "Erasable PROM" (ou mémoires à lecture seulement, mais réutilisables après effacement) sont un moyen de stockage commode, économique et de plus en plus indispensable. Malheureusement, la programmation ne peut se faire qu'en dehors de l'ordinateur, et il faut prévoir au moins deux types de dispositifs :

- pour effacer les EPROM, il faut les soumettre à un rayonnement ultraviolet de longueur d'onde approximative λ = 4000 Å, ceci pendant quinze à vingt minutes (les mémoires EPROM possèdent une petite fenêtre transparente, derrière laquelle sont situées toutes les cellules sensibles : on masque à la fin de l'effaçage cette fenêtre pour éviter toute source de lumière). Cette phase d'effacement est la plus longue, mais la plus facile à mettre en oeuvre, les effaceurs étant disponibles dans tout magasin spécialisé ;
- pour les écrire (les claquer, les brûler, etc.) le problème est légèrement plus complexe puisqu'il faut vider une partie de la mémoire RAM de l'ordinateur (celle qui contient les programmes définitifs) dans l'EPROM, tout en respectant les spécifications (tensions, délais,...) du constructeur.

Développons ce dernier point pour une EPROM de type 2764 (donc de 64 K bits, structurés en 8 K octets) une des plus répandues (sur notre plaquette d'essais, le Basic occupe deux emplacements de 2764, et tient donc sur 16 K octets) :

a) on applique l'adresse voulue sur les broches A0 à A12 ;
b) on envoie l'octet que l'on désire inscrire à cette adresse aux broches D0-D7 ;
c) la broche $\overline{OE}$ est mise à l'état haut ;
d) on porte la broche $\overline{PGM}$ à un niveau actif **bas** pendant 50 ms.

Pendant toutes ces opérations, la broche V_{pp} doit être au potentiel de 21 V.

Une électronique élémentaire faisant suite au PIA va permettre d'effectuer toutes ces opérations. On remarquera tout d'abord qu'il faut fournir, de façon stable pendant chaque cycle d'écriture treize lignes d'adresses et huit lignes de données, plus des lignes de commandes pour enchaîner ces opérations.

Il faudra donc utiliser un registre de stockage qui permettra le transfert en deux fois de l'adresse par le PIA-A (de type 74373, 74377 ou 74378), l'octet de données sera transmis par le PIA-B (qui est ici bidirectionnel ; on pourra donc aussi lire l'EPROM, à la fin d'un cycle d'écriture, pour contrôle). L'impulsion de 50 ms sera créée par le PTM (temporisateur #3, en mode monocoup, etc.) mais devra passer par un circuit NI avant d'arriver sur la broche $\overline{PGM}$. Enfin, les lignes CA2 et CB2 permettront d'agir sur le stockage d'adresse et sur le type d'action : lecture ou impression de l'EPROM. L'électronique ne nécessite en fait qu'un circuit 7400 (NAND) et un registre 74378 (**figure 13.1**). A noter que l'on pourrait se passer du circuit 7400 en gérant une des lignes CA2 ou CB2 (au niveau bas, pendant le cycle du mode monocoup) et en utilisant une des lignes d'horloge pour commander le registre 74378.

L'organigramme du programme de transfert et de contrôle est donné en **figure 13.2**.

Précautions logicielles : pour être pleinement efficace, le programme devra tout d'abord être écrit en code indépendant de la position. Sinon, il faudrait le reloger pour l'exécuter dans la zone mémoire où il a été mis au point. On devra donc exclure les adressages absolus référençant des zones de travail temporaires et remplacer ces dernières par des réservations (DC ou DS) dont l'adresse est récupérée par des LEA. Il faudra également remplacer les JSR (qui travaillent sur des adresses absolues) par des BSR.L (qui codent le déplacement relatif) lorsque les routines appelées appartiennent au programme de l'utilisateur (mais les routines du moniteur seront toujours appelées par JSR).

On utilisera avec profit l'instruction BSR.L * ± d pour transformer les JSR. Le moniteur (commande DC) permettra de s'affranchir de tout calcul fastidieux et le désassemblage de l'instruction BSR.L * permettra une vérification rapide. Pour éviter de trop nombreuses modifications de ce type pour un appel à un même sous-programme, on pourra, si le temps d'exécution n'est pas critique, utiliser une écriture différée :

```
BSR SOUP
.......
BSR SOUP
.......
SOUP: BSR.L *-2438;   seule référence par déplacement calculé.
RTS;
```

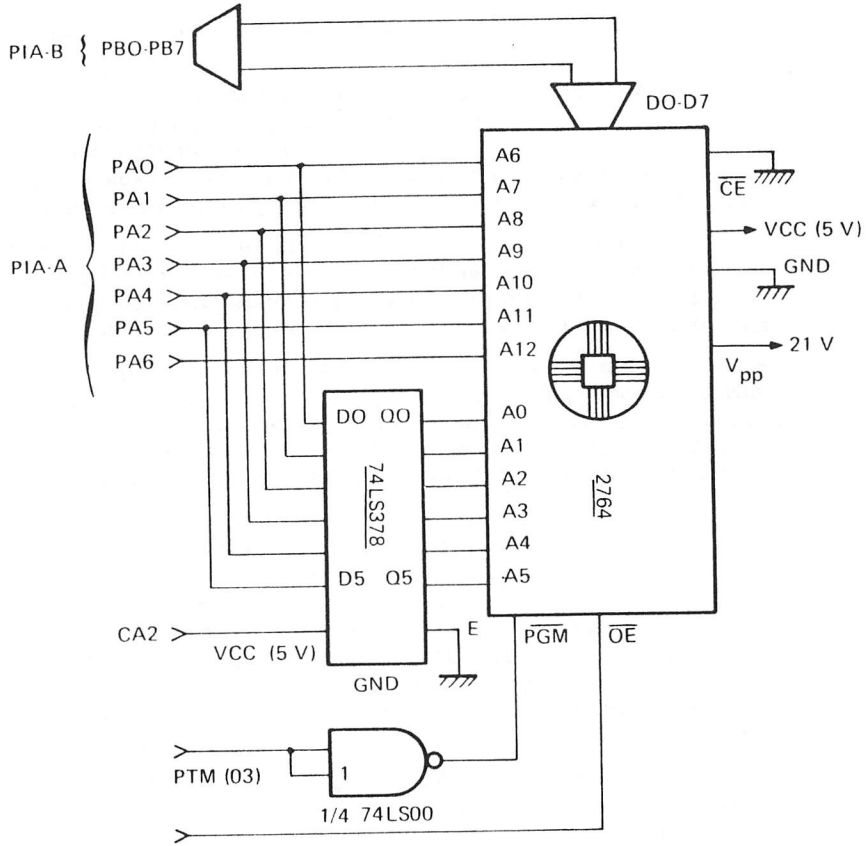



Fig. 13.1 - Programmation d'une EPROM 2764.

Pratiquement, ces écritures ne seront nécessaires que dans le cas où le programme, supposé long, a été assemblé en plusieurs fois, et où les différents segments communiquent entre eux par diverses routines.

Une autre précaution, plus élémentaire, consistera à sauvegarder les registres utilisés par une instruction MOVEM au début et à la fin de chaque sous-programme.

On notera que sur notre micro-ordinateur les deux EPROM sont sélectionnées par une même adresse (paire), ce qui permet d'accéder à deux octets simultanément. Il faudra tenir compte de cette organisation lors de la création d'EPROM.

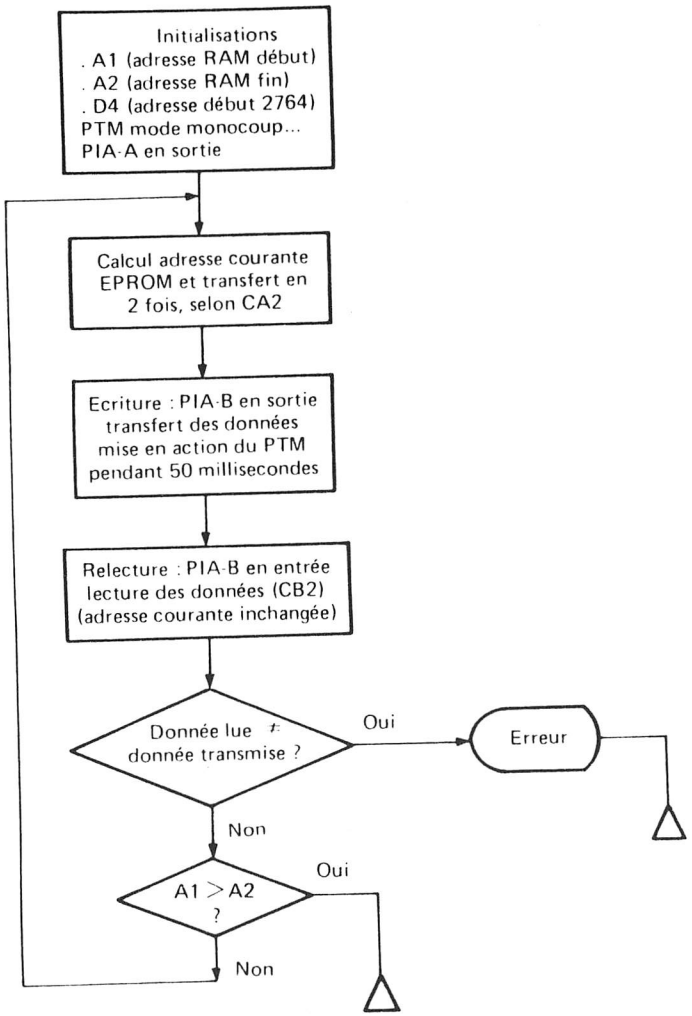


Fig. 13.2 - Organigramme du programme de contrôle et de transfert sur EPROM.


```
:e attach file10 vfile  EPROM
trad
PHYSICAL ADDRESS=0000A100 0000A534
     1   ORG $3000;
     2  LIST 3;
     3  SIZE 1;                          /* A1 contient l adresse début*/
     4  PTM:EQU @$4CF41;                 /*A2 contient l adresse de fin */
     5  PIA:EQU @$5CEF1;                 /*  D4 l adresse début EPROM */
     6  PIB:EQU PIA+4;
     7  UN:EQU $3C;
     8  ZERO:EQU $34;
     9  INIT:MOVE 0,PIA+2;
    10  MOVE 255,PIA;
    11  MOVE 4,PIA+2;
    12  MOVE ZERO,PIA+2;
    13  MOVE 0,PTM+2;
    14  MOVE $A3,PTM;
    15  MOVE $13,PTM+12;
    16  MOVE $88,PTM+14;                  /* durée du cycle 50 ms */
    17  MOVE 1,PTM+2;
    18  JSR @$21C7C;
    19  ADRESSE:MOVE.W D4,D5;
    20  MOVE D5,PIA;
    21  MOVE UN,PIA+2;
    22  NOP;
    23  MOVE ZERO,PIA+2;
    24  LSR.W 6,D5;
    25  MOVE D5,PIA;
    26  BSR ECRIRE;
    27  BSR RELIRE;
    28  CMP D6,D7;
    29  BEQ BON;
    30  ERREUR:MOVE  * ,D0;
    31  JSR @$21E24;
    32  TRAP 14;                          /* on peut arrêter apres 1 erreur */
    33  BRA LOIN;                         /* ou bien poursuivre             */
    34  BON:MOVE  ' ',D0;
    35  JSR @$21E24;
    36  LOIN:CMP.L A1,A2;
    37  BEQ.L FINI;
    38  ADD.W 1,D4;
    39  BRA ADRESSE;
    40  ECRIRE: MOVE 0,PIB+2;
    41  MOVE 255,PIB;                     /* toutes les lignes en sortie sur PIA-B */
    42  MOVE UN,PIB+2;                    /* OE inactif (haut) sur EPROM */
    43  MOVE (A1)+,D6;
    44  MOVE D6,PIB;
    45  BSR PROGRAM;
    46  MOVE ZERO,PIB+2;
    47  RTS;
    48  RELIRE:MOVE 0,PIB+2;
    49  MOVE 0,PIB;
    50  MOVE ZERO,PIB+2;                  /* toutes les lignes en entrée */
    51  MOVE PIB,D7;                      /* OE actif (lecture) sur EPROM */
    52  MOVE UN,PIB+2;
    53  RTS;
    54  PROGRAM:MOVE 1,PTM+2;
    55  MOVE 1,PTM;
    56  MOVE 0,PTM;                       /*activation PIN #3*/
    57  TST:MOVE PTM+2,D0;
```

```
                                    /* attente fin de cycle  */
   58  AND 4,D0;
   59  BEQ TST;
   60  RTS;
   61  FINI:LEA MES1,A5;
   62  LEA MES1+10,A6;
   63  JSR @$21C14;
   64  TRAP 10;
   65  MES1:DC $0D;
   66  DC $0A;
   67  DC  FIN EPROM  ;
   68  ;

STOP
10 detach file10
r 12:47 3.145 38
```

Programme 26 - Ecriture des EPROM.

conclusion

Le 68000 et ses nombreux circuits périphériques permettent les traitements les plus divers, allant de la programmation d'automatismes jusqu'à la communication aisée avec d'autres systèmes. Mais beaucoup d'autres applications, on s'en doute, peuvent être traitées avec succès par de telles machines.

Le calcul scientifique n'est pas en reste. D'une part les opérations se font assez rapidement (9 μs au maximum, pour une multiplication 16 x 16 à 8 MHz) et d'autre part, les instructions multiprécisions permettent d'étendre les opérations de base à des opérandes de longueurs indifférentes. Ainsi, les quelques programmes que nous avons donnés en exemple pourraient sans grosses difficultés traiter des nombres dépassant 32 ou 64 bits.

Par ailleurs, la philosophie développée dans le système que nous avons utilisé est particulièrement satisfaisante : le développement d'une application se fait de façon entièrement autonome, et l'on voit peu à peu fonctionner le programme en test, grâce à l'aide efficace du moniteur. Ensuite, il suffit de retirer l'assembleur-éditeur et de le remplacer par le programme au point mis sur EPROM.

Certes, l'enseignement de l'informatique a tendance à privilégier la maîtrise des progiciels les plus courants, au détriment des concepts de base dont la rentabilisation est plus longue, mais sûrement plus formatrice ; on trouve ici une machine d'enseignement, ou d'auto-apprentissage, qui peut harmonieusement compléter une formation utilitaire.

Enfin, le 68000 lui-même équipe chaque jour davantage de machines, dont le MacIntosh (Apple) promis à un bel avenir et le QL (Sinclair) qui, tel le ZX81, va submerger le marché du micro-ordinateur personnel, par une production de masse à prix compé-

titif. Plus ambitieuse encore est la SM90 qui, conçue et construite en France, commence aussi une belle carrière.

Toutes ces bonnes raisons font que la connaissance du 68000 doit faire partie du bagage de tout honnête informaticien.

ANNEXE 1
le jeu de caractères ASCII

LSB \ MSB		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

DLE	- Data Link Escape	NUL	- Null
DC	- Device Control	SOH	- Start of Heading
NAK	- Negative Acknowledge	STX	- Start of Text
SYN	- Synchronous idle	ETX	- End of Text
ETB	- End of Transmission Block	EOT	- End of Transmission
CAN	- Cancel	ENQ	- Enquiry
EM	- End of Medium	ACK	- Acknowledge
SUB	- Substitute	BELL	- Bell
ESC	- Escape	BS	- Backspace
FS	- File Separator	HT	- Horizontal Tabulation
GS	- Group Separator	LF	- Line Feed
RS	- Record Separator	VT	- Vertical Tabulation
US	- Unit Separator	FF	- Form Feed
SP	- Space (Blank)	CR	- Carriage Return
DEL	- Delete	SO	- Shift Out
		SI	- Shift In

Les valeurs de 00 à 1B peuvent se rentrer au clavier par la touche CTRL (control)

CTRL @ 00
CTRL A 01
CTRL B 02
.....
CTRL Z 1A
Escape 1B ($=27_{10}$)

ANNEXE 2
manipulation de nombres destinés à une transmission série

```
1           character*1 r(16)
2            r(1)='0'
3           r(2)='1'
4           r(3)='2'
5           r(4)='3'
6           r(5)='4'
7           r(6)='5'
8           r(7)='6'
9           r(8)='7'
10          r(9)='8'
11          r(10)='9'
12          r(11)='j'
13          r(12)='k'
14          r(13)='l'
15          r(14)='m'
16          r(15)='n'
17          r(16)='o'
18           write(9,900) 'D','D','D','D'
19          do 10 k=6000,6020
20          i2=k/256
21           i1=k-i2*256
22          i2f=i2/16
23           i2b=i2-16*i2f
24          i1f=i1/16
25           i1b=i1-16*i1f
26           i2f=i2f+1
27           i2b=i2b+1
28           i1f=i1f+1
29           i1b=i1b+1
30           write(9,900) r(i2f),r(i2b),r(i1f),r(i1b)
31    10     continue
```

suite page suivante

```
32    900    format (4a1)
33           write(9,900)'F','F','F','F'
34           stop
35           end
```

Programme Fortran permettant le transfert d'entiers codés sur 16 bits au moyen de quatre caractères ASCII.

```
1,$P#p

DDDD
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
177j
177k
1771
177m
177n
177o
1780
1781
1782
1783
1784
FFFF
```

Mise en forme des nombres entiers de 6000 à 6020.

On remarque que 1770 est en fait, 00110001 00110111 00110111 00110000 (31,37,37,30 en ASCII) ; ces quatre caractères seront mis sous la forme 0001 0111 0111 0000 par suppression des 4 bits de poids forts et ils s'interprètent en binaire : $2^4 + 2^5 + 2^6 + 2^8 + 2^9 + 2^{10} + 2^{12} = 6000$.

ANNEXE 3
émulation de consoles

```
FORCEMON 2.2 > BM 24000 28000 18000
PHYSICAL ADDRESS=00024000 00028000
PHYSICAL ADDRESS=00018000

FORCEMON 2.2 > MM 1B5D6;L
01B5D6    0000121B ?2020591B      Curseur HOME UP
01B5DA    0000121B ?2020591B
01B5DE    00000009 ?3720591B      Curseur HOME DOWN
01B5E2    3761111B ?80000000
01B5E6    00001D1B ?00004B1B      Effacement jusqu'au bout de la ligne
01B5EA    00000F1B ?00004B1B
01B5EE    0000007F ?              Effacement d'un caractère
01B5F2    80000000 ?
01B5F6    0000181B ?00006B1B      Effacement jusqu' au bas de l'écran
01B5FA    0000181B ?00006B1B
01B5FE    00000001 ?              fin du mode insertion
01B602    80000000 ?
01B606    00000002 ?              début du mode insertion
01B60A    80000000 ?
01B60E    0000000D ?              retour chariot (return)
01B612    0000000D ?
01B616    0000000A ?              ligne suivante (Line feed)
01B61A    0000000A ?
01B61E    00000C1B ?0000001A      Curseur haut
01B622    00000C1B ?0000001A
01B626    00000B1B ?0000000A      Curseur bas
01B62A    00000B1B ?0000000A
01B62E    00000010 ?00000006      Curseur à droite
01B632    00000010 ?00000006
01B636    00000008 ?00000015      Curseur à gauche
01B63A    00000008 ?00000015
01B63E    00001A1B ?0000003F      Insertion d'une ligne
01B642    00001A1B ?8000000.

FORCEMON 2.2
```

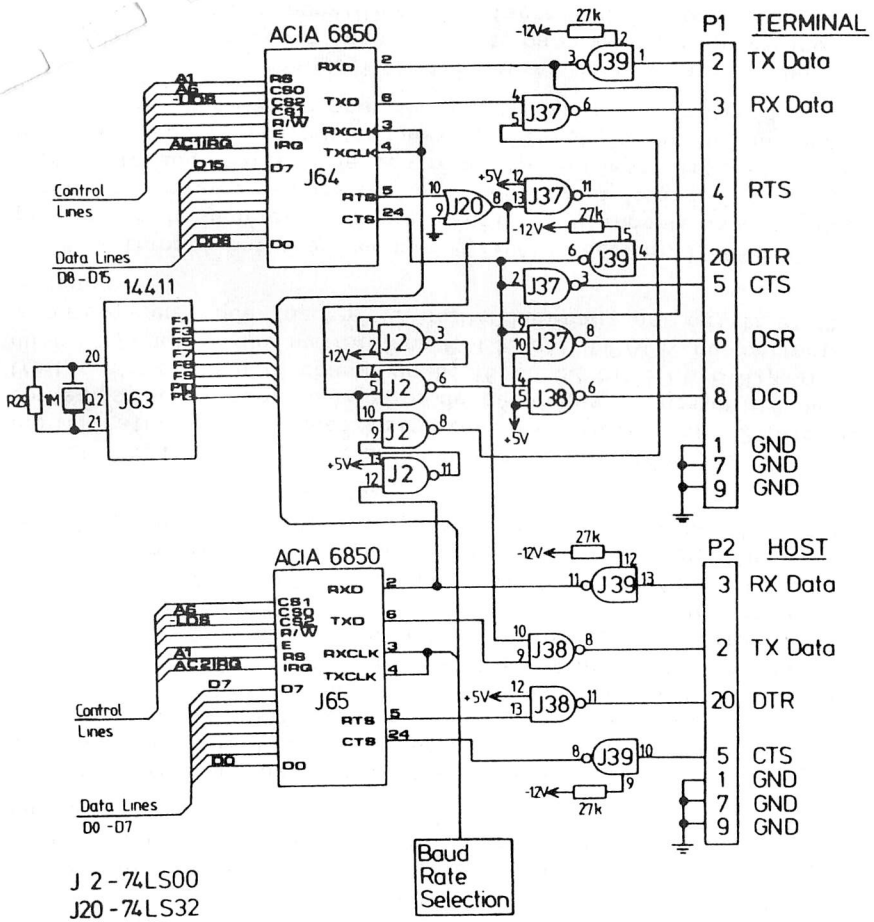
Emulation d'une console différente sans changement de l'EPROM contenant l'assembleur-éditeur. Les codes sont transmis de la droite vers la gauche à la console.

1 - on reloge l'assembleur par la commande BM pour le placer en RAM utilisateur ;
2 - on modifie tous les codes de fonction non compatibles avec ceux de la nouvelle console ;
3 - si les touches sont absentes (ou les commandes elles-mêmes) on peut en réaffecter une ou plusieurs à un usage particulier. Ici, en 1B5D6 la fonction HOME est remplacée par "Curseur" 1ère ligne, 1ère colonne. En 1B63E la touche "?" permet l'insertion d'une ligne.

Une commande contenant 80000000 est désactivée et est prise en charge par l'éditeur, quand cela est possible. Les commandes introduites permettent le fonctionnement d'une console ADDS-25.

ANNEXE 4
connexion entre les ACIA

La figure de la page suivante (document Force Computer) donne le schéma de connexion des ACIA entre l'hôte et le terminal. On remarque que les bornes 2 et 3 de P1 peuvent être reliées directement aux bornes 2 et 3 de P2 par l'intermédiaire de quatre portes NON-ET (ou NAND). Le mode TRANSPARENT met en communication de cette façon TERMINAL et HOTE par la simple activation de ces portes.



ANNEXE 5
regards sur les produits nouveaux

Le 68000 est l'élément le plus connu d'une famille qui se développe constamment, mais qui préserve une totale compatibilité ascendante : ainsi, l'investissement humain et matériel fait sur une machine n'est jamais perdu. Les nouveaux produits proposés sont plus puissants (68010, 68020) ou moins coûteux (68008).

Le 68008

Ce microprocesseur garde les possibilités d'un vrai 16 bits ; il possède un bus de données à 8 bits. Il reste compatible avec toute la famille 68000 et ses périphériques. Son cycle de lecture plus long le rend moins efficace que le 68000, dont il n'a que 60 % de la rapidité. Il est également de taille plus réduite (48 broches) et d'un prix inférieur. On pourrait le présenter comme le meilleur des micro-ordinateurs 8 bits.

Le 68010

Entièrement compatible (broche à broche) avec le 68000 il en étend les possibilités par de nouvelles instructions (transfert double MOVES ; amélioration, réalisée par le processeur, des boucles écrites avec l'instruction BDcc).

Il supporte le concept de mémoire virtuelle, ce qui permet à un utilisateur de gérer un espace mémoire bien plus grand qu'il n'a en réalité (il travaille en fait sur une mémoire extérieure de type disque, etc.). Le 68010 dispose, en effet, de la possibilité de suspendre le déroulement d'une instruction, le temps que la partie absente - mais sollicitée - soit rechargée ; dans ces conditions, l'utilisateur ne se soucie de rien. Le 68010 peut être également considéré comme une machine virtuelle (susceptible par exemple de développer le logiciel d'une autre machine dont le matériel n'est pas disponible). Le programme à développer sera écrit en mode utilisateur et un programme en mode superviseur

émule la machine absente ; pour ces raisons, quelques retouches ont été apportées au niveau des instructions privilégiées, permettant une meilleure utilisation des codes condition par le programme en développement.

Enfin, des performances électriques supérieures, en font une machine plus rapide que le 68000.

Le 68020

C'est un authentique 32 bits, apportant de grandes améliorations au 68010. Un bus de 32 bits sur les adresses et les données, des opérations sur 32 bits associées à une fréquence d'horloge de 16 MHz, en font une machine trois à quatre fois plus puissante que le 68000 ; il peut opérer entre 2 et 4 Mips. Cela est dû aussi bien sûr à la mémoire cache (d'accès plus rapide) et à la recherche anticipée des instructions (comme dans le 68010, dans le cas des boucles).

Dans l'optique de l'amélioration des vitesses, on note la possibilité d'adjoindre l'interface coprocesseur (68881) travaillant simultanément avec le 68020 ; sont alors possibles les opérations - conformes aux normes IEEE - en précision simple (32 bits) double (64) ou étendue (80) ou même, des opérations trigonométriques. Plus généralement, jusqu'à huit coprocesseurs pourront travailler simultanément en utilisant des ressources communes. A l'instruction TAS du 68000, qui permettait de constater la disponibilité d'une ressource ou non, est ajoutée maintenant l'instruction CAS qui permet d'implanter en mémoire une zone d'échange, entre tous ces coprocesseurs.

Des instructions plus classiques ont été ajoutées ; ainsi, la multiplication 32 x 32, des instructions de rotation sur 32 bits en un temps indépendant du nombre de décalages, la possibilité de traiter n'importe quel champ de 8, 16 ou 32 bits. En BCD, on note l'adjonction des instructions PACK et UNPACK permettant le passage en mode "packé" (2 digits décimaux par octet) et inversement.

Les trappes ont été étendues : TRAPV devient TRAPcc, ce qui permet de générer un déroutement sur n'importe laquelle des conditions testées. Les instructions de manipulation de bits ont encore été augmentées ; le contrôle CHK devient CHK2 et permet de tester deux bornes quelconques, CMP2 est également une instruction de comparaison nouvelle, positionnant le registre d'état.

Au niveau des commodités d'écriture, de nouveaux modes d'adressage apparaissent (déplacement, dans le mode adressage indexé, sur 32 bits ; apparition d'un facteur d'échelle, agissant sur le registre d'index).

On peut avoir aussi plusieurs tables de vecteurs d'interruptions et des instructions d'aide au développement sont intégrées : écriture simplifiée de modules translatables (avant mise en ROM, RAM, EPROM...), ou encore BKPT qui permettra de placer un point d'arrêt par programme.

Ce processeur, avec ses 200 000 transistors se situe actuellement au sommet de la hiérarchie. Il équipera les premiers ensembles dès 1986, avec un marché espéré d'un million de pièces en 1987, contre 100 000 en 1984. De récents accords entre Thomson et Motorola, permettront à ce circuit d'être produit en France, avec tous les avantages qui en découleront, pour les concepteurs et tous les utilisateurs. D'autres circuits, s'intercalant entre le 68010 et le 68020 seront commercialisés (tel le 68012) permettant de choisir le modèle le mieux adapté, au moindre coût.

Ainsi, nous voyons avec satisfaction arriver des machines très puissantes qui seront très rapidement à la portée de chacun, et qui ne nécessiteront pas un total réapprentissage.