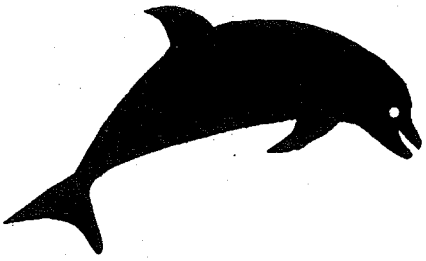
R. Moret
24.6.90

LABORATOIRE DE MICROINFORMATIQUE - EPFL

DAUPHIN 68008



A.Schmitz, J.D.Nicoud, R.Beuchat

3ème édition, mars 1988

Préface

Cette brochure présente une description matérielle du Dauphin 68008 développé au LAMI, le mode d'utilisation du programme Moniteur, la technique de codage des instructions en assembleur 68000, le téléchargement d'un programme réalisé sur Smaky 100 sur le Dauphin ainsi que l'utilisation manuelle du Dauphin. Un chapitre présente le programme moniteur du Dauphin. Le dernier chapitre documente les schémas.

Le Dauphin a été développé en 1976 pour le processeur Signetics 2650, et commercialisé en 77-79 par Stoppani Travers. Une dizaine de processeurs différents ont tourné sur cette machine; le dernier en date est un 68020 en mode 8 bits.

Ce document accompagne les manipulations suivantes :

- DAMU Initiations aux microprocesseurs
- DAMA Utilisation manuelle du Dauphin
- DATE Téléchargement du SMAKY sur Dauphin
- DAIF Réalisation d'un interface facile
- DAINT Les interruptions

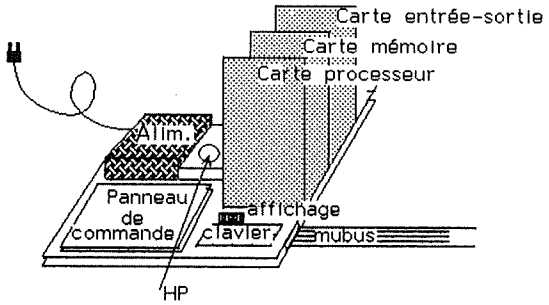
Ces manipulations permettent de comprendre les principes d'un microordinateur.

A. Schmitz, J.D. Nicoud, R. Beuchat
Lausanne, mars 1988

Table des matières

1 Le Dauphin M68008

1.1 Schéma du DAUPHIN



DAUP88

Fig. 1.1 Schéma d'un DAUPHIN

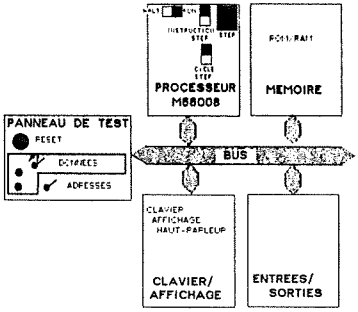
Un Dauphin se compose usuellement de 5 circuits imprimés reliés entre eux par un **bus** permettant le transfert des informations.
Ces éléments sont :

- une carte processeur (dans ce cas un 68008)
- une carte mémoire (ROM et RAM comprenant à la fois les programmes utilisateurs et le programme moniteur)
- une carte clavier-affichage permettant le dialogue avec l'utilisateur
- une carte d'entrée-sortie série
- un panneau de commande permet de réinitialiser le système (RESET) et de prendre le contrôle direct de la mémoire et des périphériques.

Sur le bus, on trouve également une prise MUBUS qui permet au Dauphin de communiquer avec le monde extérieur.

Au Dauphin minimum, il est possible d'ajouter une carte d'entrée-sortie si des transferts doivent être effectués avec un Smaky, un autre système ou un périphérique.

La représentation des différents éléments du Dauphin sous forme de schéma bloc est donné dans la figure 1.2.



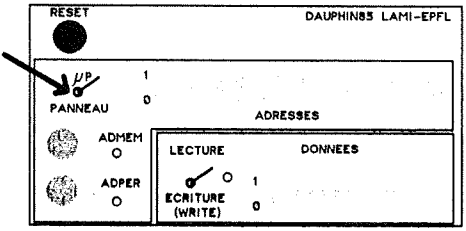
DAUPSB

Fig. 1.2 Schéma-bloc du Dauphin

1.2 Le panneau de test

Le Dauphin est un "biprocesseur" qui peut être contrôlé manuellement ou par un processeur 8 bits.

Un commutateur du panneau de test permet de choisir entre la commande manuelle et la commande par processeur.



DAUPPAN

Fig. 1.3 Panneau de test et commutateur de choix.

Le chapitre 5 explique comment utiliser le Dauphin en contrôle manuel.

1.3 Le processeur

Le processeur Motorola 68008 est entièrement compatible sur le plan logiciel avec le M68000.

Des cartes processeur Z80, 6802 et 2650 sont à disposition pour tester d'autres processeurs.

1.4 Mémoires

Seuls 14 bits d'adresses sont disponibles dans le Dauphin, permettant d'adresser au maximum 16 K-octets de mémoire (un connecteur sur la carte mémoire permet d'accéder aux autres lignes d'adresse s'il est nécessaire d'avoir une extension mémoire).

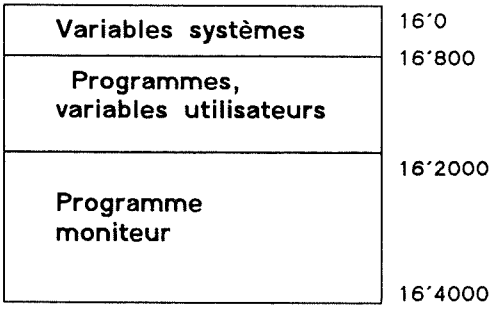


Fig. 1.4 Positions mémoires du Dauphin

Dans la partie en ROM se trouve un programme nommé MONITEUR, exécuté à l'enclenchement du système. Le programme a pour but d'offrir à l'étudiant un certain nombre de facilités :

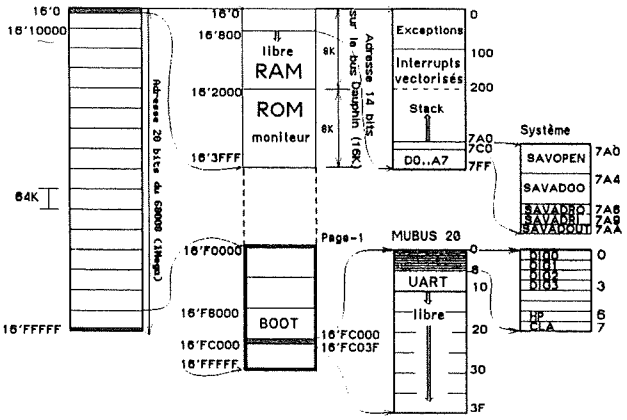
- visualisation et modification de l'état de la mémoire
- chargement du code par ligne série
- routines d'affichage

La disposition des éléments dans la mémoire est décrite dans la figure 1.5.

Les positions 0 à 16'3FFF (16K) contiennent la mémoire RAM et la mémoire ROM (qui contient le programme moniteur).

Les adresses des périphériques se trouvent à la fin de la mémoire, les adresses 16'FC000 à l'adresse 16'FC03F permettent l'accès à 64 périphériques et les adresses 16'FC040 à 16'FFFFF sont réservées; avec le 68000, toutes ces adresses sont considérées comme négatives.

L'utilisateur dispose des positions mémoires 16'800 à 16'1FFF pour ses programmes. Les adresses 0 à 16'7FF sont réservées au moniteur.



DAUPSP

Fig. 1.5 Espace d'entrée-sortie et périphériques de la carte clavier.

Le moniteur sauve les registres du processeur lors d'un arrêt du programme utilisateur à partir de l'adresse 16'7C0 (par un (**RESET**)): le registre D0.32 est sauvé de 16'7C0 à 16'7C3, etc.

1.5 Entrées/sorties

1.5.1 Introduction

Pour effectuer une entrée-sortie, il est nécessaire de sélectionner une adresse dans un groupe de 32 adresses. Ce groupe s'appelle MUBUS et se trouve à l'adresse -16'4000 = 16'FFC000.

1.5.2 Clavier-affichage

Le décodage des périphériques de la carte clavier-affichage distingue des adresses valides en écriture (les chiffres affichés) et une adresse valide en lecture seulement (le code du clavier). Des adresses sont réservées pour une extension.

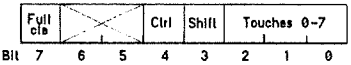
Les principaux périphériques sont:

Adresse	Description
MUBUS+0,1,2,3	**Les segments lumineux de l'affichage** (en écriture) La durée d'exécution de sélection de ces périphériques est fixée par le monostable du clavier (environ 3ms)

MUBUS+4,5 **Les douilles sur le clavier** (en écriture)
Ces douilles permettre des fonctions spéciales de déverminage.

MUBUS+6 **Le haut-parleur**
Fait basculer le flip-flop commandant la lampe et le haut-parleur. Le contenu des informations n'a pas d'importance.

MUBUS+7 **Le clavier** (en lecture).
L'action sur une des 8 touches marquées de 0 à 7 et codées sur 3 bits fait basculer le flip-flop appelé "FULL" qui reste à 1 jusqu'à la fin de la prochaine lecture de ce périphérique. Les touches SHIFT et CTRL sont des touches directes sans effet sur le "FULL" (voir la figure ci-dessous)



DAUDI

Fig. 1.6 Codage d'une touche du clavier

Pour envoyer une impulsion dans le haut-parleur l'instruction de transfert doit être précédée des déclarations :

```
;
MUBUS   = -16'4000
HP      = MUBUS+6
...
MOVE.B  D1,HP
...
```

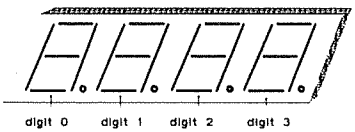
Ces périphériques sont étudiés plus en détail dans les sections suivantes.

1.5.3 Affichage

Le circuit d'affichage est composé de 4 éléments. Chaque éléments est composé de 7 segments et d'un point, il est accédé en écrivant à l'adresse MUBUS+0,1,2 ou 3.

Les périphériques d'affichage répondent aux adresses :

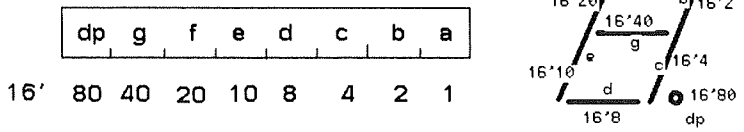
élément 0 : DIG0 = MUBUS
élément 1 : DIG1 = MUBUS + 1
élément 2 : DIG2 = MUBUS + 2
élément 3 : DIG3 = MUBUS + 3



DAUPDIG4

Fig. 1.7 Position des éléments sur l'affichage

Chaque segment correspond à un bit dans l'octet représentant un chiffre selon le format présenté à la figure 1.8. Les bits à 1 allument les segments correspondants sur l'afficheur d'adresse sélectionnée.



DAUPP

Fig. 1.8 Correspondance bit-segment affiché.

Un élément peut être allumé par l'instruction **MOVE.8 D1,DIG0**, le contenu du registre D1 contient le code des segments affichés selon le format décrit dans la figure 1.8.

Note : pour augmenter l'intensité relative des chiffres allumés par l'instruction MOVE.8 D1,DIGx, cette instruction est considérablement retardée par un monostable. Le transfert sur l'affichage en effet 3ms au lieu de 0.5μs.

Chaque élément à afficher doit être codé en binaire ou en hexadécimal. La table ci-dessous donne les déclarations fréquemment utilisées.

ZERO	=	16'3F	HUIT	=	16'7F
UN	=	16'6	NEUF	=	16'6F
DEUX	=	16'5B	letr_A	=	16'77
TROIS	=	16'4F	letr_B	=	16'7C
QUATRE	=	16'66	letr_C	=	16'58
CINQ	=	16'6D	letr_D	=	16'5E
SIX	=	16'7D	letr_E	=	16'79
SEPT	=	16'7	letr_F	=	16'71

1.5.4 Clavier

Le clavier répond à l'adresse MUBUS+7 (uniquement en lecture).

Le clavier comprend un groupe de 8 touches et 2 touches de mode se comportant de la même façon que les touches SHIFT et CTRL des terminaux.

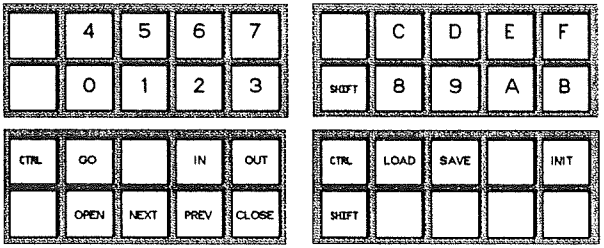


Fig. 1.9 Codes clavier

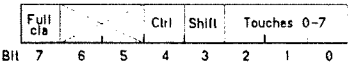


Fig. 1.10 Mot de 8 bits contenant les codes claviers

Le mot contenant le code clavier envoyé au processeur est formé de 8 bits :

- bits 0-2 codage de la touche appuyée
- bit 3 touche SHIFT enfoncée
- bit 4 touche CTRL enfoncée
- bit 5 et 6 bits inutilisés
- bit 7 indicateur qu'une touche a été enfoncée

La lecture de l'adresse périphérique MUBUS+7 permet de lire une valeur codée de 16'0 à 16'1F. Comme les bits supérieurs à 16'1F peuvent être à 1, il faut masquer la valeur lue. Une définition complète du codage du clavier est donc :

```
CLA   = 16'7   ; masque des 8 touches chiffre
SHIFT = 16'8   ; masque de la touche (SHIFT) (ordres)
CTRL  = 16'10  ; masque de la touche (CTRL) (ordres)
MCLA  = 16'1F  ; masque des bits significatifs
```

1.5.5 Haut-parleur

Le haut-parleur répond à l'adresse MUBUS+6. Il voit sa membrane alternativement déplacée à chaque écriture sur le périphérique d'adresse MUBUS+6. Si une écriture s'effectue toutes les 0,5 ms, c'est donc une fréquence de 1 kHz qui se fera entendre.

1.5.6 Interface série

La carte série répond aux adresses MUBUS+8 et MUBUS+9. Cette carte comporte une interface programmable 8251 (Intel, AMD)

1.5.7 Extensions

Les adresses MUBUS+16'10 à MUBUS+16'1F sont réservées pour des cartes d'entrées-sorties supplémentaires. Des interfaces parallèles sont à disposition.

1.5.8 Extensions sur le connecteur MUBUS

Les adresses MUBUS+16'20 à MUBUS+16'3F sont entièrement libres pour des montages en logidules.

2 Utilisation du moniteur

2.1 Introduction au moniteur

Le moniteur est un programme résidant en mémoire ROM. Il permet de visualiser à l'écran et de modifier le contenu de la RAM par des ordres donnés au clavier. Le programme se trouve dans les positions mémoires 16'2000 à 16'3FFF.

Lors de la mise en route du Dauphin (appuyer sur la touche (RESET)), le programme moniteur s'exécute. Il affiche 0000 et surveille le clavier qui permet d'intervenir dans la mémoire et sur les entrées-sorties. Si ce n'est pas le cas, vérifier la position des interrupteurs sur la carte de base (figure 1.3) et sur la carte processeur (figure 1.2).

2.2 Ordres du moniteur

Les ordres du moniteur 68008 sont donnés ci-dessous. Un nombre tapé au clavier est représenté par un rectangle et la combinaison de touches qui correspond à un ordre est représenté par une cartouche arrondie.

Les adresses et les valeurs sont données en hexadécimal.

Il faut taper d'abord les chiffres, puis l'ordre. Le fait de ne pas taper de chiffre devant un ordre modifie son interprétation.

adresse-(OPEN) "ouvre" la position d'adresse tapée, c'est-à-dire montre son contenu tant que la touche (CTRL) est maintenue pressée, avec affichage d'un "d" en 1ère position affichée

OPEN	ouvre la dernière position ouverte
valeur-NEXT	charge la valeur tapée dans la position courante et ouvre la position suivante. Tant que la touche CTRL est maintenue pressée, on voit le contenu de l'adresse avec un "d" en 1ère position affichée
NEXT	ouvre la position suivante.
valeur-PREV	charge la valeur tapée dans la position courante et ouvre la position précédente
PREV	ouvre la position précédente
valeur-CLOSE	charge la valeur tapée dans la position courante
CLOSE	ferme la position courante
adresse-GO	démarre l'exécution du programme à l'adresse donnée
GO	re-exécute à partir de la dernière adresse de GO donnée
adresse-IN	lit le périphérique d'adresse donnée
IN	relit la même adresse
valeur-OUT-adresse-OUT	écrit la valeur donnée dans le périphérique d'adresse donné. Lorsque l'adresse est attendue, un "d" est affiché en 1ère position.
valeur-OUT-OUT	écrit la valeur donnée dans la même adresse
adresse-LOAD	charge un programme via la carte série à partir de l'adresse donnée (plus l'adresse de début du programme reçu)
LOAD	charge à partir de l'adresse 0 (plus l'adresse de début de programme reçu)
adresse-SAVE	transfert la mémoire via la carte série à **adresse**
SAVE	transmet toute la mémoire vive via la carte série
INIT	initialise les positions 16'800 à 16'1FFF à zéro
TEST	Programme de test de la mémoire (prévu)
DEMO	Programme de démonstration (prévu)

2.3 Procédures d'utilisation du moniteur

Les programmes sont chargés en mémoire avec le moniteur, à partir de l'adresse 16'800.

2.3.1 Chargement d'un programme manuellement

8 0 0 - OPEN	donne le contenu actuel
byte en 800 - NEXT	écrit le 1er byte du programme
byte en 801 - NEXT	écrit le 2e byte du programme
. . .	
CLOSE	retour au moniteur

2.3.2 Vérification du contenu de la mémoire

8 0 0 - OPEN	affiche le byte en 800
NEXT	affiche le byte en 801
byte en 801 - NEXT	corrige le byte en 801 et affiche le byte en 802
. . .	
CLOSE	retour au moniteur

2.3.3 Démarrage d'un programme

8 0 0 - GO	démarre le programme à l'adresse 800

2.3.4 Chargement d'un programme à partir d'un SMAKY

La procédure à suivre est expliquée complètement dans le chapitre 4.

LOAD	prépare le Dauphin à recevoir

Sur SMAKY : PROGRA T transfert via ligne série
Le programme est chargé à partir de l'adresse 16'800+déplacement (selon le .LOC 16'X contenu dans le programme). Il démarre automatiquement à l'adresse 16'800+Y correspondant à la pseudo-instruction .START Y contenue dans le programme.

adresse -(LOAD)	prépare le Dauphin à recevoir à l'adresse donnée

Sur SMAKY : (PROGRA)(T) transfert via ligne série et démarre à l'adresse donnée+Y si le programme comprend la pseudo-instruction .START Y

Le programme transféré se trouve à l'adresse 16'adresse donnée+déplacement et démarre à cette adresse.

2.3.5 Transfert Dauphin-Smaky

(SAVE)	envoie toute la mémoire vive sur la ligne série

Sur le SMAKY, il faut être dans le programme SMILE et faire (PROGRA)(D)

2.3.6 Arret de l'exécution d'un programme

(RESET)	l'état des registres est sauvé dans les adresses de 16'3A0 (D0) à 16'3DE (A7)

2.4 Les routines

2.4.1 Description

Le moniteur se compose d'un certain nombre de routines, que l'on peut utiliser directement. Il faut insérer au sein du programme une instruction du type **CALL 32^_nom-routine** et au début du programme une pseudo-instruction **.REF M068K**. Cela suppose qu'il existe une table de symbole MO68K.SYM.

Les routines à disposition permettent d'afficher et de transférer:

Routine _AFX8	= 16'2008	routine d'affichage
Routine _INHEX	= 16'200C	lit un nombre et la touche ordre associée
Routine _DISP	= 16'2010	affiche D4.16 selon D2.8 (une fois)
Routine _LED	= 16'2014	affiche D4.16 selon D1.16 (une fois)
Routine _INUSA	= 16'2018	initalise l'usart
Routine _WUSA	= 16'201C	écriture usart
Routine _RUSA	= 16'2020	lecture usart
Routine _LOAD	= 16'2024	chargeur format PDP11 (programme)
Routine _SAVE	= 16'2028	transfert en sortie du Dauphin (programme)

```
Routine _CHGER      = 16'202C   chargeur du format PDP11 (routine)
Routine _SENDBLOC   = 16'2030   envoi d'un bloc en format PDP11
                                (routine)
```

2.4.2 Descriptions des routines

Routine _AFX8 routine d'affichage

```
;Affiche pendant 1 seconde les poids forts (D4:#32..#16)
;Affiche pendant 1 seconde les poids faibles (D4:#15..#0)
in:     D4.32   nombre 32 bits à afficher
out:    -
mod:    -
```

Routine _INHEX lit un nombre et la touche ordre associée

```
in:     D4.32   adresse (16 bits affichés)
        D4.16   ou nombre à afficher
        D2.8    si D2=0..16'7F  affiche D4.16
                si D2=-1  affiche D4.16 (pointeur vers la mémoire)
                si D2=-1 et CTRL  affiche (D4.32).8
                si D2=-2  affiche P suivi de D4.8
                si D2=-2 et CTRL  affiche (D4.32).8
                si D2=-3  affiche = et D4.8
                si D2=autre  affiche ? et D4.8 (erreur)
        D2.8    si D2=autre
out:    D2.32   =0..4 nombre de digits introduits
        D3.32   dernière touche
        D4.32   nombre 16 bits introduit
mod:    F, D2.32, D3.32, D4.32
```

Routine _DISP Affiche D4.16 selon D2.8

```
in:     D4.32   adresse (16 bits affichés)
        D4.16   ou nombre à afficher
        D2.8    si D2=0..16'7F affiche D4.16
                si D2=-1  affiche D4.16 (^mémoire)
                si D2=-1 et CTRL  affiche (D4.32).8
                si D2=-2  affiche P suivi de D4.8
                si D2=-2 et CTRL  affiche (D4.32).8
                si D2=-3  affiche = et D4.8
                si D2=autre  affiche ? et D4.8 (erreur)
        D2.8    si D2=autre
out:    D1.16   registre préparé et appelle de LED pour afficher
mod:    -
```

Routine _LED Affiche le contenu du registre D4.16 selon D1.16

```
in:     D4.16   Contenu à afficher selon D1.16
        D1.16   Si D1=16'0000>affiche les 4 digits hexa de D4.16
                Si D1=16'1100>affiche les 2 digits hexa de D4.8
                Si D1=16'7100>affiche un d puis les 2 digits hexa de D4.8
                Si D1=16'2100>affiche un P puis les 2 digits hexa de D4.8
                Si D1=16'6100>affiche un = puis les 2 digits hexa de D4.8
                Si D1=16'4100>affiche un ? puis un numéro d'erreur
                Si D1=16'3100>affiche un L
out:    -
mod:    -
```

Routine **_INUSA** Initialise l'usart

in: -
out: -
mod: -

Routine **_WUSA** Ecriture sur Usart

in: D3.8 octet à transmettre
out: -
mod: -

Routine **_RUSA** Lecture sur Usart

in: -
out: D3.8 octet lu
mod: D3.8

Routine **_SAVE** Transfert SIMSER en sortie du Dauphin

in: A0.32 pointeur vers la mémoire
D2.32 longueur de données
D3.32 octets lus
D1.16 somme de contrôle
out: -
mod: -

Routine **_CHGER** Chargeur du format PDP11

in: D4.32 offset de chargement
out: D2.32 pour INHEX
D4.32 idem
A0.32 adresse de start si NE
EQ pas de start adresse
NE adresse de start dans A0 et SAVADGO
mod: A0.32, D2.32, D4.32, F

Routine **_SENDBLOC** Envoi d'un bloc en format PDP 11

in: A0.32 ^données à envoyer
D4.16 nombre de données à envoyer
out: A0.32 ^donnée suivant la dernière envoyée
mod: A0.32

2.5 Les erreurs

Lors de l'exécution d'un programme, il se peut que celui-ci ne puisse s'exécuter correctement jusqu'à son terme; dans ce cas, le Dauphin affiche un "?" suivi d'un numéro d'erreur.

Voici liste des erreurs définies :

16'0	POP de trop
16'1	erreur de bus
16'2	adresse impaire
16'3	instruction illégale
16'4	division par zéro
16'5	CHK instruction
16'6	TRAPV instruction
16'7	instruction superviseur
16'8	TRACE
16'A	LINE A émulateur
16'F	LINE F émulateur
16'10	vecteur réservé
16'11	interrupt 1..6
16'12	TRAP 0..15
16'13	interruption vectorisée
16'20	NMI
16'10	Chargeur: pas d'adresse de start
16'31	Chargeur: erreur de chargement
16'35	Chargeur: adresse trop petite
16'FF	ordre inconnu

3 Le codage des instructions

1.85

3.1 Les registres du M68008

Les registres du processeur M68008 sont divisés en 2 groupes, les registres de données D0, D1, D2, D3, D4, D5, D6 et D7 (utilisables en 8,16 ou 32 bits) et les registres d'adresse A0 à A7 (utilisables en 8,16 ou 32 bits également). Le M68000 dispose également d'un registre SF (registre de status) qui contient les fanions (flags) : C(carry), V(overflow arithmétique), Z(égal à zéro), N(signe), X(Carry X), T(trace) et S(superviseur) et d'un registre PC qui permet d'obtenir la position mémoire courante du programme.

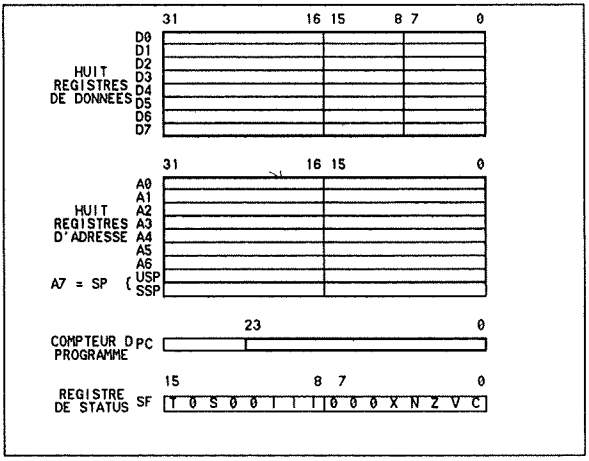


Fig. 3.1 Registres du processeur Motorolla 68000/68008

3.2 Codage des instructions

Pour permettre de réaliser de petits programmes avec le Dauphin, il est nécessaire de connaître la façon dont les instructions sont codées.

Toutes les instructions ont une longueur de 16 bits ou un multiple lorsque les opérandes ont des modes d'adressage complexe.

Une instruction DEC.8 D1 ayant comme but de décrémenter le registre D1.8 se codera 16'53 (pour les 8 bits de poids forts) suivi du code de registre D1 qui correspond ici à son numéro. Pour l'instruction DEC.8, on aura donc l'ensemble suivant de code :

```
53 00    pour DEC.8  D0
53 01    pour DEC.8  D1
53 02    pour DEC.8  D2
53 03    pour DEC.8  D3
53 04    pour DEC.8  D4
53 05    pour DEC.8  D5
53 06    pour DEC.8  D6
53 07    pour DEC.8  D7
```

Une instruction avec adressage direct 16 bits, par exemple **MOVE.16 COUNT,D1**, avec COUNT=16'243 se code (voir section 3.3.1):

code de l'instruction

= 16'243

Cette instruction ne doit pas être confondue avec l'instuction **MOVE.16 #COUNT,D1**(section 3.3.1), qui initialise D1 avec la valeur 16'243 et qui sera codé 32 3C 02 43. Dans le cas de l'instruction **MOVE.16 COUNT,D1** on ne peut connaître le contenu du registre D1 vu que l'instruction d'affectation a comme but de copier le contenu de la position mémoire 243 (8 bits) et 244 (8 bits) dans le registre D1.16.

Puisque les adresses dans lesquelles on mémorise une instruction ont une longueur de 8 bits, le codage de l'instruction **MOVE.16 COUNT,D1** à partir de l'adresse 16'800, occupe les adresses allant de 16'800 à 16'803 (16'800=32, 16'801=38, 16'802=02 et 16'803=43)

3.3 Etude de quelques codes

Les instructions données ci-dessous sont utilisées dans la manipulation DAMU, qui explique un assemblage et un chargement manuel. Les programmes plus complexes utilisé sur le Dauphin sont assemblés sur Smaky 100 avec le programme SMILE (Voir la notice "Introduction au 68000 avec SMILE", LAMI, octobre 1985), puis téléchargés.

3.3.1 Instructions de transfert

Un premier groupe de transfert d'information, d'un registre vers un autre.

MOVE.8	Di,Dj	1000+ii+jj
MOVE.16	Ri,Dj	3000+ii+jj
MOVE.A16	Ri,Aj	
MOVE.32	Ri,RJ	2000+ii+jj

	Ri	ii
Di	D0	0
	D1	1
	...	...
	D7	7
Ai	A0	8
	A1	9
	...	...
	A7	F

	RJ	jj
Dj	D0	0
	D1	200
	...	...
	D7	E00
Aj	A0	40
	A1	240
	...	...
	A7	E40

Par exemple :

MOVE.A16 D3,A2 se code 3000+3+440=3443.

Un deuxième groupe transfère une valeur imédiate donnée dans le programme vers un registre.

MOVE.8	#VAL8,Dj	103C+jj VAL8
MOVE.16	#VAL16,Dj	303C+jj VAL16
MOVE.A16	#VAL16,Aj	
MOVE.32	#VAL32,Rj	203C+jj MSB VAL32 LSB
MOVE.32	#VAL8,Dj	7000+jj+VAL8
MOVE.8	#VAL8,ADR16	11FC VAL8 ADR16

	Rj	jj
Dj	D0	0
	D1	200
	...	...
	D7	E00
Aj	A0	40
	A1	240
	...	...
	A7	E40

MOVE.16 #VAL16,ADR16

31FC
VAL16
ADR16

MOVE.32 #VAL32,ADR16

21FC
MSB VAL32 LSB
ADR16

Note : MSB = Most Significant Bit (8 bits de poids forts)
LSB = Least Significant Bit (8 bits de poids faibles)

Par exemple, MOVE.8 #"A",D3 se code 103C+600=163C pour le 1er mot et 0041 pour le 2ème mot de 16 bits, car le code du caractère ASCII "A" est 16'41 (le code 7 segments est 16'77, voir section 1.5.3)

Un troisième groupe transfère le contenu d'une adresse absolue courte (16 bits) vers un registre.

MOVE.8 ADR16,Dj

MOVE.16 ADR16,Dj
MOVE.A16 ADR16,Aj

MOVE.32 ADR16,Rj

1038+jj
ADR16

3038+jj
ADR16

2038+jj
ADR16

	Rj	jj
Dj	D0	0
	D1	200
	...	...
	D7	E00
Aj	A0	40
	A1	240
	...	...
	A7	E40

Un dernier groupe transfère le contenu d'un registre dans une adresse mémoire absolue courte (16 bits).

MOVE.8 Di,ADR16

MOVE.16 Di,ADR16

MOVE.32 Di,ADR16

11C0+ii
ADR16

32C0+ii
ADR16

21C0+ii
ADR16

	RI	II
Di	D0	0
	D1	1
	...	...
	D7	7
Ai	A0	8
	A1	9
	...	...
	A7	F

De nombreux autres groupes existent.

3.3.2 Les incrémentations et décrémentations

Des instructions souvent utilisées ont comme but d'augmenter ou de diminuer d'une unité le contenu d'un registre.

INC.8	Di	5200+ii
INC.16	Di	5240+ii
INC.32	Di	5280+ii
DEC.8	Di	5300+ii
DEC.16	Di	5340+ii
DEC.32	Di	5380+ii

Di	ii
D0	0
D1	1
...	...
D7	7

3.3.3 Les additions

Les instructions d'addition permettent d'additionner le contenu de deux registres ou d'un registre et une constante. Le résultat est toujours dans le registre qui se trouve à droite.

ADD.8	Di,Dj	D000+ii+jj
ADD.16	Di,Dj	D040+ii+jj
ADD.32	Di,Dj	D080+ii+jj
ADD.8	#VAL8,Di	0600+ii VAL8
ADD.16	#VAL16,Di	0640+ii VAL16
ADD.32	#VAL32,Di	0680+ii MSB VAL32 LSB

Di	ii
D0	0
D1	1
...	...
D7	7

Dj	jj
D0	0
D1	200
...	...
D7	E00

3.3.4 Les soustractions

Les instructions de soustractions permettent de soustraire le contenu de deux registres ou d'un registre et une constante. Le résultat est toujours dans le registre qui se trouve à droite.

SUB.8	DI,DJ	9000+ii+jj
SUB.16	DI,DJ	9040+ii+jj
SUB.32	Di,Dj	9080+ii+jj
SUB.8	#VAL8,Di	0400+ii VAL8
SUB.16	#VAL16,Di	0440+ii VAL16
SUB.32	#VAL32,Di	0480+ii MSB VAL32 LSB

DI	II
D0	0
D1	1
...	...
D7	7

DJ	JJ
D0	0
D1	200
...	...
D7	E00

3.3.5 Les sauts

Pour sauter à une adresse quelconque du programme, il existe des sauts inconditionnels (saute dans tous les cas) ou des sauts soumis à une condition (dans ce cas, on teste l'un des fanions).

JUMP	ADR16	4EF8 ADR16
JUMP	RELAD	6000+DEPL
JUMP,NE	RELAD saut si "non equal" (Z=0)	6600+DEPL
JUMP,EQ	RELAD saut si "equal" (Z=1)	6700+DEPL
JUMP,LO	RELAD saut si "lower" (C=1)	6500+DEPL
JUMP,LS	RELAD saut si "lower or same" (C=1 ou Z=1)	6300+DEPL
JUMP,HS	RELAD saut si "higher or same" (C=0)	6400+DEPL
JUMP,HI	RELAD saut si "higher" (C=0 et Z=0)	6200+DEPL

Le déplacement DEPL se calcule en soustrayant de l'adresse destination RELAD l'adresse x qui suit l'instruction du saut. Cette différence doit être un mot arithmétique de 8 bits, donc compris entre +127 et -128 (en décimal).

On a donc JUMP x+(RELAD-x) où (RELAD-x)=DEPL

3.3.6 Les comparaisons

Les instructions de comparaison documentées ici permettent de comparer le contenu de deux registres ou d'un registre et d'une constante. Le résultat d'une comparaison est de modifier les fanions.

COMP.8	Di,Dj	B000+ii+jj
COMP.16	Di,Dj	B040+ii+jj
COMP.32	Di,Dj	B080+ii+jj
COMP.8	#VAL8,Di	0C00+ii VAL8
COMP.16	#VAL16,Di	0C40+ii VAL16
COMP.32	#VAL32,Di	0C80+ii MSB VAL32 LSB

Di	ii
D0	0
D1	1
...	...
D7	7

Dj	jj
D0	0
D1	200
...	...
D7	E00

4 Téléchargement

1.1

4.1 Introduction

Le téléchargement (ou "down-load") est l'opération de transfert d'un ordinateur vers un autre par ligne série.

Le téléchargement est utilisé pour transférer des programmes du Smaky vers le Dauphin ou transférer le contenu de la mémoire du Dauphin vers le Smaky.

Dans le cadre du laboratoire, il est possible de mettre au point un programme sur SMAKY100 à l'aide de l'éditeur-assembleur SMILE, de connecter un Dauphin 68008 à un Smaky par une liasion série et de transférer l'information dans un sens ou dans l'autre.

4.2 Procédure pour télécharger du Smaky vers le Dauphin

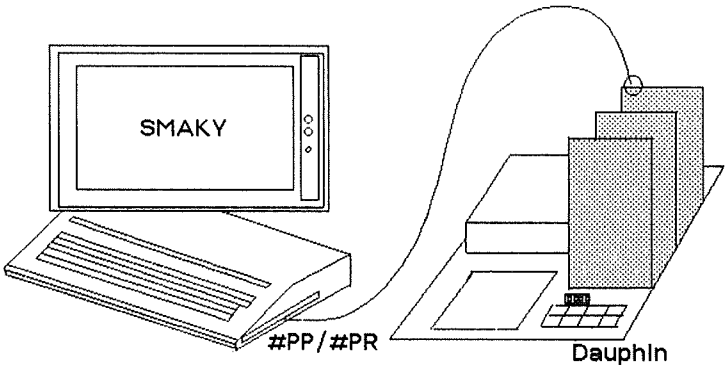
Procédure à suivre pour télécharger un programme :

- relier un Dauphin au Smaky
  - relier la prise #PP/#PR du Smaky 100 à la prise de l'interface série du Dauphin
- chargement du programme sous SMILE
  - taper **SMILE**↵ dans le CLE du Smaky
  - taper votre programme ou charger-le en faisant **SHOW** **DEFINE** **nom de fichier** dans SMILE
  - une fois le programme chargé sur l'écran, assembler et corriger les erreurs éventuelles.

- préparation du Dauphin
  - le commutateur principal du Dauphin est sur μP
  - les commutateurs de la plaque processeur sur ROM, HOLD, RUN
  - l'affichage montre 0000
- téléchargement
  - CTRL SHIFT 4 sur le Dauphin, ce qui équivaut au LOAD
  - PROGRA T sur Smaky

Le programme se charge (sauf ordre supplémentaire) à l'adresse 16'800, l'exécution commence immédiatement si le programme contient un .START ADRDEBUT

Pour arrêter le programme, appuyer sur RESET. Après le RESET, redémarrez le programme avec adresse -GO.



DAUPTE

Fig. 4.1 Connection Smaky-Dauphin par ligne série

4.2.1 Procédure pour télécharger du Dauphin vers le Smaky

Il peut être utile de vider le contenu de la mémoire Dauphin dans l'écran du Smaky.

- relier le Dauphin au Smaky (voir section 4.2)
- charger SMILE sur le Smaky
- téléchargement
  - SAVE sur le Dauphin
  - PROGRA D sur le Smaky

5 Utilisation manuelle

1.1

5.1 Introduction

La section 1.2 présente le panneau de test du Dauphin. Ce panneau permet de prendre le contrôle manuel du Dauphin.

Ce chapitre présente comment utiliser le Dauphin dans ce monde manuel.

5.2 Prise de contrôle du panneau de commande

Le commutateur est sur "PANNEAU" et le processeur sur "HALT". Faire un (RESET) afin de ne pas démarrer le programme moniteur.

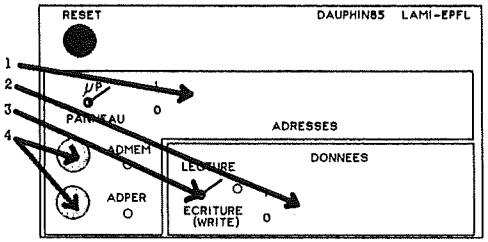
Les opérations que vous pouvez effectuer sont des opérations de lecture ou d'écriture, en mémoire ou sur un périphérique.

La sélection "écriture" ou "lecture" se fait grâce au commutateur situé sur le panneau de commande. L'opération de lecture ou d'écriture se fait en appuyant sur le bouton ADMEM si l'on fait une opération avec la mémoire ou sur ADPER s'il s'agit d'une opération avec un périphérique.

L'écriture a comme but d'écrire une donnée à une adresse. (Par exemple, écrire un mot "2'10000000" à l'adresse 16'00001 aura comme conséquence d'afficher un point décimal sur le deuxième digit). On aura ainsi la possibilité de lire des périphériques (ADPER) ou des adresses en mémoires (ADMEM).

Pour une écriture, les opérations à effectuer sont :

- sélection d'une adresse d'écriture (digits=0 à 3)
- sélection de la donnée
- sélection du mode ECRITURE
- envoi d'une impulsion sur ADPER ou ADMEM (suivant qu'il s'agit d'une écriture sur un périphérique ou d'une écriture en mémoire)

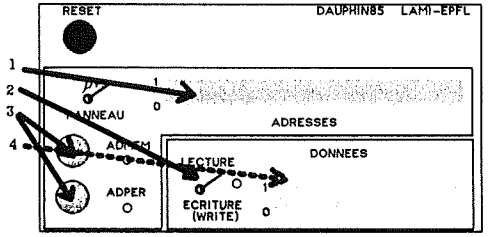


DAUPEC

Fig. 5.1 Opérations à faire pour une écriture

Pour la lecture, les opérations à effectuer sont :

- sélection d'une adresse de lecture (clavier=7)
- sélection du mode LECTURE
- envoit d'une impulsion sur ADPER ou ADMEM (suivant qu'il s'agit d'un périphérique ou d'une adresse mémoire)
- RESULTAT: valeur des données sur les affichages



DAUPLE

Fig. 5.2 Opérations à faire pour une lecture

6 Programme moniteur

```
13   000000                    .TITLE    DAUMON.ASM
14   000000                    .PROC     M68000
15   000000
22   000000
```

Moniteur pour Dauphin 68008

```
23   000000
```

Listage du programme en EPROM

```
24   000000
25   000000          Le programme initialise les vecteurs d'interruption en RAM.
26   000000          Puis il attend sur le clavier et interprète les ordres.
27   000000          Les adresses ne peuvent être données que sur 16 bits.
28   000000          Seulement 14 bits sont câblés sur le bus.
29   000000          Les données sont 8 bits seulement.
30   000000
32   000000          Ordres reconnus (m,m' 16 bits, n 8 bits, [optionnel]):
33   000000
```

CTRL	GO		IN	OUT
SHIFT	OPEN	NEXT	PREV	CLOSE

IMON1

Fig. 6.1 Ordres en CTRL

```
34   000000
35   000000          m OPEN          Ouvre la position m
36   000000          OPEN            Ouvre la dernière position ouverte
37   000000          [n] NEXT        [Charge n et] ouvre la pos suivante
38   000000          [n] PREV        [Charge n et] ouvre la position précédente
39   000000          [n] CLOSE       [Charge n et] ferme
40   000000          m GO            Exécute à partir de m
41   000000          GO              Ré-exécute à partir de la dernière valeur
42   000000          n IN            Lit Mubus à l'adresse n
43   000000          IN              Relit la même adresse
44   000000          n OUT p OUT     Ecrit la valeur n à l'adresse p et sauve ces deux valeurs
45   000000          n OUT OUT       Ecrit la valeur n à la même adresse
46   000000          OUT             Ecrit la valeur suivante sur la même adresse
47
48   000000
```

CTRL	LOAD	SAVE		
SHIFT			TEST	DEMO

DAUMON2

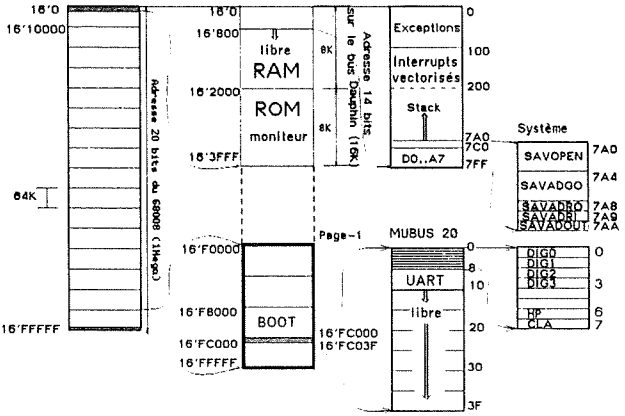
Fig. 6.2 Ordres en SHIFT-CTRL

```
49
50  000000          [m] LOAD          Charge à partir de l'adresse m via Simser (adresse Mubus+10)
51  000000          LOAD              Charge à partir de 0 (+ adresse de début de bande)
52  000000          [m] SAVE [m']     "Perfore" ou transmets de m à m'
53  000000          SAVE              Transmets toute la mémoire vive (2k)
54  000000          TEST              Programme de test: à faire ...
55  000000          DEMO              Petite Demo : à faire ...
56
57  000000          Les autres ordres sont libres
58
59  000000
61  000000          Les erreurs sont rendues avec un ? suivi d'un numéro
62  000000
63  000000          0  POP de trop          1  Erreur de bus
64  000000          2  Adresse impaire      3  Instruction illégale
65  000000          4  Div par Zero         5  CHK instruction
66  000000          6  TRAPV instr          7  Instr superviseur
67  000000          8  TRACE                9
68  000000          A  LINE A               F  LINE F
69  000000          10 Vect réservé         11 Interrupt 1..6
70  000000          12 TRAP 0..15           13 Interrupt vectorisés
71  000000          20 NMI                  30 Pas d'adresse de start
72  000000          31 Erreur chargement    32 Adresse trop petite
73  000000          FF Ordre inconnu
74  000000
75  000000          Définition des PC
76  000000
77  000000  00000000  PC_CODE   =  0        Pour zone contenant le code
78  000000  00000001  PC_VAR    =  1        Pour zone contenant les variables
79  000000
81  000000
```

```
82
83   000000
```

Constantes | Mémoire et I/O | Définitions des adresses principales

```
84
85   000000
```

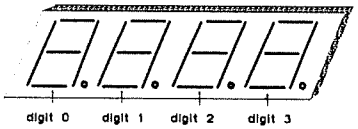


JMON3

Fig. 6.3 Plan mémoire

```
86
87   000000                Les Constantes
88   000000
89   000000   00002000   ROM      =   16'2000          Adresse physique de la ROM
90   000000   00000000   RAM      =   0               Adresse physique de la RAM
91
92   000000   00002000   LRAM     =   16'2000          Longueur RAM
93   000000   00000800   LRAMSYS  =   16'800           Longueur RAM pour système (pile comprise)
94   000000   00001800   LRAMUSE  =   LRAM-LRAMSYS     Longueur RAM pour l'utilisateur
95
96   000000   00000000   DRAM     =   RAM              Début RAM
97   000000   00000000   DRAMSYS  =   RAM              Début RAM système
98   000000   00000800   DRAMUSE  =   RAM+LRAMSYS      Début RAM système
99   000000   00002000   FRAM     =   DRAM+LRAM        Fin de la RAM
100  000000   00000800   FRAMSYS  =   DRAMSYS+LRAMSYS  Adresse fin RAM système (non incluse)
101  000000   00002000   FRAMUSE  =   DRAMUSE+LRAMUSE  Adresse fin RAM utilisateur (non incluse)
102  000000   00000040   LSAVREG  =   10'8*10'2*10'4   Longueur nécessaire pour sauver tous les registres
103
104  000000   00002000   MON      =   ROM              Adresse de la rom
105  000000   00000000    PAGEMON =   MON/16'10000     Pour permettre des tables 16 bits
106
107  000000                Les erreurs
108  000000
109  000000   00000000   ERPOP    =   16'0    POP de trop
110  000000   00000001   ERBUS    =   16'1    Erreur de bus
111  000000   00000002   ERADR    =   16'2    Adresse impaire
112  000000   00000003   ERINSIL  =   16'3    Instruction illégale
113  000000   00000004   ERDIVZ   =   16'4    Division par zero
114  000000   00000005   ERCHK    =   16'5    CHK instruction
115  000000   00000006   ERTRAPV  =   16'6    TRAPV instruction
116  000000   00000007   ERINSSU  =   16'7    Instruction superviseur
117  000000   00000008   ERTRAC   =   16'8    TRACE
118  000000   0000000A   ERLINA   =   16'A    LINE A emulateur
119  000000   0000000F   ERLINF   =   16'F    LINE F emulateur
120  000000   00000010   ERVECR   =   16'10   Vecteur réservé
121  000000   00000011   ERINT    =   16'11   Interrupt 1..6
122  000000   00000012   ERTRAP   =   16'12   TRAP 0..15
123  000000   00000013   ERINTV   =   16'13   Interrupt vectorisée
```

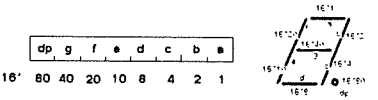
```
124  000000  00000020  ERNMI    =  16'20   NMI
125  000000  00000030  ERNOSTAD =  16'30   Loader: pas d'adresse de start
126  000000  00000031  ERCHAR   =  16'31   Loader: erreur de chargement
127  000000  00000035  ERADLOW  =  16'35   Loader: adresse trop petite
128  000000  000000FF  ERORDIN  =  16'FF   Ordre inconnu
129
130  000000             Adresses des périphériques
131
132  000000
133  000000  FFFF8000  BOOT     =  -16'8000          Commutation de la rom
134  000000  FFFFC000  HUBUS    =  -16'4000          Hubus et adresse de base des périphériques
135
136  000000             Adresses affichage
137  000000             Attention: les chiffres sont adressés de gauche à droite
138
139  000000
```



DAUHON4

Fig. 6.4 Position des digits

```
140
141  000000  FFFFC000  DIG0     =  HUBUS
142  000000  FFFFC001  DIG1     =  DIG0+1
143  000000  FFFFC002  DIG2     =  DIG1+1
144  000000  FFFFC003  DIG3     =  DIG2+1
145  000000  00000004  NBDIG    =  4                 Nombre de digits
146  000000             Codes 7segments
147
148  000000
```

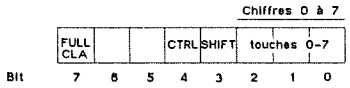


DAUHON5

Fig. 6.5 Codage des segments

```
149  000000
150  000000  0000003F  C0       =  16'3F
151  000000  00000006  C1       =  16'6
152  000000  0000005B  C2       =  16'5B
153  000000  0000004F  C3       =  16'4F
154  000000  00000066  C4       =  16'66
155  000000  0000006D  C5       =  16'6D
156  000000  0000007D  C6       =  16'7D
157  000000  00000007  C7       =  16'7
158  000000  0000007F  C8       =  16'7F
159  000000  0000006F  C9       =  16'6F
160  000000  00000077  CA       =  16'77
161  000000  0000007C  CB       =  16'7C
162  000000  00000058  CC       =  16'58
163  000000  0000005E  CD       =  16'5E
164  000000  00000079  CE       =  16'79
165  000000  00000071  CF       =  16'71
166
167  000000  00000038  CL       =  16'38
168  000000  00000073  CP       =  16'73
169
170  000000  00000040  MOINS    =  16'40
171  000000  00000048  EGAL     =  16'48
172  000000  00000053  QUEST    =  16'53
173  000000  00000080  DOT      =  16'80
174  000000
```

```
175  000000                 Adresses clavier et Haut-Parleur
176
177  000000   FFFFC006  HP       =   HUBUS+6   Accès HP = accès LED rouge
178  000000               Chaque fois que l'on presse sur une touche 0..7 le bit BFULCLA s'active.
179  000000               Les touches SHIFT et CTRL n'activent PAS le bit BFULCLA lorsqu'elles sont ap
180  000000
181
182  000000
```



UMONG

Fig. 6.6 Mot lu du clavier

```
183  000000
184  000000   FFFFC007  CLA      =   HUBUS+7          Adresse du Clavier
185  000000   00000007   BFULCLA  =   7               Bit touche en attente
186  000000   00000080   FULCLA   =   2**BFULCLA
187  000000   00000010   MDIG     =   16'10           Masque des touches chiffre
188  000000   0000001F   MCLA     =   16'1F           Masque touches significatives CTRL+shift+chiffres
189  000000   0000000F   MCHIFFR  =   16'F            Masque shift et chiffres
190  000000   00000003   BSHIFT   =   3               Bit touche SHIFT
191  000000   00000004   BCTRL    =   4               Bit touche CTRL
192
193  000000              Adresses USART
194  000000              L'USART employé sur le dauphin est un 8251
195  000000   FFFFC008  USA      =   HUBUS+16'8       Données
196  000000   FFFFC009  SUSA     =   USA+1            Status
197  000000   00000000   BRDYS    =   0               Bit prêt à envoyer
198  000000   00000001   BFULL    =   1               Bit un caractère est reçu
199
200  000000   00000002   DUMMY    =   16'2            Rien de spécial en mode et en commande
201  000000   000000CE   MODUSA   =   16'CE           2 stops bits
202  000000                                           Pas de parité
203  000000                                           8 bits
204  000000                                           Baud rate 16x clock
205  000000   00000015   ENAUSA   =   16'15           Transmit enable
206  000000                                           Receive enable
207  000000                                           DTR inactif
208  000000                                           Reset erreurs
209  000000                                           RTS inactif
210  000000   00000040   RSTUSA   =   16'40           Reset soft USART
211
212  000000
220  000000
223  000000
```

Variables **Les variables** **Réservation de la place mémoire pour les variables systèmes**

```
224
225  000000   00000001            .APC                PC_VAR
226  000000   00000000            .LOC                DRAMSYS
227  000000
228  000000   00000100  PNTINT:     .BLK.8   16'100    Réservé interruptions
229  000100   00000100  PNTINTV:    .BLK.8   16'100    Reserve interruptions vectorisées
230  000200            MAXPILE:                        Adresse "max" de la pile
231
232  0007A0   000007A0            .LOC     FRAMSYS-16'60
233
234  0007A0            PILE:                           "Bas" de la pile
235  0007A0   000005A0   LGPILE   =        PILE-MAXPILE Taille de la pile
236  0007A0   00000001  SAVOPEN:    .BLK.32  1         "RAM lors de l'open. next. prev. close
237  0007A4   00000001  SAVADGO:    .BLK.32  1         Adresse de démarrage
238  0007A8   00000001  SAVADRO:    .BLK.8   1         Adresse périphérique OUT (8 bits)
239  0007A9   00000001  SAVADRI:    .BLK.8   1         Adresse périphérique IN (8 bits)
```

```
240  0007AA  00000001  SAVDAOUT:  .BLK.8   1          Donnée envoyée sur pér. OUT (8 bits)
241  0007AC  00000001  .EVEN
245  0007AC

246  0007C0  000007C0            .LOC      FRAMSYS-LSAVREG  Sauve les registres au Reset
247
248  0007C0  00000040  SAVREG:    .BLK.8    LSAVREG    Sauvetage des registres
249
250  000800
251  000800
```

Constantes en début de ROM

```
252  000800
253
254  000800            Au démarrage seule la ROM est décodée en lecture.
255  000800            Elle se trouve dédoublée entre 16'0000-16'1FFF et entre 16'2000 et 16'3FFF.
256  000800            C'est à dire que tout accès s'effectura en EPROM lors du Reset.
257  000800            Ceci tant que le périphérique BOOT ne sera pas accédé
258
259  000800            A partir de ce moment les adresses déclarées précédemment seront valides.
260  000800
261  000000  00000000            .APC      PC_CODE
262  002000  00002000            .LOC      MON                Celle adresse est dédoublée en zero entre le RESET et l'accès au
263
264  002000 000007A0             .32       PILE               Adresse du Stack Superviseur au Reset
265  002004 0000204C             .32       DEBUTMON           Adresse de démarrage au Reset
266
267  002008            BASEIND:                               Adresse de base des indirections des appels systèmes
268  002008 60000462   _AFX8:    JUMP      R16^AFX8
269  00200C 60000358   _INHEX:   JUMP      R16^INHEX
270  002010 600003A0   _DISP:    JUMP      R16^DISP
271  002014 60000408   _LED:     JUMP      R16^LED
272  002018 6000032C   _INUSA:   JUMP      R16^INUSA
273  00201C 60000304   _WUSA:    JUMP      R16^WUSA
274  002020 60000312   _RUSA:    JUMP      R16^RUSA
275  002024 600001AA   _LOAD:    JUMP      R16^LOAD
276  002028 600001BC   _SAVE:    JUMP      R16^SAVE
277  00202C 600001E2   _CHGER:   JUMP      R16^CHGER
278  002030 60000282   _SENDBLOC: JUMP     R16^SENDBLOC
279
280  002034  00002034  PCC       =         APC                PC lors de l'assemblage
281
282  002034 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                                 .FILL.32  BASEIND+10'50-PCC,-16'1
283  00204C                      Laisse de la place pour appels futurs
284
285  00204C
286  00204C
```

Structure générale

```
287  00204C
288  00204C
289  00204C            Initialisation      (pile, vecteurs)
290  00204C            Attente clavier     Affiche en attendant un résultat precedent eventuel (adresse cont
291  00204C            Sort de l'attente dès qu'une touche ordre est pressée
292  00204C            Si [OPEN]           Ouvrir SAVOPEN
293  00204C            Si m [OPEN]         Ouvrir m et mettre a jour SAVOPEN
294  00204C            Si [NEXT]           Incrémenter SAVOPEN et ouvrir
295  00204C            Si n [NEXT]         Charger n dans (SAVOPEN+) et ouvrir
296  00204C            Si [PREV]           Décrémenter SAVOPEN et ouvrir
297  00204C            Si n [PREV]         Charger n dans (SAVOPEN-) et ouvrir
298  00204C            Si [CLOSE]          Ne rien faire
299  00204C            Si n [CLOSE]        Charger n dans (SAVOPEN)
300  00204C            Si [GO]             Exécuter à partir de SAVLGO (16'400 après un reset)
301  00204C            Si m [GO]           Sauver m dans SAVLGO et exécuter
```

```
00204C              SI [LOAD]          Charger à partir de DRAHUSE (tous les programmes ont un .LOC 0)
00204C              SI m [LOAD]        Charger en ajoutant m à toutes les adresses
00204C              SI [IN]            Lit le périphérique d'adresse HUBUS+(SAVADRI)
00204C              SI p [IN]          Lit le périphérique d'adresse p et sauve dans SAVADRI
00204C              SI [OUT] [OUT]     Ecrit (SAVDAOUT+) à l'adresse HUBUS+(SAVADRO)
00204C              SI [OUT] p [OUT]   Ecrit (SAVDAOUT+) à l'adresse p
00204C              SI n [OUT] [OUT]   Ecrit n à l'adresse HUBUS+(SAVADRO)
00204C              SI n [OUT] p [OUT] Ecrit n à l'adresse HUBUS+p et sauve p dans SAVADRO
00204C
00204C
00204C
```

Programme MONITEUR — Programme moniteur

```
00204C              Commutation immédiate des adresses EPROM, pour les mettre à la bonne place
00204C              Sauvetage des registres (pour observation par le moniteur)
00204C              Initialisation de la mémoire système (sauf la zone contenant les registres sauv
00204C              Initialisation des pointeurs aux routines d'interruptions
00204C              Initialisation de l'usart 8251
00204C              Boucle dans le moniteur
00204C

00204C              DEBUTMON:
00204C 11C08000             MOVE.B    D0,BOOT             Commute la rom
002050 48F8FFFF07C0         MOVEM.32  D0..A7,SAVREG       Sauve tous les registres a partir de 16^3C0
002056

002056                      Effacement de la pile et de la mémoire SYSTEME (sauf sauvetage
002056 303C03DF             MOVE.16   #(LRAMSYS-LSAVREG)/2-1,D0
00205A                      Taille à initialiser
00205A 43F80000             MOVE.32   #DRAMSYS,A1         Adresse début RAM système
00205E              L1$:
00205E 4259                 CLR.16    (A1+)               Mise à zéro
002060 51C8FFFC             DJ.16,NHO D0,L1$
002064 61000454             CALL      INIVECT             Initialise tables des vecteurs d'interruptions
002068 610002DC             CALL      INUSA               Init USART
00206C

00206C                      Attend les ordres du clavier et va les exécuter

00206C              REHON:
00206C 4284                 CLR.32    D4                  Affichage initial
00206E 4282                 CLR.32    D2                  Nombre de digits
002070              REHON1:
002070 2E7C000007A0         MOVE.32   #PILE,SP            Remet le ^pile à sa base
002076 610002EE             CALL      INHEX               D2 <- nombre de byte valide dans D4
00207A                                                    D3 <- numéro dernière touche avec CTRL
00207A                                                    D4 <- nombre tappé
00207A E548                 SL.16     #2,D3               *4 pour sauter dans les JUMP R16^
00207C 4A02                 TEST.8    D2                  Prépare pour tous les ordres
00207E                                                    Y a-t-il eu un chiffre tappé?
00207E 4EBB3004             CALL      R8^TAORDRES+A16^(D3)      Saut à la routine moniteur
002082 60EC                 JUMP      REMON1
002084

002084                      Table des indirections
002084                      chaque JUMP R16^ prend 4 bytes de code

002084              TAORDRES:
002084 6000003E             JUMP      R16^OPEN            Ouvre une position memoire
002088 60000046             JUMP      R16^NEXT            Va voir la suivante
00208C 60000054             JUMP      R16^PREV            Va voir la precédente
002090 60000062             JUMP      R16^CLOSE           Ferme la position mémoire courante
002094 60000072             JUMP      R16^GO              Demarre un programme
002098 60000128             JUMP      R16^NO              Libre
00209C 60000078             JUMP      R16^IN              Lecture sur périphérique
0020A0 60000092             JUMP      R16^OUT             Envoi sur périphérique
0020A4 6000011C             JUMP      R16^NO              Libre
0020A8 60000118             JUMP      R16^NO              Libre
```

```
365   0020AC 600000C8          JUHP     R16^TESTRAM     Programme de test (à faire)
366   0020B0 60000508          JUHP     R16^DEMO        Programme de démo (à faire)
367   0020B4 6000011A          JUHP     R16^LOAD        Chargement d'un binaire par simser
368   0020B8 6000012C          JUHP     R16^SAVE        Sauvetage d'un binaire par simser
369   0020BC 60000104          JUHP     R16^NO          Libre
370   0020C0 600000EC          JUHP     R16^INITMEM     Initialise RAM utilisateur
371
372
373   0020C4
```

Routines des ordres du moniteur à exécuter

```
374   0020C4
375   0020C4          Un TEST.8 D2 à été effectué avant de sauter à ces appels
376   0020C4          Pour toutes les routines d'ordres qui suivent:
377
378   0020C4             In:   D4.16   Nombre tappé
379   0020C4                   D3.16   numéro de l'ordre
380   0020C4                   D2.16   nombre de caractères tappés 0..4
381
382   0020C4
```

Routine OPEN

```
383   0020C4
384   0020C4
385   0020C4          m OPEN       m -> SAVOPEN
386   0020C4          OPEN         SAVOPEN -> affichage
387   0020C4
388   0020C4
389
390   0020C4          OPEN:
391   0020C4 67000006           JUHP,EQ   L10$
392   0020C8 21C407A0           MOVE.32   D4,SAVOPEN
393   0020CC           L10$:
394   0020CC 60000030           JUHP      AFADCOUR        autre possibilité: CALL AFADCOUR / RET
395
396   0020D0
```

Routine NEXT

```
397   0020D0
398   0020D0
399   0020D0          n NEXT       n -> (SAVOPEN+), SAVOPEN -> affichage
400   0020D0          NEXT         Incrémente SAVOPEN, SAVOPEN -> affichage
401   0020D0
402   0020D0
403
404   0020D0          NEXT:
405   0020D0 67000008           JUHP,EQ   L2$
406   0020D4 2B7807A0           MOVE.32   SAVOPEN,A4
407   0020D8 1884               MOVE.8    D4,(A4)
408   0020DA            L2$:
409   0020DA 52B807A0           INC.32    SAVOPEN
410   0020DE 6000001E           JUHP      AFADCOUR
411
412   0020E2
```

Routine PREV

```
413   0020E2
414   0020E2
415   0020E2          n PREV       n -> (SAVOPEN-),  SAVOPEN -> affichage
416   0020E2          PREV         Décremente SAVOPEN,  SAVOPEN -> affichage
417   0020E2
418   0020E2
419
420   0020E2          PREV:
421   0020E2 67000008           JUHP,EQ   L2$
```

```
422   0020E6 287807A0            MOVE.32   SAVOPEN,A4
423   0020EA 1884                MOVE.8    D4,(A4)
424   0020EC               L2$:
425   0020EC 53B807A0            DEC.32    SAVOPEN
426   0020F0 6000000C            JUMP      AFADCOUR
427
428   0020F4
```

Routine CLOSE

```
429   0020F4
430   0020F4
431   0020F4               n CLOSE       n -> (SAVOPEN), SAVOPEN -> affichage
432   0020F4               CLOSE         SAVOPEN -> affichage
433   0020F4
434   0020F4
435
436   0020F4               CLOSE:
437   0020F4 67000008            JUMP,EQ   AFADCOUR
438   0020F8 287807A0            MOVE.32   SAVOPEN,A4
439   0020FC 1884                MOVE.8    D4,(A4)
440   0020FE               ;     JUMP      AFADCOUR
441
442   0020FE
```

Routine AFADCOUR

```
443   0020FE               Prépare l'adresse courante, et le mode de visualisation
444   0020FE               L'affichage s'effectue en continu dans REMON
445   0020FE                  out:  D4.32   adresse courante
446   0020FE                        D2.16   mode
447   0020FE                  mod:  D4.32,  D2.16
448
449   0020FE               AFADCOUR:
450   0020FE 283807A0            MOVE.32   SAVOPEN,D4
451   002102 343CFFFF            MOVE.16   #-1,D2          Affiche l'adresse
452   002106 4E75                RET
453
454   002108
```

Routine GO

```
455   002108
456   002108
457   002108               m GO          m -> SAVADGO,  Goto (SAVADGO)
458   002108               GO            Goto (SAVADGO)
459   002108
460   002108
461
462   002108               GO:
463   002108 67000006            JUMP,EQ   L2$
464   00210C 21C407A4            MOVE.32   D4,SAVADGO
465   002110               L2$:
466   002110 287807A4            MOVE.32   SAVADGO,A4
467   002114 4ED4                JUMP      (A4)
468
469   002116
```

Routine IN

```
470   002116
471   002116
472   002116               p IN          $(p) -> affichage, p -> SAVADRI
473   002116               IN            $(SAVADRI) -> affichage
474   002116
475   002116
476
477   002116               IN:
478   002116 67000006            JUMP,EQ   L2$
```

```
479  00211A 11C407A9            MOVE.B   D4,SAVADRI
480  00211E              L2$:
481  00211E 183807A9            MOVE.B   SAVADRI,D4
482  002122 0284000000FF        AND.W    #$FF,D4          adresse périphérique sur 8 bits
483  002128 0684FFFFC000        ADD.W    #HUBUS,D4        décodée à partir de HUBUS
484  00212E                                               adresse à afficher
485  00212E                                               dans INHEX affichera le contenu si CTRL pressé
486  00212E 343CFFFE            MOVE.W   #-2,D2           pour INHEX
487  002132 4E75                RTS
488
489  002134
```

Routine OUT

```
490  002134
491  002134
492  002134                     n OUT p OUT    n -> $(p)
493  002134                                    n -> SAVDAOUT, p -> SAVADRO
494  002134                     n OUT OUT      n -> $(SAVADRO)
495  002134                                    n -> SAVDAOUT
496  002134                     OUT p OUT      (SAVDAOUT+) -> $(p)
497  002134                                    p -> SAVADRO
498  002134                     OUT OUT        (SAVDAOUT+) -> $(SAVADRO)
499
500  002134                     Après le 1er et le 2e OUT, SAVADRO -> affichage
501  002134
502  002134
503
504  002134                     OUT:
505  002134 6600000A                    JUMP,NE  L1$
506  002138 523807AA                    INC.B    SAVDAOUT        Pas de data
507  00213C 183807AA                    MOVE.B   SAVDAOUT,D4
508  002140                       L1$:
509  002140 11C407AA                    MOVE.B   D4,SAVDAOUT
510  002144 183807A8                    MOVE.B   SAVADRO,D4
511  002148 343CFFFD                    MOVE.W   #-3,D2
512
513  00214C 61000218                    CALL     INHEX           Attente adresse
514  002150 4A02                        TEST.B   D2              Chiffres tappé?
515  002152 67000006                    JUMP,EQ  L4$
516  002156 11C407A8                    MOVE.B   D4,SAVADRO
517  00215A                       L4$:
518  00215A 183807A8                    MOVE.B   SAVADRO,D4
519  00215E 0284000000FF                AND.W    #$FF,D4         On ne peut pas masquer dans un registre d'adresse!
520  002164 0684FFFFC000                ADD.W    #HUBUS,D4       Add adresse base HUBUS
521  00216A 2844                        MOVE.W   D4,A4
522  00216C 18B807AA                    MOVE.B   SAVDAOUT,(A4)   Copie data dans périphérique
523  002170 343CFFFE                    MOVE.W   #-2,D2          Pour INHEX
524  002174 4E75                        RTS
525
526  002176
```

Routine TESTRAM — Test de la RAM

```
527  002176              Petit programme de test de la mémoire
528  002176              Ecrit un motif dans la mémoire, le relit et le compare
529  002176              Si KO, affiche l'adresse et le contenu
530
531  002176              TESTRAM:
532  002176 41F80000            MOVE.W   #RAM.A0
533  00217A 2808                MOVE.W   A0.D4
534  00217C 610002EE            CALL     AFX8
535  002180              L0$:
536  002180 20BC12345678        MOVE.W   #$12345678,(A0)
537  002186              L1$:
538  002186 0C9012345678        COMP.W   #$12345678,(A0)
539  00218C 66000012            JUMP,NE  L10$
540  002190              L11$:
```

```
541   002190 5888                     ADD.32    #4,A0
542   002192 203C00002000             MOVE.32   #LRAM,D0
543   002198 B1C0                     CONP.32   D0,A0
544   00219A 6700FEB0                 JUMP,EQ   DEBUTMON
545   00219E 60E0                     JUMP      L0$
546   0021A0                L10$:
547   0021A0 2808                     MOVE.32   A0,D4
548   0021A2 610002C8                 CALL      AFX8
549
550   0021A6 2810                     MOVE.32   (A0),D4
551   0021A8 610002C2                 CALL      AFX8
552   0021AC 60E2                     JUMP      L11$
553
554   0021AE
```

Routine **INITMEM** Initialise toute la RAM utilisateur à 0

```
555
556   0021AE                INITMEM:
557   0021AE 303C0BFF                 MOVE.16   #LRAMUSE/2-1,D0      Taille mémoire
558   0021B2 207C00000800             MOVE.32   #DRAMUSE,A0          Début de la mémoire
559   0021B8                L1$:
560   0021B8 30FC0000                 MOVE.16   #0,(A0+)             Mise à zéro
561   0021BC 51C8FFFA                 DJ.16,NMO D0,L1$
562   0021C0 4E75                     RET
563
564   0021C2
```

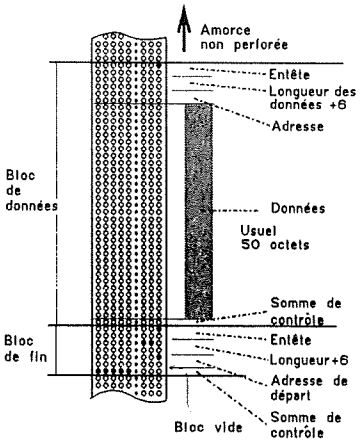
Routine **NO** Numéro d'ordre inexistant

```
565
566   0021C2                NO:
567   0021C2 20380003                 MOVE.32   3.D0
568   0021C6 343CFFFC                 MOVE.16   #-4,D2
569   0021CA 183C00FF                 MOVE.8    #16'FF,D4           No d'erreur à revoir ?
570   0021CE 4E75                     RET
571
572   0021D0
```

```
573   0021D0
```

Routine **LOAD**

```
574   0021D0
575   0021D0              a LOAD        Charge depuis l'adresse a (ajoute a à l'adresse en début d
576   0021D0              LOAD   Charge en DRAMUSE = 16'800
577   0021D0
578
579   0021D0
```



DAUHOH7

Fig. 6.7 Format PDP11

```
580
581   0021D0              LOAD:
582   0021D0 66000008               JUMP,NE   L1$
583   0021D4 283C00000800           MOVE.32   #DRAMUSE,D4          Offset de chargement
584   0021DA               L1$:
585   0021DA 61000034               CALL      CHGER                Charge les données (le code)
586   0021DE 67000004               JUMP,EQ   NOSTART              = 1, il faudra faire un GO
587   0021E2 4E90                   CALL      (A0)                 A0 adr start
588
589   0021E4              NOSTART:
590   0021E4 4E75                   RET
591
592   0021E6
```

Routine **SAVE** **Transfert SIMSER en sortie du Dauphin**

```
593   0021E6              Envoie toute la mémoire vive (8 k, de 0 à 16'2000) par blocs de 64 octets sel
594   0021E6
595   0021E6   00000080   LSAVBLOC  = 10'128
596   0021E6
597   0021E6
598   0021E6              SAVE          Transfert toute la mémoire
599   0021E6
600   0021E6
601
602   0021E6              Utilisation des registres:
603   0021E6                D1 somme de contrôle
604   0021E6                D2 longueur des données
605   0021E6                D3 octet lu
606   0021E6                A0 pointeur mémoire
607
608   0021E6              SAVE:
609   0021E6 48E7F080               PUSHM.32  D0..D3|A0
610   0021EA 41F80000               MOVE.32   #0,A0                 initial
```

```
611   0021EE 383C0080            MOVE.16   #LSAVBLOC,D4          Longueur à sauver
612   0021F2 303C003F            MOVE.16   #FRAM/LSAVBLOC-1,D0
613   0021F6                     Nombre de blocs à transmettre
614   0021F6               L10$:
615   0021F6 610000BC            CALL      SENDBLOC              Envoi d'un bloc
616   0021FA 51C8FFFA            DJ.16,NH0 D0,L10$               Et on continue
617
618   0021FE               Envoi dernier bloc
619
620   0021FE 41F80001            MOVE.32   #1,A0                pas d'adresse à donner
621   002202 383C0000            MOVE.16   #0,D4                 Longueur à sauver
622   002206 610000AC            CALL      SENDBLOC
623
624   00220A 4CDF010F            POPM.32   D0..D3|A0
625   00220E 4E75                RET
626
627   002210
```

Routines utilitaires du loader

```
628   002210
629
630   002210
```

Routine CHGER — Chargeur du format PDP11

```
631   002210                     in:   D4.32   offset de chargement
632   002210                     out:  D2.32   pour INHEX
633   002210                           D4.32   idem
634   002210                           A0.32   adresse de start si NE
635   002210                           EQ      Pas de start adresse
636   002210                           NE      Adresse de start dans A0 et SAVADGO
637   002210                     mod:  A0.32, F
638
639   002210               Utilisation des registres:
640   002210                D1 somme de contrôle
641   002210                D2 longueur des données
642   002210                D3 octet lu
643   002210                A0 pointeur mémoire
644   002210                D4 offset
645
646   002210               CHGER:
647   002210 48E75000            PUSHM.32  D1|D3
648   002214               L1$:
649   002214 4281                CLR.32    D1                    Init cheksum
650   002216 61000102            CALL      RBYTE                 Attente entête
651   00221A 4A03                TEST.8    D3
652   00221C 67F6                JUMP,EQ   L1$
653   00221E
654   00221E 5303                DEC.8     D3                    1er car = 1
655   002220 66F2                JUMP,NE   L1$
656   002222
657   002222 610000F6            CALL      RBYTE
658   002226 4A03                TEST.8    D3                    2e car = 0
659   002228 66EA                JUMP,NE   L1$
660   00222A
661   00222A 4283                CLR.32    D3
662   00222C 610000EC            CALL      RBYTE                 Longueur du bloc (poids faibles)
663   002230 E14B                SL.16     #10'8,D3
664   002232 610000E6            CALL      RBYTE                 Poids forts
665   002236 E15B                RL.16     #10'8,D3
666   002238 5D43                SUB.16    #6,D3                 Longueur des données corrigée
667   00223A 3403                MOVE.16   D3,D2                 D2.16 <- Nombre de caracteres a lire
668   00223C
669   00223C 610000DC            CALL      RBYTE                 Poids faible de l'adresse
670   002240 E14B                SL.16     #10'8,D3              Les mets aux bits 15..8
671   002242 610000D6            CALL      RBYTE                 Poids fort de l'adresse
```

```
672  002246 E15B                RL.16     #10'8,D3          Remet en place
673
674  002248 2043                MOVE.32   D3,A0             où mettre les données
675  00224A D1C4                ADD.32    D4,A0             + correction éventuelle
676
677  00224C           Un affichage de contrôle serait bien, mais l'affichage Dauphin est trop lent
678  00224C
679  00224C 4A42                TEST.16   D2                Bloc de fin? (longueur nulle)
680  00224E 67000020            JUMP,EQ   L4$
681
682  002252 B1FC00000800        COMP.32   #FRAMSYS,A0       Chargement interdit par dessus les vecteurs et  variables
683  002258 6500004C            JUMP,LO   ERLDLOW
684
685  00225C           Lecture d'un bloc
686
687  00225C           L2$:
688  00225C 610000BC            CALL      RBYTE
689  002260 10C3                MOVE.8    D3,(A0+)          Charge en mémoire
690  002262 5342                DEC.16    D2                Décompte nombre bytes à lire
691  002264 66F6                JUMP,NE   L2$
692  002266
693  002266 610000B2            CALL      RBYTE             Somme de contrôle nulle?
694  00226A 67A8                JUMP,EQ   L1$               OK. on lit le bloc suivant
695  00226C 60000028            JUMP      LDERR1
696
697  002270           Bloc de fin, start automatique si adresse ≠ 1
698  002270           L4$:
699  002270 3F03                PUSH.16   D3
700  002272 610000A6            CALL      RBYTE             Somme de contrôle
701  002276 66000026            JUMP,NE   LDERR2
702  00227A
703  00227A 361F                POP.16    D3
704  00227C 21C807A4            MOVE.32   A0,SAVADGO
705  002280 4284                CLR.32    D4
706  002282 4282                CLR.32    D2
707  002284 0C430001            COMP.16   #1,D3             Test dernier caractère reçu
708  002288           FCHGER:
709  002288 4CDF000A            POPM.32   D1|D3
710  00228C 4E75                RET
711  00228E
712  00228E           LDERR0:                               Erreur de chargement
713  00228E 383C0031            MOVE.16   #ERCHAR,D4        No d'erreur
714  002292 60000016            JUMP      AFERR
715  002296           LDERR1:                               Erreur de chargement
716  002296 383C0032            MOVE.16   #ERCHAR+1,D4      No d'erreur
717  00229A 6000000E            JUMP      AFERR
718  00229E           LDERR2:                               Erreur de chargement
719  00229E 383C0033            MOVE.16   #ERCHAR+2,D4      No d'erreur
720  0022A2 60000006            JUMP      AFERR
721  0022A6
722  0022A6           ERLDLOW:                              Adresse de chargement trop petite
723  0022A6 383C0035            MOVE.16   #ERADLOW,D4
724  0022AA           ;         JUMP      AFERR
725
726  0022AA           AFERR:
727  0022AA 143C00FC            MOVE.8    #-4.D2          Mode d affichage
728  0022AE 003C0004            SETZ                      Active Bit Z)ero
729  0022B2 60D4                JUMP      FCHGER
730
731  0022B4
```

Routine SENDBLOC — Envoi d'un bloc en format PDP 11

```
0022B4                  in:     A0.32   'données à envoyer
0022B4                          D4.16   nombres de données à envoyer
0022B4                  out:    A0.32   'donnée suivant la dernière envoyée
0022B4                  mod:    A0.32

0022B4              SENDBLOC:
0022B4 48E7D800             PUSHM.32  D0|D1|D3|D4
0022B8              Envoi amorce
0022B8 303C001F             MOVE.16   #10'32-1,D0       Nombre de zéro (amorce)
0022BC 163C0000             MOVE.8    #0,D3
0022C0              L15$:
0022C0 61000050             CALL      WBYTE             Envoi amorce
0022C4 51C8FFFA             DJ.16,NH0 D0,L15$
0022C8 4281                 CLR.32    D1                Mise à zéro cheksum
0022CA 163C0001             MOVE.8    #1,D3
0022CE 61000042             CALL      WBYTE             Envoi 1 d'entête
0022D2 163C0000             MOVE.8    #0,D3
0022D6 6100003A             CALL      WBYTE             Envoi 0 d'entête

0022DA              Envoi longueur
0022DA 3604                 MOVE.16   D4,D3
0022DC 5C43                 ADD.16    #10'6,D3          Correction longueur
0022DE 61000032             CALL      WBYTE             Envoi poids faible de la longueur
0022E2 E15B                 RL.16     #10'8,D3
0022E4 6100002C             CALL      WBYTE             Envoi poids fort

0022E8              Envoi adresse
0022E8 3608                 MOVE.16   A0,D3             Adresse poids faible
0022EA 61000026             CALL      WBYTE
0022EE E15B                 RL.16     #10'8,D3          Adresse poids fort
0022F0 61000020             CALL      WBYTE

0022F4              Envoi d'un bloc
0022F4 3004                 MOVE.16   D4,D0             Nombre d'octets par blocs
0022F6 6700000C             JUMP,EQ   L45$              Fini?
0022FA              L40$:
0022FA 1618                 MOVE.8    (A0+),D3
0022FC 61000014             CALL      WBYTE             Envoi d'un octe et calcul cheksum
002300 5344                 DEC.16    D4
002302 66F6                 JUMP,NE   L40$
002304

002304              L45$:
002304              Envoi cheksum
002304 4401                 NEG.8     D1                16'100-Cheksum
002306 1601                 MOVE.8    D1,D3
002308 61000008             CALL      WBYTE             Envoi somme de contrôle
00230C 4CDF001B             POPM.32   D0|D1|D3|D4
002310 4E75                 RET

002312
```

Routine WBYTE — routine spécifique au chargeur, calcule la somme de contrôle dans D1.8

```
002312                  in:     D1.8    Cheksum avant nouveau caractère envoyé
002312                          D3.8    Nouveau caractère à envoyer
002312                  out:    D1.8    Cheksum des caractères reçus
002312                  mod:    D1.8, D3.8

002312              WBYTE:
002312 6100000E             CALL      WUSA
002316 D203                 ADD.8     D3,D1             Calcul cheksum
002318 4E75                 RET

00231A
```

Routine RBYTE — Routine spécifique au chargeur, calcule la somme de contrôle dans D1.8

```
794   00231A                in:    D1.8     Cheksum avant nouveau caractère
795   00231A                out:   D1.8     Cheksum des caractères reçus
796   00231A                       D3.8     Nouveau caractère
797   00231A                mod:   D1.8, D3.8
798
799   00231A           RBYTE:
800   00231A 61000018            CALL     RUSA
801   00231E D203                ADD.B    D3,D1          Calcul cheksum
802   002320 4E75                RET
803
804   002322
```

Routine WUSA — Ecriture USART

```
805   002322           Envoi d'un octet
806   002322           Attente active que le buffer d'émission soit vide
807   002322              in:    D3.8     Octet à transmettre
808   002322              out:            Octet transmis sur usart
809   002322              mod:   F
810
811   002322           WUSA:
812   002322 08380000C009        TEST.B   SUSA:#BRDYS    Test si buffer d'émission prêt
813   002328 67F8                JUMP,BC  WUSA
814   00232A 11C3C008            MOVE.B   D3,USA         Envoi du caractère
815   00232E 11C0C006            MOVE.B   D0,HP
816   002332 4E75                RET
817
818   002334
```

Routine RUSA — Lecture USART

```
819   002334           Attente d'un octet
820   002334           Attente active que le buffer de réception ait reçu un caractère
821   002334           Petit bip pour chaque octet reçu
822   002334              in:
823   002334              out:   D3.8     Octet
824   002334              mod:   D3.8  (Important le .8 !!)
825   002334

826   002334           RUSA:
827   002334 08380001C009        TEST.B   SUSA:#BFULL    Test si un caractère est reçu
828   00233A 67F8                JUMP,BC  RUSA
829   00233C 11C0C006            MOVE.B   D0,HP          Bip et lampe qui clignote
830   002340 1638C008            MOVE.B   USA,D3         Lit le caractère
831   002344 4E75                RET
832
833   002346
```

Routine INUSA — Init USART

```
834   002346           Initialisation de l'usart 8251
835   002346           Il faut savoir que ce circuit possède deux registres différents que
836   002346           l'on atteint à la même adresse SUSA
837   002346           Le premier programme le MODE d'utilisation soit:
838   002346           - Synchrone/asynchrone, parité, nbr stop bit, longueur caractère
839   002346           Le second permet d'envoyer des COMMANDES au circuit
840   002346           - Transmission permise
841   002346           - DTR actif/inactif
842   002346           - Réception permise
843   002346           - Envoi d'un BREAK
844   002346           - RTS actif/inactif
845   002346           - Reset des erreurs
846   002346           - Reset Interne
847   002346           Le MODE est atteint après un reset externe ou interne
848
849   002346           INUSA:
850   002346 11FC0002C009        MOVE.B   #DUHHY,SUSA    Rien si en commande
851   00234C                                             Passe en commande, si en mode
```

```
852  00234C 11FC0002C009          MOVE.B   #DUMMY,SUSA        Encore une fois pour rien
853  002352 11FC0040C009          MOVE.B   #RSTUSA,SUSA       Reset soft
854  002358 11FC00CEC009          MOVE.B   #MODUSA,SUSA       Selectionne le mode
855  00235E 11FC0015C009          MOVE.B   #ENAUSA,SUSA       Permet les réceptions et émissions
856  002364 4E75                  RET
857
858
859  002366
```

Routines utilitaires

```
860  002366
861
862  002366
```

Routine INHEX — Lit un nombre et la touche ordre associée

```
863  002366              Routine principale d'affichage et de lecture
864  002366              En attendant un nouvel ordre, affiche les paramètres donnés en entrée selon d
865  002366              Selon le mode, D4 contient une valeur à afficher ou une adresse où lire la don
866  002366                 in:   D4.32   Adresse (16 bits affichés)
867  002366                       D4.16   ou nombre à afficher
868  002366                       D2.8  SI D2 = 0 .. 16'7F  Affiche D4.16
869  002366                          SI D2 = -1            Affiche D4.16 ('mémoire)
870  002366                          SI D2 = -1 et CTRL    Affiche (D4.32).8
871  002366                          SI D2 = -2            Affiche P suivi de D4.8 ('I/O)
872  002366                          SI D2 = -2 et CTRL    Affiche (D4.32).8
873  002366                          SI D2 = -3            Affiche = et D4.8 (milieu d'ordre O
874  002366                          SI D2 = autre         Affiche ? et D4.8 (erreur)
875  002366                 out:  D2.32 = 0..4 Nombre de digits introduits
876  002366                       D3.32  Dernière touche
877  002366                       D4.32  Nombre 16 bits introduit
878  002366                 mod:  F  D2.32 D3.32 D4.32
879
880  002366              Utilisation des registres:
881  002366                       D2.32   compteur de digits introduit
882  002366                       D4.32   Nombre entré (masqué 16 bits)
883
884  002366              INHEX:
885  002366 2F00                  PUSH.32  D0
886  002368 61000038              CALL     TESTCLA            Attend une touche du clavier
887  00236C 4284                  CLR.32   D4
888  00236E 4282                  CLR.32   D2
889
890  002370              Nouvelle touche
891  002370              L10$:
892  002370 02800000001F          AND.32   #HCLA,D0           Masque les bits inutiles
893  002376 08800004              TCLR.32  D0:#BCTRL          Les ordres sont donnés lorsque la touche CTRL est appuyée
894  00237A                       le bit est mis à zéro pour permettre une recherche de l'ordre plus aisée
895  00237A 6600000E              JUMP,BS  L90$               OK on sort ->90$
896  00237E E98C                  SL.32    #4,D4
897  002380 8880                  OR.32    D0,D4              insertion du nombre (ADD.16 serait OK)
898  002382 5282                  INC.32   D2
899  002384 6100001C              CALL     TESTCLA
900  002388 60E6                  JUMP     L10$               On va attendre la touche suivante
901
902  00238A              Touche d'ordre appuyée
903
904  00238A              L90$:
905  00238A 02840000FFFF          AND.32   #16'FFFF.D4        Sur Dauphin actuel on n utilise que 16 bits
906  002390 2600                  MOVE.32  D0.D3              Copie touche fonctions appuyee
907  002392 0C8200000004          COMP.32  #NBDIG.D2          Nombre de chiffres tapes - nbr max
908  002398 6F000004              JUMP.LE  L95$               non -> 95$
909  00239C 7404                  MOVE.32  #NBDIG,D2          D2 <- nbr max de chiffres
910  00239E              L95$:
911  00239E 201F                  POP.32   D0
912  0023A0 4E75                  RET
913
914
915  0023A2
```

Routine TESTCLA Attend une touche chiffre appuyée sur la clavier

```
0023A2                  in:   -
0023A2                  out:  D0.8      $CLA
0023A2                  mod:  D0.8

0023A2              TESTCLA:
0023A2 6100000E             CALL    DISP            Affichage en continu
0023A6 1038C007             MOVE.8  CLA,D0          Lecture du clavier
0023AA 08000007             TEST.32 D0:#BFULCLA     Nouvelle touche appuyée? (0..7)
0023AE 67F2                 JUMP,BC TESTCLA         Non, on attend
0023B0 4E75                 RET
```

Routines d'affichage

```
0023B2
0023B2              La routine DISP prépare les paramètres pour LED et affiche le contenu de A4 o
0023B2              La routine LED affiche une fois le contenu de D4 en fonction du contenu de D1
0023B2
```

Routine DISP Affiche D4.16 selon D2.8

```
0023B2                  in:   D4.32   Adresse (16 bits affichés)
0023B2                        D4.16   ou nombre à afficher
0023B2                        D2.8  Si D2 = 0 .. 16'7F  Affiche D4.16
0023B2                           Si D2 = -1             Affiche D4.16 ('mémoire)
0023B2                           Si D2 = -1 et CTRL     Affiche d suivi(D4.32).8
0023B2                           Si D2 = -2             Affiche P suivi de D4.8 ('I/O)
0023B2                           Si D2 = -2 et CTRL     Affiche (D4.32).8
0023B2                           Si D2 = -3             Affiche = et D4.8 (milieu d'ordre O
0023B2                           Si D2 = autre          Affiche ? et D4.8 (erreur)
0023B2                  out:  Prépare D1.16 et appelle LED pour afficher
0023B2                  mod:  -

0023B2              DISP:
0023B2 48E74808             PUSHM.32 D1|D4|A4
0023B6 7200                 MOVE.32  #16'0000,D1        Affichage de tout xxxx
0023B8 08020007             TEST.32  D2:#7              Bit de signe D2.8
0023BC 67000056             JUMP,BC  L90$               Nombre normal (frappe)

0023C0 0C0200FF             COMP.8   #-16'1,D2
0023C4 6700001A             JUMP,EQ  L10$               = -1 (mémoire)
0023C8 0C0200FE             COMP.8   #-16'2,D2
0023CC 67000028             JUMP,EQ  L20$               = -2 (I/O)
0023D0 0C0200FD             COMP.8   #-16'3,D2
0023D4 6700003A             JUMP,EQ  L30$               = -3 (pendant ordre OUT)

0023D8              Erreur          > -3 (erreur)
0023D8 323C4100             MOVE.16  #16'4100,D1        Affichage ?␣xx
0023DC 60000036             JUMP     L90$               Va afficher
0023E0              Mémoire
0023E0              L10$:
0023E0 08380004C007         TEST.8   CLA:#BCTRL
0023E6 6700002C             JUMP.BC  L90$               Saut si touche CTRL relâchée
0023EA              Affichage contenu

0023EA 323C7100             MOVE.16  #16'7100,D1        Affichage de blank xx
0023EE 3844                 MOVE.A16 D4,A4              On utilise l'adresse sur 16 bits
0023F0 1814                 MOVE.8   (A4),D4
0023F2 60000020             JUMP     L90$               Va afficher
0023F6              I/O
0023F6              L20$:
0023F6 323C2100             MOVE.16  #16'2100,D1        Affichage P␣xx
0023FA 08380004C007         TEST.8   CLA:#BCTRL
```

```
977   002400 67000012              JUMP,BC   L90$              Saut si touche pressée. afl contenu
978
979   002404 323C1100              MOVE.16   #16'1100,D1      Affichage ␣␣xx
980   002408 3844                  MOVE.A16  D4,A4            Version 4 digits. on prend ad sur 16 bits!!
981   00240A 1814                  MOVE.8    (A4),D4
982   00240C 60000006              JUMP      L90$             Va afficher
983   002410
984   002410 323C6100    L30$:     MOVE.16   #16'6100,D1      Affichage =␣xx
985   002414
986   002414             L90$:
987   002414 61000008              CALL      LED              Routine d'affichage
988   002418 4CDF1012              POPM.32   D1|D4|A4
989   00241C 4E75                  RET
990
991   00241E
```

Routine LED Affiche D4.16 selon D1.16

```
992   00241E                 in:   D4.16 Contenu à afficher selon mode de D1.16
993   00241E                       D1.16 type d'affichage
994   00241E                        Si D1=16'0000 les 4 digits hexa de D4.16 (aff. d'une adress
995   00241E                        Si D1=16'1100 les 2 digits hexa de D4.8 (contenu)
996   00241E                        Si D1=16'7100 un d puis les 2 digits hexa de D4.8 (contenu
997   00241E                        Si D1=16'2100 un P puis les 2 digits hexa de D4.8 (address
998   00241E                        Si D1=16'6100 un = puis les 2 digits hexa de D4.8 (address
999   00241E                        Si D1=16'4100 un ? puis un numéro d'erreur
1000  00241E                        Si D1=16'3111 un L
1001  00241E                 out:  -
1002  00241E                 mod:  -
1003
1004  00241E             LED:
1005  00241E 48E7F880              PUSHM.32  D0..D4|A0
1006  002422 207CFFFFC004          MOVE.32   #DIG0+NBDIG,A0   Adresse digit de droite
1007  002428 343C0003              MOVE.16   #NBDIG-1,D2      Compteur par nbr de digits
1008  00242C             L1$:
1009  00242C 1004                  MOVE.8    D4,D0            D0: registre de travail
1010  00242E E85C                  RR.16     #4,D4            Décale pour préparer le prochain tour
1011  002430 0240000F              AND.16    #16'F,D0         Ne prend que le digit de droite
1012
1013  002434 1601                  MOVE.8    D1,D3            Masque d'affichage de ce digit
1014  002436 E859                  RR.16     #4,D1            Decale pour préparer le prochain tour
1015  002438 0243000F              AND.16    #16'F,D3         Ne prend que le digit de droite
1016  00243C 67000008              JUMP,EQ   L2$              Pas de masquage -> 2$
1017
1018  002440 1003                  MOVE.8    D3,D0            Il faut afficher un signe spécial
1019  002442 06000010              ADD.8     #16'10,D0        Les signes spéciaux sont après les chiffres
1020  002446             L2$:
1021  002446 113B000C              MOVE.8    R8~TADIG+A16~(D0),(-A0)   Envoie directement la valeur sur le digit lumineux
1022  00244A 51CAFFE0              DJ.16,NHO D2,L1$
1023  00244E 4CDF011F              POPM.32   D0..D4|A0
1024  002452 4E75                  RET
1025  002454
```

Table des codes à afficher

```
1026  002454             Table des codes à afficher
1027  002454             TADIG:
1028  002454                       Chiffres en Hexa
1029  002454 3F065B4F              .8        C0.C1.C2.C3
1030  002458 666D7D07              .8        C4.C5.C6.C7
1031  00245C 7F6F777C              .8        C8.C9.CA.CB
1032  002460 585E7971              .8        CC.CD.CE.CF
1033  002464                       Signes spéciaux
1034  002464 00007338              .8        0.0.CP.CL
1035  002468 5340485E              .8        QUEST,MOINS.EGAL.CD
1036  00246C   00000000    .EVEN
1037
1038
1039  00246C
```

Routine AFX8 Affiche D4 sur 32 bits (8 digits)

```
1040  00246C                Affiche ---- pendant 1 sec
1041  00246C                Affiche poids forts de D4 (D4:#31..#16) pendant 1 sec
1042  00246C                Affiche poids faibles de D4(D4:#15,.0) pendant 1sec
1043  00246C                  in:   D4.32   nombre 32 bits à afficher
1044  00246C                  mod:  -
1045
1046  00246C              AFX8:
1047  00246C 48E7E800             PUSHM.32  D0..D2|D4
1048  002470 303C0064             MOVE.16   #10'100,D0
1049  002474 2F04                 PUSH.32   D4
1050  002476              L40$:
1051  002476 323C5555             MOVE.16   #16'5555,D1
1052  00247A 61A2                 CALL      LED
1053  00247C 51C8FFF8             DJ.16,NMO D0,L40$
1054  002480 281F                 POP.32    D4
1055
1056  002482 303C0064             MOVE.16   #10'100,D0
1057  002486 2F04                 PUSH.32   D4
1058  002488 4844                 SWAP.32   D4
1059  00248A 02840000FFFF         AND.32    #16'FFFF,D4
1060  002490              L50$:   Affiche poids forts 1 sec
1061  002490 143C0000             MOVE.8    #0,D2
1062  002494 6100FF1C             CALL      DISP
1063  002498 51C8FFF6             DJ.16,NMO D0,L50$
1064
1065  00249C 303C0064             MOVE.16   #10'100,D0
1066  0024A0 281F                 POP.32    D4
1067  0024A2 02840000FFFF         AND.32    #16'FFFF,D4
1068  0024A8              L51$:   Affiche poids faibles 1 sec
1069  0024A8 143C0000             MOVE.8    #0,D2
1070  0024AC 6100FF04             CALL      DISP
1071  0024B0 51C8FFF6             DJ.16,NMO D0,L51$
1072  0024B4 4CDF0017             POPM.32   D0..D2|D4
1073  0024B8 4E75                 RET
1074
1075
1076  0024BA
```

Routines de bas niveau d'initialisation

```
1077  0024BA
1078  0024BA
```

Routine INIVECT Routine d'initialisation des pointeurs aux routines d'interruptions

```
1079  0024BA                    mod:  A0.32, A1.32, D0.32, D1.32
1080
1081  0024BA              INIVECT:
1082  0024BA 41FA003C             MOVE.32   #R16'TAVEC,A0       'table des vecteurs
1083  0024BE 43F80000             MOVE.32   #0,A1               Pointe la RAM en zero
1084  0024C2
1085  0024C2 22D8                 MOVE.32   (A0+),(A1+)         Positions en principes jamais atteinte.
1086  0024C4 22D8                 MOVE.32   (A0+),(A1+)         le démarrage se fait en ROM
1087
1088  0024C6                  Chargement en RAM des vecteurs d'interruption par défaut
1089  0024C6                  Utilisation des registres
1090  0024C6                    A0.32   'dans la table
1091  0024C6                    A1.32   'en mémoire
1092  0024C6                    D0.8    Type de données
1093  0024C6                            TYPVECTDIF: Vecteurs différents
1094  0024C6                            TYPVECTIDE: Vecteurs identiques
1095  0024C6                            FININIVECT: Fin init vecteur
1096  0024C6                    D1.8    Nombre de transfers avec ce type -1
1097
1098  0024C6              BINITVECT:
```

```
1099   0024C6 4241                CLR.16    D1
1100   0024C8 1018                MOVE.8    (A0+),D0          Type de données
1101   0024CA 1218                MOVE.8    (A0+),D1          Nombre de transfert-1
1102
1103   0024CC 0C000001            COMP.8    #TYPVECTDIF,D0    Vecteur différents?
1104   0024D0 6700000C            JUMP,EQ   RVECTDIF
1105
1106   0024D4 0C000002            COMP.8    #TYPVECTIDE,D0    Vecteurs identiques
1107   0024D8 67000010            JUMP,EQ   RVECTID
1108
1109   0024DC                     On a finit d'initialiser les pointeurs aux routines d'interruptions
1110   0024DC 4E75                RET
1111
1112   0024DE                     RVECTDIF:
1113   0024DE                     L1$:
1114   0024DE 32FC0000            MOVE.16   #PAGEMON,(A1+)     Adresse de poids fort
1115   0024E2 32D8                MOVE.16   (A0+),(A1+)       Adresse de poids faible
1116   0024E4 51C9FFF8            DJ.16,NMO D1,L1$            Nbr d'ad copiée
1117   0024E8 60DC                JUMP      BINITVECT
1118
1119   0024EA                     RVECTID:
1120   0024EA                     L1$:
1121   0024EA 32FC0000            MOVE.16   #PAGEMON,(A1+)     Adresse de poids fort
1122   0024EE 32D0                MOVE.16   (A0),(A1+)        Adresse de poids faible
1123   0024F0 51C9FFF8            DJ.16,NMO D1,L1$            Nbr d'ad copiée
1124   0024F4 5488                ADD.32    #2,A0             Pour pointé sur ad suivante
1125   0024F6 60CE                JUMP      BINITVECT
1126
1127   0024F8                     La table des vecteurs à copier en ROM a une structure particulière pour éviter
1128   0024F8                     De plus, c'est une table de mots de 16 bits étant donné que toutes les routin
1129   0024F8
1130   0024F8   00000001   TYPVECTDIF   =          1 Vecteurs différents :rror
1131   0024F8   00000002   TYPVECTIDE   =          2 Vecteurs identiques :rror
1132   0024F8   000000FF   FININIVECT   =          16'FF  Fin init vecteur :rror
1133   0024F8
1134   0024F8
1135   0024F8
```

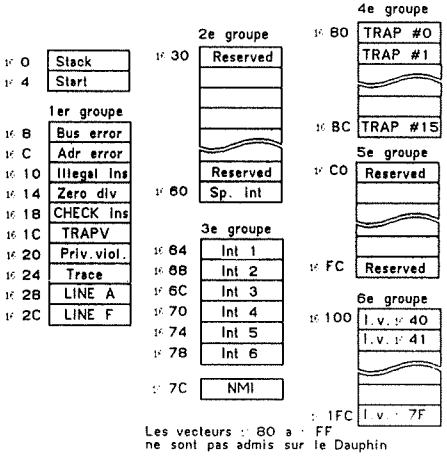


Fig. 6.8 Début mémoire

```
1136   0024F8
```

Adresses des routines d'interruptions

```
0024F8
0024F8                  TAVEC:
0024F8 60000034                 JUMP    ERPILE          En cas de JUMP 0
002500   00002500               .LOC    TAVEC+16'8

002500                  1er groupe
002500 01                       .8.8    TYPVECTDIF      NBVEC1
002501 253A             TVEC1:  .16     RERBUS.AND.16'FFFF  Bus error
002503 2566                     .16     RERADR.AND.16'FFFF  Address err
002505 256C                     .16     RERINSTR.AND.16'FFFF    Instr err
002507 2572                     .16     RERZERO.AND.16'FFFF Div par zero
002509 2578                     .16     RERCHK.AND.16'FFFF  CHK error
00250B 257E                     .16     RERTRAPV.AND.16'FFFF    TRAP overflow
00250D 2584                     .16     RERPRIV.AND.16'FFFF Instr privilégiée
00250F 258A                     .16     RTRACE.AND.16'FFFF  Trace C
002511 2590                     .16     RLINEA.AND.16'FFFF  Emul A
002513 2596                     .16     RLINEF.AND.16'FFFF  aussi instr EXIT
002515   00000009       NBVEC1  =       (APC-TVEC1)/2-1

002515                  2e groupe
002515 020C                     .8.8    TYPVECTIDE,NBVEC2
002517 259C                     .16     RIMPOSS.AND.16'FFFF Zone réservée impossible
002519   0000000C       NBVEC2  =       (16'64-16'30)/4-1

002519                  3e groupe
002519 0205                     .8.8    TYPVECTIDE,NBVEC3
00251B 25A2                     .16     RINT.AND.16'FFFF    Int 1 à 6
00251D   00000005       NBVEC3  =       (16'7C-16'64)/4-1

00251D 0200                     .8.8    TYPVECTIDE,NBVEC3B
00251F 25B4                     .16     RNMI.AND.16'FFFF
002521   00000000       NBVEC3B =       (16'80-16'7C)/4-1

002521                  4e groupe
002521 020F                     .8.8    TYPVECTIDE,NBVEC4
002523 25A8                     .16     RTRAPI.AND.16'FFFF  Traps 0 à 15
002525   0000000F       NBVEC4  =       (16'C0-16'80)/4-1

002525                  5e groupe
002525 020F                     .8.8    TYPVECTIDE,NBVEC5
002527 259C                     .16     RIMPOSS.AND.16'FFFF Jusqu'en 16'100
002529   0000000F       NBVEC5  =       (16'100-16'C0)/4-1

002529                  6e groupe
002529 023F                     .8.8    TYPVECTIDE,NBVEC6
00252B 25AE                     .16     RIVECT.AND.16'FFFF  Jusqu'en 16'200
00252D   0000003F       NBVEC6  =       (16'200-16'100)/4-1

00252D FF                       .8      FININIVECT
00252E   00000000               .EVEN


00252E
```

Gestion des erreurs, Rout. d'interruptions et d'exceptions

```
00252E

00252E
```

Routine **ERPILE** Si jump 0 effectué

```
1193   00252E                ERPILE:
1194   00252E 183C0000               MOVE.B    #0,D4
1195   002532 143C00FC               MOVE.B    #-4,D2          Affichage d'erreur
1196   002536 6000FB38               JUMP      REMON1
1197   00253A
1198   00253A
```

Routine **RERBUS** Bus Error

```
1199   00253A                RERBUS:
1200   00253A 183C0001               MOVE.B    #ERBUS,D4
1201   00253E                ;       JUMP      EXCEPT
1202   00253E
1203   00253E
```

Routine **EXCEPT** Affichage de l'erreur

```
1204   00253E                Affiche le numéro de l'erreur
1205   00253E                Repart dans le moniteur
1206   00253E                   in:  D4        numéro de l'erreur
1207   00253E                   mod:D0, D1, D4, SP
1208
1209   00253E                EXCEPT:
1210   00253E 303C0064               MOVE.W    #10'100,D0
1211   002542                L1$:
1212   002542 223C00004100           MOVE.L    #16'4100,D1     Affiche ? numéro d'erreur
1213   002548 4EB900002014           CALL      32^_LED
1214   00254E 51C8FFF2               DJ.W,NHO  D0,L1$          Prend PC au moment de l'interruption
1215   002552 282F0004               MOVE.L    (SP)+4,D4       et l'affiche
1216   002556 4EB900002008           CALL      32^_AFX8
1217   00255C 2E7C000007A0           MOVE.L    #PILE,SP        réinitialise le 'pile
1218   002562 6000FB08               JUMP      REMON
1219
1220   002566
```

Routine **RERADR** Erreur d'adresse

```
1221
1222   002566                RERADR:
1223   002566 183C0002               MOVE.B    #ERADR,D4
1224   00256A 60D2                   JUMP      EXCEPT
1225
1226   00256C
```

Routine **RERINSTR** Instruction illégale

```
1227
1228   00256C                RERINSTR:
1229   00256C 183C0003               MOVE.B    #ERINSIL,D4
1230   002570 60CC                   JUMP      EXCEPT
1231
1232   002572
```

Routine **RERZERO** Division par zéro

```
1233
1234   002572                RERZERO:
1235   002572 183C0004               MOVE.B    #ERDIVZ,D4
1236   002576 60C6                   JUMP      EXCEPT
1237
1238   002578
```

Routine **RERCHK** Erreur avec instruction CHK

```
1239
1240   002578                RERCHK:
1241   002578 183C0005               MOVE.B    #ERCHK,D4
1242   00257C 60C0                   JUMP      EXCEPT
1243
```

```
1244   00257E
```

Routine **RERTRAPV** Trap sur Overflow

```
1245
1246   00257E              RERTRAPV:
1247   00257E 183C0006             MOVE.B    #ERTRAPV,D4
1248   002582 60BA                 JUMP      EXCEPT
1249
1250   002584
```

Routine **RERPRIV** Instruction superviseur exécutée en mode utilisateur

```
1251
1252   002584              RERPRIV:
1253   002584 183C0007             MOVE.B    #ERINSSU,D4
1254   002588 60B4                 JUMP      EXCEPT
1255
1256   00258A
```

Routine **RTRACE** Trace

```
1257
1258   00258A              RTRACE:
1259   00258A 183C0008             MOVE.B    #ERTRAC,D4
1260   00258E 60AE                 JUMP      EXCEPT
1261
1262   002590
```

Routine **RLINEA** Instruction 16'Axxx

```
1263
1264   002590              RLINEA:
1265   002590 183C000A             MOVE.B    #ERLINA,D4
1266   002594 60A8                 JUMP      EXCEPT
1267
1268   002596
```

Routine **RLINEF** Instruction 16'Fxxx

```
1269
1270   002596              RLINEF:
1271   002596 183C000F             MOVE.B    #ERLINF,D4
1272   00259A 60A2                 JUMP      EXCEPT
1273
1274   00259C
```

Routine **RIMPOSS** Zone vecteur réservée

```
1275
1276   00259C              RIMPOSS:
1277   00259C 183C0010             MOVE.B    #ERVECR,D4
1278   0025A0 609C                 JUMP      EXCEPT
1279
1280   0025A2
```

Routine **RINT** Interruption autovectorisée non initialisée

```
1281
1282   0025A2              RINT:
1283   0025A2 183C0011             MOVE.B    #ERINT,D4
1284   0025A6 6096                 JUMP      EXCEPT
1285
1286   0025A8
```

Routine **RTRAPI** Instruction Trap #xx

```
1287
1288   0025A8              RTRAPI:
1289   0025A8 183C0012             MOVE.B    #ERTRAP,D4
1290   0025AC 6090                 JUMP      EXCEPT
1291
1292   0025AE
```

Routine RIVECT Interruption vectorisée non initialisée

```
1293
1294   0025AE                RIVECT:
1295   0025AE 183C0013               MOVE.B    #ERINTV,D4
1296   0025B2 608A                   JUMP      EXCEPT
1297
1298   0025B4
```

Routine RNMI Interruption non masquable

```
1299
1300   0025B4                RNMI:
1301   0025B4 183C0020               MOVE.B    #ERNMI,D4
1302   0025B8 6084                   JUMP      EXCEPT
1303   0025BA
1304
1305   0025BA
```

Routine DEMO Routine de démonstration à écrire

```
1306
1307   0025BA
1308   0025BA                n DEMO          Quelques affichages et mélodies pour remplir la
1309   0025BA
1310   0025BA                DEMO:
1311   0025BA 6000FAB0               JUMP      REMON
1312
1313   0025BE                .END
Second pass complete  34 sec
218/182/649 all symbols/global symbols/references
8/1/8 if/else/endif
Source file 655 useful lines long
Binary file 1470/16'5BE bytes long
Assembly time: 50 sec    1572 lines/min
Warning: No .START
   Generating cross reference table   Unused memory: 93302
   Name      Type     Value    Set in  Lines
   ?ASCOND   Val            0          17
   AFADCOUR  Add         20FE DAUHON     394   410   426   437   449
   AFERR     Add         22AA DAUHON     714   717   720   726
   AFX0      Add         246C DAUHON     268   534   548   551  1046
   BASEIND   Add         2008 DAUHON     267   282
   BCTRL     Val            4 DAUHON     191   893   965   976
   BFULCLA   Val            7 DAUHON     185   186   923
   BFULL     Val            1 DAUHON     198   827
   BINITVECT
             Add         24C6 DAUHON    1098  1117  1125
   BOOT      Val     FFFF8000 DAUHON     133   321
   BRDYS     Val            0 DAUHON     197   812
   BSHIFT    Val            3 DAUHON     190
   C0        Val           3F DAUHON     150  1029
   C1        Val            6 DAUHON     151  1029
   C2        Val           5B DAUHON     152  1029
   C3        Val           4F DAUHON     153  1029
   C4        Val           66 DAUHON     154  1030
   C5        Val           6D DAUHON     155  1030
   C6        Val           7D DAUHON     156  1030
   C7        Val            7 DAUHON     157  1030
   C8        Val           7F DAUHON     158  1031
   C9        Val           6F DAUHON     159  1031
   CA        Val           77 DAUHON     160  1031
   CB        Val           7C DAUHON     161  1031
   CC        Val           58 DAUHON     162  1032
   CD        Val           5E DAUHON     163  1032  1035
   CE        Val           79 DAUHON     164  1032
   CF        Val           71 DAUHON     165  1032
```

```
LSAVBLOC Val          80 DAUMON     595   611   612
LSAVREG  Val          40 DAUMON     102   242   246   248   325
MAXPILE  Add         200 DAUMON     230   235
MCHIFFR  Val           F DAUMON     189
MCLA     Val          1F DAUMON     188   892
MDIG     Val          10 DAUMON     187
MODUSA   Val          CE DAUMON     201   854
MOINS    Val          40 DAUMON     170  1035
MON      Val        2000 DAUMON     104   105   262
MUBUS    Val    FFFFC000 DAUMON     134   141   177   184   195   483   520
NBDIG    Val           4 DAUMON     145   907   909  1006  1007
NBVEC1   Val           9 DAUMON    1154
NBVEC2   Val           C DAUMON    1157  1159
NBVEC3   Val           5 DAUMON    1162  1164
NBVEC3B  Val           0 DAUMON    1166  1168
NBVEC4   Val           F DAUMON    1171  1173
NBVEC5   Val           F DAUMON    1176  1178
NBVEC6   Val          3F DAUMON    1181  1183
NEXT     Add        20D0 DAUMON     356   404
NO       Add        21C2 DAUMON     360   363   364   369   566
NOSTART  Add        21E4 DAUMON     586   589
OPEN     Add        20C4 DAUMON     355   390
OUT      Add        2134 DAUMON     362   504
PAGEMON  Val           0 DAUMON     105  1114  1121
PCC      Val        2034 DAUMON     280   282
PC_CODE  Val           0 DAUMON      77   261
PC_VAR   Val           1 DAUMON      78   225
PILE     Add         7A0 DAUMON     234   235   264   341  1217
PNTINT   Add           0 DAUMON     228
PNTINTV  Add         100 DAUMON     229
PR       Val    FFFFFFFF DAUMON      18    21    60   213   222
PREV     Add        20E2 DAUMON     357   420
QUEST    Val          53 DAUMON     172  1035
RAM      Val           0 DAUMON      90    96    97    98   532
RBYTE    Add        231A DAUMON     650   657   662   664   669   671   688
                                   693   700   799
REMON    Add        206C DAUMON     337  1218  1311
REMON1   Add        2070 DAUMON     340   349  1196
RERADR   Add        2566 DAUMON    1145  1222
RERBUS   Add        253A DAUMON    1144  1199
RERCHK   Add        2578 DAUMON    1148  1240
RERINSTR Add        256C DAUMON    1146  1228
RERPRIV  Add        2584 DAUMON    1150  1252
RERTRAPV Add        257E DAUMON    1149  1246
RERZERO  Add        2572 DAUMON    1147  1234
RIMPOSS  Add        259C DAUMON    1158  1177  1276
RINT     Add        25A2 DAUMON    1163  1282
RIVECT   Add        25AE DAUMON    1182  1294
RLINEA   Add        2590 DAUMON    1152  1264
RLINEF   Add        2596 DAUMON    1153  1270
RNMI     Add        25B4 DAUMON    1167  1300
ROM      Val        2000 DAUMON      89   104
RSTUSA   Val          40 DAUMON     210   853
RTRACE   Add        258A DAUMON    1151  1258
RTRAPI   Add        25A8 DAUMON    1172  1288
RUSA     Add        2334 DAUMON     274   800   826   828
RVECTDIF Add        24DE DAUMON    1104  1112
RVECTID  Add        24EA DAUMON    1107  1119
SAVADGO  Add         7A4 DAUMON     237   464   466   704
SAVADRI  Add         7A9 DAUMON     239   479   481
SAVADRO  Add         7A8 DAUMON     238   510   516   518
SAVDAOUT Add         7AA DAUMON     240   506   507   509   522
SAVE     Add        21E6 DAUMON     276   368   608
SAVOPEN  Add         7A0 DAUMON     236   392   406   409   422   425   438
                                   450
SAVREG   Add         7C0 DAUMON     248   322
SENDBLOC Add        22B4 DAUMON     278   615   622   737
```

```
ST          Val     FFFFFFFF DAUMON      16   20   80   213
SUSA        Val     FFFFC009 DAUMON     196  812  827  850  852  853  854
                                855
TADIG       Add         2454 DAUMON    1021 1027
TAORDRES    Add         2084 DAUMON     348  354
TAVEC       Add         24F8 DAUMON    1082 1138 1140
TESTCLA     Add         23A2 DAUMON     886  899  920  924
TESTRAM     Add         2176 DAUMON     365  531
TRUE        Val     FFFFFFFF             16   18
TVEC1       Add         2501 DAUMON    1144 1154
TYPVECTDIF
            Val            1 DAUMON    1103 1130 1143
TYPVECTIDE
            Val            2 DAUMON    1106 1131 1157 1162 1166 1171 1176
                                1181
USA         Val     FFFFC008 DAUMON     195  196  814  830
WBYTE       Add         2312 DAUMON     743  748  750  755  757  761  763
                              770  778  788
WUSA        Add         2322 DAUMON     273  789  811  813
_AFX8       Add         2008 DAUMON     268 1216
_CHGER      Add         202C DAUMON     277
_DISP       Add         2010 DAUMON     270
_INHEX      Add         200C DAUMON     269
_INUSA      Add         2018 DAUMON     272
_LED        Add         2014 DAUMON     271 1213
_LOAD       Add         2024 DAUMON     275
_RUSA       Add         2020 DAUMON     274
_SAVE       Add         2028 DAUMON     276
_SENDBLOC
            Add         2030 DAUMON     278
_WUSA       Add         201C DAUMON     273
```

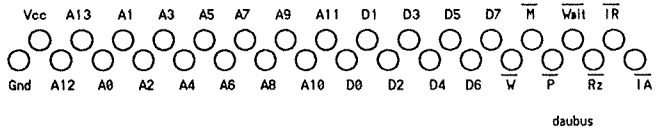
7 Schémas du Dauphin 68008

7.1 Bus système

Le Dauphin est un système microprocesseur basé sur un bus avec 8 bits de données et 14 bits d'adresse, avec des signaux de commande dérivés de Mubus [1], [2]. La liste des signaux est donnée dans la figure 7.1 et le brochage du connecteur dans la figure 7.2. Un "O" caractérise une sortie (par rapport au maître) et un "I" une entrée.

A13..A0	O	Adresses mémoire et entrée/sortie
D7..D0	I/O	Données
$\overline{Wr}$	O	Ecriture (direction du transfert)
$\overline{M}$	O	Sélection espace mémoire (16K octets)
$\overline{P}$	O	Sélection espace périphérique (64 octets)
$\overline{Wait}$	I	Attente sur périphérique lent
$\overline{IR}$	I	Demande d'interruption
$\overline{IA}$	O	Sélection d'un vecteur d'interruption (selon processeur)
$\overline{Rz}$	I	Remise à zéro

Fig. 7.1 Liste des signaux du bus du Dauphin



DAUBUS

Fig. 7.2 Connecteur Dauphin 31 broches DIN 41617

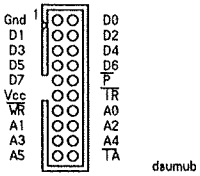
A noter que les Dauphin Stoppani, dont quelques copies existent encore dans les archives du LAMI, ont à la place des lignes d'adresse A13 et A12 les tensions +12V et -12V qu'il ne faut pas oublier de déconnecter.

7.2 Mubus

Un sous ensemble du bus système est relié à un connecteur à 20 broches permettant des accès d'entrée-sortie avec un nombre limité d'adresses. Les signaux de ce connecteur sont donnés dans la figure 7.3 et le brochage du connecteur dans la figure 7.4. Ce connecteur est compatible avec le logidule triple longueur no 118 "Mubus".

A5..A0	Adresse
D7..D0	Données
$\overline{Wr}$	Cycle d'Ecriture si actif
$\overline{P}$	Sélection espace périphérique
$\overline{IR}$	Requête d'interruption
$\overline{IA}$	Lecture d'un vecteur d'interruption (selon processeur)

Fig. 7.3 Liste des signaux Mubus



DAUMUB

Fig. 7.4 Connecteur Mubus

7.3 Carte de base

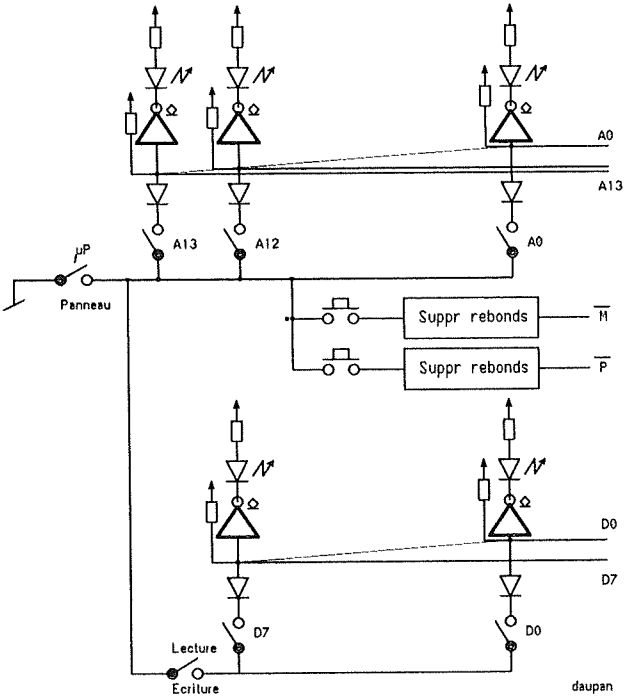
La carte de base porte l'alimentation, les connecteurs du bus, le connecteur Mubus 20 et les interrupteurs de commande directe du bus. La logique de commande est simplifiée pour abaisser les coûts. Son schéma de principe est donné dans la figure 7.5.

Le 1er but du panneau de test est d'afficher l'état du bus en mode fonctionnement et pas à pas. Des diodes lumineuses précédées d'inverseur suffisent dans ce but.

Le 2e but est de commander directement les mémoires et interfaces, à la place du processeur et seulement si celui-ci est déconnecté du bus. Etant donné que le bus ne transporte pas un signal de suspension d'activité ("Hold"), il faut agir indépendamment sur la carte processeur et sur le panneau de test dans le bon ordre.

L'interrupteur de prise de contrôle du passeur de test s'appelle "Panneau/μP". Dans la position "μP", le processeur est le maître du bus et active les LEDs; les commutateurs et poussoirs du panneau sont

inactifs, sauf le bouton "Reset".



DAUPAN

Fig. 7.5 Schéma de principe du panneau de test du Dauphin

Dans la position "Panneau", (il faut avoir préalablement mis le processeur dans le mode "Halt", l'état des interrupteurs d'adresse est copié sur le bus. Si de plus l'interrupteur "Lecture/Ecriture" est en mode "Ecriture", les interrupteurs de données sont aussi activés (données en entrée sur les mémoires et périphériques).

Au lieu de portes à 3 états ou de portes en collecteurs ouverts (figure 7.5), le schéma utilise directement les interrupteurs et des diodes d'isolation pour imposer les états. Ceci conduit à des états zéro à la limite des spécifications (0,6V), mais est encore acceptable à faible vitesse.

Les signaux $\overline{M}$ et $\overline{P}$ ne doivent pas avoir de rebonds de contact, ce qui est obtenu par filtrage avec réseau RC et bascule de Schmitt.

Le schéma complet est donné à la page suivante (figure 7.6).

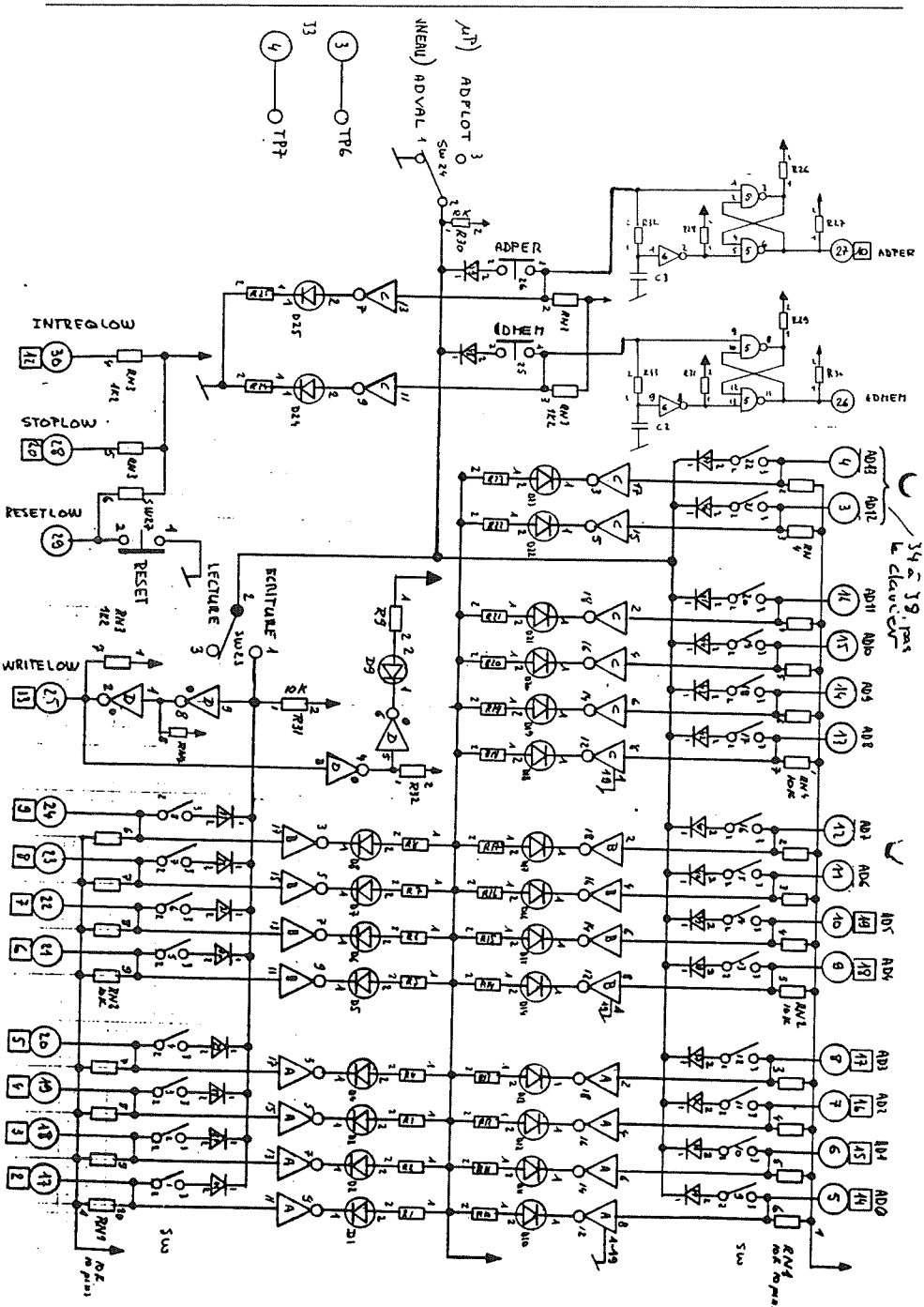


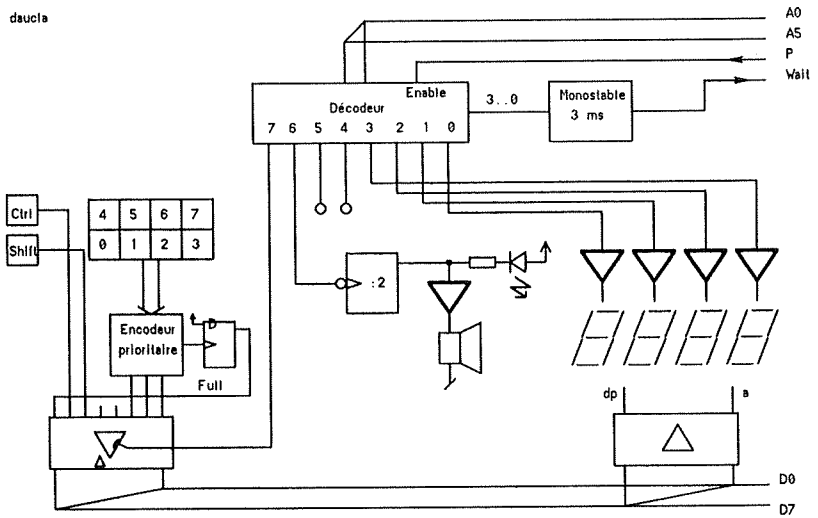
Fig. 7.6 Schéma du panneau de test

7.4 Carte clavier-affichage

La carte clavier-affichage est un périphérique occupant les 8 premières adresses de l'espace d'entrée-sortie.

Adresse 0 (écriture)	Digit de gauche
Adresse 1 ..	..
Adresse 2 ..	..
Adresse 3 ..	Digit de droite
Adresse 4 (écriture ou lecture)	Douille de sélection
Adresse 5	Idem
Adresse 6 (écriture ou lecture)	Changement d'état de la bascule contrôlant la lampe ou le haut-parleur
Adresse 7 (lecture)	Lecture du clavier

Le schéma de principe est donné dans la figure 7.7



DAUCLA

Fig. 7.7 Schéma bloc du clavier

Un décodeur sélectionne les périphériques. Un monostable est activé lorsque l'un des périphériques 0 à 3 est accédé. L'effet du monostable est de prolonger la durée d'affichage des digits pour qu'ils soient mieux visibles. Le processeur est bloqué par le signal WAIT pendant 3ms environ.

Le contrôle des segments de chaque digit dépend directement du mot fourni par le processeur. Toutes les combinaisons de segments sont donc possibles. Pour être visible, chaque digit doit être activé périodiquement par le processeur. Il n'y a pas de mémorisation de la

valeur affichée entre le processeur et l'affichage. Le processeur doit donc effectuer suffisament souvent les instructions de rafraîchissement de l'affichage.

Une bascule (diviseur par 2) contrôle une lampe et un petit haut-parleur.

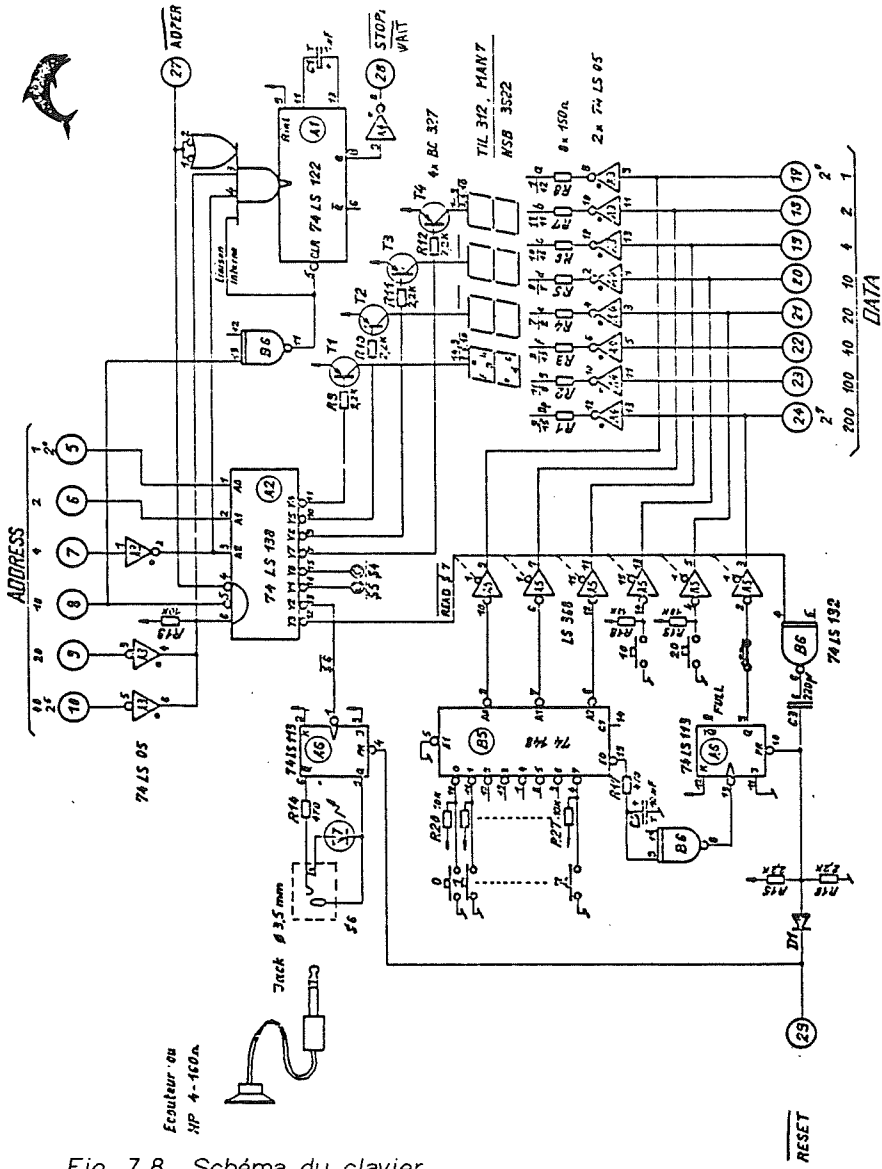


Fig. 7.8 Schéma du clavier

Le codage des touches 0..7 du clavier est réalisé par un circuit encodeur prioritaire à 8 entrées. Les 3 sorties codées sont transmises au bus de données par l'intermédiaire d'un passeur à 3 états contrôlés par la sélection du périphérique d'adresse 7. Les 2 touches FONCTION ("Shift" et "Ctrl") sont également reliées au passeur et permettent de donner 4 significations différentes aux 8 touches 0 à 7.

Lorsque l'une de ces 8 touches est pressée, la bascule FULL s'active et reste active (état 1) jusqu'à ce que le processeur ait lu l'information venant du clavier. Une impulsion brève de remise à zéro est générée à la fin de l'impulsion de lecture du périphérique 7 (clavier).

Cette bascule se justifie par le fait que le processeur est très rapide par rapport à l'opérateur humain. Il ne faut pas que le processeur, qui peut lire le clavier 20000 fois par seconde, ait l'impression que l'utilisateur a pesé 10000 fois sur la même touche, alors qu'il y a simplement maintenu son doigt une demi-seconde. Seuls les codes lus avec le bit FULL=1 sont considérés comme des nouvelles touches pressées.

A noter encore que les rebonds de contact sont supprimés par un condensateur de 10 μF.

En toute rigueur, le schéma proposé a un défaut, et il peut arriver qu'une action sur une touche soit perdue (simultanéité de l'action sur la touche et de l'instruction de lecture). Cela se produit toutefois moins d'une fois sur 1000 et est perçu comme une erreur de manipulation.

Lorsque FULL = 1, une interruption pourrait être créée. La liaison correspondante n'est pas câblée, mais pourrait être ajoutée.

A noter encore que les broches 3 et 4 du connecteur clavier n'ont pas leur spécification A13 et A12, mais correspondent au haut-parleur qui a été déplacé sur la carte de base pour faciliter sa fixation mécanique.

7.5 Carte mémoire

Le schéma de la carte mémoire "84" est très simple (figure 7.9). Un décodeur coupe l'espace d'adressage de 16K en deux tranches de 8K, entièrement occupées. Un circuit RAM 8Kx8 (6264) répond aux adresses 16'2000 à 16'3FFF. Un circuit EPROM 8Kx8 (2764) répond aux adresse 0 à 16'1FFF. Une ROM 16Kx8 (27128) peut être utilisée, mais seule la première moitié de ses adresses peut être accédée.

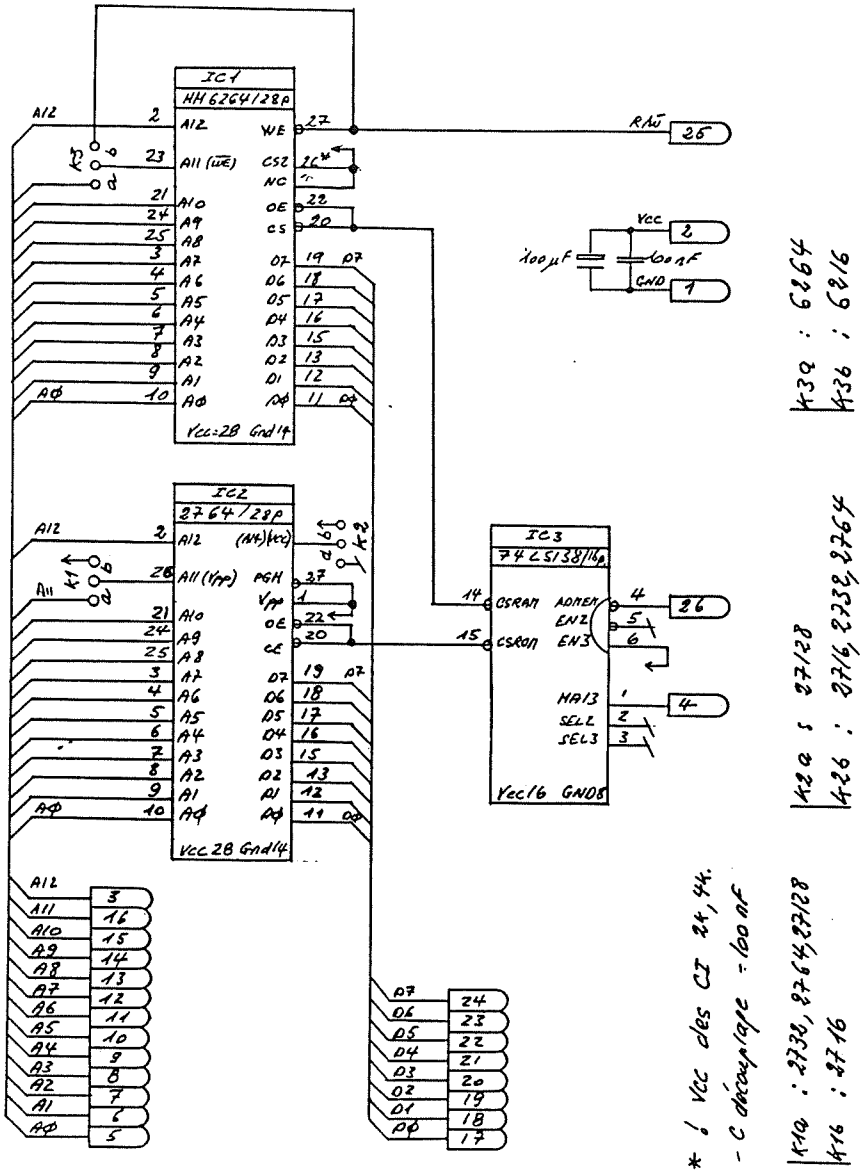


Fig. 7.9 Schéma de la carte mémoire

7.6 Carte série

La carte interface série comporte un circuit USART 8251 avec une interface SIMSER et une interface cassette. Son schéma est donné dans la figure 7.10.

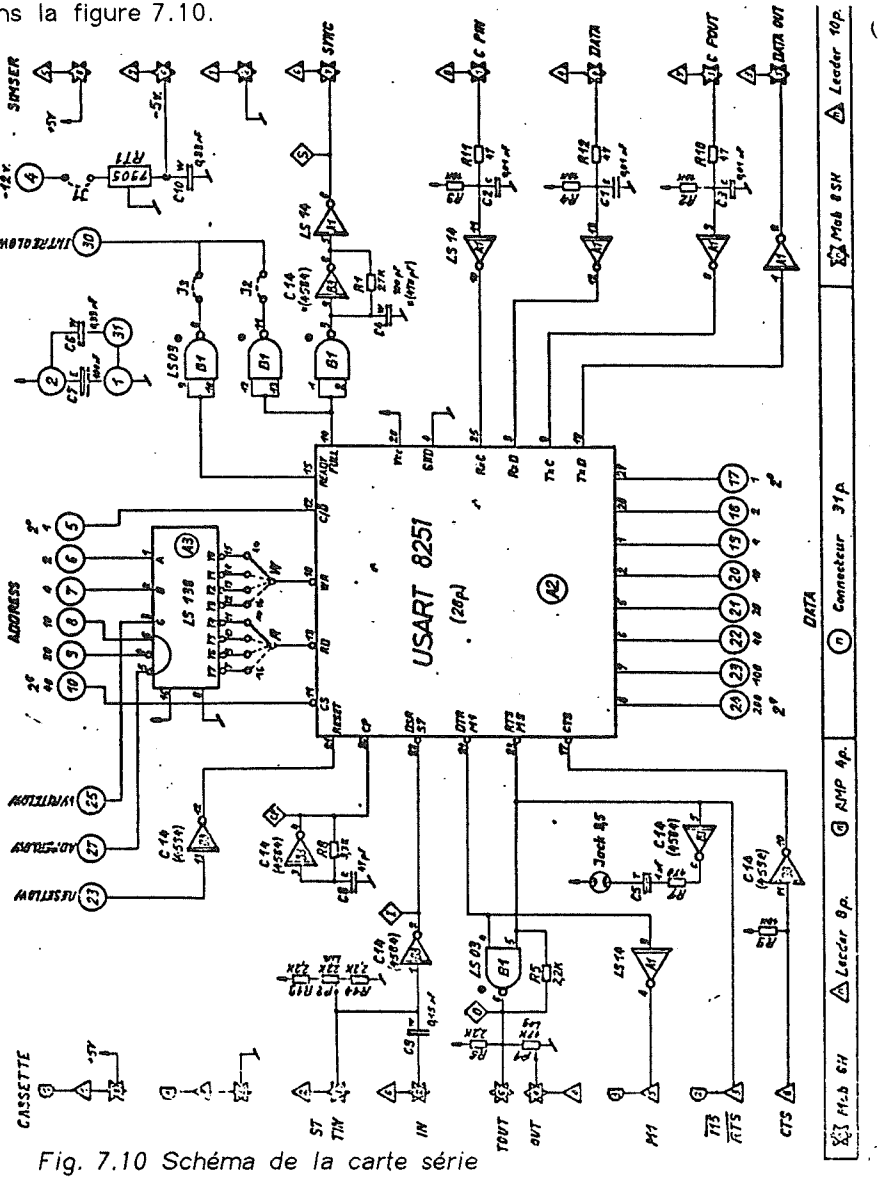


Fig. 7.10 Schéma de la carte série

7.7 Carte processeur

La carte processeur 68008 a été développée par A. Simik comme travail de 8e semestre (juin 84). Un projet de semestre terminé en mars 88 a conduit à une carte 68020 (en mode 8bits) compatible, mais un peu chère.

Des signaux d'adresse et de commande supplémentaires ont été prévus sur un connecteur câble plat additionnel pour permettre des applications spéciales éventuelles.

La logique de pas à pas est réalisée avec deux bascules JK et une suppression de rebonds de contact formé de deux inverseurs interconnectés.

Le processeur est stoppé au milieu d'un cycle d'accès à la mémoire, tant que le poussoir STEP n'est pas activé. Le signal DTACK n'est simplement pas fourni par la PAL (Programmable Logic Array). Il est ainsi possible d'examiner sur les lampes de la carte de base les adresses fournies par le processeur et les données lues ou écrites.

Le commutateur INSTR/CYCLE permet de sélectionner si le pas à pas est effectué pour tous les cycles d'accès à la mémoire (ou aux périphériques), ou seulement pour les lectures du code de l'instruction à exécuter (fetch)

Le décodage du type de cycle est effectué par une PAL. Le signal INSTR indique que c'est la lecture d'une instruction qui est en cours.

Le décodage du sous-espace d'entrée sortie (adresses négatives), le mode d'interruption vectorisé ou non, ainsi que la correspondance mémoire sont effectués dans la PAL.

Le bus Dauphin transporte des adresses physiques, avec la mémoire morte située en 0, et la mémoire vive en 16'2000 (section 7.5). Le 68000 doit démarrer en 0 sur une mémoire morte, mais étant donné que ses vecteurs d'exception sont situés à partir de l'adresse 8 et doivent pouvoir être modifiés par logiciel, il faut de la mémoire vive à ces adresses. Pour satisfaire ces exigences contradictoires, une bascule appelée BOOT (cachée dans la PAL) s'active au "Reset", indiquant que l'EPROM doit être décodée à la fois en zero et en 16'2000 (il n'y a donc pas de RAM accessible). Le processeur saute à une adresse ROM (16'204C dans le moniteur MO68k), puis exécute une instruction qui désactive la bascule BOOT et fait apparaître la RAM aux adresses "zero".

Cette correspondance s'obtient en modifiant la ligne d'adresse A13, dès que le processeur désactive BOOT. La compréhension complète du schéma de la figure 7.11 et du listage de la PAL de la

figure 7.12 déborde les objectifs du premier cours Microinformatique; les cours Microinformatique II et Microprocesseur I apportent les connaissances nécessaires à cette compréhension.

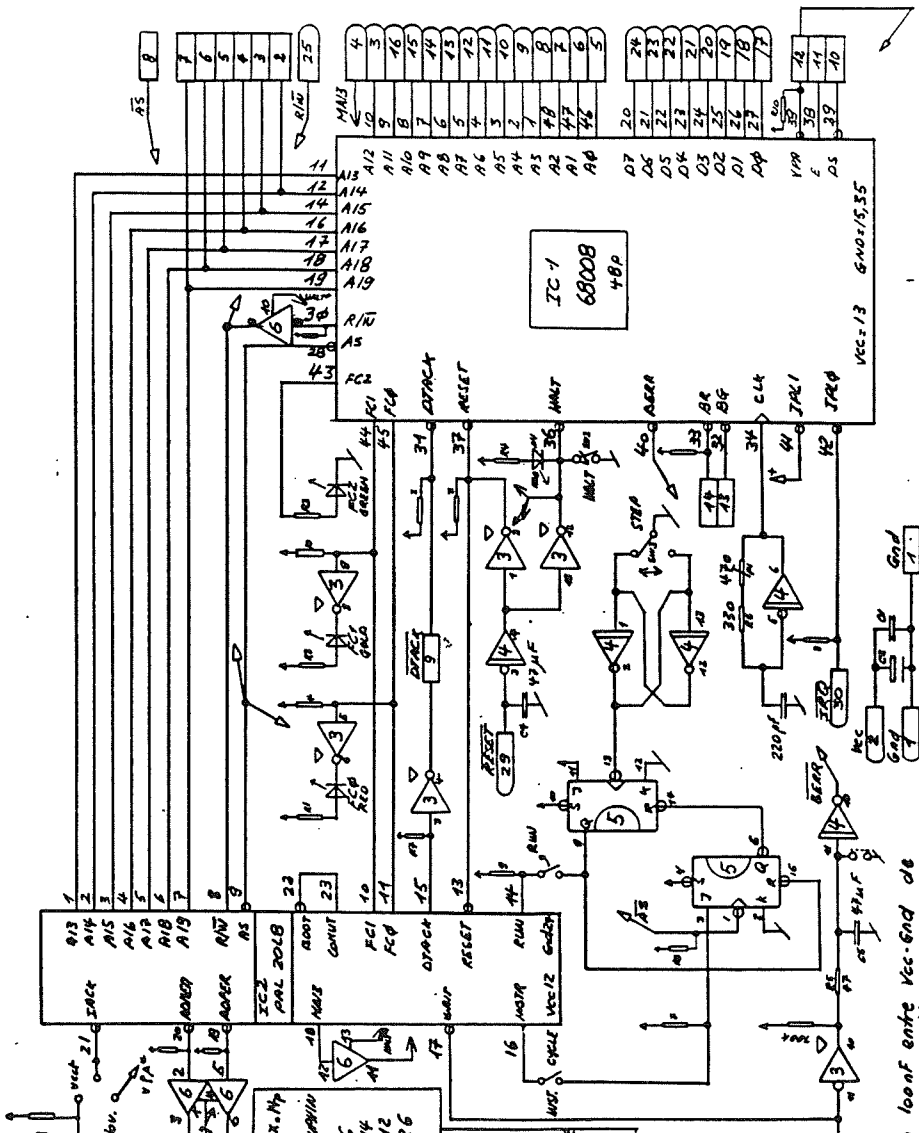


Fig. 7.11 Schéma de la carte 68008

17 mars 1988 DAUPAL.PAL

```
Pal du Dauphin 680008
; RB    Modif MA13   850816

.NAME   PAL20L8

.PIN
A13     = 1     ; Adresse 68000
A14     = 2     ; Adresse 68000
A15     = 3     ; Adresse 68000
A16     = 4     ; Adresse 68000
A17     = 5     ; Adresse 68000
A18     = 6     ; Adresse 68000
A19     = 7     ; Adresse 68000
/WR     = 8     ; Cycle écriture si actif
/AS     = 9     ; Adresse Valide en provenance du 68000
FC1     = 10    ; Code de fonction du 68000
FC0     = 11    ; Code de fonction du 68000
/RZ     = 13    ; Reset du 68000
RUN     = 14    ; "1" -> le cycle peut s'exécuter
                ; "0" -> attend (permet le pas à pas)

DACK    = 15    ; Envoi du signal Data Ack au 68000
INSTR   = 16    ; "actif" -> cycle Programme (Utilisateur ou Superviseur)
/WAIT   = 17    ; Wait du bus dauphin
MA13    = 18    ; ligne adresse 13 envoyé à la mémoire
/ADPER  = 19    ; accès Périphérique
/ADMEM  = 20    ; accès Mémoire
/IACK   = 21    ; cycle "Interrupt acknowledge"
/BOOT   = 22    ; Phase de démarrage
/COMUT  = 23    ; Commutte l'Eprom
VCC     = 24

.EQUATION

ADMEM   = /A19*/A18*/A17*/A16*/A15*/A14*AS

ADPER   = A19*A18*A17*A16*A15*A14*AS

BOOT    = COMUT*/RZ
          + A19*A18*A17*A16*A15*/A14*AS

/DACK   = /AS
          + WAIT
          + /RUN

/INSTR  = FC0
          + /FC1

/MA13   = A13*/WR
          +/WR*/COMUT

IACK    = FC0*FC1*AS

.END
```

DAUPAL.PAL

Fig. 7.12 Listage de la PAL de la carte 68008