rogrammer en assembleur permet de multiplier la uissance de son ordinateur : certains programmes e peuvent être réalisés qu'en assembleur : jeu, mulation, utilitaire, ...

objet de ce livre, constitué de deux parties, est une part de vous faire découvrir les étonnantes ossibilités de votre EXELVISION, d'autre part de ous apprendre à programmer.

a première partie décrit les caractéristiques techni- ues de la famille de processeurs TMS 7000 et de architecture des ordinateurs EXELVISION.

a seconde partie propose un apprentissage pro- ressif de la programmation en langage assembleur ar le biais de la réalisation de programmes simples, tiles et adaptés aux besoins du programmeur.

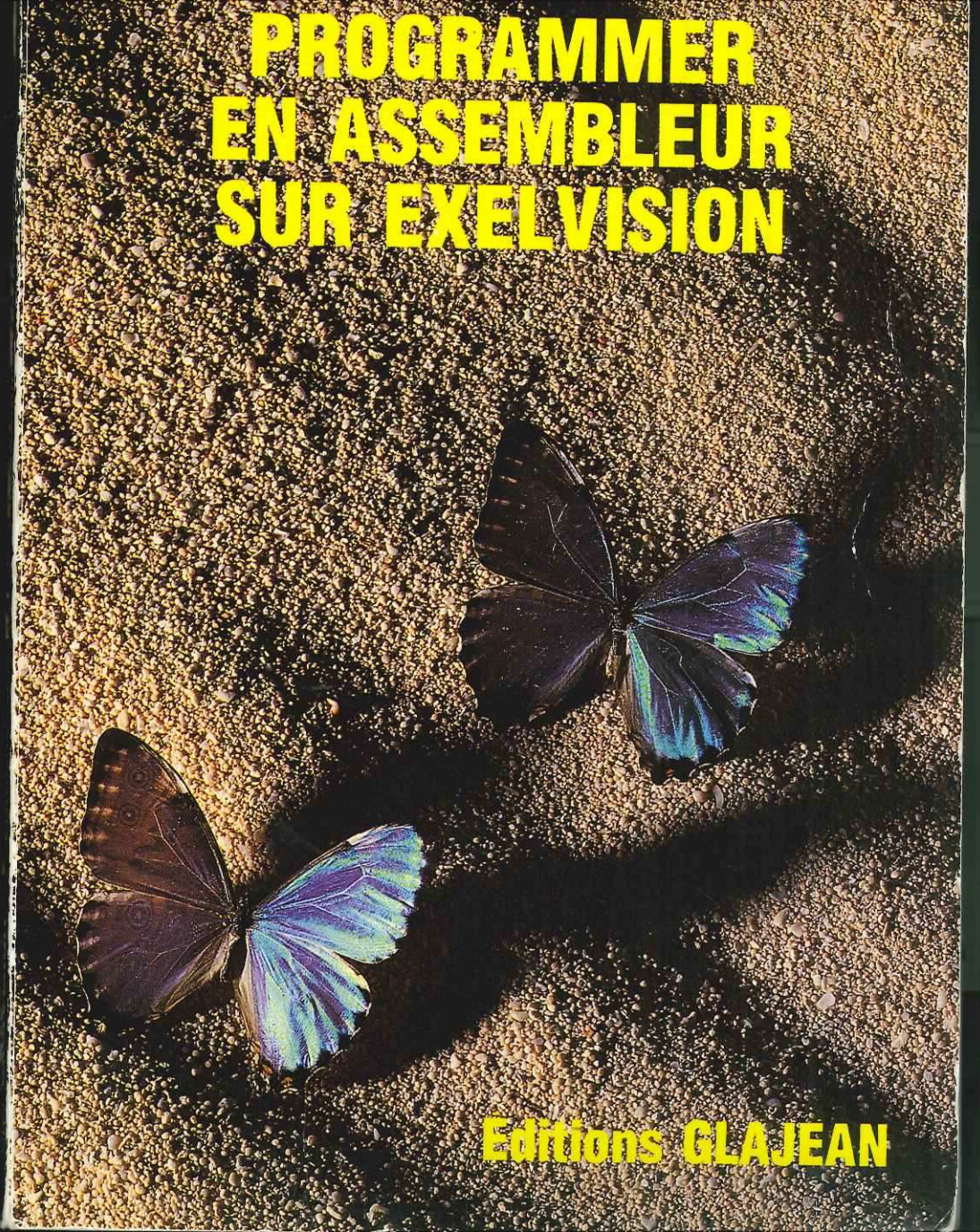
e débutant commencera par la seconde partie, tout n se référant à la première pour obtenir les rensei- nements nécessaires à la bonne compréhension es exemples.

amateur éclairé, ayant déjà programmé en assem- eur, trouvera dans la première partie les éléments dispensables à la réalisation de logiciels, et en fin e seconde partie, des exemples pratiques de pro- rammation, groupés par thème : l'écran, le clavier, c.

49 FF TTC

n° ISBN 2-905451-09-02

PROGRAMMER EN ASSEMBLEUR SUR EXELVISION



Editions GLAJEAN

P. GLAJEAN - F. DONNETTE - L. PENON

Programmer en Assembleur sur Exelvision

Dépôt Légal : 2e Trimestre 1987

ÉDITIONS GLAJEAN
93 RUE DE MAUBEUGE, 75010 PARIS

Chez le même Éditeur

LIVRES

- En compagnie d'Exelogo : les micro-mondes Logo
- En compagnie d'Exelogo : La conduite de projets
- Programmer en Basic sur Exelvision

LOGICIELS

- EXELDUMP : Moniteur des assembleurs
- MICROLOGO : Initiation à Logo avec une tortue pivotante
- EXELIRE : Entraînement à la lecture
  en collaboration avec l'A.F.L.
- EXELASS : Assembleur
- TICIEL 30 : Simulation calculatrice
- TICIEL 57 : Simulation calculatrice programmable

AVERTISSEMENT

Ce livre est constitué de deux parties distinctes mais complémentaires.

La première partie décrit les caractéristiques techniques de la famille de processeurs TMS 7000 et de l'architecture des ordinateurs EXELVISION.

La seconde partie propose un apprentissage progressif de la programmation en langage assembleur par le biais de la réalisation de programmes simples, utiles et adaptés aux besoins du programmeur.

Le débutant commencera par la seconde partie, tout en se référant à la première pour obtenir les renseignements nécessaires à la bonne compréhension des exemples.

L'amateur éclairé, ayant déjà programmé en assembleur, trouvera dans la première partie les éléments indispensables à la réalisation de logiciels, et en fin de seconde partie, des exemples pratiques de programmation, groupés par thème : l'écran, le clavier, etc.

Les auteurs

SOMMAIRE

AVANT-PROPOS

Pour fixer les idées

Un micro-ordinateur est constitué d'une unité centrale associée à d'autres composants : mémoires, entrées/sorties, circuits spécialisés (son, graphisme), etc.

L'unité centrale regroupe l'unité arithmétique et logique (UAL) et une unité de commande.

L'unité arithmétique et logique effectue des opérations arithmétiques et logiques de base : addition, soustraction, ET logique, OU logique, décalage, etc.

L'unité de commande synchronise l'exécution des différentes fonctions du système et des instructions connues du processeur. Elle contrôle en particulier l'échange d'informations entre l'unité centrale et d'autres composants : lecture en mémoire, écriture en mémoire, etc.

Dans le cas d'un microprocesseur, l'unité centrale est intégrée sur une seule pastille aux dimensions microscopiques.

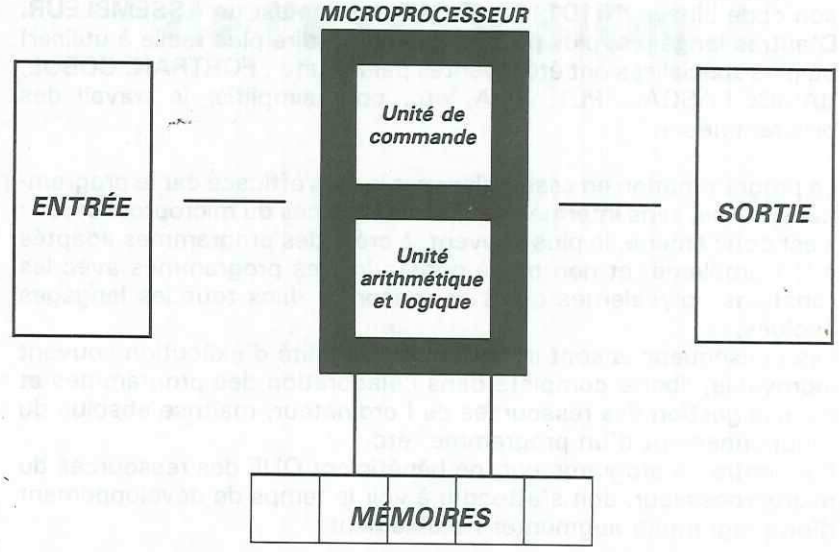


Schéma fonctionnel d'un microprocesseur

L'essentiel de l'activité d'un microprocesseur se résume à :
• des opérations de transfert d'informations entre l'unité centrale et des mémoires ;
• des opérations arithmétiques et logiques élémentaires.

Un ensemble d'instructions (un jeu d'instructions), plus ou moins complet, codées en binaire (électronique oblige !), permet d'accéder à ces différentes opérations et donc de programmer le microprocesseur.
La solution à un problème est présentée sous la forme d'une suite d'opérations (un algorithme) qui, après traduction en code binaire, sera exécutée par le microprocesseur.

Le langage binaire est particulièrement difficile à utiliser et la traduction d'un algorithme en code binaire un travail fastidieux : pour éviter les maux de tête systématiques, il a fallu inventer d'autres langages de programmation des microprocesseurs.

Le programmeur dispose en premier lieu d'un **LANGAGE ASSEMBLEUR** qui associe à chaque code binaire décrivant une instruction, un **NOM** (un mnémonique), ce qui facilite l'écriture et la relecture des programmes. Une addition, par exemple, codée par le nombre 10101100 en binaire sera désignée par le mot ADD, beaucoup plus parlant !
Un programme assurera, par la suite, la traduction du mot ADD en son code binaire 10101100. Ce programme est un **ASSEMBLEUR.**
D'autres langages, plus puissants (c'est à dire plus facile à utiliser) ou plus spécialisés ont été inventés par la suite : FORTRAN, COBOL, BASIC, PASCAL, PL1, ADA, etc. pour simplifier le travail des programmeurs.

La programmation en assembleur est la plus efficace car le programmeur utilise, sans intermédiaire, les ressources du microprocesseur : il est donc amené, le plus souvent, à créer des programmes adaptés à UN problème, et non pas à construire ses programmes avec les fonctions polyvalentes que l'on rencontre dans tous les langages évolués.
Les conséquences sont importantes : rapidité d'exécution souvent incroyable, liberté complète dans l'élaboration des programmes et dans la gestion des ressources de l'ordinateur, maîtrise absolue du fonctionnement d'un programme, etc.
Par contre, le programmeur, ne bénéficiant QUE des ressources du microprocesseur, doit s'attendre à voir le temps de développement d'un programme augmenter notablement.

L'objet de ce livre est d'une part de vous apprendre à programmer en assembleur, d'autre part de vous faire découvrir les possibilités et les ressources étonnantes de votre **EXEL 100** ou de votre **EXELTEL.**

Première Partie

Le processeur TMS 7000

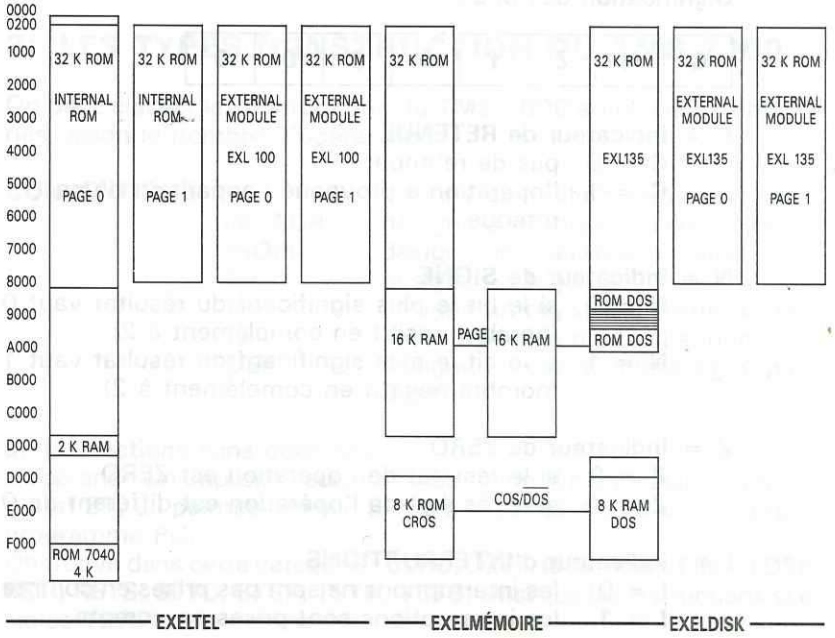
Architecture de l'EXELVISION

LE PROCESSEUR TMS 7000

1) GÉNÉRALITÉS

Le terme TMS 7000 désigne en fait une famille de micro-processeurs incluant en particulier le TMS 7000, le TMS 7020 et le TMS 7040 qui équipe les ordinateurs EXELVISION.
Les informations données dans ce chapitre sont valables pour l'ensemble de la famille.

Le processeur **TMS 7000** est un processeur qui peut manipuler des mots de 8 bits (octets).
Il dispose de trois registres internes de 128 registres externes d'utilité générale, et de 128 registres (ports) d'entrée/sortie.
Il peut adresser directement 64 Ko (65536 bytes).
La zone >0000 à >007F est réservée aux registres généraux et à la pile.
La zone >0100 à >01FF est réservée aux registres d'entrée sortie.
La zone >0200 à >E7FF est une zone d'extension de mémoire (61.5 K).
La zone >F800 à >FFFF est une zone de mémoire morte (4 K).
Dans le cas de l'**EXELTEL,** la zone >0200 à >F7FF est occupée comme l'indique le schéma.



2) ORGANISATION INTERNE

a) Les registres internes

PC PC désigne le compteur-programme (*programme counter*). Il contient toujours l'adresse en mémoire de la prochaine instruction à exécuter. C'est un registre 16 bits auquel on ne peut accéder directement.

SP SP désigne le pointeur de pile (*stack pointer*). Il contient un pointeur 8 bits sur le haut de la pile. L'espace consacré à la pile se trouve en mémoire vive de >0000 à >007F. Le haut de la pile est fixé par l'instruction **LDSP**, qui initialise SP, le pointeur de pile SP. La pile s'étend alors de l'adresse >0000 + SP à >007F.

ATTENTION : La pile occupe une partie de la zone mémoire allouée aux registres généraux.

ST ST désigne le registre d'état (*status register*). Il se lit bit à bit et l'état des indicateurs du microprocesseur après chaque instruction. L'utilisation de ce registre 8 bits, fondamentale, est décrite tout au long de cet ouvrage.

Signification des bits :

C	N	Z	I	0	0	0	0

C = indicateur de RETENUE
- **C = 0** pas de retenue
- **C = 1** l'opération a provoqué l'apparition d'une retenue

N = indicateur de SIGNE
- **N = 0** si le bit le plus significatif du résultat vaut 0 (nombre positif en complément à 2)
- **N = 1** si le bit le plus significatif du résultat vaut 1 (nombre négatif en complément à 2)

Z = indicateur de ZERO
- **Z = 0** si le résultat de l'opération est ZERO
- **Z = 1** si le résultat de l'opération est différent de 0

I = indicateur d'INTERRUPTIONS
- **I = 0** les interruptions ne sont pas prises en compte
- **I = 1** les interruptions sont prises en compte.

b) Registres externes

Ils sont situés en mémoire vive, des adresses >0000 à >007F et sont notés R0 à R128. Presque toutes les instructions du **TMS7000** utilisent ces registres. En particulier, les registres R0 et R1 coïncident avec les accumulateurs A et B. Ces 128 registres peuvent servir indifféremment de compteurs, de pointeurs d'accumulateurs temporaires, etc.

ATTENTION : L'espace consacré à la pile se trouve aussi dans la zone >0000 à >007F. Le haut de la pile est fixé par l'instruction LDSP qui initialise le pointeur de pile SP. La pile s'étend alors de l'adresse SP à >007F.

C) Les registres d'entrée/sortie

Ils sont situés en mémoire vive des adresses >0100 à >01FF et sont notés P0 à P255.

3) LES TYPES D'INSTRUCTION DU TMS 7000

On peut classer les instructions du **TMS 7000** en plusieurs catégories, selon le nombre d'opérandes.

CONVENTIONS :

>	désigne une valeur hexadécimale
@ ADR	désigne une adresse hexadécimale
%OPI	désigne un opérande immédiat
Rn	désigne un registre
Pn	désigne un registre d'entrée/sortie
*****	est le symbole de l'indirection
DEP	désigne un déplacement signé par 8 bits.

a) Instructions sans opérande

L'opérande est implicite : suivant l'instruction, il peut s'agir du registre d'état ST, du pointeur de pile SP, des registres A et B ou compteur-programme PC.
On trouve dans cette catégorie : **CLRC, DINT, IDLE, NOP, EINT, LDSP, RETI, RETS, SETC, STSP, TSTA, TSTB,** ainsi que les instructions spéciales **TRAP.**

b) Instructions à un opérande

L'opérande peut être une adresse 16 bits, un registre ou un déplacement relatif sur un octet.

1. L'opérande est une adresse 16 bits.
   Il peut s'agir d'une adresse absolue ou d'une adresse calculée (adressage indexé).
   FORME : **[INSTRUCTION] @ ADR**
   [INSTRUCTION] @ ADR (B)
   On y trouve : **BR, CALL, CMPA, LDA, STA.**
2. L'opérande est un registre
   FORME : **[INSTRUCTION] Rn**
   [INSTRUCTION] * Rn
   Ex. **DEC R10**
   CLR A
   BR * R12
   Ce registre est désigné par son nom (**R10, R22**) ou par **A** ou **B** pour les registres **R0** et **R1**. On trouve dans cette catégorie : **BR, CALL, CLR, DEC, INC, DECD, INV, POP, PUSH, RL, RLC, RR, RRC, SWAP, XCHB.**

REMARQUE : L'instruction DECD est une instruction à un registre dans son écriture mais agit sur deux registres.

---

3. L'opérande est un déplacement relatif 8 bits signé.
   Cette catégorie regroupe les instructions de branchement conditionnel :
   FORME : **[INSTRUCTION] DEP** ★ *Le déplacement est calculé à l'assemblage*
   Ex. **JEQ TERMINE**
   JC, JEQ, JHS, JL, JMP, JN, JNC, JNE, JNZ, JP, JPZ, JZ

c) Instructions à deux opérandes

Ces instructions utilisent deux opérandes : l'opérande SOURCE et l'opérande DESTINATION.

1. L'opérande SOURCE est un registre ou une valeur immédiate, l'opérande DESTINATION est un registre.
   FORME : **[INSTRUCTION] %OPI, Rn**
   [INSTRUCTION] Rn, Rn
   Ex. **MOV %15,A**
   ADD R10,R15
   On trouve dans cette catégorie : **ADC, ADD, AND, DAC, CMP, DSB, MOV, MOVD, MPY, OR, SBB, SUB, XOR.**
2. l'opérande SOURCE est **A,B** ou une valeur immédiate, l'opérande DESTINATION est un registre d'entrée/sortie.

FORME : **[INSTRUCTION] A,Pn**
[INSTRUCTION] B,Pn
[INSTRUCTION] %OPI,Pn
Ex. **ANDP A,P32**
XORP B,P20
ORP %10,P80
On trouve dans cette catégorie : **ANDP, ORP, XORP, MOVP.**

* **MOVP**, qui assure la lecture et l'écriture des registres d'entrée/sortie, supporte les FORMES complémentaires :
MOVP A,Pn
MOVP B,Pn

3. L'opérande SOURCE est un registre, l'opérande DESTINATION est un déplacement relatif 8 bits signé.
   FORME : **[DJNZ] Rn,DEP**
   Ex. **DJNZ A,SOMME** ★ *Le déplacement est calculé à l'assemblage*
   DJNZ R10,TEST

d) Instructions à trois opérandes

Cette catégorie regroupe les instructions réalisant un branchement conditionnel après une comparaison bit à bit de l'opérande SOURCE et de l'opérande DESTINATION.
L'opérande SOURCE est une valeur immédiate ou un registre, l'opérande DESTINATION, un registre et le déplacement un déplacement signé sur 8 bits.

FORME : **[INSTRUCTION] %OP,Rn,DEP**
[INSTRUCTION] Rn,Rn,DEP
Ex. **BTJO %10,A,FIN** ★ *Le déplacement est calculé à l'assemblage*
BTJZP A,P12,SCRUTE

★ Dans le cas des instructions concernant les registres d'entrée/sortie, l'opérande SOURCE ne peut être que les registres **A** et **B** ou une valeur immédiate.

4) LES MODES D'ADRESSAGES DU TMS 7000

Il existe sept modes d'adressage :
— l'adressage **IMPLICITE,**
— l'adressage **IMMÉDIAT,**
— l'adressage **DIRECT,**
— l'adressage **ABSOLU,**
— l'adressage **INDEXE,**
— l'adressage **INDIRECT,**
— l'adressage **RELATIF.**

a) L'adressage IMPLICITE

Il n'y a pas d'opérande (NOP, IDLE), ou alors il est sous entendu :
Registre **ST** dans les instructions **CLRC, DINT, EINT, SETC.**
Registre **PC** dans les instructions **RETI, RETS.**
Registre **SP** dans les instructions **LDSP, STSP.**
Registre **A** ou **B** dans les instructions **TSTA, TSTB.**

b) L'adressage IMMÉDIAT

L'opérande SOURCE est une constante 8 bits, l'opérande DESTINATION est un registre ou un registre d'entrée/sortie.
La présence d'un opérande immédiat est signalée par les symbole **%**.

Ex. **ADD %15,R10** : ajoute 15 (décimal) au registre R10.
MOVP % >AO,P23 : remplace le contenu de P23 par >AO (héxadécimal).

★ Cas particulier :
L'instruction **MOVD** manipule des valeurs 16 bits. La paire de registres constituant l'opérande DESTINATION est désignée par le registre de **POIDS LE PLUS FORT** (numéro le plus grand) :
MOVD % >FO98,R12 : copie la constante 16 bits >FO98 dans dans R11 et R12. R11 = > FO,
R12 = > 98.

c) L'adressage DIRECT

L'opérande SOURCE et l'opérande DESTINATION sont des registres ou des registres d'entrée/sortie.
Les instructions utilisent alors le CONTENU des registres.
Le résultat est toujours contenu dans le registre **destination.**
Ex. **SUB R2,R3** : ajoute le contenu de R2 à celui de R3.
Le résultat est rangé dans R3.

★ Cas particulier :
L'instruction **DECD** agit sur une paire de registres, qui est désignée par le registre de **POIDS LE PLUS FORT** (numéro le plus grand) :
MOVD % >A200,R15 : copie % >A200 dans R14 et R15,
DECD R15 décrémente la paire de registres R14,R15,
qui contient alors >A1FF.

d) L'adressage ABSOLU

On utilise un emplacement mémoire repéré par une adresse 16 bits.
Ces adresses sont précédées du symbole @ (arobas).

Ex. **LDA @ >F000** : copie le contenu de l'adresse >F000 dans A.
STA @ >A200 : copie le contenu de A à l'adresse >A200.
BR @1234 : branchement à l'adresse décimale 1234.

e) L'adressage INDEXE

L'adresse de l'opérande SOURCE ou de l'opérande DESTINATION est calculée en ajoutant le contenu du registre **B** considéré comme un nombre non signé, à une adresse de base.
L'adressage indexé est signalé par la présence du registre **B** entre parenthèses après l'adresse.

Ex. **MOV %>0B,B** : initialise le registre **B.**
LDA @ >9F00(B) : copie dans A le contenu de l'adresse >9F0B (>9F00 + >0B).

f) L'adressage INDIRECT

L'adresse de l'opérande est contenu dans deux registres contigus.
L'adressage indirect est signalé par le symbole * .

ATTENTION : Dans tous les cas, la paire de registre utilisée est définie par le numéro du registre de POIDS LE PLUS FORT (de plus grand numéro).

Ex. **MOV % >7F.A** : >7F dans A.
STA @ >8150 : >7F en @ >8150.
MOVD % >8150,R11 : copie % >8150 dans R10 et R11.
LDA * R11 : récupère le contenu de l'adresse pointée par R10 et R11 (soit @ >8150) dans A (soit >7F).

g) L'adressage RELATIF

L'adressage relatif est utilisé exclusivement par les instructions de saut (sauf **BR**).
L'adresse de branchement est relative à **PC.**
Elle est calculée en additionnant l'adresse de l'instruction suivant l'instruction de saut au déplacement **DEP** donné, considéré comme une valeur signée sur 8 bits ($-128 \leqslant$ **DEP** $\leqslant +127$).
L'assembleur se charge de calculer, à partir de la position courante de **PC** et de l'adresse destination, la valeur du déplacement.

Ex.
```
DÉBUT   CMP   %00,A
        JEQ   NUL      : sauf à l'adresse NUL si A vaut 0.
        MOV   %12,B
FIN     RETS
NUL     MOV   %>FF,B
        RETS
```

ADC (add with carry)

SYNTAXE : ADC (s).(d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) + (d) + C –> (d)
EXEMPLE : ADC R12,R14
INDICATEURS : C = 1 si le résultat est supérieur à > FF.
Z, N selon le résultat.

ADC additionne l'opérande SOURCE et la retenue à l'opérande DESTINATION.
Le résultat est stocké dans l'opérande DESTINATION.

ADD (add)

SYNTAXE : ADD (s).(d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) + (d) –> (d)
EXEMPLE : ADD A,R10
INDICATEURS : C = 1 si le résultat est supérieur à > FF.
Z, N selon le résultat.

ADD additionne l'opérande SOURCE à l'opérande DESTINATION.
Le résultat est stocké dans l'opérande DESTINATION.

AND (add with carry)

SYNTAXE : AND (s).(d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) ET (d) –> (d)
EXEMPLE : AND % > A2,B
INDICATEURS : C = 0
Z, N selon le résultat.

AND effectue un ET logique entre l'opérande SOURCE et l'opérande DESTINATION.
Le résultat est stocké dans l'opérande DESTINATION.

ANDP (and peripheral register)

SYNTAXE : ANDP (s).(d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) ET (d) –> (d)
EXEMPLE : AND %> A2,B
INDICATEURS : C = 0
Z, N selon le résultat.

ANDP effectue un ET logique entre l'opérande SOURCE et l'opérande DESTINATION, qui est un registre d'entrée/sortie.
Le résultat est stocké dans l'opérande DESTINATION.

BTJO (bit test and jump if one)

SYNTAXE : BTJO (s).(d).(dep)
TYPE : DOUBLE REGISTRE et RELATIF
RÉSULTAT : PC = PC + (dep) si [(s) et (d)] <> 0
EXEMPLE : BTJO %> E2,R12,BOUCLE
INDICATEURS : C = 0
Z, N selon le résultat.

BTJO effectue un ET logique entre l'opérande SOURCE et l'opérande DESTINATION. Si le résultat est différent de zéro, il y a un saut à l'adresse indiquée.
Les opérandes ne sont pas modifiés.

BTJOP (bit test and jump if one peripheral)

SYNTAXE : BTJOP (s).(d).(dep)
TYPE : REGISTRE PÉRIPHÉRIQUE et RELATIF
RÉSULTAT : PC = PC + (dep) si [(s) et (d)] <> 0
EXEMPLE : BTJOP %> 30,P20,SCRUTE
INDICATEURS : C = 0
Z, N selon le résultat.

BTJOP effectue un ET logique entre l'opérande SOURCE et l'opérande DESTINATION, qui est un registre d'entrée/sortie. Si le résultat est différent de zéro, il y a un saut à l'adresse indiquée.
Les opérandes ne sont pas modifiés.

BTJZ (bit test and jump if zero)

SYNTAXE : BTJZ (s).(d).(dep)
TYPE : DOUBLE REGISTRE et RELATIF
RÉSULTAT : PC = PC + (dep) si [(s) et NON(d)] <> 0
EXEMPLE : BTJZ %> E2,R12,BOUCLE
INDICATEURS : C = 0
Z, N selon le résultat.

BTJZ effectue un ET logique entre l'opérande SOURCE et le complément à un (NON LOGIQUE) de l'opérande DESTINATION. Si le résultat est différent de zéro, il y a un saut à l'adresse indiquée.
Les opérandes ne sont pas modifiés.

BTJZP (bit test and jump if zero peripheral)

SYNTAXE : BTJZP (s).(d).(dep)
TYPE : REGISTRE PÉRIPHÉRIQUE et RELATIF
RÉSULTAT : PC = PC + (dep) si [(s) et NON(d)] <> 0
EXEMPLE : BTJZP %> E2,R12,BOUCLE
INDICATEURS : C = 0
Z, N selon le résultat.

BTJZP effectue un ET logique entre l'opérande SOURCE et le complément à un (NON LOGIQUE) de l'opérande DESTINATION, qui est un registre d'entrée/sortie. Si le résultat est différent de zéro, il y a un saut à l'adresse indiquée.
Les opérandes ne sont pas modifiés.

BR (branch)

SYNTAXE : BR (d)
TYPE : ÉTENDU
RÉSULTAT : (d) –> PC
EXEMPLE : BR @TABLE(B)
INDICATEURS : aucune action

Branchement à l'adresse 16 bits spécifiée. Cette adresse peut être obtenue au moyen d'un des trois modes d'adressage de l'adressage étendu.

CALL (call)

SYNTAXE : CALL (d)
TYPE : ÉTENDU
RÉSULTAT : SP + 1 –> SP, PC MSB –> PILE
: SP + 1 –> SP, PC LSB –> PILE
: (d) –> PC
EXEMPLE : CALL @TEST
INDICATEURS : AUCUNE ACTION

CALL permet l'appel à un sous-programme. La valeur courante de PC (adresse de l'instruction suivant immédiatement CALL) est empilée (octet de poids fort puis octet de poids faible) et il y a branchement à l'adresse 16 bits spécifiée. Cette adresse peut être obtenue au moyen d'un des trois modes d'adressage de l'adressage étendu.

CLR (clear)

SYNTAXE : CLR (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : 0 –> (d)
EXEMPLE : CLR B
INDICATEURS : C = 0, N = 0, Z = 1

Remplace le contenu de l'opérande DESTINATION, qui est un registre, par zéro.

CLRC (clear carry)

SYNTAXE : CLRC
TYPE : IMPLICITE
RÉSULTAT : 0 –> C
EXEMPLE : CLRC
INDICATEURS : C = 0, N et Z suivant le contenu de A

CLRC permet de mettre à zéro l'indicateur de retenue. Cette instruction est la même que TSTA : les indicateurs N et Z sont donc positionnés selon la valeur de A.

CMP (compare)

SYNTAXE : CMP (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : Calcul de [(d) – (s)]. Le résultat positionne les indicateurs du registre d'état.
EXEMPLE : CMP A,R12
INDICATEURS : C = 1 si (d) est logiquement supérieur ou égal à (s),
N = 1 si le résultat de la soustraction est négatif,
Z = 1 si (d) est égal à (s).

Les indicateurs du registre d'état sont positionnés d'après le résultat de la soustraction entre le contenu de l'opérande DESTINATION et de l'opérande SOURCE. Les opérandes ne sont pas modifiés.

CMPA (compare A)

SYNTAXE : CMPA (s)
TYPE : ÉTENDU
RÉSULTAT : Calcul de [(A) – (s)]. Le résultat positionne les indicateurs du registre d'état.
EXEMPLE : CMPA * R10
INDICATEURS : C = 1 si (A) est logiquement supérieur ou égal à (s),
N = 1 si (A) est arithmétiquement inférieur à (s),
Z = 1 si (A) est égal à (s).

Les indicateurs du registre d'état sont positionnés d'après le résultat de la soustraction entre le contenu de A et celui de l'adresse s, qui peut être obtenue au moyen d'un des trois modes d'adressage de l'adressage étendu.
Les opérandes ne sont pas modifiés.

DAC (decimal add with carry)

SYNTAXE : DAC (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) + (d) + C –> (d)
EXEMPLE : DAC R12,R14
INDICATEURS : C = 1 si le résultat est supérieur ou égal à 100,
Z, N selon le résultat

DAC additionne l'opérande SOURCE et la retenue à l'opérande DESTINATION. Cette addition est réalisée en BCD (décimal codé binaire). Les deux opérandes doivent être des valeurs BCD correctes pour que le résultat soit cohérent.
Le résultat est stocké dans l'opérande DESTINATION.

DEC (decrement)

SYNTAXE : DEC (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : (d) – 1 –> (d)
EXEMPLE : DEC R10
INDICATEURS : C = 0 s'il y a passage de >00 à >FF,
Z, N selon le résultat.

DEC soustrait 1 à une copie du contenu de l'opérande DESTINATION qui est un registre.
Le résultat est stocké dans l'opérande DESTINATION.

DECD (decrement double)

SYNTAXE : DECD (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : (d) – 1 –> (d)
EXEMPLE : DECD R14
INDICATEURS : C = 0 si le MSB passe de >00 à >FF,
N, Z selon la valeur du MSB.

DECD soustrait 1 à une copie du contenu de la paire de registres spécifiée.
L'opérande DESTINATION est le registre de poids faible (LSB) de la paire de registres (dans l'exemple : LSB = R14, MSB = R13, on décrémente R13-R14).
C'est donc une opération sur 16 bits. Les indicateurs du registre d'état sont positionnés selon le contenu du registre de poids fort (MSB).
Le résultat est stocké dans la paire de registre.

DINT (disable interrupts)

SYNTAXE : DINT
TYPE : IMPLICITE
RÉSULTAT : 0 – 1 –> I
EXEMPLE : DINT
INDICATEURS : C = 0, N = 0, Z = 0, I = 0

DINT interdit toutes les interruptions en mettant à zéro l'indicateur I.

DJNZ (decrement and jump if non zero)

SYNTAXE : DJNZ (d) , (dep)
TYPE : SIMPLE REGISTRE ET RELATIF
RÉSULTAT : (d) – 1 –> (d)
si (d) <> 0 alors PC + (dep) –> PC
EXEMPLE : DJNZ R12,BOUCLE
INDICATEURS : AUCUNE ACTION.

DJNZ soustrait 1 à l'opérande DESTINATION (un registre) et range le résultat dans l'opérande DESTINATION. Si le résultat de l'opération n'est pas zéro, il y a saut à l'adresse indiquée.

DSB (decimal substract with borrow)

SYNTAXE : DBS (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (d) – (s) – 1 + C –> (d)
EXEMPLE : DSB R12,R14
INDICATEURS : C = 1 si le résultat est inférieur à 0,
Z, N selon le résultat.

DSB soustrait l'opérande SOURCE et la retenue de l'opérande DESTINATION.
Cette soustraction est réalisée en BCD (décimal codé binaire). Les deux opérandes doivent être des valeurs BCD correctes pour le résultat soit cohérent.
Le résultat est stocké dans l'opérande DESTINATION.

EINT (enable interrupts)

SYNTAXE : DINT
TYPE : IMPLICITE
RÉSULTAT : 1 –> I
EXEMPLE : DINT
INDICATEURS : C = 1, N = 1, Z = 1, I = 1.

EINT autorise toutes les interruptions en mettant à zéro l'indicateur I.

IDLE (idle until interrupt)

SYNTAXE : IDLE
TYPE : IMPLICITE
RÉSULTAT : attente d'interruption
EXEMPLE : IDLE
INDICATEURS : AUCUNE ACTION.

IDLE place le processeur en attente d'interruption.

INC (increment)

SYNTAXE : DEC (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : (d) + 1 –> (d)
EXEMPLE : INC A
INDICATEURS : C = 1 s'il y a passage de >FF à >00,
N, Z selon le résultat.

DEC ajoute 1 à une copie du contenu de l'opérande DESTINATION qui est un registre.
Le résultat est stocké dans l'opérande DESTINATION.

INV (invert)

SYNTAXE : INV (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT :[NON (d)] –> (d)
EXEMPLE : INV A
INDICATEURS : C = 0
N, Z selon le résultat.

INV réalise le complément à un de (NON LOGIQUE) de l'opérande DESTINATION qui est un registre.
Le résultat est stocké dans l'opérande DESTINATION.

JMP (jump)

SYNTAXE : JMP (dep)
TYPE : RELATIF
RÉSULTAT : PC + (dep) —> PC
EXEMPLE : JMP SUITE
INDICATEURS : AUCUNE ACTION

JMP permet de faire un saut inconditionnel à une adresse calculée à partir de PC et du déplacement (dep), considéré comme un nombre signé compris entre −128 et +127.

J(cond) (jump on condition)

Les instructions du groupe **J(cond)** permettent de réaliser ou non des sauts selon la position des indicateurs du registre d'état.
SYNTAXE : J(cond) (dep)
TYPE : RELATIF
RÉSULTAT : si la condition est remplie : PC + (dep) —> PC
EXEMPLE : JEQ TERMINÉ
INDICATEURS : AUCUNE ACTION.

Les instructions **J(cond)** permettent de faire un saut conditionnel à une adresse calculée à partir de PC et de déplacement (dep), considéré comme un nombre signé compris entre −128 et +127.
Les trois indicateurs C, N et Z peuvent être testés séparément.
Les instructions disponibles sont :

			ÉTAT DES BITS POUR SAUT		
INSTRUCTION	MNÉMONIQUE	CODE	C	N	Z
saut si retenue	JC	E3	1	×	×
saut si égal à zéro	JEQ	E2	×	×	1
saut si supérieur ou égal	JHS	E3	1	×	×
saut si inférieur	JL	E7	0	×	×
saut si négatif	JN	E3	×	1	×
saut si pas de retenue	JNC	E7	0	×	×
saut si non égal	JNE	E6	×	×	0
saut si différent de zéro	JNZ	E6	×	×	0
saut si positif	JP	E4	×	0	0
saut si positif ou nul	JPZ	E5	×	0	×
saut si nul	JZ	E2	×	×	1
saut si plus petit	JLT	E1	×	1	×
saut si plus grand	JGT	E4	×	0	0
saut si plus grand ou égal	JGE	E5	×	0	×

LDA (load A register)

SYNTAXE : LDA (s)
TYPE : ÉTENDU
RÉSULTAT : (s) —> (A)
EXEMPLE : LDA @>F800
INDICATEURS : C = 0
N, Z selon la valeur chargée.

LDA permet de récupérer dans A une valeur stockée à n'importe quelle adresse s.
L'adresse s peut être obtenue au moyen d'un des trois modes d'adressage de l'adressage étendu.

LDSP (load stack pointer)

SYNTAXE : LDSP
TYPE : IMPLICITE
RÉSULTAT : (B) —> SP
EXEMPLE : LDSP
INDICATEURS : AUCUNE ACTION

LDSP permet d'initialiser le pointeur de pile, avec le contenu du registre B.

MOV (move)

SYNTAXE : MOVE (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) —> (d)
EXEMPLE : MOV A,R14
MOV %10,R12
INDICATEURS : C = 0
Z, N selon la valeur copiée.

MOV permet de lire ou d'écrire dans un registre.

MOVD (move double)

SYNTAXE : MOVD (s) , (d)
TYPE : SPÉCIAL
RÉSULTAT : (s) —> (d)
EXEMPLE : MOVD R11,R21
MOVD %>1020,B
INDICATEURS : C = 0
Z, N selon l'octet de poids fort de la valeur copiée.

MOVD permet de transférer une valeur de 16 bits dans une paire de registres, spécifiée par le registre de poids faible (LSB).
Les indicateurs du registre d'état sont positionnés selon la valeur de l'octet de poids fort (MSB).
Adressages autorisés :

- immédiat	MOVD %>1020,B	: >1020	—>	(A,B)
- direct	MOVD R11,R21	: (R10,R11)	—>	(R20,R21)
- indexé	MOVD %TABLE(B),R16	: TABLE+B	—>	(R15,R16)

MOVP (move to/from peripheral)

SYNTAXE : MOVP (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) —> (d)
EXEMPLE : MOVP A,P4
MOVP P16,B
INDICATEURS : C = 0,
Z, N selon la valeur copiée.

MOVP permet de lire ou d'écrire dans un registre d'entrée/sortie

MPY (multiply)

SYNTAXE : MPY (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (S) * (d) —> (A,B)
EXEMPLE : MPY R12,R14
MPY %10,R12
INDICATEURS : C = 0,
Z, N selon l'octet de poids fort du résultat.

MPY permet d'effectuer une multiplication entre deux opérandes de 8 bits, le résultat sur 16 bits se trouvant toujours dans A et B, A contenant l'octet de poids fort (MSB).

NOP (no operation)

SYNTAXE : NOP
TYPE : IMPLICITE
RÉSULTAT : PC + 1 —> PC
EXEMPLE : NOP
INDICATEURS : AUCUNE ACTION.

NOP est souvent utilisée pour éliminer des instructions erronées sans modifier un programme.

OR (or)

SYNTAXE : OR (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) OU (d) —> (d)
EXEMPLE : OR %>FE,A
INDICATEURS : C = 0,
Z, N selon le résultat.

OR réalise un OU LOGIQUE entre l'opérande SOURCE et l'opérande DESTINATION.
Le résultat est stocké dans l'opérande DESTINATION.

ORP (or peripheral)

SYNTAXE : ORP (s) , (d)
TYPE : REGISTRE PÉRIPHÉRIQUE
RÉSULTAT : (s) OU (d) —> (d)
EXEMPLE : ORP %>04,P10
INDICATEURS : C = 0,
Z, N selon le résultat.

OR réalise un OU LOGIQUE entre l'opérande SOURCE et l'opérande DESTINATION, qui est un registre d'entrée/sortie.
Le résultat est stocké dans l'opérande DESTINATION.

POP (pop from stack)

SYNTAXE : POP (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : PILE(SP) —> (d)
SP – 1 —> SP
EXEMPLE : POP A
POP ST ★ *Cas particulier*
INDICATEURS : C = 0,
N, Z selon la valeur dépilée.

POP permet de dépiler une valeur : la valeur située au sommet de la pile est transférée dans le registre destination, puis le pointeur de pile est décrémenté.
POP ST place la valeur située au sommet de la pile dans le registre d'état.

PUSH (push on stack)

SYNTAXE : PUSH (s)
TYPE : SIMPLE REGISTRE
RÉSULTAT : SP + 1 —> SP
(s) —> PILE(SP)
EXEMPLE : PUSH R12
PUSH ST ★ *Cas particulier*
INDICATEURS : C = 0,
N, Z selon la valeur empilée.

PUSH permet d'empiler une valeur : la valeur contenue dans le registre source est placée au sommet de la pile, puis le pointeur de pile est incrémenté.
PUSH ST place le contenu du registre d'état au sommet de la pile.

RETI (return form interrupt)

SYNTAXE : RETI
TYPE : IMPLICITE
RÉSULTAT : PILE(SP) —> PC (LSB) puis SP – 1 —> SP
PILE(SP) —> PC (MSB) puis SP – 1 —> SP
PILE(SP) —> ST (LSB) puis SP – 1 —> SP
EXEMPLE : RETI
INDICATEURS : LE REGISTRE D'ÉTAT EST CHARGÉ A PARTIR DE LA PILE.

RETI est la dernière instruction d'une routine de gestion d'interruption. L'état du processeur est restauré et l'exécution continue à l'endroit où l'interruption a été prise en compte.

RETS (return from subroutine)

SYNTAXE : RETS
TYPE : IMPLICITE
RÉSULTAT : PILE(SP) —> PC (LSB) puis SP – 1 —> SP
PILE(SP) —> PC (MSB) puis SP – 1 —> SP
EXEMPLE : RETS
INDICATEURS : AUCUNE ACTION.

RETS est la dernière instruction d'un sous-programme. L'exécution continue à l'adresse de l'instruction qui suit immédiatement l'appel (CALL).

RL (rotate left)

SYNTAXE : RL (s)
TYPE : SIMPLE REGISTRE
RÉSULTAT : BIT(N) —> BIT(N – 1)
C —> BIT(0)
BIT(7) —> C
EXEMPLE : RL A
INDICATEURS : C = BIT(7) DE LA VALEUR INITIALE
N, Z selon le résultat.

RL effectue un décalage d'une position vers la gauche de tous les bits de l'opérande SOURCE. Le bit 7 de la valeur initiale est alors copié dans C et dans le bit 0 du résultat.

RLC (rotate left through carry)

SYNTAXE : RLC (s)
TYPE : SIMPLE REGISTRE
RÉSULTAT : BIT(N) —> BIT(N – 1)
C —> BIT(0)
BIT(7) —> C
EXEMPLE : RLC R10
INDICATEURS : C = BIT(7) DE LA VALEUR INITIALE
N, Z selon le résultat.

RLC effectue un décalage d'une position vers la gauche de tous les bits de l'opérande SOURCE. Le bit 0 est remplacé par le contenu de C, puis C est remplacé par le bit 7 de la valeur initiale.

RR (rotate right)

SYNTAXE : RR (s)
TYPE : SIMPLE REGISTRE
RÉSULTAT : BIT(N + 1) —> BIT(N)
BIT(0) —> BIT(7) et C
EXEMPLE : RR B
INDICATEURS : C = BIT(0) DE LA VALEUR INITIALE
N, Z selon le résultat.

RR effectue un décalage d'une position vers la droite de tous les bits de l'opérande SOURCE. Le bit 0 de la valeur initiale est alors copié dans C et dans le bit 7 du résultat.

RRC (rotate right through carry)

SYNTAXE : RRC (s)
TYPE : SIMPLE REGISTRE
RÉSULTAT : BIT(N + 1) —> BIT(N)
C —> BIT(7)
BIT(0) —> C
EXEMPLE : RRC A
INDICATEURS : C = BIT(0) DE LA VALEUR INITIALE
N, Z selon le résultat.

RRC effectue un décalage d'une position vers la droite de tous les bits de l'opérande SOURCE. Le bit 7 est remplacé par le contenu de C, puis C est remplacé par le bit 0 de la valeur initiale.

SBB (substract with borrow)

SYNTAXE : SBB (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (d) − (s) − 1 + C —> (d)
EXEMPLE : SBB %>12,A
INDICATEURS : C = 0 s'il y a une retenue. C = 1 sinon,
Z, N selon le résultat.

SETC (set carry)

SYNTAXE : SETC
TYPE : IMPLICITE
RÉSULTAT : 1 —> C
EXEMPLE : SETC
INDICATEURS : C = 1, N = 0, Z = 1.

SETC permet de mettre à un l'indicateur de retenue.

STA (store A register)

SYNTAXE : STA (d)
TYPE : ÉTENDU
RÉSULTAT : A —> (d)
EXEMPLE : STA @>A200
INDICATEURS : C = 0,
N, Z selon la valeur stockée.

STA permet de ranger le contenu de A dans n'importe quelle adresse s. L'adresse s peut être obtenue au moyen d'un des trois modes d'adressage de l'adressage étendu.

STSP (store stack pointer)

SYNTAXE : LDSP
TYPE : IMPLICITE
RÉSULTAT : (SP) —> B
EXEMPLE : STSP
INDICATEURS : AUCUNE ACTION.

STSP permet de récupérer la valeur du pointeur de pile dans le registre B.

SUB (substract)

SYNTAXE : SUB (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (d) – (s) –> (d)
EXEMPLE : SUB %>12,B
INDICATEURS : C = 1 si le résultat est supérieur ou égal à zéro,
Z, N selon le résultat.

SUB soustrait l'opérande SOURCE de l'opérande DESTINATION. Le résultat est stocké dans l'opérande DESTINATION.

TRAP (trap to subroutine)

SYNTAXE : TRAP(n)
TYPE : SPÉCIAL
RÉSULTAT : SP + 1 –> SP puis PC MSB –> PILE (SP)
SP + 1 –> SP puis PC LSB –> PILE (SP)
adresse du TRAP –> PC
EXEMPLE : TRAP 8
INDICATEURS : AUCUNE ACTION

Un **TRAP** est une forme particulière d'appel à un sous-programme : l'adresse de la routine se trouve dans une table située en ROM de >FFD0 à >FFF0.
L'opérande est un numéro qui indique une adresse particulière dans la table.
La table est ainsi construite :

>FFFF	TRAP 0	ADRESSE BASSE (LSB)
>FFFE	TRAP 0	ADRESSE HAUTE (MSB)
>FFFD	TRAP 1	ADRESSE BASSE (LSB)
>FFFC	TRAP 1	ADRESSE HAUTE (MSB)
>FFFB	TRAP 2	ADRESSE BASSE (LSB)
>FFFA	TRAP 2	ADRESSE HAUTE (MSB)
.....	.....	
.....	.....	
.....	.....	
>FFE1	TRAP 15	ADRESSE BASSE (LSB)
>FFE0	TRAP 15	ADRESSE HAUTE (MSB)
.....	.....	
.....	.....	
.....	.....	
>FFD1	TRAP 23	ADRESSE BASSE (LSB)
>FFD0	TRAP 23	ADRESSE HAUTE (MSB)

TSTA (test A register)

SYNTAXE : TSTA
TYPE : IMPLICITE
RÉSULTAT : positionne les indicateurs d'état suivant la valeur de A.
EXEMPLE : TSTA
INDICATEURS : C = 0,
N et Z suivant le contenu de A.

TSTA positionne les indicateurs N et Z selon la valeur de A. C est mis à zéro. Cette instruction est la même que CLRC.

TSTB (test B register)

SYNTAXE : TSTB
TYPE : IMPLICITE
RÉSULTAT : positionne les indicateurs d'état suivant la valeur de B
EXEMPLE : TSTB
INDICATEURS : C = 0,
N et Z suivant le contenu de B.

TSTB positionne les indicateurs N et Z selon la valeur de B. C est mis à zéro.

XCHB (test B register)

SYNTAXE : XCHB (d)
TYPE : SIMPLE REGISTRE
RÉSULTAT : (B) < – > (d)
EXEMPLE : XCHB
INDICATEURS : C = 0,
N et Z suivant le contenu initial de B.

XCHB réalise un échange entre le contenu du registre B et le contenu du registre destination.

XOR (exclusive or)

SYNTAXE : XOR (s) , (d)
TYPE : DOUBLE REGISTRE
RÉSULTAT : (s) OU EXCLUSIF (d) —> (d)
EXEMPLE : XOR %>12,A
INDICATEURS : C = 0,
Z, N selon le résultat.

XOR réalise un OU EXCLUSIF entre l'opérande SOURCE et l'opérande DESTINATION. Le résultat est stocké dans l'opérande DESTINATION.

XORP (exclusif or peripheral)

SYNTAXE : ORP (s) , (d)
TYPE : REGISTRE PÉRIPHÉRIQUE
RÉSULTAT : (s) OU (d) —> (d)
EXEMPLE : ORP %>04,P10
INDICATEURS : C = 0,
Z, N selon le résultat.

XOR réalise un OU EXCLUSIF entre l'opérande SOURCE et l'opérande DESTINATION, qui est un registre d'entrée/sortie.

LA GESTION DE L'ÉCRAN

Les échanges entre le CPU (micro-processeur central) et l'écran de visualisation se font via un processeur spécialisé, le **TMS 3556**, que l'on désigne par VDP : Video Display Processor. Ce processeur, qui utilise le mode RVB (Rouge Vert Bleu ou RGB : Red Green Blue) pour l'affichage des couleurs, est programmable par l'intermédiaire de registres de contrôle et dispose d'une mémoire propre (VRAM) qui peut aller jusqu'à 64 Ko.
C'est dans la VRAM que l'utilisateur définira, selon ses besoins, les emplacements mémoires alloués à la page-écran et aux différents générateurs de caractères, si nécessaire. La place disponible pourra alors être utilisée comme une zone de mémoire RAM normale, à une exception près : on ne pourra y lancer l'exécution de routines en code machine car cette mémoire n'est pas DIRECTEMENT accessible par le CPU.

REMARQUE : l'objectif de ce chapitre est de donner les éléments d'information indispensables à la réalisation de routines en assembleur utilisant l'écran. Pour une étude plus technique, reportez-vous au MANUEL DE L'UTILISATEUR DU TMS 3556.

1) LES MODES D'AFFICHAGE

1. Le mode TEXTE (Text Mode) ou SEMI-GRAPHIQUE

L'écran est organisé en 25 lignes de 40 caractères.
Dans ce mode, seuls les caractères dont les dessins ont été prédéfinis peuvent être affichés. Les dessins de ces caractères sont stockés en VRAM, dans des zones mémoire appelées « générateurs de caractères ».

a. Organisation de la mémoire
On utilise 2 octets pour décrire un caractère : le premier contient les attributs du caractère : couleur, taille, ..., le second contient le code du caractère sur 7 bits, le bit restant permettant de coder un attribut supplémentaire.
Une page d'écran en mode texte occupe donc 2 × 25 × 40 octets, c'est à dire 2 000 octets (figures 1 et 2).
L'adresse en mémoire de l'écran, qui porte le nom de BAPA, doit être initialisée (voir : initialisation du VDP).

b. Les caractères et les générateurs de caractères

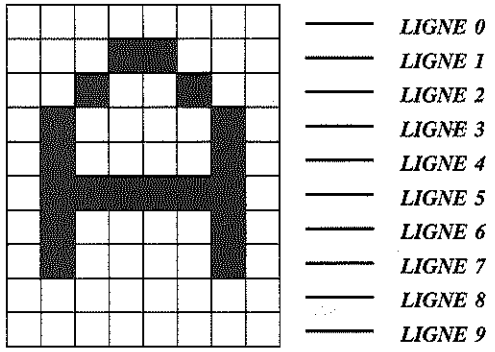
Le VDP peut afficher 128 caractères différents, codés par un nombre de 0 à 127. Chaque caractère est défini par son CODE et sa GRAPHIE, ou plus simplement, le DESSIN qui lui est associé.

Le code d'un caractère renvoit en fait le VDP à un emplacement en mémoire contenant la description du caractère. La zone mémoire regroupant le dessin des 128 caractères affichables est appelée « GÉNÉRATEUR DE CARACTÈRES ».

1. Description d'un caractère

Chaque caractère est dessiné sur 10 lignes de 8 points (on dit une matrice 8 × 10).

Exemple :



Chaque ligne correspond à un octet. Un point est « allumé » quand le bit correspondant dans l'octet est à 1.

2. Description d'un générateur

Un générateur de caractères contient 128 caractères et un caractère est défini par 10 octets : un générateur de caractères occupe donc 128 × 10 octets, soit 1 280 octets.

La zone mémoire associée à un générateur de caractères est désignée par l'adresse de son premier octet, qui porte le nom :

BAGC0 pour l'adresse de base du générateur de caractères 0
BAGC1 pour l'adresse de base du générateur de caractères 1
BAGC2 pour l'adresse de base du générateur de caractères 2
BAGC3 pour l'adresse de base du générateur de caractères 3

Les dessins des caractères sont stockés à partir de cette adresse d'une façon assez particulière.

L'adresse BAGCn (avec n = 0, 1, 2 ou 3) contient la dernière ligne (ligne 9) du premier caractère (code 0), l'octet suivant contient la dernière ligne (ligne 9) du deuxième caractère (code 1), et ainsi de suite, jusqu'à l'adresse BAGCn + 127, qui contient la dernière ligne (ligne 9) du dernier caractère (code 127).

Cette structure est reprise à partir de BAGCn + 128 pour stocker l'avant-dernière ligne (ligne 8) des 128 caractères, et ainsi de suite jusqu'à la première ligne (ligne 0) des 128 caractères (figure 3).

3. Les deux types de générateurs

Le VDP n'affiche pas de la même façon les caractères des quatre générateurs de caractères. En fait, il distingue deux types de générateurs :

- les générateurs alphanumériques,
- les générateurs alphamosaïques.

La différence réside dans la façon dont sont gérés les attributs des caractères lors de l'affichage du caractère (voir paragraphe attributs) : l'organisation du générateur en mémoire reste identique.

Les générateurs BAGC0 et BAGC1 sont toujours considérés comme des générateurs de type ALPHANUMÉRIQUE.

Le générateur BAGC2 est toujours considéré comme un générateur de type ALPHAMOSAÏQUE.

Le générateur BAGC3 peut être de type ALPHANUMÉRIQUE ou ALPHAMOSAÏQUE, selon les besoins du programmeur. Le choix est fait en programmant un des registres de contrôle du VDP (voir : initialisation du VDP).

c. Attributs des caractères

Pour afficher un caractère, le VDP dispose de deux octets, soit 16 bits, contenant des informations sur le caractère.

Les 9 premiers bits (premier octet + MSB du second octet) contiennent la description des attributs du caractère, les 7 autres le code du caractère. Suivant le type du générateur, les 9 bits d'attributs sont interprétés différemment.

1. Générateur de type alphamosaïque

BF	GF	RF	CG1	CG0	BB	GB	RB	FLS	CODE DU CARACTÈRE

BF GF RF : fixent la couleur des points du caractère (FOREGROUND)

0	0	0	: NOIR	1	0	0	: BLEU
0	0	1	: ROUGE	1	0	1	: MAGENTA
0	1	0	: VERT	1	1	0	: CYAN
0	1	1	: JAUNE	1	1	1	: BLANC

CG1 CG2 : contiennent le numéro du générateur de caractères utilisé.

0	0	: BAGC0
0	1	: BAGC2
1	0	: BAGC1
1	1	: BAGC3

ATTENTION : il faut inverser les bits par rapport à la numérotation normale

BB GB RB : fixent la couleur du fond du caractère (BACKGROUND)

0	0	0	: NOIR	1	0	0	: BLEU
0	0	1	: ROUGE	1	0	1	: MAGENTA
0	1	0	: VERT	1	1	0	: CYAN
0	1	1	: JAUNE	1	1	1	: BLANC

FLS : mode clignotant

0	: affichage normal
1	: affichage clignotant

CODE DU CARACTÈRE : contient le code du caractère sur 7 bits.

2. Générateur de type alphanumérique

Ce type de générateur permet d'afficher des caractères de taille variable. Par contre, on ne peut pas spécifier la couleur du fond par caractère : elle est fixée lors de l'initialisation du registre de contrôle CM4 (voir : initialisation du VDP). Cette couleur de fond est alors utilisée pour l'affichage de tous les caractères.

BF	GF	RF	CG1	CG0	INV	DH	DW

FLS	CODE DU CARACTÈRE

BF GF RF : permettent de fixer la couleur du caractère

0	0	0	: NOIR	1	0	0	: BLEU
0	0	1	: ROUGE	1	0	1	: MAGENTA
0	1	0	: VERT	1	1	0	: CYAN
0	1	1	: JAUNE	1	1	1	: BLANC

CG1 CG2 : contiennent le numéro du générateur de caractères utilisé.

0	0	: BAGC0
0	1	: BAGC2
1	0	: BAGC1
1	1	: BAGC3

ATTENTION : il faut inverser les bits par rapport à la numérotation normale

INV : inversion des couleurs

0	: affichage normal
1	: la couleur du fond et la couleur du caractère sont inversées

DH DW : permet de fixer la taille du caractère

0	0	: affichage en taille normale
0	1	: affichage en double largeur
1	0	: affichage en double hauteur
1	1	: affichage en double taille

FLS : mode clignotant

0	: affichage normal
1	: affichage clignotant

L'affichage de caractères en plusieurs tailles obéit aux règles suivantes :

- Taille normale : Chaque caractère occupe une position sur l'écran. Le code et l'attribut du caractère n'apparaissent qu'une seule fois.
- Double hauteur : Le caractère occupe 2 lignes sur 1 colonne. Le code et l'attribut du caractère doivent apparaître 2 fois, sur deux lignes contigües et sur la même colonne.
- Double largeur : Le caractère occupe 2 colonnes sur 1 ligne. Le code et l'attribut du caractère doivent apparaître 2 fois, sur la même ligne et sur 2 colonnes contigües.
- Double taille : Le caractère occupe 4 positions : 2 colonnes sur 2 lignes. Le code et l'attribut du caractère doivent apparaître 4 fois.

3. Attribut par bloc

Dans les 2 cas précédents, 9 bits permettent de définir l'attribut caractère par caractère : il s'agit alors d'une « définition d'attribut par caractère ».

Il est possible de fixer l'attribut non pas d'un seul caractère mais d'un groupe de caractères : il s'agit alors d'une « définition d'attribut par bloc ».

L'étude de ce type de définition, très proche du mode d'affichage VIDEOTEX, par exemple, dépasse le cadre de cet ouvrage : reportez-vous au MANUEL DE L'UTILISATEUR DU TMS 3556 pour une approche complète.

2. Le mode HAUTE RÉSOLUTION (bit-mapped mode) ou GRAPHIQUE

En mode haute résolution, l'écran est constitué d'un ensemble de 250 lignes de 320 points que l'on peut « allumer » individuellement dans une des 8 couleurs disponibles. Les générateurs de caractères sont inutiles.

a. Organisation de la mémoire

Comme en mode TEXTE, BAPA pointe sur une zone mémoire contenant la description de l'image affichée par le VDP.

Il faut 3 bits pour coder un point qui peut apparaître dans une des 8 couleurs possibles. 3 octets consécutifs contiennent les trois couleurs d'un groupe de 8 points consécutifs. Il faut 40 triplets, soit 120 octets, pour représenter une ligne (figures 4 et 5).

b. Gestion de la couleur de la marge

A chaque groupe de 120 octets représentant une ligne sont ajoutés deux octets.

BM	GM	RM	XX	MR	LR	IR

XX	OCTET DE CONTRÔLE

BM GM RM : permettent de définir la couleur de la marge DE LA LIGNE SUIVANTE

MR : Mis à 1, MR provoque l'affichage de la ligne suivante dans la couleur BM, GM, RM.

LR : doit être à zéro.

IR : est lié à la gestion des signaux de sortie et au mode de synchronisation de l'écran

Le 122e octet contrôle l'état des signaux de sortie de l'écran. En mode haute résolution, il faut 250 × 122 octets, soit 30 500 octets (presque 30 Ko), pour représenter une page d'écran.

3. Le mode MIXTE (mixed mode)

Ce mode autorise le mélange, ligne à ligne, du mode texte et du mode graphique. Dans tous les cas, la première ligne de l'écran doit être une ligne « texte ». Une ligne texte occupe bien entendu l'équivalent de 10 lignes graphiques. On ne peut donc définir que des groupes de 10 lignes graphiques.

Dans ce mode, chaque ligne texte (80 octets) ou graphique (120 octets) est complétée par les deux octets suivants :

BM	GM	RM	C/G	MR	LR	IR

XX	OCTET DE CONTRÔLE

BM GM RM : permettent de définir la couleur de la marge DE LA LIGNE SUIVANTE

C/G : indique le mode de la ligne suivante : graphique (1) ou texte (0)

MR LR IR : dépendent du mode de la ligne suivante :

ligne texte : ils agissent comme les bits du même nom du registre de contrôle CM4,

ligne graphique : ils agissent comme les bits du même nom du 121e octet de la ligne.

Le 82e (ou 122e) octet contrôle l'état des signaux de sortie de l'écran. L'espace mémoire occupé par un écran en mode mixte peut être calculé au moyen de la formule :

$$Espace\ occupé = (T \times 82) + (G \times 122)$$

avec : T = nombre de lignes textes

et G = nombre de lignes graphiques.

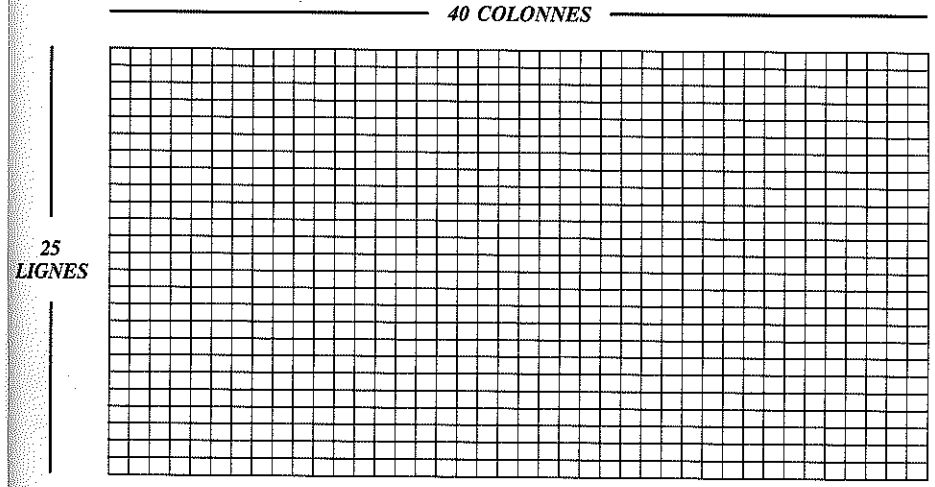


Figure 1 : Ecran en mode texte

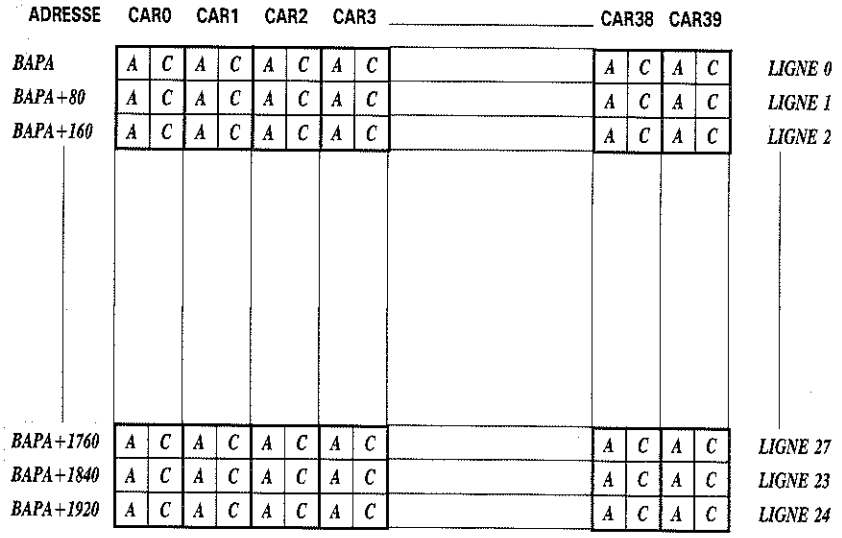


Figure 2 : Organisation de la mémoire en mode texte.

ADRESSE								
BAGCn	*CAR0 LIGNE 9*	*CAR1 LIGNE 9*	*CAR2 LIGNE 9*		*CAR125 LIGNE 9*	*CAR126 LIGNE 9*	*CAR127 LIGNE 9*	*LIGNE 9*
BAGCn + 128	*CAR0 LIGNE 8*	*CAR1 LIGNE 8*	*CAR2 LIGNE 8*		*CAR125 LIGNE 8*	*CAR126 LIGNE 8*	*CAR127 LIGNE 8*	*LIGNE 8*
BAGCn + 256								
BAGCn + 384								
BAGCn + 512								
BAGCn + 640								
BAGCn + 768								
BAGCn + 896								
BAGCn + 1024	*CAR0 LIGNE 1*	*CAR1 LIGNE 1*	*CAR2 LIGNE 1*		*CAR125 LIGNE 1*	*CAR126 LIGNE 1*	*CAR127 LIGNE 1*	*LIGNE 1*
BAGCn + 1152	*CAR0 LIGNE 0*	*CAR1*	*CAR2*		*CAR125 LIGNE 0*	*CAR126 LIGNE 0*	*CAR127 LIGNE 0*	*LIGNE 0*

Figure 3 : Organisation d'un Générateur

B B B B B B B B	*G G G G G G G G*	*R R R R R R R R*
Couleur du N^e point	*Couleur du N + 7^e point*	

Figure 4 :
Organisation en mémoire de 8 points consécutifs en mode graphique

	8 POINTS					TYPE DE LA LIGNE SUIVANTE		
ADRESSE								
BAPA	*B*	*G*	*R*	*B*		*ROW ATTRIBUTES*	*I OUTPUT*	*LINE 0*
BAPA+122	*B*	*G*	*R*	*B*		"	"	*LINE 1*
BAPA+244	*B*	*G*	*R*	*B*		"	"	*LINE 2*
BAPA+366	*B*	*G*	*R*	*B*		"	"	*LINE 3*
								LINE 244
BAPA+29 890	*B*	*G*	*R*	*B*		*ROW ATTRIBUTES*	*I OUTPUT*	*LINE 245*
BAPA+30 012	*B*	*G*	*R*	*B*		"	"	*LINE 246*
BAPA+30 134	*B*	*G*	*R*	*B*		"	"	*LINE 247*
BAPA+30 256	*B*	*G*	*R*	*B*		"	"	*LINE 248*
BAPA+30 378	*B*	*G*	*R*	*B*		"	"	*LINE 249*

250 LIGNES PAR PAGE

122 OCTETS PAR LIGNE

Figure 5 :
Organisation mémoire en mode graphique

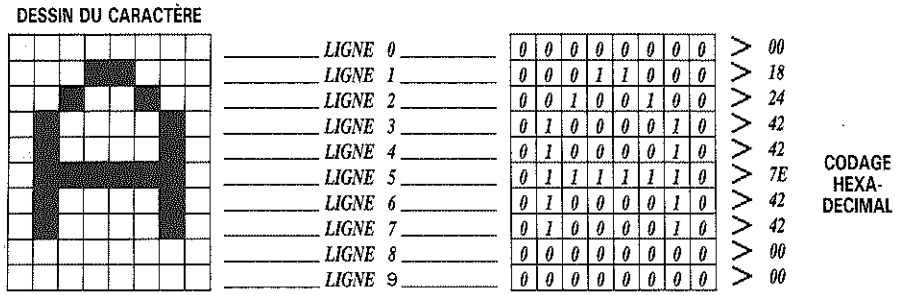


Figure 6 :
Codage des points « allumés » du dessin du caractère

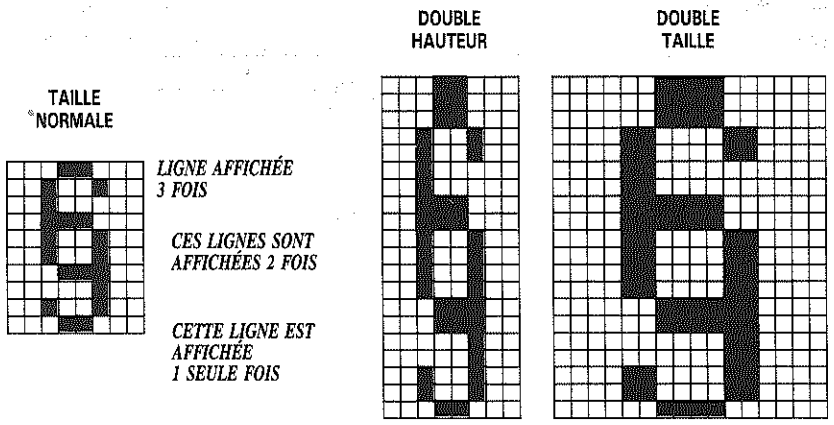


Figure 7 :
Effet de l'affichage en double taille sur le dessin des caractères

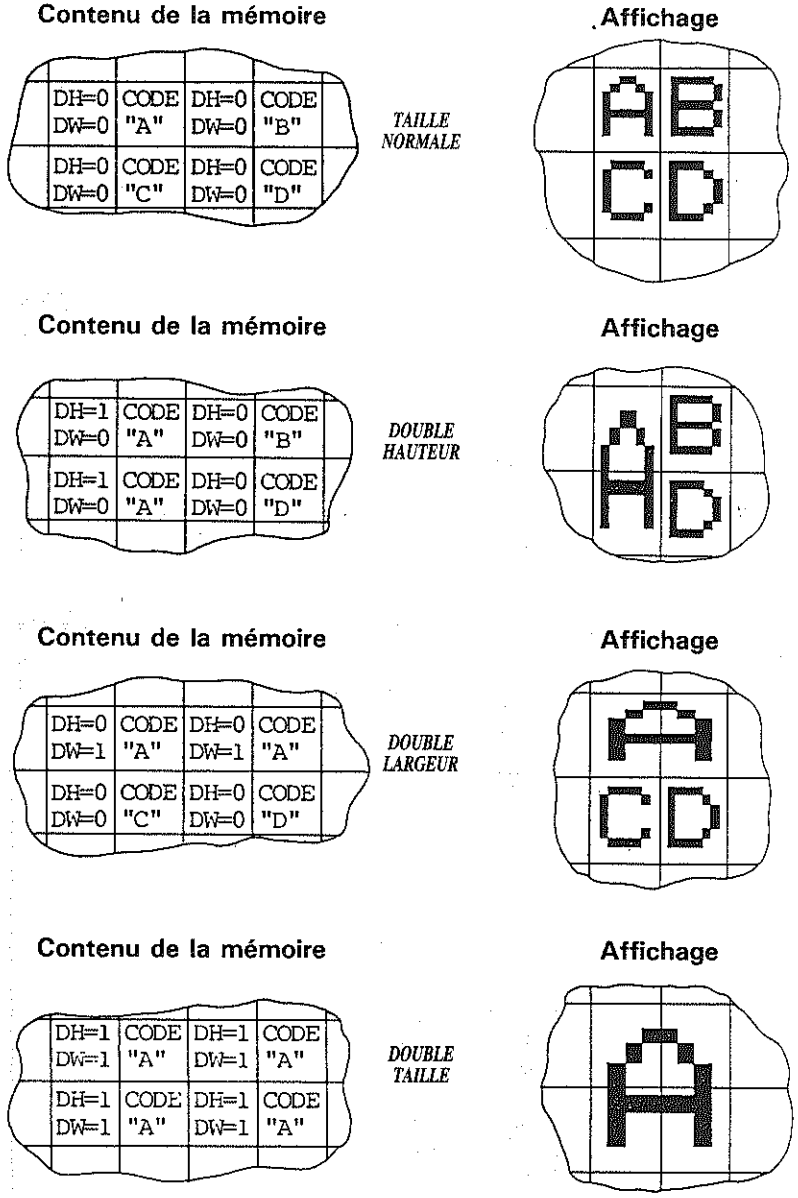


Figure 8 : Application de l'affichage en différentes tailles

2) COMMUNICATION AVEC LE VDP

Le processeur central doit pouvoir communiquer avec le VDP pour réaliser deux types d'opérations :

— l'initialisation du VDP :adresses de l'écran et des générateurs, mode d'affichage, couleur du bord de l'écran, etc. ;

— l'accès à la VRAM en écriture et en lecture, soit pour agir sur l'écran (affichage de caractères...), soit pour agir sur les générateurs de caractères (changement du dessin d'un caractère...), soit pour gérer des emplacements mémoires (tables...).

1. Initialisation du VDP

On communique avec le VDP par l'intermédiaire de 16 registres VDP, qui ont chacun leur fonction et qui sont numérotés de 0 à 15 (ou de >0 à >F).

Parmi ces 16 registres VDP, 8 sont des registres 8 bits ou chaque bit joue un rôle, les 8 autres étant des registres 16 bits destinés à contenir des adresses. Ils portent chacun un nom qui précise leur fonction :

REGISTRE	NUMÉRO	CONTENU	MSB b7	b6	b5	b4	b3	b2	b1	LSB b0
POINTEUR	*0*	*POINTEUR*	*RT8*	*RT4*	*RT2*	*RT1*	*AC8*	*AC4*	*AC2*	*AC1*
COL	*1*	*ADRESSE LSB*								
ROW	*2*	*ADRESSE MSB*								
STATUS	*3*	*ETAT DU VDP*	*ST1*	*ST2*	*ST3*	*ST4*	*ST5*	*ST6*	*ST7*	*ST8*
CM1	*4*	*BASE DE TEMPS*	*BT1*	*BT2*	*BT3*	*BT4*	*BT5*	*x*	*x*	*x*
CM2	*5*	*DECODEUR*	*DC1*	*DC2*	*DC3*	*DC4*	*DC5*	*DC6*	*DC7*	*DC8*
CM3	*6*	*MODE D'AFFICH.*	*CT1*	*CT2*	*CT3*	*CT4*	*CT5*	*CT6*	*x*	*x*
CM4	*7*	*BORD D'ÉCRAN*	*BM*	*GM*	*RM*	*x*	*MR*	*LR*	*IR*	*x*
BAMP	*8*	*ADRESSE DU DÉBUT DU BUFFER DES DONNÉES*								
BAMP	*9*	*ADRESSE DU DÉPART DES ACCÈS AUTO-INCRÉMENTÉS*								
BAPA	*10*	*ADRESSE DE L'ÉCRAN*								
BAGC0	*11*	*ADRESSE DU GÉNÉRATEUR DE CARACTÈRES 0*								
BAGC1	*12*	*ADRESSE DU GÉNÉRATEUR DE CARACTÈRES 1*								
BAGC2	*13*	*ADRESSE DU GÉNÉRATEUR DE CARACTÈRES 2*								
BAGC3	*14*	*ADRESSE DU GÉNÉRATEUR DE CARACTÈRES 3*								
BAMTF	*15*	*ADRESSE DE LA FIN DU BUFFER DES DONNÉES*								

a. Les types de registres

On distingue deux groupes de registres :

• Les registres 0 à 7
Ce sont des registres 8 bits.
Le registre 0 contient l'adresse du registre auquel on veut accéder.
Le registre 3, accessible uniquement en lecture, est le registre d'état du VDP. Il est inutilisé en ce qui nous concerne.
Les registres 1 à 7 sont tous programmables au moyen du port P45, qui assure la liaison entre le CPU et le VDP.
Cet accès se fait en deux temps :
— adressage du registre ;
— écriture dans le registre.
Exemple : mettre la valeur >10 dans le registre VDP numéro 4.
MOVP %>04,P45 : registre VDP numéro 4
MOVP %>10,P45 : stocké %>10

• Les registres 8 à 15
Les registres 8 à 15 sont des registres 16 bits destinés à contenir les adresses de base des différentes zones de la VRAM.
Comme le bus de communication entre le VDP et le CPU est un bus 8 bits, ces registres sont programmés en quatre étapes :
— l'octet de poids faible de l'adresse (LSB) est rangé dans le COL ;
— l'octet de poids fort de l'adresse (MSB) est rangé dans ROW ;
— le registre d'adresse concerné est sélectionné ;
— l'adresse est transmise en écrivant 2 fois dans le POINTER REGISTER.

ATTENTION : Lors du transfert de l'adresse de (ROW.COL) au registre d'adresse concerné, l'adresse est systématiquement incrémentée par le VDP. Cette incrémentation respecte la règle suivante :
★ si le registre concerné est BAGC0, BAGC1, BAGC2 ou BAGC3, le VDP ajoute 2 à l'adresse, avant le transfert.
S'il s'agit d'un autre registre, le VDP ajoute 1 à l'adresse avant le transfert.

Exemples :

1 - Initialiser le registre BAPA à l'adresse >2000.
Dans le cas du registre BAPA, le VDP ajoute 1 à la valeur contenue dans (ROW.COL) avant le transfert. Il faut donc stocker >1FFF, soit >2000 —>1, dans (ROW.COL) pour initialiser BAPA à >2000.

```
MOVP %>01.P45      : sélectionne le registre COL
MOVP %>FF.P45      : écrit %>FF : LSB de l'adresse

MOVP %>02.P45      : sélectionne le registre ROW
MOVP %>1F.P45      : écrit %>1F : MSB de l'adresse

MOVP %>0A.P45      : sélectionne le registre BAPA

MOVP %>00.P45      : transfert LSB
MOVP %  00.P45     : transfert MSB
```

2 - Initialiser le registre BAGC2 à l'adresse >1A00
Dans le cas du registre BAGC2, le VDP ajoute 2 à la valeur contenue dans (ROW-COL) avant le transfert. Il faut donc stocker >19FE, soit >1A00 —>2 dans (ROW-COL) pour initialiser BAGC2 à >1A00.

```
MOVP %>01.P45      : sélectionne le registre COL
MOVP %>FE.P45      : écrit %>FE : LSB de l'adresse

MOVP %>02.P45      : sélectionne le registre ROW
MOVP %>19.P45      : écrit %>19 : MSB de l'adresse

MOVP %>0D.P45      : sélectionne le registre BAGC2

MOVP %>00.P45      : transfert LSB
MOVP %  00.P45     : transfert MSB
```

b. Rôle et utilisation des registres importants.

1. Registre 8 bits
Seules sont précisées les fonctions des bits importants.
• ROW et COL (1 et 2) sont utilisés pour communiquer une adresse à un registre 16 bits. ROW contiendra toujours le MSB de l'adresse et COL le LSB.
• CM1 doit toujours être initialisé à >10.
• CM2 le bit DC2 permet d'afficher (1) ou non (0) la ligne d'écran 0 ;
le bit DC5 fixe le type du générateur BAGC3 : alphanumérique (0) ou alphamosaïque (1) ;
le bit DC6 permet de superposer (1) ou non (0) une grille sur l'écran

• CM3
les bits CT1 et CT2 permettent de fixer le mode d'affichage :

CT1	CT2	
0	0	pas d'affichage (screen off)
0	1	mode texte
1	0	mode graphique
1	1	mode mixte

• CM4
Les bits BM GM RM indiquent la couleur du bord de l'écran et la couleur de fond des caractères pour les générateurs alphanumériques.

0	0	0	NOIR
0	0	1	ROUGE
0	1	0	VERT
0	1	1	JAUNE
1	0	0	BLEU
1	0	1	MAGENTA
1	1	0	CYAN
1	1	1	BLANC

Les bits MR LR IR gèrent le masquage, le soulignement et l'incrustation, dans le cadre d'une définition d'attribut par bloc.

2 - Registres 16 bits

Il s'agit des registres BAPA, BAGC0, BAGC1, BAGC2, BAGC3 qui contiennent une adresse en VRAM :

BAPA contient l'adresse du premier octet de l'écran ;

BAGC0 contient l'adresse du premier octet du générateur de caractères 0 ;

BAGC1 contient l'adresse du premier octet du générateur de caractères 1 ;

BAGC2 contient l'adresse du premier octet du générateur de caractères 2 ;

BAGC3 contient l'adresse du premier octet du générateur de caractères 3.

2. Accès à la VRAM

Pour accéder à la VRAM, il faut, d'une part, pouvoir définir l'adresse qui nous intéresse, d'autre part réaliser l'opération de lecture ou d'écriture.

Pour simplifier l'écriture des programmes, EXELVISION a développé un ensemble de sous-programmes accessibles par l'instruction TRAP, qui permettent de se positionner à une adresse en VRAM et deux instructions microcodées permettant de lire ou d'écrire un octet en VRAM.

a. Positionnement en VRAM

On dispose de deux pointeurs indépendants, le pointeur d'écriture et le pointeur de lecture, qui peuvent contenir des adresses différentes.

• TRAP 8
TRAP 8 permet d'initialiser le pointeur d'écriture UNIQUEMENT.
L'adresse d'écriture doit être placée dans TEMP1 (R13.R14).

Exemple :

```
MOVD %>0A00.TEMP1     : adresse d'écriture
TRAP 8                : initialisation du pointeur
```

• TRAP 9
TRAP 9 permet d'initialiser le pointeur de lecture et le pointeur d'écriture A LA MEME ADRESSE.
L'adresse doit être placée dans TEMP1 (R13.R14).

Exemple :

```
MOVD %>0A00.TEMP1     : adresse de lecture et d'écriture
TRAP 9                : initialisation des pointeurs
```

• TRAP 10
TRAP 10 permet d'initialiser le pointeur de lecture et le pointeur d'écriture A DEUX ADRESSES DIFFERENTES.
L'adresse d'écriture doit être placée dans TEMP1 (R13.R14).
L'adresse de lecture dans TEMP2 (R15.R16).

Exemple :

```
MOVD %>0A00.TEMP1     : adresse d'écriture
MOVD %>4000.TEMP2     : adresse de lecture
TRAP 10               : initialisation des pointeurs
```


b. Ecriture et lecture d'un octet

LVDP (adressage implicite) permet de lire un octet à l'adresse courante indiquée par le pointeur de lecture. L'octet est placé dans A. WVDP permet d'écrire un octet à l'adresse courante indiquée par le pointeur d'écriture. L'octet peut être donné de façon immédiate ou se trouver dans A ou B : WVDP %15. WDP A. WVDP B.

Remarques : • Les pointeurs ne sont pas accessibles en LECTURE. Il est donc impossible de connaître les adresses pointées par ces registres.
• Les pointeurs sont auto-incrémentés après chaque accès à la VRAM, ce qui permet de réaliser facilement les accès séquentiels.

Exemples :

```
MOVD %>0A00.TEMP1     : adresse d'écriture
MOVD %>8000.TEMP2     : adresse de lecture
TRAP 10               : initialise les pointeurs
LVDP                  : lit dans A un octet en VRAM en >8000
MOV A.B               : sauve l'octet dans B
LVDP                  : lit dans A un octet VRAM en >8001
WVDP A                : écrit (A) en VRAM en >0A00
WVDP A                : écrit (A) en VRAM en >0A01
WVDP B                : écrit (B) en VRAM en >0A02
WVDP B                : écrit (B) en VRAM en >0A03
```

Après exécution, le pointeur de lecture vaut >8002 et le pointeur d'écriture >0A04.

CM1 (=>10)

BIT	ÉTAT	FONCTION
BT1	*0*	
BT2	*0*	*TV STANDARD*
BT3	*0*	*TOUJOURS A 0*
BT4	*1*	*SYNCHRONISATION*
BT5	*0*	
x	*0*	
x	*0*	
x	*0*	

CM2

BIT	ÉTAT	FONCTION
DC1	*1*	*ANTIOPE*
DC2	*0*	*PAS DE LIGNE 0*
	1	*AFFICHAGE LIGNE 0*
DC3	*0*	
DC4	*0*	
DC5	*0*	*BAGC3 ALPHANUM.*
	1	*BAGC3 ALPHAMOS.*
DC6	*0*	*PAS DE GRILLE*
DC7	*0*	*AFFICHAGE GRILLE*
DC8	*0*	

BIT	ÉTAT		FONCTION
	CT1	CT2	
CT1	*0*	*0*	*ECRAN ETEINT*
CT2	*0*	*1*	*MODE TEXTE*
	1	*0*	*MODE GRAPHIQUE*
	1	*1*	*MODE MIXTE*
CT3	*0*		*TOUJOURS A 0*
CT5	*1*		
CT6	*0*		
x	*0*		
x	*0*		

CM3

BIT	ÉTAT	FONCTION	
BM	*1*	*BLEU*	*COULEUR DU BORD ET DU FOND DES CARACTÈRES*
GM	*1*	*VERT*	
RM	*1*	*ROUGE*	
x	*0*		
MR	*0*		
CR	*0*		
IR	*0*		
x	*0*		

CM4

RÉSUMÉ DE L'ÉTAT DES REGISTRES DE CONTRÔLE DU VDP

ATTRIBUT			CODE	
BIT			BIT	
7	BLEU	*COULEUR*	7	MODE CLIGNOTANT
6	VERT	*DES*	6	
5	ROUGE	*POINTS*	5	*CODE*
4	—	*NUMÉRO DU*	4	*DU*
3	—	*GÉNÉRATEUR*	3	*CARACTÈRE*
2	BLEU	*COULEUR*	2	
1	VERT	*DU*	1	
0	ROUGE	*FOND*	0	

Description de l'attribut et du code d'un caractère de type alphamosaïque

ATTRIBUT			CODE	
BIT			BIT	
7	BLEU	*COULEUR*	7	MODE CLIGNOTANT
6	VERT	*DU*	6	
5	ROUGE	*CARACTÈRE*	5	*CODE*
4	—	*NUMÉRO DU*	4	*DU*
3	—	*GÉNÉRATEUR*	3	*CARACTÈRE*
2	INVERSION		2	
1	DOUBLE LARGEUR		1	
0	DOUBLE HAUTEUR		0	

Description de l'attribut et du code d'un caractère de type alphanumérique

LES TRAPS

La ROM interne du 7020 de l'EXL 100 contient des sous-programmes.
Ils sont accessibles par l'instruction **TRAP**. Les quatre premiers sont réservés aux traitements des diverses interruptions.

Voici les fonctions réalisées par ces sous-programmes :

TRAP 0
TRAP 0 permet la gestion du reset.

TRAP 1
TRAP 1 permet la gestion de l'interruption 1 qui contrôle la communication avec le 7041/7042. Les échanges se font grâce à la « MAIL BOX » située sur le port 48 et les messages sont constitués d'un code suivi ou non de renseignements.
A chaque réception d'une interruption, la routine d'interruption teste l'état de l'écran. Si l'écran est éteint, la routine place le contenu du registre VDPO6 dans le registre 6 du VDP. Si le registre SUICOM est nul, on se trouve en présence d'un code fonction. La routine parcourt alors sa table de traitement, puis se branche à l'adresse de la fonction découverte.

Les fonctions sont les suivantes :

CODE	EXL 100	EXELTEL	SIGNIFICATIONS
00	*OUI*	*OUI*	*Non utilisée*
01	*OUI*	*OUI*	*Réception joystick 0*
02	*OUI*	*OUI*	*Réception joystick 1*
03	*OUI*	*OUI*	*Début buffer speech*
04	*OUI*	*OUI*	*Fin du buffer speech*
05	—	*OUI*	*Série*
06	—	—	*Non utilisé*
07	*OUI*	—	*Ecran introduction (logo)*
08	*OUI*	*OUI*	*7041/7042 initialisé*
09	*OUI*	*OUI*	*Interface série 7041/7042 prête*
10	*OUI*	*OUI*	*Interface série 7041/7042 non prête*
11	*OUI*	*OUI*	*Ecran éteint*
12	—	*OUI*	*Début buffer speech*
13	—	*OUI*	*Contrôle CRC 6100 ou 7042*
14	—	*OUI*	*Test mail box, lecture code pays*
15	—	*OUI*	*Lecture 6100 (répétition des données)*

TRAP 2
TRAP 2 assure la gestion de l'interruption TIMER, elle est d'ailleurs très simple. Seul les registres A et B sont sauvegardés. La routine teste ensuite le contenu de BRTIME et BRTIME + 1. Il peut alors se présenter deux cas :
• le contenu de BRTIME et BRTIME + 1 est nul : il y a restitution des registres A et B et l'interruption est terminée ;
• BRTIME et BRTIME + 1 contiennent une adresse non nulle : un appel de la routine située à cette adresse est effectué par une instruction CALL.
La routine devra donc se terminer par un RETS afin de permettre la restitution des registres A et B.

TRAP 3
TRAP 3 assure la gestion de l'interruption 3.
Un contrôle du contenu de BRINT3 + 1 et BRINT3 + 2 est réalisé :
• leur contenu est nul. Il y a un retour d'interruption par RETI ;
• leur contenu n'est pas nul. Un branchement (BR) est réalisé à l'adresse BRINT3. Cette adresse doit contenir l'instruction BR et les deux suivantes l'adresse de la routine de gestion de cette interruption ;

Cette routine doit se terminer par RETI. Dans le cas de l'EXELTEL, le registre A est sauvegardé lors de l'appel et il ne faut pas oublier de renseigner V$INT3.

TRAP 4
Ce TRAP initialise le synthétiseur de paroles.
Il suffit de mettre l'adresse de la table de son dans le registre PTRSON. S'il s'agit d'un son répétitif, il faut mettre dans le registre SONFON l'adresse de la table du son répétitif. Exemples :
• Son ponctuel
MOVD %table.PTRSON
TRAP 4
• Son répétitif
MOVD %table.PTRSON
MOVD PTRSON.SONFON
TRAP 4

A la fin d'un son ponctuel, si l'adresse contenue dans SONFON est nulle, on positionne à 0 le bit 1 du registre FLGCOM (il est à 1 si un son est en cours). Dans le cas contraire, SONFON est chargé dans PTRSON et il y a réinitialisation du son.
• Registres utilisés : **A, B, WRKSON**

TRAP 5
Ce TRAP permet d'arrêter le synthétiseur de paroles.
Le bit du registre FLGGGCOM est remis à 0. Un contrôle du registre SONFON est effectué. S'il n'est pas nul, il y a relance du son répétitif :

- Arrêt d'un son ponctuel : **TRAP 5**
- Arrêt d'un son répétitif : **MOVD %>0000.SONFON**
  TRAP 5
- Registres utilisés : **A, B.**

TRAP 6
Cette routine envoit un octet vers le 7041/7042. Cet octet doit être chargé dans le registre A avant appel. Suivant la valeur de cet octet, une fonction est choisie.
En voici la liste :

CODE	EXL 100	EXELTEL	FONCTIONS
00	*OUI*	*OUI*	*Provoque un RESET du 7041*
01	—	*OUI*	*Fonction nulle (dangereux)*
02	*OUI*	*OUI*	*Lecture de la valeur courante joystick 0*
03	*OUI*	*OUI*	*Lecture de la valeur courante joystick 1*
04	*OUI*	*OUI*	*Teste la disponibilité de l'interface série*
05	*OUI*	*OUI*	*Emission d'un octet via l'interface série*
06	*OUI*	*OUI*	*Initialisation du port série 7041/7042*
07	—	*OUI*	*Emission du contenu de la ROM 6100*
08	*OUI*	*OUI*	*RESET « HARD » du synthétiseur de parole*
09	*OUI*	*OUI*	*Démarrage du synthétiseur de parole*
10	*OUI*	*OUI*	*Suite de données pour le synthétiseur*
11	*OUI*	*OUI*	*Demande du générateur standard*
12	—	*OUI*	*Traitement du CRC du 7042 (émission)*
13	*OUI*	—	*Envoi du LOGO exelvision*
13	—	*OUI*	*Démarrage du son ROM 6100*
14	—	*OUI*	*Suite de données sur la ROM pour le synthé.*
15	—	*OUI*	*Ne pas décoder les touches du joystick 0*
16	—	*OUI*	*Ne pas décoder les touches du joystick 1*
17	—	*OUI*	*Décoder les touches du joystick 0*
18	—	*oui*	*Décoder les touches du joystick 1*
19	—	*oui*	*Test MAIL BOX.Renvoit la donnée transmise*
20	—	*oui*	*Mise en mode veille*
21	—	*oui*	*Lecture en code pays dans la ROM 6100*
22	—	*oui*	*Positionnement DSR 7042 sans initialisation*
23	—	*oui*	*Traitement des sons 6100 avec l'adresse*
24 et +	—	—	*Provoque un RESET du 7041*

— signifie qu'un RESET du 7041/7042 est très possible.

- Mode d'emploi
  MOV %donnée.A
  TRAP 6

- Registres utilisés : **A.B**

TRAP 7
Il ne faut jamais utiliser cette fonction.

TRAP 8, 9 et 10
Ces différentes routines permettent le positionnement des pointeurs de lecture et d'écriture de la RAM VDP.

TRAP 8 permet le positionnement du pointeur d'écriture seulement.

TRAP 9 permet le positionnement des pointeurs de lecture et d'écriture à la même adresse.

TRAP 10 permet le positionnement des pointeurs de lecture et d'écriture à deux adresses différentes.

Avant appel, il faut fournir l'adresse d'écriture dans le registre TEMP1 pour TRAP 8 et TRAP 9, et l'adresse de lecteur dans le registre TEMP2 pour TRAP 10. Le registre TEMP1 contient toujours l'adresse d'écriture. Après un TRAP 10, il faut exécuter en premier une lecture de la RAM VDP afin de positionner correctement les pointeurs.

- Mode d'emploi

TRAP 8 : **MOVD %adresse écriture.TEMP1**
TRAP 8

TRAP 9 : **MOVD %adresse écriture/lecture.TEMP1**
TRAP 9

TRAP 10 : **MOVD %adresse écriture.TEMP1**
MOVD %adresse lecture.TEMP2
TRAP 10

Les pointeurs sont auto-incrémentés après chaque accès. On peut donc réaliser facilement une lecture ou une écriture séquentielle.

- Registres utilisés : **A, B, TEMP1, TEMP2**

TRAP 11
Cette routine permet d'afficher une image codée en haute résolution. L'image est constituée d'un ensemble d'octets, chaque octet réalisant une des fonctions de la table suivante :

B7 B6	CODE INSTRUCTION
0 0	*Incrémenter X*
0 1	*Tracer un point*
1 0	*Incrémenter Y*
1 1	*Répétition d'un groupe d'instructions*
B5 B4 B3	*Nombre de répétitions de l'opération de base*
B2 B1 B0	*Code couleur ou, dans le cas d'une boucle, nombre d'instructions du corps de la boucle.*

Avant l'appel du TRAP 11, il faut initialiser plusieurs registres :

— RAMADR doit contenir l'adresse du point (0,0).
— TEMP7 doit contenir l'adresse de la table.
— TEMP10 doit contenir l'abscisse X.
— TEMP11 doit contenir l'ordonnée Y.

• Mode d'emploi
MOVD %XOYO.RAMADR
MOVD %table,TEMP7
MOVD %X,TEMP10
MOV %Y,TEMP11
TRAP 11

• Registres utilisés : **A, B, TEMP1, TEMP2, TEMP3 – 1, TEMP4, TEMP5, TEMP6, TEMP7, TEMP8.**

TRAP 12
Cette routine permet d'afficher un point en haute résolution.
Il est nécessaire de fournir les paramètres suivants :
— TEMP4 : abscisse X
— TEMP3 – 1 : ordonnée Y
— TEMP3 : couleur C du point (B0 rouge, B1 vert, B2 bleu)
— TEMP5 : adresse du point (X0,Y0)
— TEMP8 : longueur L d'une ligne (généralement 122)

• Mode d'emploi : **MOV %L,TEMP8**
MOVD %X,TEMP4
MOV %Y,TEMP3.1
MOVD %XOYO,TEMP5
TRAP 12

• Registres utilisés : **A, B, TEMP1, TEMP4, TEMP8**

TRAP 13
Cette routine permet de charger le générateur de caractère ASCII à partir du 7041. Il suffit d'indiquer dans le registre TEMP8 l'adresse en VRAM du générateur.

ATTENTION : — Il est obligatoire d'autoriser les interruptions.
— Cette routine détruit le contenu de la RAM CPU à partir de (C100 sur 1280 octets.

• Mode d'emploi : **MOVD %adresse du générateur.TEMP8**
TRAP 13

• Registres utilisés : **A, B, TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP7, TEMP9**

TRAP 14
Cette routine gère le port cassette qui correspond au bit 3 du port 6. (C'est aussi le son du moniteur.)

1 - TYPE DE FICHIER
Cette routine effectue l'enregistrement d'un bloc mémoire du 7040 ou de la RAM VDP délimité par deux adresses. Lors de la re-lecture, le bloc mémoire enregistré peut être placé à une adresse elle même enregistrée sur la bande ou précisée par l'utilisateur.

2 - CODAGE DES INFORMATIONS
Un bit est représenté par deux périodes d'un signal audio. La fréquence dépend de la valeur du bit. Si le bit est à 1, la fréquence est de 2,5 kHz, sinon elle est de 1,25 kHz. La vitesse de transmission dépend donc du bit enregistré ou lu. Sa valeur moyenne est de 940 bauds, soit 117 octets par seconde.

3 - CONSTITUTION DE L'ENTETE DU FICHIER MAGNETIQUE
Une série de 2400 bits (alternativement à 1 puis à 0) sert à synchroniser le programme de lecture et l'éventuel contrôle automatique d'enregistrement du magnétophone. Le premier octet, qui vaut (70, identifie le début du fichier. Les quatre octets suivants contiennent le nom du fichier donné par l'utilisateur. Les quatre octets suivants contiennent les adresses d'implantation du bloc mémoire. Un dernier octet contient un offset.

4 - UTILISATION DU FLAG DE SORTIE
Lorsque le TRAP 14 est terminé, le registre TEMP10 indique le résultat de l'opération :
— bit 7 = 1 si le fichier sur la cassette n'est pas le fichier demandé. Le nom du fichier trouvé est dans TEPNAM (4 octets. >C01B).
— bit 6 = 1 si il y a erreur de lecture.
— bit 5 = 1 si la routine de lecture n'a pas détecté de variation de niveau pendant soixantes secondes.

Dans tous les cas, la zone FILNAM (4 octets de (C025 à (C028) doit contenir le nom du fichier à lire ou à enregistrer.
Il est possible de lire le premier fichier trouvé, en positionnant à 1 le bit 3 de TEMP10 avant l'appel du TRAP 14.

5 - INITIALISATION DES PARAMÈTRES SELON LA FONCTION CHOISIE

FONCTION	TEMP10	TEMP10 – 1	TEMP9	TEMP8	
OUTRAM	*0*	—	*adr. fin*	*adr. déb.*	*Enregistrement*
OUTVDP	*4*	*offset*	*adr. fin*	*adr. déb.*	
INRAMU	*2*	—	*adr. fin*	*adr. déb.*	*Lecture*
INRAMK	*3*	—	—	—	
INVDPU	*6*	*offset*	*adr. fin*	*adr. déb*	
INVDPK	*7*	—	—	—	

Les signes — signifient qu'il n'est pas nécessaire d'initialiser le registre en question.
Liste des fonctions :
— OUTRAM : enregistrement d'un bloc mémoire 7040
— OUTVDP : enregistrement d'un bloc mémoire VDP
— INRAMU : lecture et stockage en mémoire 7040 aux adresses fixées par l'utilisateur
— INRAMK : lecture et stockage en mémoire 7040 aux adreses sauvegardées sur la bande
— INVDPU : comme INRAMU mais en mémoire VDP
— INVDPK : comme INRAMK mais en mémoire VDP

• Mode d'emploi

```
MVD %fonction.TEMP10
MOV %offset.TEMP10 – 1
MOVD %fin.TEMP9
MOVD %début.TEMP8
CALL @SETNAM        : mettre le nom dans la zone FILNAM
TRAP 14
CALL @GESTER        : gestion des erreurs
```

• Registres utilisés : **A, B, TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP6.1**

TRAP 15
Cette routine permet d'initialiser le générateur de nombre pseudo-aléatoires qui se trouve dans la zone SEED (de >C00F à >C014). Le nombre créé se trouve en PROD (de >C015 à >C01A).
La méthode la plus simple est l'appel direct du TRAP 15.
Une méthode plus élaborée est conseillée : initialiser le générateur à partir d'une action extérieure non contrôlable par l'ordinateur (appui sur une touche, par exemple). L'initialisation doit se faire au moyen d'un nombre impair.

• Mode d'emploi
— Simple :

```
TRAP 15
```

— Elaboré :

```
      MOVD %  6001.B   : nombre impair.B
WAIT  DECD B
      DECD B
      CALL @KEY        : scrute clavier,
                         par exemple
      JEQ WAIT         : pas de touche, attente
      STA @SEED
      STA @SEED+2
      STA @SEED+4
      XCHB A
      STA @SEED+1
      STA @SEED+3
      STA @SEED+5
```

• Registres utilisés : **A, B**

TRAP 16
Cette routine fournira dans TEMP8 un nombre pseudo-aléatoire 16 bits inférieur au maximum + 1 donné dans TEMP6. La valeur maximum est >7FFF.

• Mode d'emploi :
MOVD %maximum + 1.TEMP6
TRAP 16

• Registres utilisés : **A, B, TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP6, TEMP7, TEMP8**

TRAP 17
Cette routine effectue une division 16 bits. En entrée, TEMP6 contient le diviseur et TEMP8 le dividende. En sortie, TEMP8 contient le reste et TEMP6 le quotient. Si TEMP6 est nul en entrée, aucun calcul n'est effectué.

• Mode d'emploi : diviser 7412 par 29
MOVD %7412.TEMP8
MOVD %29.TEMP6
TRAP 17

• Registres utilisés : **A, B, TEMP1, TEMP5, TEMP6, TEMP8**

TRAP 18
TRAP 19
Ces deux routines gèrent les générateurs de caractères. Elles permettent de charger un générateur personnalisé ou de modifier un générateur existant. Paramètres à fournir :

REGISTRE	PARAMÈTRE
TEMP4–1	*Nombre de caractères à charger*
TEMP7	*Numéro du premier caractère*
TEMP3	*Adresse de la table du générateur en RAM CPU*
TEMP2	*Adresse d'implantation en RAM VDP*

TRAP 18 charge un générateur entier (128 caractères), après une remise à zéro de la zone de destination. TEMP7 doit être mis à zéro avant l'appel, le premier caractère portant le numéro zéro.

• Mode d'emploi :
MOVD %géné.TEMP3 : adresse des dessins des caractères
MOVD %adresse.TEMP2 : adresse du générateur en VRAM
CLR TEMP7 : premier caractère = code 0
MOV %128.TEMP4–1 : modifier 128 caractères
TRAP 18

TRAP 19 permet la modification d'un générateur existant.
Les dessins des caractères doivent être stockés dans une table, chaque octet représentant une ligne, en commençant par la ligne du bas. Il faut indiquer le numéro du premier caractère et le nombre de caractères à modifier avant l'appel.

• Mode d'emploi : modifier les caractères 65 à 70 (6 caractères).
MOVD %caract.TEMP3 : adresse des dessins des caractères
MOVD %adresse.TEMP2 : adresse du générateur en VRAM
MOV %65.TEMP7 : premier caractère = code 65
MOV %6,TEMP4-1 : 6 caractères à modifier
TRAP 19

• Registres utilisés : **A, B, TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP6, TEMP7, TEMP8, TEMP9**

TRAP 20
TRAP 20 assure l'affichage d'une chaîne de caractères sur l'écran. La chaîne doit se terminer par >00. Les caractères de contrôle ne sont pas gérés par cette routine, et la ligne et la colonne ou doit apparaître le message doivent être initialisées séparément.
TEMP2 doit contenir l'adresse du début du message et TEMP3 l'attribut.
• Mode d'emploi :

```
        MOVD %TEXTE.TEMP2
        MOV %attrib.TEMP3
        TRAP 20
        RETS
TEXTE   BYTE 'MESSAGE',0
        END
```

TRAP 21
TRAP 22
Ces deux TRAP assurent la gestion de l'interface série.
TRAP 21 permet l'émission de l'octet contenu dans A vers l'interface.

TRAP 22 permet de vérifier la disponibilité de l'interface (bit 4 de FLGCOM à 1 si interface OK).
La réception d'un octet à partir de l'interface série dépend de l'ordinateur :

EXL100 : tout se passe comme si l'octet provenait du clavier : sa valeur est dans le registre VALEU0.

EXELTEL : l'octet se trouve en SPESER + 1, SPESER contenant le résultat de la lecture :
bit 7 : overrun dû au 7040
bit 6 : overrun de communication série
bit 5 : nombre de bits stop incorrect
bit 4 : erreur de parité
bit 3
à
bit 0 : nombre d'erreurs depuis le dernier transfert.

L'initialisation de l'interface série est réalisé en envoyant 6 octets au 7041.

octet 1 : code fonction (6) 0000 0110

octet 2 : parité, longueur des mots x1xx xx10
bit 7 : 0 —> 1 bit stop
1 —> 2 bits stop
bit 6 : 1
bit 5.4 : 00 —> pas de parité
01 —> parité impaire
11 —> parité paire
bit 3.2 : 00 —> 5 bits
01 —> 6 bits
10 —> 7 bits
11 —> 8 bits
bit 1 : 1
bit 0 : 0

octet 3 : réception/transmission ?0?1 ?x?x
bit 7 : ?
bit 6 : 0
bit 5 : ?
bit 4 : 1
bit 3 : ?
bit 2 : 0 —> réception off
1 —> réception on
bit 1 : ?
bit 0 : 0 —> émission off
1 —> émision on

octet 4 : précompteur vitesse de transmission ?1?? 00xx
bit 7 : ?
bit 6 : 1
bit 5 : ?
bit 4 : ?
bit 3 : 0
bit 2 : 0
bit 1.0 : 00 —> vitesse >110 bauds
01 —> vitesse =110 bauds

octet 5 : vitesse de transmission
>AE : 110 bauds >2A : 900 bauds >03 : 9600 bauds
>FF : 150 bauds >1F : 1200 bauds >01 : 19200 bauds
>7F : 300 bauds >0F : 2400 bauds >00 : 38400 bauds
>3F : 600 bauds >07 : 4800 bauds

octet 6 : état des signaux DSR et DTR 0000 00xx
bit 7 : 0
bit 6 : 0
bit 5 : 0
bit 4 : 0
bit 3 : 0
bit 2 : 0
bit 1 : 0 —> on line si état haut
1 —> on line si état bas
bit 0 : 0 —> DSR émis à l'état haut (+ 12V)
1 —> DSR émis à l'état bas (- 12V)

A la mise sous tension, l'initialisation est réalisée par le 7042 et l'interface est dans l'état suivant :
1200 bauds, 7 bits/mot, parité paire, 1 bit stop, DSR et DTR état haut.

TRAP 23
Ce trap est disponible. Il suffit de mettre en BRTRAP + 1 et BRTRAP + 2 l'adresse de la routine associée au TRAP 23. Dans le cas de l'EXELTEL, il ne faut pas oublier d'indiquer la page de la routine dans V$INT1.

Remarque : un TRAP est équivalent à un CALL.

PARTICULARITÉS DE L'EXELTEL

Dans l'EXELTEL certaines routines ont été intégrées. Elles ont été regroupées en fin de zone mémoire dans une table.

Voici son contenu :

ADRESSE	FONCTION
>F006	*Version du moniteur*
>F007	*Pagination, instruction BR*
>F00A	*Pagination, instruction CALL*
>F00D	*Pagination, instruction LDA*
>F013	*Scrool en 80 colonnes*
>F016	*Chargement du générateur en 80 colonnes*
>F019	*Modification du générateur en 80 colonnes*
>F01C	*Affichage d'un caractère en 80 colonnes*
>F01F	*Affichage d'un message en 80 colonnes*
>F022	*Cls en 80 colonnes*
>F025	*Pagination, instruction STA*
>F028	*Sélection de page*

>F006 : Version du moniteur
Cet octet contient la version du moniteur. Pour l'EXL100, sa valeur est >FF et pour l'EXELTEL, il commence à 2.

>F007 : Pagination, instruction BR
La pagination permet de travailler avec 8 zones de 32 K situées physiquement à la même adresse. Mais pour s'y retrouver, le moniteur doit connaître le numéro de la page courante. C'est ce numéro qui est placé dans V$ACT.Il doit également savoir dans quelle page se situent les interruptions. Le numéro de la page sera stocké pour chaque interruption X dans le vecteur V$INTX, ou X est le numéro de l'interruption. De plus le vecteur DISKT devra contenir la valeur du port 64.
BR : — adresse dans TEMP1
— page dans B

>F00A : Pagination, instruction CALL
CALL : — adresse dans TEMP1
— page dans B

>F00D : Pagination, instruction LDA
LDA : — adresse dans TEMP1
— page dans B
— valeur lue dans A
— utilise A et B

>F013 : Scroll en 80 colonnes
Pour pouvoir travailler en 80 colonnes, il faut être en mode graphique puis passer en 80 colonnes par l'intermédiaire de P53. Il faudrait aussi charger le générateur.
La routine du scrool permet de remonter le texte d'une ligne vers le haut, la dernière ligne étant effacée. Pour lancer le scrool, il suffit de faire : **CALL @>F013**
La routine utilise les registres **A, B, TEMP1-1, TEMP1, TEMP2-1 et TEMP2**

>F016 : Chargement du générateur en 80 colonnes
Cette routine détruit la zone mémoire débutant en >C100 sur une longueur de 1280 octets. Pour lancer cette routine, faire :
CALL @>F016
La routine utilise les mêmes registres que ceux lors du TRAP 13.

>F019 : Modification du générateur en 80 colonnes
La constitution de la table est la même que pour un TRAP 19.
L'indice du premier caractère à modifier doit être dans A.
Le nombre de caractères dans B et l'adresse de la table de codes dans TEMP1. Pour lancer, faire : **CALL @>F019**
Les registres utilisés sont : **A, B, TEMP1-1, TEMP1, TEMP2-1, TEMP2.**

>F01C : Affichage d'un caractère en 80 colonnes
Cette routine gère le scrolling ainsi que les retours chariots. La position du curseur est stockée en RAM dans COL80 et LIG80.
La colonne 0 est à gauche et la ligne 0 en haut. Il est possible d'afficher en inverse vidéo en mettant le code du caractère plus 128. Le code du caractère à afficher doit être mis dans A.
On lance la routine en faisant : **CALL @>F01C**
Les registres utilisés sont **A, B, TEMP1-1, TEMP1, TEMP2-1, TEMP2, TEMP3-1, TEMP3, TEMP4-1 et TEMP4.**

>F01F : Affichage d'un message en 80 colonnes
L'utilisation est la même que le TRAP 20. Le message doit se terminer par (00. L'adresse du début du message sera mis dans TEMP1.
CALL %>00
Les registres utilisés sont les mêmes que pour le TRAP 20 plus TEMP8-1 et TEMP8.

>F022 : Cls en 80 colonnes
Cette routine efface l'écran et repositionne le curseur en haut à gauche. L'écran est alors blanc et LIG80 = COL80 = 0. Elle se lance par : **CALL @>F022**
Les registres utilisés sont **A, B, TEMP1-1, TEMP1**

>F025 : Pagination, instruction STA
STA : — adresse dans TEMP1
— page dans B
— valeur à écrire dans A
— utilise A et B

>F028 : Sélection de page

ATTENTION : votre routine de pagination doit être en mémoire >C100 à >C7FF. Le numéro de page choisi doit être dans A. Puis faire : CALL @>F028

Tableau de correspondance des pages

N° PAGE	PAGE
0	*RAM supplémentaire*
1	*RAM disquette*
2	*ROM interne EXELTEL page 0*
3	*ROM interne EXELTEL page 1*
4	*ROM externe disquette page 0*
5	*ROM externe disquette page 1*
6	*ROM externe EXELTEL page 0*
7	*ROM externe EXELTEL page 1*

ORGANISATION DE LA MÉMOIRE CPU

1) Cartographie

	EXL 100 EXELTEL	EXELMEMOIRE	EXL135 ou EXELDISK	
>FFF	*ROM INTERNE AU MICRO-PROCESSEUR*			4 octets
>F00	*LIBRE*	*ROM (CROS)*	*RAM (DOS)*	8 octets
>D000	*LIBRE*		*ROM (DOS)*	2 octets
>C800	*RAM SYSTEME*			2 octets
>C000	*LIBRE*	*RAM*		8 octets
>A000	*LIBRE*	*RAM*	*ROM (DOS)*	8 octets
>8000	*MODULE ROM*		*RAM*	32 octets 512 octets
>0200	*LIBRE*			128 octets
>0180	*RESERVE*			48 octets
>0150	*LIBRE*		*ACCÈS AU CONTRÔLEUR DE DISQUETTES*	16 octets
>0140	*VOIR DÉTAIL DES PORTS*			64 octets
>0100	*LIBRE*			128 octets
>0080	*REGISTRES*			128 octets
>0000				

2) Détail des emplacements

>0000 à >007F :
Il s'agit de la zone ou se situent les registres. Le registre A (RO) se situe à l'adresse >0000 et le registre 127 (R127) à l'adresse >007F.

>0080 à >00FF :
Cette zone est libre. Vous pouvez y mettre de la mémoire ou des périphériques de votre choix. Il faudra pour cela effectuer un décodage d'adresse.

>0100 à >013F :
Zone des ports. Réservée à la gestion de l'ordinateur. Pour plus d'informations, voir le détail des ports en fonction de l'ordinateur.

>0140 à >014F :
Zone utilisée par le lecteur de disquette. Voir le détail sur cette zone dans le manuel du DOS.

>0150 à >017F :
Zone réservée par le constructeur pour de futurs périphériques.

>0180 à >01FF :
Zone de ports libre pour l'utilisateur. Voir un exemple de décodage d'adresse dans la partie réalisation.

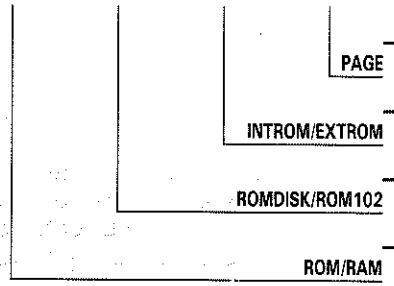
>0200 à >7FFF :
Pour cette zone deux possibilités se présentent :

— L'EXL100 :
Dans le cas de l'unité centrale seule, on ne peut avoir dans cette zone que la cartouche face avant de l'EXL100.
Dans le cas où l'on possède l'EXL135 ou l'EXELDISK, on peut paginer entre le module face avant de l'unité disquette ou la RAM interne de cette même unité. La pagination est effectuée en fonction du bit b6 du port 64 (P64).

b6 de P64 ROM / $\overline{\text{RAM}}$	PAGINATION
0	*RAM interne disquette*
1	*Module face avant disquette*

— L'EXELTEL :
Sur l'EXELTEL, on a la possibilité de paginer sur 8 zones différentes et ce grâce à 4 signaux de commandes.

b6 de P64	P56	P57	b2 de P6 (PORT B)	PAGINATION
0	*XXX*	*XXX*	*0*	*RAM EXELDISK*
0	*XXX*	*XXX*	*1*	*RAM SUPPLÉMENTAIRE*
1	*READ*	*READ*	*0*	*MODULE EXTERNE EXELTEL PAGE 1*
1	*READ*	*READ*	*1*	*MODULE EXTERNE EXELTEL PAGE 1*
1	*READ*	*WRITE*	*0*	*MODULE INTERNE EXELTEL PAGE 0*
1	*READ*	*WRITE*	*1*	*MODULE INTERNE EXELTEL PAGE 0*
1	*WRITE*	*XXX*	*0*	*MODULE EXTERNE EXELDISK PAGE 1*
1	*WRITE*	*XXX*	*1*	*MODULE EXTERNE EXELDISK PAGE 0*



	0 ou READ	1 ou WRITE
PAGE	*PAGE 1*	*PAGE 0*
INTROM/EXTROM	*EXTERNE*	*INTERNE*
ROMDISK/ROM102	*EXELTEL*	*EXELDISK*
ROM/RAM	*RAM*	*ROM*

XXX = QUELCONQUE

>8000 à >9FFF :
On peut paginer cette zone entre la ROM BOOTSTRAP ou la RAM de l'EXELMEMOIRE. On obtient cette pagination grâce au bit b4 de P64.

b4 de P64 CRAM / $\overline{\text{BOOT}}$	PAGINATION
0	*RAM BOOTSTRAP*
1	*RAM EXELMÉMOIRE*

>F000 à >FFFF :
Il s'agit d'une zone de RAM de l'EXELMÉMOIRE.

>C000 à >C7FF :
Zone contenant la RAM système. Pour plus de détails, se reporter à la table d'équivalence de cette zone en annexe.

>C800 à >CFFF :
Il s'agit d'une zone de ROM de l'unité disquette. Elle contient les routines de l'interface utilisateur. Pour plus de détail sur ces routines, se reporter au manuel du DOS.

>D000 à >EFFF :
On peut paginer cette zone entre la ROM du CROS et la RAM du DOS, et ce par l'intermédiaire du bit b5 de P64.

b5 de P64 COS / $\overline{\text{DOS}}$	PAGINATION
0	*RAM DU DOS*
1	*ROM DU CROS*

>F000 à >FFFF :
Cette zone contient les vecteurs d'interruptions, les vecteurs des TRAPS, ainsi que les TRAPS eux-mêmes. Il s'agit d'une zone ROM interne au microprocesseur. Dans le cas de l'EXELTEL, cette zone contient également certaines routines dont le détail vous est donné au chapitre *« Particularités de l'EXELTEL »*.

3) Les registres utilisés par le moniteur

A EQU R0 **B EQU R1** **FLGCOM EQU R2**

Bit	Signification au niveau « 1 »
7	*Attente d'aknowledge communication 7041/7042*
6	*Ne pas traiter le screen off*
5	*Ecran éteint (screen off)*
4	*Interface série prête*
3	*Communication avec le 7041/7042 en cours*
2	*Module connaissant la pagination (EXELTEL uniquement)*
1	*Speech en cours*
0	*Réservé*

VALEU0 EQU R3
Contient le code courant reçu du clavier ou de la manette 0 (tir orange).

VALEU1 EQU R4
Contient le code courant de la manette 1 (tir blanc).

SUICOM EQU R6
Adresse de traitement suite com.

WRKSON EQU R8
Registre de travail du speech.

PIRSON EQU R10
Adresse de la table de son aléatoire.

SONFON EQU R12
Adresse de la table de son répétitif.

TEMP1 EQU R14

TEMP2 EQU R16

TEMP3 EQU R18

TEMP4 EQU R20

TEMP5 EQU R22

TEMP6 EQU R24

TEMP7 EQU R26

TEMP8 EQU R28

TEMP9 EQU R30

TEMP10 EQU R32

4) Les ports

4.1. L'EXL100

ADD	PORT	FONCTION
>0100	P0	Registre de contrôle du TMS 7040
>0102	P2	Donnée du timer du TMS 7040
>0103	P3	Contrôle du timer du TMS 7040
>0104	P4	Port A
>0106	P6	Port B
>0124	P36	Lecture de la mémoire écran
>0125	P37	Lecture d'un registre du VDP
>0127	P39	Initialisation de l'interface VDP
>0128	P40	Initialisation de la lecture de la mémoire d'écran
>012D	P45	Ecriture dans un registre du VDP
>012E	P46	Ecriture dans la mémoire d'écran
>0130	P48	Accès à la boîte aux lettres du TMS 7041
>0131	P49	Port d'accès au connecteur d'extension X1
>0132	P50	Port d'accès au connecteur d'extension X1
>0133	51	Port d'accès au connecteur d'extension X2

PO : IOCTL

Registre de contrôle du TMS 7040

7	6	5	4	3	2	1	0	
1	0	INT 3 FLAGS	INT 3 ENABLE	INT 2 FLAGS	INT 2 ENABLE	INT 1 FLAGS	INT 1 ENABLE	READ
1	0	INT 3 CLEAR	INT 3 ENABLE	INT 2 CLEAR	INT 2 ENABLE	INT 1 CLEAR	INT 1 ENABLE	WRITE

INT X ENABLE valide les interruptions X à l'état « 1 ».
INT X FLAGS est à l'état « 1 » si il y a eu une interruption
INT X CLEAR lorsqu'il est mis à « 1 » repositionne INT FLAGS X à « 0 ».

ATTENTION : lors d'un RESET seules les interruptions 1 et 2 sont validées.

P2 - P3 : Ports d'accès du TIMER
Il est possible de générer des interruptions à intervalles réguliers. On utilise pour cela le timer.
Le port P2 contient en écriture la valeur à partir de laquelle on va décompter. En lecture, elle contient la valeur courante du compteur. P2 est appelé le diviseur du TIMER. L'interruption TIMER sera générée lorsque P2 passera à « 0 ». Il sera alors automatiquement rechargé avec la valeur initiale (valeur écrite dans ce port).
Le port P3 permet de gérer le fonctionnement du TIMER ainsi que le prédiviseur. Le prédiviseur fonctionne sur le même principe que le diviseur. Il est incrémenté tous les 16 cycles d'horloge. Lorsqu'il passe par « 0 », le diviseur est alors décrémenté.

bit	7	6	5	4	3	2	1	0
P3	TIMER ENABLE	0	0	PRÉDIVISEUR				

Le bit 7 détermine la mise en marche (état « 1 ») ou non du TIMER. La formule permettant de calculer la fréquence des interruptions est la suivante :

$$F = \frac{307200}{(PT + 1) * (DT + 1)} \text{ Hertz}$$

ou PT est le prédiviseur TIMER et DT est le diviseur TIMER.

P4 : Port A
On accède aux bits de ce port ainsi que du port B directement sur le boîtier du TMS 7040 (voir le plan).
En voici le détail :

bit	7	6	5	4	3	2	1	0
P4	SAPARIN7	SPARIN6	SPARIN5	CASSIN	NXMB	NMDM	NXMBSYIN	REV3

SPARIN7 : entrée supplémentaire sur le connecteur X2.
SPARIN6 : entrée supplémentaire sur le connecteur X1.
SPARIN5 : entrée permettant la vérification d'une carte incrustation avant la coupure de la commutation lente.
CASSIN : entrée du signal cassette.
NXMB : entrée disponible sur le connecteur X2
NMDM : entrée permettant la vérification de la présence d'une carte d'extension sur le connecteur X1.
NXMBSYIN : demande de transfert de données par le WD 1770/1772 (contrôleur de disquette).
REV3 : type d'interruption (dialogue avec le TMS 7041).

P6 : Port B
Les 4 bits de poids fort de ce port sont utilisés pour la gestion des bus d'adresses et de données.

bit	7	6	5	4	3	2	1	0
P6	*X*	*X*	*X*	*X*	*CASSOUT*	*NXINTOUT*	*REV4*	*REV2*

CASSOUT : sortie du signal cassette.
NXINTOUT : signal de pagination utilisé dans l'unité disquette.
REV4. REV2 : signaux de protocole avec le TMS 7041.
P49 : Port d'accès au connecteur X1.
Lorsqu'on accède à ce port (en lecture ou en écriture), le signal MDMEM1 du connecteur X1 passe au niveau « 0 », permettant la validation d'une carte d'extension.
P50 : Port d'accès au connecteur X1.
Lorsqu'on accède à ce port (en lecture ou en écriture), le signal MDMEM2 du connecteur X1 passe au niveau « 1 », permettant la validation d'une carte d'extension.
P51 : Port d'accès au connecteur X2.
Lorsqu'on accède à ce port (en lecture ou en écriture), le signal XMBEN du connecteur X2 passe au niveau « 1 », permettant la validation d'une carte d'extension.

4.1. L'EXELTEL

ADD	PORT	FONCTION
>0100	*P0*	*Registre de contrôle du TMS 7040*
>0102	*P2*	*Donnée du timer du TMS 7040*
>0103	*P3*	*Contrôle du timer du TMS 7040*
>0104	*P4*	*Port A*
>0106	*P6*	*Port B*
>0124	*P36*	*Lecture de la mémoire écran*
>0125	*P37*	*Lecture d'un registre du VDP*
>0127	*P39*	*Initialisation de l'interface VDP*
>0128	*P40*	*Initialisation de la lecture de la mémoire d'écran*
>012D	*P45*	*Ecriture dans un registre du VDP*
>012E	*P46*	*Ecriture dans la mémoire d'écran*
>0130	*P48*	*Accès à la boîte aux lettres du TMS 7041*
>0131	*P49*	*Port d'accès au connecteur d'extension X1*
>0132	*P50*	*Port d'accès au connecteur d'extension X1*
>0133	*51*	*Port d'accès au connecteur d'extension X2*
>0134	*P52*	*Horloge du synthétiseur vocal*
>0135	*P53*	*Sélection de l'affichage 40 ou 80 colonnes*
>0136	*P54*	*Sélection du quartz*
>0137	*P55*	*Initialisation du prédiffusé*
>0138	*P56*	*Sélection ROM EXELDISK / ROM EXELTEL*
>0139	*P57*	*Sélection ROM interne / ROM externe*
>013A	*P58*	*Accès au modem intégré*
>013B	*P59*	*Accès au modem intégré*

PO : IOCTL

Registre de contrôle du TMS 7040

7	6	5	4	3	2	1	0	
1	*0*	*INT 3 FLAGS*	*INT 3 ENABLE*	*INT 2 FLAGS*	*INT 2 ENABLE*	*INT 1 FLAGS*	*INT 1 ENABLE*	*READ*
1	*0*	*INT 3 CLEAR*	*INT 3 ENABLE*	*INT 2 CLEAR*	*INT 2 ENABLE*	*INT 1 CLEAR*	*INT 1 ENABLE*	*WRITE*

INT X ENABLE valide les interruptions X à l'état « 1 ».
INT X FLAGS est à l'état « 1 » si il y a eu une interruption
INT X CLEAR lorsqu'il est mis à « 1 » repositionne INT FLAGS X à « 0 ».

ATTENTION : lors d'un RESET seules les interruptions 1 et 2 sont validées.

P2 - P3 : Ports d'accès du TIMER
Il est possible de générer des interruptions à intervalles réguliers. On utilise pour cela le timer.
Le port P2 contient en écriture la valeur à partir de laquelle on va décompter. En lecture, elle contient la valeur courante du compteur. P2 est appelé le diviseur du TIMER. L'interruption TIMER sera générée lorsque P2 passera à « 0 ». Il sera alors automatiquement rechargé avec la valeur initiale (valeur écrite dans ce port).
Le port P3 permet de gérer le fonctionnement du TIMER ainsi que le prédiviseur. Le prédiviseur fonctionne sur le même principe que le diviseur. Il est incrémenté tous les 16 cycles d'horloge. Lorsqu'il passe par « 0 », le diviseur est alors décrémenté.

bit	7	6	5	4	3	2	1	0
P3	*TIMER ENABLE*	*0*	*0*	*PRÉDIVISEUR*				

Le bit 7 détermine la mise en marche (état « 1 ») ou non du TIMER.
La formule permettant de calculer la fréquence des interruptions est la suivante :

$$F = \frac{307200}{(PT + 1) * (DT + 1)} \text{ Hertz}$$

ou PT est le prédiviseur TIMER et DT est le diviseur TIMER.

P4 : Port A
On accède aux bits de ce port ainsi que du port B directement sur le boîtier du TMS 7040 (voir le plan).
En voici le détail :

bit	7	6	5	4	3	2	1	0
P4	*EEPROMDO*	*TAPESTAT*	*CLSTAT*	*CASSIN*	*DS*	*PERPRES*	*DRQ*	*TYPINT*

EEPROMDO : entrée pour les données de la EEPROM.
TAPESTAT : entrée permettant de vérifier si l'EXELRECORDER est prêt à enregistrer (niveau « 1 » si oui)
CLSTAT : entrée permettant le test de présence d'une carte incrustation avant la coupure de la commutation lente.
CASSIN : entrée du signal cassette.

DS : entrée de détection de sonnerie. Passe à « 0 » à une fréquence de 100 Hertzs si la sonnerie retentit.
PERPRES : entrée de détection d'une carte d'extension sur le connecteur X1 (« 0 » si présente).
DRQ : demande de transfert de donnée par le WD 1770/1772 (contrôleur de disquette).
TYPINT : type d'interruption (dialogue avec le TMS 7042).

P6 : Port B
Les 4 bits de poids fort de ce port sont utilisés pour la gestion des bus d'adresse et de donnée.

bit	7	6	5	4	3	2	1	0
P6	*X*	*X*	*X*	*X*	*CASSOUT*	*PAGE*	*ACK40*	*MBFULL*

CASSOUT : sortie du signal cassette.
PAGE : signal de pagination permettant la sélection de la page de 32K haute ou basse.
ACK40.MBFULL : signaux de protocole avec le TMS 7041.

P49 : Port d'accès au connecteur X1
Lorsqu'on accède à ce port (en lecture ou en écriture), le signal PEREN1 du connecteur X1 passe au niveau « 0 », permettant la validation d'une carte d'extension.

P50 : Port d'accès au connecteur X1
Lorsqu'on accède à ce port (en lecture ou en écriture), le signal PEREN2 du connecteur X1 passe au niveau « 1 », permettant la validation d'une carte d'extension.

P51 : Télécommande du magnétophone.
Ecriture :active le relais.
Lecture : mets le relais au repos.

P52 : Horloge du synthétiseur vocal.
Ecriture : horloge à 400 K hertzs.
Lecture : horloge à 320 K hertzs.

P53 : Sélection de l'affichage 40 ou 80 colonnes.
Ecriture : mode 80 colonnes.
Lecture : mode 40 colonnes.

P54 : Sélection du type de quartz.
Ecriture : quartz de 4.9152 M hertz.
Lecture : quartz de 9.8304 M hertz.

P55 : Initialisation du prédiffusé.
Le circuit prédiffusé de l'EXELTEL doit être initialisé. Pour cela on effectue une lecture ou une écriture de ce port. Après initialisation, l'EXELTEL se trouve dans l'état suivant :
— mode 40 colonnes.
— interface VDP initialisée.
— horloge du synthétiseur vocal à 320K hertz.
— horloge du mocroprocesseur à 9.8304M hertz.

P56 : Sélection ROM EXELDISK / ROM EXELTEL.
Ecriture : sélectionne la ROM EXELDISK.
Lecture : sélectionne la ROM EXELTEL.

P57 : Sélectionne ROM interne / ROM externe.
Ecriture : sélectionne la ROM interne.
Lecture : sélectionne la ROM externe.

RÉALISATION D'UN DÉCODEUR D'ADRESSE

Cette partie est consacrée à la description d'une carte de décodage d'adresse pouvant se relier au connecteur X2. La zone décodée sera située à partir du port P128, c'est-à-dire à partir de l'adresse >0180. On réalisera un décodage de 8 zones de 4 octets chacune.

add	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*1*	*0*	*0*	#	#	#	*	*

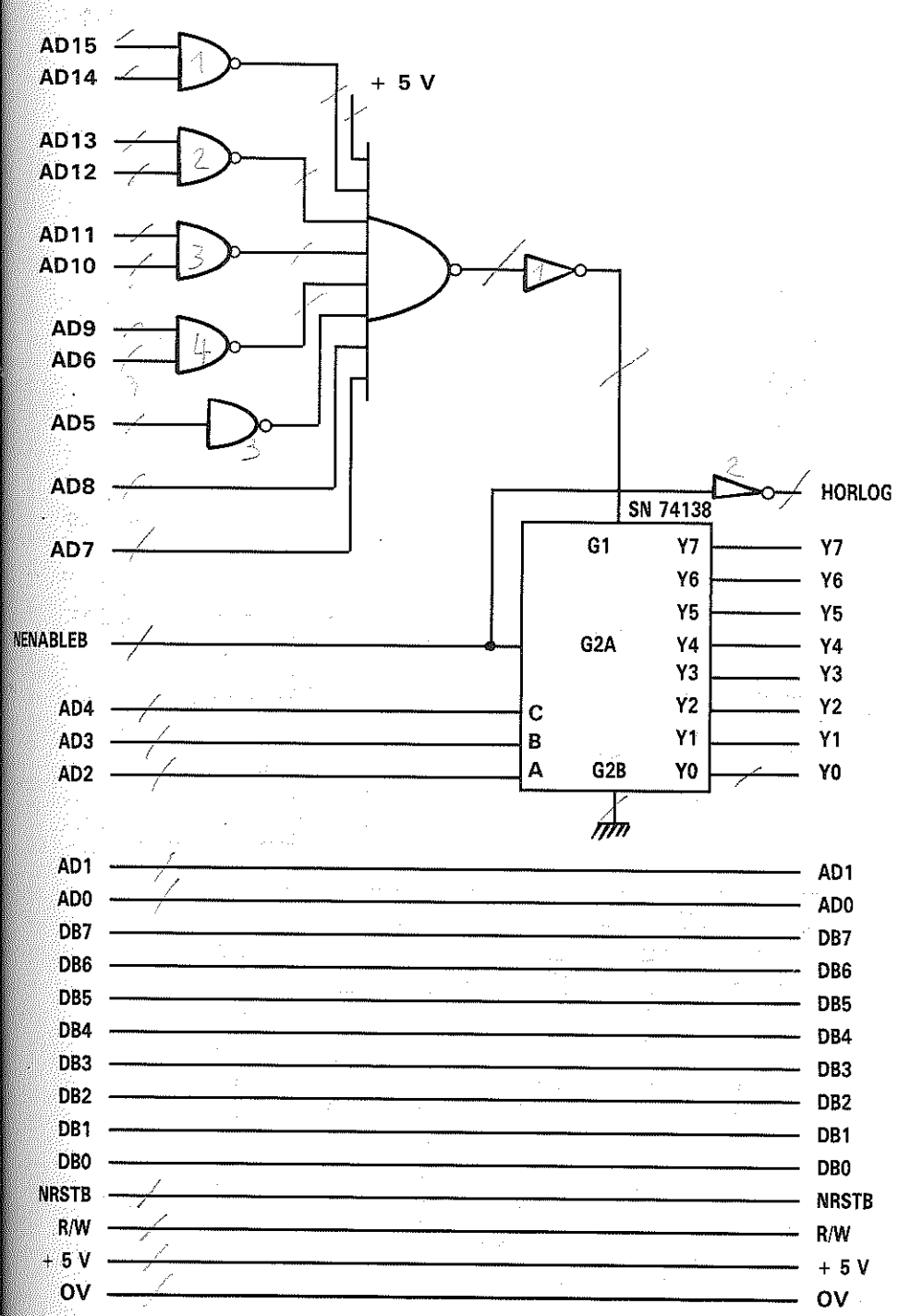
Le tableau ci-dessus représente l'état de chacun des bits d'adresse. Les « # » correspondent aux bits d'adresses servant à la sélection de la zone. Les « * » indiquent les bits servant au choix de l'octet dans la zone. Ce type de décodage permettra la mise en oeuvre rapide de PIA du type MC 6821. La sélection d'une zone parmi huit se fera à l'aide d'un circuit SN 74138. Il s'agit d'un démultiplexeur 1 parmi 8. Les 8 sorties sont valides à l'état « 0 ». On pourra, après l'avoir inversé, utiliser le signal NENABLEB comme signal d'horloge. Ce signal passe au niveau « 0 » lorsque les données sont valides. On s'en servira donc comme signal de sélection pour le démultiplexeur. Pour ce faire on le reliera à la broche G2A, l'autre (G2B) étant relié à la masse. Les bits A2, A3, et A4 seront respectivement reliés aux entrées A, B, et C du multiplexeur.

D'où la correspondance entre les adresses et les sorties de sélection :

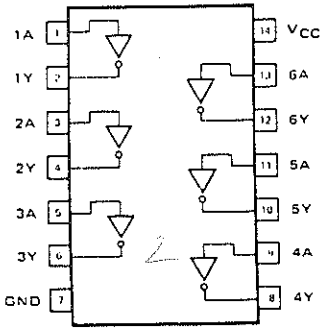
— >0180 à >0183 active Y0 (actif à « 0 »)
— >0184 à >0187 active Y1 (actif à « 0 »)
— >0188 à >018B active Y2 (actif à « 0 »)
— >018C à >018F active Y3 (actif à « 0 »)
— >0190 à >0193 active Y4 (actif à « 0 »)
— >0194 à >0197 active Y5 (actif à « 0 »)
— >0198 à >019B active Y6 (actif à « 0 »)
— >019C à >019F active Y7 (actif à « 0 »)

Le reste du circuit de décodage est construit autour de 3 composants. Il s'agit d'une NAND à 8 entrées (SN 7430), de 4 NOR à 2 entrées (SN 7402) et de 3 portes inverseuses (SN 7404). Les lignes de données (DB0 à DB7), la ligne de RESET (NRSTB), la ligne R/W ainsi que les lignes d'alimentations peuvent être reprises directement. Les lignes d'adresses AD0 et AD1 seront elles aussi directes. Pour plus de détails vous pouvez vous reporter aux plans, ainsi qu'aux fiches sur les composants utilisés.

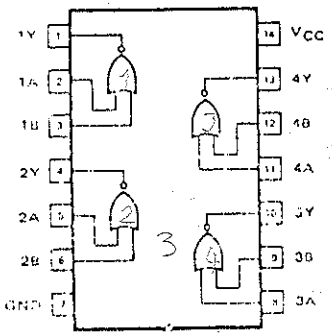
REMARQUE : Les broches B1 des connecteurs X1, X2 et X3 correspondent aux broches situées à l'extrême gauche des faces supérieures du circuit, lorsque l'on regarde l'ordinateur de l'arrière. Les broches A1 se trouvent sous les broches B1. De plus, le connecteur à la sortie de l'EXELMEMOIRE comporte deux broches supplémentaires. Il s'agit des broches A20 et B20, correspondant aux tensions de la pile de sauvegarde.



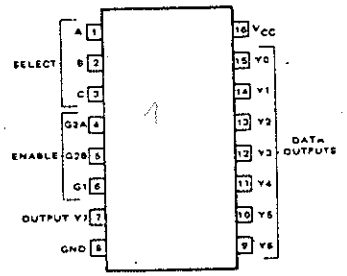
SN7404



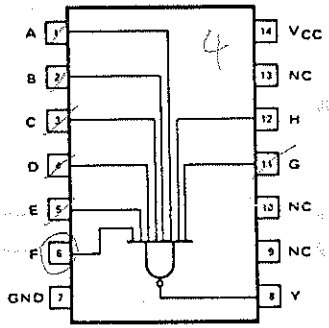
SN7402



SN74138



SN7430



FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*G2 = G2A + G2B
H = high level, L = low level, X = irrelevant

Deuxième Partie

Programmation en Assembleur

L'écran, le clavier, le synthétiseur

DÉCIMAL, BINAIRE, HEXADÉCIMAL

1) CODAGE BINAIRE

Tous les ordinateurs utilisent un système de codage pour représenter l'information.
En fait, ce système est le plus simple qui soit : il signale par deux codes différents l'absence ou la présence d'un objet.
Tous les phénomènes à deux états (on dit : binaires) peuvent être représentés directement par ce système à deux codes (code binaire).
Prenons un exemple :

> Lorsqu'un automobiliste arrive à un carrefour, deux possibilités lui sont offertes :
>
> le feu est **vert** → il peut passer
> le feu est **rouge** → il ne peut pas passer

Pour rendre compte de ce phénomène, on peut alors choisir une autre écriture. Plus simple à manipuler : 0 le feu est rouge, 1 le feu est vert. On obtient ainsi l'équivalence :

> feu **rouge** → 0 → on ne passe pas.
> feu **vert** → 1 → on passe.

Selon les cas, on pourra choisir une écriture plus significative pour représenter ces deux états :

[rouge/vert] [0/1] [noir/blanc] [oui/non]
[vrai/faux] [allumé/éteint] [ouvert/fermé]

2) LES OBJETS ET LES BITS

Dans le jargon de l'informaticien, on nomme BIT (résultat de la contraction des deux mots anglais 'binary digit') une information binaire.

En fait, un ordinateur travaille non pas avec un seul bit, mais avec un ensemble de 8, 16, voire 32 bits (on dit des MOTS, WORDS en anglais), selon son architecture.

Le microprocesseur équipant votre EXELVISION est conçu pour manipuler des OCTETS (BYTE en anglais), c'est-à-dire des mots de 8 bits. Par extension, on parle de processeur 8 bits.
Un octet est donc un groupe de 8 bits (numérotés de 0 à 7) que l'on traite dans son ensemble.

Exemple : 01011101 est un octet dont les différents bits sont les suivants :

7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	1

Le bit le plus à gauche, noté BIT(7), est le MSB : Most Significant Bit. (bit le plus significatif).
Le bit le plus à droite, noté BIT(0), est le LSB : Least Significant Bit. (bit le moins significatif).

3) DÉCIMAL, BINAIRE, ET HEXADÉCIMAL

a) La numération décimale

Les nombres que nous utilisons quotidiennement sont construits à l'aide des dix chiffres 0,1,2,3,4,5,6,7,8 et 9, ces dix chiffres représentant eux-mêmes un nombre. On parle de numération EN BASE 10.

Exemple :

la quantité huit est représentée par	8	
la quantité dix est représentée par	10	(1 puis 0)
la quantité douze est représentée par	12	(1 puis 2)
la quantité cent cinq est représentée par	105	(1 puis 0 puis 5)

Le système est simple : on décale un nombre vers la gauche chaque fois qu'on le multiplie par 10.

Inversement, nous savons déterminer la quantité représentée par un nombre, grâce à la position des chiffres qui le composent :

3	(3 unités)	vaut	3x1
17	(1 dizaine 7 unités)	vaut	1x10 + 7x1
405	(4 centaines 0 dizaine 5 unités)	vaut	4x100 + 0x10 + 5x1

Dans certains cas, seule la position des chiffres permet de distinguer les quantités représentées par deux nombres : 435 et 345 par exemple.

Les valeurs multiplicatives que nous désignons par 'unité', 'dizaine', 'centaines', etc., sont les puissances de 10 consécutives.

Exemple :
5418 vaut 5x1000+4x100+1x10+8x1
ou $5x10^3 + 4x10^2 + 1x10^1 + 8x10^0$

En résumé, on utilise des chiffres (10 en base 10, de 0 à 9) pour écrire un nombre. Ce nombre représente une quantité définie par la valeur et la position des chiffres qui le composent.

b) La numérotation binaire

On applique la même méthode de codage qu'en base 10, mais avec les deux chiffres 0 et 1 (base 2).

Les valeurs multiplicatives associées aux positions sont, cette fois, les puissances de 2, 1, 2, 4, 8, 16, 32, 64, 128, 256, ...

1) Lecture d'un nombre binaire

Nous lisons couramment 'le décimal' : nous passons près de quatre ans, à l'école, pour apprendre à compter !
Nous sommes donc contraints de transformer l'écriture binaire en écriture décimale, la seule que nous maîtrisions parfaitement.
On effectue la même série de multiplications et d'additions qu'en base 10, mais avec les puissances de 2, pour obtenir un nombre en base 10, que nous pouvons lire directement.

Exemple :
POSITION : 543210
NOMBRE : 101011
vaut $1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$
soit 1x32 + 0x16 + 1x8 + 0x4 + 1x2 + 1x1 = 43

2) Conversion d'un nombre décimal en binaire

Il suffit de soustraire les puissances de 2 successives.

Exemple : 195

195 = 128 + 67 − 1 en position 7 ($128 = 2^7$)
67 = 64 + 3 − 1 en position 6 ($64 = 2^6$)
3 = 2 + 1 − 1 en position 1 ($2 = 2^1$)
1 = 1 − 1 en position 0 ($1 = 2^0$)

Seules les positions 7, 6, 2 et 1 contiennent un 1. Les autres contiennent donc 0. On obtient :

POSITION : 76543210
NOMBRE : 11000011

195 s'écrit donc 11000011.

c) L'écriture hexadécimale

On utilise l'écriture hexadécimale (base 16) pour deux raisons :

- l'écriture en binaire est très longue et difficile à relire,
- la conversion BINAIRE → DECIMAL est lente et fastidieuse.

L'écriture hexadécimale condense un octet sur 2 caractères et la conversion DECIMAL → HEXADECIMAL est pratiquement immédiate.

Le système hexadécimal utilise 16 chiffres notés 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. Ces 16 chiffres correspondent respectivement aux quantités suivantes en décimal : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 et 15.

Les chiffres hexadécimaux correspondent aux 16 premiers chiffres binaires exprimés sur 4 bits (c'est-à-dire sur un QUARTET, soit la moitié d'un octet).

On obtient la table de correspondance suivante :

Système décimal	Système binaire	Système hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Correspondance entre les bases usuelles

Nous disposons maintenant de trois bases pour représenter un nombre. Pour les distinguer nous utiliserons :

- le symbole % pour un nombre binaire : %01010110.
- le symbole > pour un nombre hexadécimal : >1F.
- pas de symbole pour un nombre décimal : 123

Nous nous contenterons, par ailleurs, de travailler avec des mots de 8 ou 16 bits.

1) Conversion hexadécimal → décimal

On effectue la même série de multiplications et d'additions qu'en base 10, mais avec les puissances de 16, pour obtenir un nombre en base 10, que nous pouvons lire directement.

Exemple : > 3FA

POSITION : 210
NOMBRE : > 3FA vaut $3x16^2 + 0x16^1 + 1x16^0$
soit $3x256 + 15x16 + 10x1 = 1018$

2) Conversion décimal → hexadécimal

Il suffit de diviser le nombre à convertir par les puissances de 16 successives en commençant par la plus forte.

Exemple : 424

424 = 1x256 + 168 − 1 en position 2 (256 = 16^2 et > 1 = 1)
168 = 10x16 + 8 − A en position 1 (16 = 16^1 et > A = 10)
8 = 8x1 − 8 en position 0 (1 = 16^0 et > 8 = 8)

POSITION : 210
NOMBRE : 1A8

424 s'écrit donc > 1A8

3) Conversion hexadécimal → binaire

On utilise la table de correspondance.

Exemple : > 7A
7 correspond au quartet 0111
A correspond au quartet 1010
donc : > 7A = %01111010

4) Conversion binaire → hexadécimal

On découpe le nombre à convertir en quartets (on peut compléter le nombre par des 0 à gauche, si nécessaire) puis on utilise la table de correspondance.

Exemples :

%10110011
%10110011 devient 1011 0011, soit B 3, et enfin > B3

%10
%10 devient 0010 soit 2 et enfin > 2.

d) Arithmétique binaire

Le calcul en base 2 (binaire) obéit aux même règles que le calcul en base 10 (décimal).
On ajoute à l'opération usuelle + des opérations logiques : OU (OR) ET (AND) OU EXCLUSIF (XOR).

Voici les tables de ces opérations :

0 + 0 = 0	0 OR 0 = 0	0 AND 0 = 0	0 XOR 0 = 0
0 + 1 = 1	0 OR 1 = 1	0 AND 1 = 0	0 XOR 1 = 1
1 + 0 = 1	1 OR 0 = 1	1 AND 0 = 0	1 XOR 0 = 1
1 + 1 = 0	(C) 1 OR 1 = 1	1 AND 1 = 1	1 XOR 1 = 0

Remarque :

1 + 1 = 0, avec une retenue (carry en anglais) que nous avons signalée par (C). Il faudrait disposer d'un bit supplémentaire pour visualiser cette retenue :
%1 + %1 = %10 soit 0 si l'on ne garde que le premier bit ! Le phénomène existe en base 10 : 9 + 1 = 10, qui fait 0 si l'on ne garde que le premier chiffre!

Exemples :

%01001001	%10100101	%11101010	%00111101
+ %10001101	OR %11101101	AND %00101110	XOR %10100101
%11010110	%11101101	%00101010	%10011000

4) REPRÉSENTATION DE L'INFORMATION

Pour réaliser des programmes adaptés à différentes situations, il est nécessaire de manipuler des informations de natures différentes : nombres, lettres, instructions, etc.
Notre micro-processeur ne sait manipuler quant à lui, que des octets. Un même octet pourra donc contenir une valeur qui sera interprétée selon le cas, comme un nombre, une lettre, une instruction, etc.

1) Représentation des nombres

a) Binaire pur

La valeur de l'octet est prise telle quelle. On peut alors manipule des nombres compris entre 0 et 255 inclus (soit 256 valeurs).
La plus grande valeur que l'on peut coder un octet est, en effet, 255 (> FF ou %11111111).

Exemples :

>32 = %00110010 = 50
>80 = %10000000 = 128
>FF = %11111111 = 255

Ce système ne permet ni le codage des grands nombres ni celui des nombres négatifs.

b) Représentation en signe et grandeur

Dans ce système, le bit 7 (MSB) représente le signe du nombre : 0 représente le signe +, 1 le signe −. On peut alors manipuler des nombres compris entre −127 et +127.

Exemples :

>32 = %00110010 = +50
>7F = %01111111 = +127
>AO = %10100000 = −32
>FF = %11111111 = −127

Le problème est que l'on obtient deux 0 (%00000000 et %10000000) !

c) Représentation en complément à deux. Soustraction binaire

Dans ce système, le bit (MSB) représente aussi le signe du nombre : 0 représente le signe +, 1 le signe −.
Les nombres positifs vont de %00000000 à %01111111 (0 à 127)
Les nombres négatifs vont de %10000000 à %11111111 (−128 à −1)
On peut alors manipuler des nombres compris entre −128 et +127.

Exemples :

>32 = %00110010 = +50
>7F = %01111111 = +127
>80 = %10000000 = −128
>FF = %11111111 = −1

Pour obtenir le complément à deux d'un nombre, il suffit :
- d'inverser chacun des bits du nombre (0 → 1 et 1 → 0).
  On dit aussi : prendre le complément à un.
- d'ajouter 1 au nombre obtenu.

Exemple :

Prendre le complément à deux de >46

%01000110 = >46 = +70 (le bit 7 est à 0)

Complément à un : %10111001 = >B9 = −70 (le bit 7 est à 1)
+ 1 : %00000001

%10101010 = >AA = −70 (le bit 7 est à 1)

Pour effectuer une soustraction il suffit alors d'ajouter le complément à deux.

Exemple :

Calculer %01100010 − %00010111 (>62 −>17)

>17 = %00010111

complément à un = %11101000

+ 1 = %11101001 = complément à deux.

%01100010
+ %11101001

(1) 01001011 = >4B

Le (1) près du résultat binaire indique la présence d'une retenue.

2) Représentation des caractères.

Il est indispensable de pouvoir représenter des caractères, et en particulier les lettres majuscules ou minuscules, les chiffres, les signes de ponctuation, etc.
Un code, construit à cet effet, est aujourd'hui utilisé par tous : le code ASCII (American Standard Code for Information Interchange).
Le caractère A, par exemple est codé › 41, l'étoile *, › 2A.

Bits 4	3	2	1	Bits 7,6,5 / Hex 1	000	001	010	011	100	101	110	111
				Hex 0	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	'	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	''	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	¦
1	1	0	1	D	CR	GS	—	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	_	°	DEL

Codes ASCII

PREMIERS PROGRAMMES

Les programmes que vous alliez découvrir dans ce chapitre après quelques pages de présentation sont destinés à vous familiariser avec le mode de raisonnement du programmeur en assembleur. Ces programmes sont des exemples typiques des techniques mises en œuvre et des outils mis à la disposition du programmeur.

Les instructions utilisées ne sont pas forcément décrites en détail, l'important étant de comprendre leurs rôles de façon globale. On pourra se reporter, si nécessaire, au chapitre dédié à la description complète des instructions.

Certains programmes demandent l'utilisation de codes comme le code ASCII : il faut dans ce cas le consulter si nécessaire.
Dans tous les cas, les programmes sont destinés à être ESSAYÉS. Reportez-vous au mode d'emploi du logiciel que vous possédez (ASSEMBLEUR, MONITEUR, etc.)

Vous trouverez en fin de chapitre un complément d'information concernant les particularités de la programmation en assembleur sur TMS 7040. particularités abordées en vrac dans les différents programmes : le registre d'état ST, la gestion de la pile...

ADRESSAGE, INSTRUCTIONS, LANGAGE ASSEMBLEUR

I. Adressage et instructions

L'activité essentielle d'un micro-processeur est articulée autour de 2 points :
– les opérations sur les octets (déplacements, comparaisons, calculs...)
– les déplacements dans la mémoire, pour l'exécution des instructions.

A. Opérations sur les octets

Ces octets se trouvent dans la mémoire associée au processeur. L'ensemble de la mémoire accessible est organisée en cases contenant un octet. Ces cases sont repérées par des numéros : les adresses, presque toujours exprimées en hexadécimal. Certaines zones de la mémoire sont privilégiées : la zone > 00 à > 7F, qui contient les registres, notés R0 à R127, et la zone > 100 à > 1FF qui contient les ports d'entrée/sortie, notés P0 à P255. Deux registres sont eux-mêmes privilégiés : R0 et R1, appelés aussi A et B, A étant l'accumulateur (registre ou s'accumulent les résultats des calculs), B étant le registre d'index.

Dans tous les cas, il faut pouvoir accéder aux octets. Le TMS 7040 autorise plusieurs techniques d'accès (on dit techniques d'adressage).

1) Adressage implicite : STSP

Dans le cas de l'adressage implicite, les registres ou adresses concernées sont implicitement définis par l'instruction. Pour STSP, il s'agit du registre interne SP et du registre B : la valeur courante de SP est transférée dans le registre B. Exemple :

STSP : transfert le contenu de SP dans B.

2) Adressage direct, immédiat et par registre : CLR, MOV

★ Adressage direct : CLR

L'instruction CLR permet de mettre à 0 le contenu d'un des registres R0 à R127. Un registre est directement concerné et l'action à effectuer sur ce registre est liée à la définition de l'instruction. Exemples :

CLR R10 : mettre à 0 le registre R10
CLR A : mettre à 0 l'accumulateur A
CLR B : mettre à 0 le registre B

★ Adressage immédiat et par registre : MOV

L'instruction MOV assure le transfert d'une valeur située à une adresse appelée SOURCE dans une adresse appelée DESTINATION.

L'adresse DESTINATION est toujours un registre (A,B,R12, etc.). L'adresse SOURCE, par contre, peut être désignée de différentes façons :

a) Adressage immédiat

La valeur à transférer est donnée de façon immédiate : elle n'est pas située à une adresse. Exemples :

MOV %15,A : la valeur décimale 15 est transférée dans A.
MOV %> A2,B : la valeur hexadécimale > A2 est transférée dans B.
MOV %100,R12 : la valeur décimale 100 est transférée dans R12.

b) Adressage par registre

La valeur à transférer est contenue dans un des registres. Exemples :

MOV R12,A : la valeur contenue dans R12 est transférée dans A.
MOV R14,R15 : la valeur contenue dans R14 est transférée dans R12.
MOV A,R20 : la valeur contenue dans A est transférée dans R20.
MOV A,B : la valeur contenue dans A est transférée dans B.

3) Adressage étendu : LDA

L'adresse contenant la valeur est une adresse quelconque : elle peut être indiquée de trois façons : absolue, indexée, indirecte.

LDA permet de transférer le contenu de l'adresse indiquée dans le registre A.

★ **Adresse absolue**

L'adresse est donnée par son nom. Exemples :

```
LDA@1275        : transfert dans A du contenu de l'adresse déci-
                  male 1275.
LDA@F000        : transfert dans A du contenu de l'adresse
                  hexadécimale >F000.
```

★ **Adresse indexée**

L'adresse contenant la valeur à transférer est calculée en ajoutant une adresse de base ou contenu du registre d'index B. Exemples :

```
MOV %>0A,B      : initialise B à >0A (10 décimal)
LDA@1000(B)     : transfert la valeur contenue dans >100A
                  dans A.
                : [>100A=>1000+(B)]
```

★ **Adresse indirecte**

L'adresse contenant la valeur à transférer est elle-même contenue dans une paire de registres. On indique, dans l'instruction, le registre de plus petit numéro. Exemples :

```
MOV %>AA,A      : initialise A à >AA
STA %>A012      : stocke (A) en >A012 (>A012 contient donc
                  >AA).
CLR A           : efface 1 (A=0)
MOV %>A0,R10    : transfert >A0 dans R10
MOV %>12,R11    : transfert >12 dans R11. Ici (R10,R11) con-
                  tient >A012.
LDA *R11        : charge dans A la valeur contenue dans
                  l'adresse
                : pointée par (R10,R11), soit >AA.
```

B. Déplacements dans la mémoire

Les instructions qu'un processeur doit exécuter se trouvent elles aussi dans la mémoire. Le processeur les exécute les unes après les autres, à partir de l'adresse que nous lui avons indiquée.

Ces instructions peuvent être placées les unes à la suite des autres, c'est-à-dire de façon séquentielle, mais cette organisation ne permet

que très rarement de résoudre un problème donné. Suivant le résultat d'un test, par exemple, il peut être nécessaire d'exécuter telle ou telle partie du programme, ou d'éviter l'exécution d'un groupe d'instructions.

Les instructions qui facilitent ce contrôle du déroulement des programmes sont les instructions de saut. Elles agissent sur le registre PC, de deux manières différentes :

1) Instructions de saut absolu : BR

Lors de l'exécution de BR, l'adresse indiquée est chargée dans PC. Le programme continue de se dérouler à partir de cette nouvelle adresse. Exemples :

Adresse	Instruction	Commentaire
>7400	MOV %5,A	: on met 5 dans A
>7402	BR @>7600	: branchement à l'adresse >7600
. . . . .		
. . . . .		
>7600	. . . . .	: l'exécution se poursuit ici

2) Instructions de saut relatif : JMP

Lors de l'exécution de JMP, l'adresse de branchement est calculée à partir de la valeur courante de PC (adresse de l'instruction de saut +2) et du déplacement (offset) considéré comme un nombre signé en complément à 2. L'adresse de branchement est RELATIVE à PC. Le programme continue de se dérouler à partir de cette nouvelle adresse. Exemples :

Adresse	Instruction	Commentaire
>7400	MOV %5,A	: on met 5 dans A
>7402	JMP +>10	: branchement à l'adresse >7414 (>7404+>10)
>7404	. . . . .	
. . . . .		
>7412	. . . . .	: l'exécution se poursuit ici

II. Langage assembleur

Un langage assembleur est un langage qui assure, en particulier, la traduction en code binaire directement utilisable par le processeur d'un langage plus facile à manipuler.

Le programme écrit en langage assembleur est nommé le programme SOURCE ou code source, et le langage issu de la traduction par l'assembleur, le code OBJET.

Globalement, un assembleur traduit un programme source en code binaire et stocke les codes obtenus en mémoire, à l'adresse que nous lui avons indiquée.

Parmi les nombreux avantages qu'offre l'utilisation d'un langage assembleur on peut en dégager quelques uns :

★ La clarté et la simplicité du code source

Les instructions binaires sont définies par des MNEMONIQUES, tels que CLR, MOV, CMP, qui décrivent de façon condensée le rôle de chaque instruction. L'assembleur se chargera, lors de la traduction, d'associer son code à chaque mnémonique et de vérifier la cohérence éventuelle de l'adressage utilisé.

★ La possibilité d'utiliser des étiquettes

Un assembleur permet de désigner les adresses par des noms que l'on appelle étiquettes.
Ces étiquettes servent à repérer une adresse particulière : adresse d'un saut, adresse d'une table, adresse d'un sous-programme, etc.

★ Le calcul automatique d'adresses

Un assembleur prend en charge le calcul des adresses lors de l'assemblage. Il calcule alors les adresses réelles correspondant aux étiquettes ainsi que les déplacements relatifs issus des instructions de saut.

★ La possiblité de commenter le programme

Cela semble évident, pourtant c'est un des plus gros avantages ! On peut ainsi reprendre un programme longtemps après sa création pour le corriger, etc. Ces commentaires n'apparaissent pas dans le code objet.

Exemple de ce qu'il faut programmer pour additionner 5 et 30 :

1) Sans assembleur (en hexadécimal)

adresse	code (les codes sont pris dans les tables d'instructions).
0000	22 05
0002	52 1E
0004	38 00
0006	0A

2) Avec un assembleur

```
:
: ADDITION 5+30
:
DEB         MOV %05,A     : METTRE 05 DANS A
            MOV %30,B     : METTRE 30 DANS B
            ADD A,B       : ADDITIONNER
            RETS          : FIN DU PROGRAMME
:
            END           : FIN DU CODE SOURCE
```

Les commentaires sont précédés du point-virgule (:), on peut utiliser des étiquettes (DEB), le programme est directement lisible.

La phrase d'assemblage correspond à la traduction du code source (type 2) en code objet (type 1). Exemple : listing du programme ci-dessus après assemblage.

```
0000              :
0000              : ADDITION 5+30
0000              :
0000 2205   DEB       MOV %05,A     : METTRE 05 DANS A
0002 521E             MOV %30,B     : METTRE 30 DANS B
0006 3800             ADD A,B       : ADDITIONNER
0008 0A               RETS          : FIN DU PROGRAMME
                  :
0009                  END           : FIN DU CODE SOURCE
0000 error
```

La première colonne contient l'adresse de l'instruction en mémoire. Chaque instruction est codée sur un, deux, trois voire quatre octets : c'est la deuxième colonne. On reconnaît au passage les valeurs 5 et 30 (> 05 et > 1E en hexadécimal).

1. Addition 8 bits

L'addition de deux nombres de 8 bits ne pose pas de réel problème car l'instruction ADD est prévue à cet effet.

Le programme suivant additionne la valeur contenue dans A à celle contenue dans B. Le résultat est dans B.

On initialise A et B par MOV. RETS signale la fin du programme. Après exécution, le résultat est dans B, et A n'a pas été modifié.

Essayez d'additionner des valeurs plus grandes : 30 et 60, 130 et 100, etc., en vérifiant le contenu de B après chaque essai, sans oublier que le registre B est un registre 8 bits qui ne peut contenir que des nombres compris entre 0 et 255.

```
                        ;
                        ;**********************
                        ;*                     *
                        ;*  ADDITION 8 BITS    *
                        ;*                     *
                        ;**********************
                        ;
                        ;
0000 2205               MOV   %05.A     ;INIT A
0002 521E               MOV   %30.B     ;INIT B
0004 3800               ADD   A.B       ;ADDITION
0006 0A                 RETS            ;TERMINE
                        ;
0007                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

2. Addition 16 bits

Un nombre de 16 bits occupe deux octets ; l'octet de poids fort et l'octet de poids faible. Exemple : > 12A0. L'octet de poids fort est > 12, l'octet de poids faible > A0.
Pour réaliser une addition entre deux nombres de 16 bits, on additionne les deux octets de poids faible, puis les deux octets de poids fort en tenant compte de la retenue.
Un octet ne peut contenir que des nombres compris entre 0 et 255. Si le résultat d'une addition dépasse 255, le bit de retenue C est mis à 1 par le processeur, sinon C est à 0. Exemple : 254 + 12 = 266 qui, étant supérieur à 255, donne 11 avec C = 1 (266 = 255 + 11).
On peut ensuite tenir compte de cette retenue lorsque l'on additionne les octets de poids fort.

Le programme suivant additionne deux nombres de 16 bits contenus l'un dans la paire de registre (A,B), l'autre dans la paire de registres (R10,R11). Le résultat est dans (A,B).

On initialise (A,B) et (R10,R11) par un MOVD.
L'addition se fait en deux temps et l'on tient compte de la retenue éventuelle grâce à l'instruction ADC.

```
                        ;
                        ;**********************
                        ;*                     *
                        ;* ADDITION 16 BITS    *
                        ;* VERSION 1           *
                        ;*                     *
                        ;**********************
                        ;
0000 88007D01           MOVD  %125.B    ;INIT (A.B)
0004 88015E0B           MOVD  %350.R11  ;INIT (R10.R11)
0008 380B               ADD   R11.B     ;POIDS FAIBLES
000A 190A               ADC   R10.A     ;POIDS FORTS
000C 0A                 RETS            ;TERMINE
                        ;
000D                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

On peut envisager de résoudre le problème d'une autre façon : les deux nombres sont accessibles en mémoire et le résultat doit être stocké lui aussi en mémoire.
On utilise un des avantages de l'assembleur : la possibilité d'associer un nom à un emplacement mémoire, à l'aide de la pseudo-instruction EQU.
Dans cet exemple, en en tenant compte du fait que les valeurs manipulées occupent deux octets, on associe à chaque nombre les étiquettes :

VAL1H et VAL1L pour le premier nombre, octet haut (HIGH) et octet bas (LOW).
VAL2H et VAL2L pour le second nombre, octet haut (HIGH) et octet bas (LOW).
RESH et RESL pour le résultat, octet haut (HIGH) et octet bas (LOW).

Les deux nombres, qui ne sont donc accessibles que par LDA, sont copiés dans les deux paires de registres (R10,R11) et (R12,R13), qui permettent la réalisation de l'addition. Le résultat est rangé en mémoire par STA, après copie dans (A,B) par MOVD.

Dans le cas du programme proposé, il faut stocker les valeurs en (> C7F0, > C7F1) et (> C7F2, > C7F3) et lire le résultat en (> C7FA4, > C7F5).

```
                    *
                    *************************
                    *                       *
                    * ADDITION 16 BITS      *
                    * VERSION 2             *
                    *                       *
                    *************************
                    *
0000 C7F0       =   VAL1H   EQU   >C7F0
0000 C7F1       =   VAL1L   EQU   >C7F1
0000 C7F2       =   VAL2H   EQU   >C7F2
0000 C7F3       =   VAL2L   EQU   >C7F3
0000 C7F4       =   RESH    EQU   >C7F4
0000 C7F5       =   RESL    EQU   >C7F5
                    *
0000 8AC7F0             LDA   @VAL1H    :RECUPERE
0003 D00A               MOV   A,R10     :VALEUR 1
0005 8AC7F1             LDA   @VAL1L    :16 BITS DANS
0008 D00B               MOV   A,R11     :REGISTRES
                    *
000A 8AC7F2             LDA   @VAL2H    :RECUPERE
000D D00C               MOV   A,R12     :VALEUR 2
000F 8AC7F3             LDA   @VAL2L    :16 BITS DANS
0012 D00D               MOV   A,R13     :REGISTRES
                    *
```



```
0014 480B0D             ADD   R11,R13   :ADDITIONNE
0017 480A0C             ADC   R10,R12
                    *
001A 980D01             MOVD  R13,B     :COPIE RESULTAT
001D 8BC7F4             STA   @RESH     : EN MEMOIRE
0020 52                 MOV   B,A
0021 8BC7F5             STA   @RESL
                    *
0024 0A                 RETS
                    *
0025                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

La version 3 de ce programme ne diffère que très légèrement de la version 1, on a seulement donné un nom aux deux paires de registres destinés à contenir les deux nombres.
Le programme devient plus facile à lire et à modifier.

```
                    *
                    *************************
                    *                       *
                    * ADDITION 16 BITS      *
                    * VERSION 3             *
                    *                       *
                    *************************
                    *
0000 001E       =   R1H     EQU   R30
0000 001F       =   R1L     EQU   R31
0000 0020       =   R2H     EQU   R32
0000 0021       =   R2L     EQU   R33
                    *
0000 C7F0       =   VAL1H   EQU   >C7F0
0000 C7F1       =   VAL1L   EQU   >C7F1
0000 C7F2       =   VAL2H   EQU   >C7F2
0000 C7F3       =   VAL2L   EQU   >C7F3
0000 C7F4       =   RESH    EQU   >C7F4
0000 C7F5       =   RESL    EQU   >C7F5
                    *
0000 8AC7F1             LDA   @VAL1L    :RECUPERE
0003 C0                 MOV   A,B       :VALEUR 1
0004 8AC7F0             LDA   @VAL1H    :16 BITS DANS
0007 98011F             MOVD  B,R1L     :REGISTRES
                    *
```

```
000A 8AC7F3             LDA   @VAL2L     :RECUPERE
000D C0                 MOV   A,B         :VALEUR 2
000E 8AC7F2             LDA   @VAL2H     :16 BITS DANS
0011 980121             MOVD  B,R2L      :REGISTRES
                        :
0014 481F21             ADD   R1L,R2L    :ADDITIONNE
0017 491E20             ADC   R1H,R2H
                        :
001A 1220               MOV   R2H,A      :COPIE RESULTAT
001C 8BC7F4             STA   @RESH      :EN MEMOIRE
001F 1221               MOV   R2L,A
0021 8BC7F5             STA   @RESL
                        :
0024 0A                 RETS
                        :
0025                    END
0000 Error(s)
```

3. Conversion chiffre décimal ➡ valeur binaire

La conversion d'un code en un autre code est peut-être le problème le plus souvent rencontré, avec le déplacement d'octets, par le programmeur en assembleur.

Un nombre, en particulier, doit être traduit dans ses différentes représentations selon son utilisation: calculs, affichage, etc.

Pour coder un chiffre décimal (de 0 à 9) que l'on doit afficher, par exemple, on utilise le code ASCII, qui associe un code à chaque caractère:

0 est codé 48 (> 30)	5 est codé 53 (> 35)
1 est codé 49 (> 31)	6 est codé 54 (> 36)
2 est codé 50 (> 32)	7 est codé 55 (> 37)
3 est codé 51 (> 33)	8 est codé 56 (> 38)
4 est codé 52 (> 34)	9 est codé 57 (> 39)

De la même façon, un caractère reçu de l'extérieur (du clavier, par exemple) est généralement codé sous forme ASCII.

Les valeurs permettant de calculer réellement, par contre, sont des valeurs binaires: il faut donc effectuer une conversion.

La conversion est simple:

0 codé > 30 donne > 00	5 codé > 35 donne > 05
1 codé > 31 donne > 01	6 codé > 36 donne > 06
2 codé > 32 donne > 02	7 codé > 37 donne > 07

3 codé > 33 donne > 03	8 codé > 38 donne > 08
4 codé > 34 donne > 04	9 codé > 39 donne > 09

Il suffit donc de soustraire > 30 au code ASCII pour obtenir la valeur correspondante. La soustraction est réaliser grâce à SUB.

Dans le programme suivant, le code du chiffre (entre > 30 et > 39) doit être stocké à l'adresse CAR, la valeur binaire est lue en BIN après exécution.

```
                        :
                        :**************************
                        :*                        *
                        :* CONVERSION CARACTERE   *
                        :* NUMERIQUE DECIMAL      *
                        :* EN BINAIRE             *
                        :* VERSION 1              *
                        :*                        *
                        :**************************
                        :
0000 C7F0       = CAR   EQU   >C7F0
0000 C7F1       = BIN   EQU   >C7F1
                        :
0000 8AC7F0             LDA   @CAR       :RECUPERE CAR
0003 2A30               SUB   %>30,A     :CONVERSION
0005 8BC7F1             STA   @BIN
                        :
0008 0A                 RETS
                        :
0009                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

La version 2 réalise la conversion d'une autre façon: on supprime par une opération logique tous les bits qui ne nous intéressent pas. Ceci est possible grâce à la constatation suivante (les quartets ont été séparés pour faciliter la lecture):

> 30 = ?0011 0000 et l'on veut obtenir ?0000 0000
> 31 = ?0011 0001 et l'on veut obtenir ?0000 0001
> 32 = ?0011 0010 et l'on veut obtenir ?0000 0010
> 33 = ?0011 0011 et l'on veut obtenir ?0000 0011
> 34 = ?0011 0100 et l'on veut obtenir ?0000 0100

> 35 = ?0011 0101 et l'on veut obtenir ?0000 0101
> 36 = ?0011 0110 et l'on veut obtenir ?0000 0110
> 37 = ?0011 0111 et l'on veut obtenir ?0000 0111
> 38 = ?0011 1000 et l'on veut obtenir ?0000 1000
> 39 = ?0011 1001 et l'on veut obtenir ?0000 1001

Il suffit donc d'annuler les bits du quartet de poids fort. AND agit de la manière suivante sur chacun des bits: si les 2 bits sont à 1, le bit résultant est à 1, sinon il est à 0.
En effectuant un AND entre un nombre de > 30 à > 39 et le nombre > 0F, on annule le premier quartet sans modifier le second. Ex:

nombre 1	> 32	?0011 0010
nombre 2	> 0F	?0000 1111
AND:		
	> 02	?0000 0010

```
                    :
                    :************************
                    :*                      *
                    :* CONVERSION CARACTERE *
                    :* NUMERIQUE DECIMAL    *
                    :* EN BINAIRE           *
                    :* VERSION 2            *
                    :*                      *
                    :************************
                    :
0000 C7F0       = CAR     EQU   >C7F0
0000 C7F1       = BIN     EQU   >C7F1
                    :
0000 8AC7F0             LDA   @CAR        :RECUPERE CAR
0003 230F               AND   %>0F.A      :CONVERSION
0005 8BC7F1             STA   @BIN
                    :
0008 0A                 RETS
                    :
0009                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

Les 2 versions précédentes ne vérifient pas le contenu de CAR: on peut y mettre n'importe quoi: le résultat ne sera cohérent que si la valeur stockée en CAR est le code ASCII d'un chiffre.

Cette nouvelle version vérifie si le code donné est un code ASCII valide. Si c'est le cas, l'adresse ERR est mise à 0, la conversion est

effectuée et le résultat stocké en BIN. Dans le cas inverse, l'adresse ERR est mise à 1.

On utilise l'instruction CMP qui positionne les indicateurs du registre d'état ST en fonction des deux valeurs comparées. Les instructions de saut permettent ensuite d'agir en conséquence.

En particulier, JL impose un saut à l'adresse indiquée si la valeur destination est strictement inférieure à la valeur source et JHS un saut à l'adresse indiquée si la valeur destination est supérieure ou égale à la valeur source.

Un code incorrect est soit strictement inférieur à > 30 (code du caractère '0'), soit supérieur ou égal à > 3A (> 39 + > 1, code du caractère ':'). On peut effectuer la comparaison au moyen des caractères, ce qui est plus parlant.

Après exécution, on vérifiera avant tout le contenu de ERR.

```
                    :
                    :************************
                    :*                      *
                    :* CONVERSION CARACTERE *
                    :* NUMERIQUE DECIMAL    *
                    :* EN BINAIRE           *
                    :* VERSION 3            *
                    :*                      *
                    :************************
                    :
0000 C7F0       = CAR     EQU   >C7F0
0000 C7F1       = BIN     EQU   >C7F1
0000 C7F2       = ERR     EQU   >C7F2
                    :
0000 8AC7F0             LDA   @CAR        :RECUPERE CAR
                    :
0003 2D30               CMP   %'0'.A      :VERIFIE SI
0005 E70E               JL    ERREUR      :CHIFFRE VALIDE
0007 2D3A               CMP   %':'.A
0009 E30A               JHS   ERREUR
                    :
000B 230F               AND   %>0F.A      :CONVERSION
000D 8BC7F1             STA   @BIN
0010 B5                 CLR   A           : 0 DANS A
0011 8BC7F2             STA   @ERR        : EFFACE ERR
0014 0A                 RETS              : TERMINE
                    :
0015 2201       ERREUR  MOV   %1.A        : INDIQUE
0017 8BC7F2             STA   @ERR        : L'ERREUR
001A 0A                 RETS
                    :
001B                    END
0000 Error(s)
```

4. Conversion chiffre hexadécimal ➡ valeur binaire

Le même système peut être utilisé pour convertir un caractère hexadécimal en sa valeur binaire.

Un caractère hexadécimal peut être un des 10 chiffres 0 à 9 ou une des lettres A,B,C,D,E,F, qui sont respectivement codées >41, >42, >43, >44, >45 et >46.

car: 0 codé >30 et l'on veut obtenir >00
car: 1 codé >31 et l'on veut obtenir >01
car: 2 codé >32 et l'on veut obtenir >02
car: 3 codé >33 et l'on veut obtenir >03
car: 4 codé >34 et l'on veut obtenir >04
car: 5 codé >35 et l'on veut obtenir >05
car: 6 codé >36 et l'on veut obtenir >06
car: 7 codé >37 et l'on veut obtenir >07
car: 8 codé >38 et l'on veut obtenir >08
car: 9 codé >39 et l'on veut obtenir >09

car: A codé >41 et l'on veut obtenir >0A
car: B codé >42 et l'on veut obtenir >0B
car: C codé >43 et l'on veut obtenir >0C
car: D codé >44 et l'on veut obtenir >0D
car: E codé >45 et l'on veut obtenir >0E
car: F codé >46 et l'on veut obtenir >0F

Le problème est plus complexe : séparons-le en deux étapes :

1) on vérifie si le caractère est un chiffre.
   Si oui, on le convertit et le travail est terminé.
   Si non il faut vérifier si le caractère est une lettre valide.

2) on vérifie si le caractère est une lettre valide.
   Si oui, on le convertit en remarquant que :
   >41 –> 07 => 3A
   >42 –> 07 => 3B
   >43 –> 07 => 3C
   >44 –> 07 => 3D
   >45 –> 07 => 3E
   >46 –> 07 => 3F

et qu'il suffit d'annuler le quartet de poids fort pour obtenir ce que l'on cherche.

Si non, le caractère proposé n'est pas un caractère hexadécimal. L'indicateur d'erreur est mis à 1.


```
                    *
                    *****************************
                    *                           *
                    * CONVERSION CARACTERE      *
                    * NUMERIQUE HEXADECIMAL     *
                    * EN BINAIRE                *
                    * VERSION 1                 *
                    *                           *
                    *****************************
                    *
0000 C7F0     = CAR     EQU   >C7F0
0000 C7F1     = BIN     EQU   >C7F1
0000 C7F2     = ERR     EQU   >C7F2
                    *
0000 BAC7F0         *       LDA   @CAR        *RECUPERE CAR
                    *
0003 2D30           *       CMP   %'0'.A      *VERIFIE SI
0005 E722           *       JL    ERREUR      *CHIFFRE VALIDE
0007 2D3A           *       CMP   %':'.A
0009 E30A           *       JHS   LETTRE
                    *
000B 230F           *       AND   %>0F.A      *CONVERSION
000D 8BC7F1         *       STA   @BIN
0010 B5             *       CLR   A           * 0 DANS A
0011 8BC7F2         *       STA   @ERR        * EFFACE ERR
0014 0A             *       RETS              * TERMINE
                    *
0015 2D41           LETTRE  CMP   %'A'.A      * LES LETTRES
0017 E710           *       JL    ERREUR      * A -> F
0019 2D47           *       CMP   %'G'.A      * SONT
001B E30C           *       JHS   ERREUR      * AUTORISEES
                    *
001D 2A07           *       SUB   %07.A       * CONVERSION
001F 230F           *       AND   %>0F.A
                    *
0021 8BC7F1         *       STA   @BIN
0024 B5             *       CLR   A           * 0 DANS A
0025 8BC7F2         *       STA   @ERR        * EFFACE ERR
0028 0A             *       RETS              * TERMINE
                    *
0029 2201           ERREUR  MOV   %1.A        * INDIQUE
002B 8BC7F2         *       STA   @ERR        * L'ERREUR
002E 0A             *       RETS
                    *
002F                *       END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

L'utilisation de sous-programmes dans cette deuxième version permet de clarifier l'écriture.

Un sous-programme est un ensemble d'instructions permettant de résoudre un problème ou réalisant une fonction particulière. Lors de l'appel à un sous-programme par CALL, l'adresse qui suit l'appel est sauvegardé dans la pile puis il y a branchement à l'adresse indiquée après le CALL. Le déroulement se poursuit jusqu'à la rencontre d'une instruction RETS, qui force le processeur à 'dépiler' l'adresse de retour (qui devient donc la nouvelle adresse d'exécution).

```
                    :
                    :**************************
                    :*                        *
                    :* CONVERSION CARACTERE   *
                    :* NUMERIQUE HEXADECIMAL  *
                    :* EN BINAIRE             *
                    :* VERSION 2              *
                    :**************************
                    :
0000 C7F0       = CAR     EQU   >C7F0
0000 C7F1       = BIN     EQU   >C7F1
0000 C7F2       = ERR     EQU   >C7F2
                    :
0000 8AC7F0             LDA   @CAR        :RECUPERE CAR
                    :
0003 8E001B             CALL  @CHIFF
0006 E305               JC    CBON        : C'EST 1 CHIF
0008 8E0029             CALL  @LETTRE
000B E708               JNC   ERREUR      : PAS 1 LETTRE
                    :
000D 8BC7F1     CBON    STA   @BIN        : A = VAL BIN
0010 B5                 CLR   A           : 0 DANS A
0011 8BC7F2             STA   @ERR        : EFFACE ERR
0014 0A                 RETS              : TERMINE
                    :
0015 2201       ERREUR  MOV   %1,A        : INDIQUE
0017 8BC7F2             STA   @ERR        : L'ERREUR
001A 0A                 RETS
                    :
001B 2D30       CHIFF   CMP   %'0',A      ;VERIFIE SI
001D E708               JL    PASCH       :CHIFFRE VALIDE
001F 2D3A               CMP   %':',A
0021 E304               JHS   PASCH
0023 230F               AND   %>0F,A      ;CONVERSION
0025 07                 SETC              : INDIQUE OK
0026 0A                 RETS              : ET RETOUR
0027 B0         PASCH   CLRC              : INDIQUE ERR
0028 0A                 RETS              : ET RETOUR
                    :
0029 2D41       LETTRE  CMP   %'A',A      ; LES LETTRES
002B E70A               JL    PASLET      : A -> F
```



```
002D 2D47               CMP   %'G',A      : SONT
002F E306               JHS   PASLET      : AUTORISEES
0031 2A07               SUB   %07,A       : CONVERSION
0033 230F               AND   %>0F,A
0035 07                 SETC              : INDIQUE OK
0036 0A                 RETS              : ET RETOUR
0037 B0         PASLET  CLRC              : INDIQUE ERR
0038 0A                 RETS              : ET RETOUR
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

On a défini deux sous-programmes :

CHIFF qui teste le contenu de A et renvoit :

C = 0 si le contenu de A n'est pas le code ASCII d'un chiffre.

C = 1 si le contenu de A est le code ASCII d'un chiffre.
A contient alors la valeur binaire correspondante (0 à 9).

LETTRE qui teste le contenu de A et renvoit :

C = 0 si le contenu de A n'est pas le code ASCII d'une lettre valide (A à F).

C = 1 si le contenu de A est le code ASCII d'une lettre valide.
A contient alors la valeur binaire correspondante (> 0A à > 0F).

Dans les deux cas, C joue le rôle d'un indicateur que nous positionnons nous-mêmes et qui n'est lié ni au résultat d'un calcul (ADD, ...) ni au résultat d'une comparaison (CMP, ...).

5. Conversion 2 chiffres décimaux ➨ valeur binaire

Le problème posé par la conversion de 2 caractères décimaux en un nombre binaire nous permet d'introduire l'instruction MPY, qui réalise une multiplication entre l'opérande source et l'opérande destination (valeurs 8 bits), le résultat se trouvant toujours dans la paire de registres (A,B).

En effet, on remarque que pour obtenir la valeur binaire d'un nombre de 2 chiffres décimaux, il faut multiplier le premier par 10 puis lui ajouter le second. Exemple :

$$23 = 2 \times 10 + 3$$

On utilise par ailleurs les sous-programmes définis antérieurement (on sait qu'ils fonctionnent, alors...), une adresse mémoire supplémentaire CAR2 destinée à contenir le second chiffre, et un registre permettant de stocker temporairement la valeur intermédiaire.

Il faut placer les deux caractères en CAR1 et CAR2 puis lire le résultat en BIN après exécution.

```
                    ;
                    ;*****************************
                    ;*                           *
                    ;* CONVERSION 2 CARACTERES   *
                    ;* NUMERIQUES DECIMAUX       *
                    ;* EN BINAIRE                *
                    ;* VERSION 1                 *
                    ;*                           *
                    ;*****************************
                    ;
0000 001E       = TEMP1   EQU   R30
                    ;
0000 C7F0       = CAR1    EQU   >C7F0
0000 C7F1       = CAR2    EQU   >C7F1
0000 C7F2       = BIN     EQU   >C7F2
0000 C7F3       = ERR     EQU   >C7F3
                    ;
0000 8AC7F0             LDA   @CAR1      :RECUP CAR 1
0003 8E0027             CALL  @CHIFF     ;
0006 E719               JNC   ERREUR
0008 2C0A               MPY   %10,A
000A D11E               MOV   B,TEMP1
                    ;
000C 8AC7F1             LDA   @CAR2      :RECUP CAR 2
000F 8E0027             CALL  @CHIFF
0012 E70D               JNC   ERREUR
0014 48001E             ADD   A,TEMP1
                    ;
0017 121E               MOV   TEMP1,A
0019 8BC7F2             STA   @BIN
```



```
001C B5                 CLR   A          ; 0 DANS A
001D 8BC7F3             STA   @ERR       ; EFFACE ERR
0020 0A                 RETS             ; TERMINE
                    ;
0021 2201       ERREUR  MOV   %1,A       ; INDIQUE
0023 8BC7F3             STA   @ERR       ; L'ERREUR
0026 0A                 RETS
                    ;
                    ;
0027 2D30       CHIFF   CMP   %'0',A     ;VERIFIE SI
0029 E708               JL    PASCH      ;CHIFFRE VALIDE
002B 2D3A               CMP   %':',A
002D E304               JHS   PASCH
002F 230F               AND   %>0F,A     ;CONVERSION
0031 07                 SETC             ; INDIQUE OK
0032 0A                 RETS             ; ET RETOUR
                    ;
0033 B0         PASCH   CLRC             ; INDIQUE ERR
0034 0A                 RETS             ; ET RETOUR
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

6. Recherche d'un caractère dans une table

On est assez souvent amené à rechercher une valeur dans une liste de valeur (dans une table). Cette valeur peut être un symbole, le code d'une touche, le numéro d'un sous-programme, un nombre quelconque.

Le registre B joue dans ce cas un rôle particulièrement important : celui de registre d'index.

Connaissant l'adresse de base d'une table, on peut accéder à l'un de ses éléments de deux façons :

– en donnant l'adresse de l'élément,
– en calculant l'adresse de l'élément en ajoutant le contenu du registre B à l'adresse de base.

Exemple :
Soit la table de 10 éléments, située en > C700 et contenant les lettres E,X,E,L,V,I,S,I,O,N dans cet ordre.

On peut accéder à la lettre O en donnant son adresse : LDA %> C708.
On peut accéder à cette même lettre en initialisant B à 8 et en utilisant l'adressage indexé :

```
TABLE   EQU >C700
:
        MOV  %8,B
        LDA  @TABLE(B)
```

8 valant 8 et TABLE valant >C700, l'adresse concernée vaut >C700+>8 c'est-à-dire >C708.

Cet adressage simplifie la résolution des problèmes demandant le parcours d'une table (recherche, tri, etc.) : il suffit de faire varier B sans modifier l'adresse de la table.

```
                    :
                    :*******************************
                    :*                             *
                    :* RECHERCHE D'1 CARACTERE     *
                    :* DANS UNE LISTE              *
                    :* VERSION 1                   *
                    :*                             *
                    :*******************************
                    :
0000 001E       = RCAR    EQU   R30
0000 001F       = RLONG   EQU   R31
                    :
0000 C6FC       = CAR     EQU   >C6FC
0000 C6FD       = LONG    EQU   >C6FD
0000 C6FE       = POS     EQU   >C6FE
0000 C6FF       = RES     EQU   >C6FF
                    :
0000 C700       = TABLE   EQU   >C700
                    :
0000 8AC6FC             LDA   @CAR       : CAR RECHERCHE
0003 D01E               MOV   A,RCAR
0005 8AC6FD             LDA   @LONG      : LONGUEUR DE
0008 D01F               MOV   A,RLONG    : LA TABLE
                    :
000A C5                 CLR   B          :ON COMMENCE
000B AAC700     CONT    LDA   @TABLE(B)  : AU DEBUT
000E 4D001E             CMP   A,RCAR
0011 E20F               JEQ   TROUVE     : C'EST BON
0013 C3                 INC   B
0014 4D011F             CMP   B,RLONG    : MAX = LONG
0017 E6F2               JNE   CONT
0019 62         INEX    MOV   B,A        :SAUVE DERNIERE
001A 8BC6FE             STA   @POS       : POSITION
001D B5                 CLR   A          : INDIQUE
001E 8BC6FF             STA   @RES       : INCONNU
0021 0A                 RETS
                    :
0022 62         TROUVE  MOV   B,A        :SAUVE POSITION
0023 8BC6FE             STA   @POS       : DU CARACTERE
0026 2201               MOV   %1,A       : INDIQUE
```



```
0028 8BC6FF             STA   @RES       : TROUVE
002B 0A                 RETS
                    :
002C                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

Le programme suivant réalise la recherche d'un caractère dont le code est dans CAR dans une table contenant N caractères (N doit être inférieur ou égal à 255 car B est un registre 8 bits). Le nombre de caractères est stocké avant exécution dans LONG. Le programme renvoit dans RES et dans POS le résultat de la recherche :

RES = 0 le caractère n'est pas dans la table.
POS contient la longueur de la table.

RES = 1 le caractère est dans la table.
POS contient sa position dans la table.

Le caractère cherché et la longueur de la table sont copiés dans deux registres, afin de simplifier l'écriture. La longueur de la table sert de critère d'arrêt : si l'on atteint la fin de la table (B = RLONG), c'est que le caractère n'y est pas.

```
                    :
                    :*******************************
                    :*                             *
                    :* RECHERCHE D'1 CARACTERE     *
                    :* DANS UNE LISTE              *
                    :* VERSION 2                   *
                    :*                             *
                    :*******************************
                    :
0000 001E       = RCAR    EQU   R30
0000 001F       = RLONG   EQU   R31
                    :
0000 C6FC       = CAR     EQU   >C6FC
0000 C6FD       = LONG    EQU   >C6FD
0000 C6FE       = POS     EQU   >C6FE
0000 C6FF       = RES     EQU   >C6FF
                    :
0000 C700       = TABLE   EQU   >C700
                    :
0000 8AC6FC             LDA   @CAR       : CAR CHERCHE
0003 D01E               MOV   A,RCAR
0005 8AC6FD             LDA   @LONG      : NBRE DE CAR
0008 D01F               MOV   A,RLONG
```

```
000A 321F              MOV   RLONG.B    : ON COMMENCE
000C AAC6FF    CONT    LDA   @TABLE-1(B) : PAR LA FIN
000F 4D001E            CMP   A.RCAR
0012 E20B              JEQ   TROUVE
0014 CAF6              DJNZ  B.CONT     : JUSQU'A B=0
0016 62        INEX    MOV   B.A        : SAUVE DER
0017 8BC6FE            STA   @POS       : POSITION
001A B5                CLR   A          : INDIQUE
001B 8BC6FF            STA   @RES       : INCONNU
001E 0A                RETS
               :
001F 62        TROUVE  MOV   B.A        :SAUVE POSITION
0020 8BC6FE            STA   @POS       : DU CARACTERE
0023 2201              MOV   %1.A       : INDIQUE
0025 8BC6FF            STA   @RES       : TROUVE
0028 0A                RETS
               :
0029           *       END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

On peut résoudre le même problème en partant du bas de la table, c'est-à-dire de son dernier élément. Dans ce cas on utilise l'instruction DJNZ qui fonctionne de la façon suivante :

Le processeur soustrait 1 au registre concerné.
Si le registre ne contient pas 0, il y a saut à l'adresse indiquée, sinon le registre contenant 0, il y a exécution de l'instruction suivante.

★ Attention au décalage : le premier caractère de la table étant situé à l'adresse TABLE+0 et non TABLE+1, on utilise l'adresse TABLE−1 comme base de la table, car B=1 lors du dernier test.

```
               :
               :**********************************
               :*                                *
               :* RECHERCHE D'1 CARACTERE        *
               :* DANS UNE LISTE                 *
               :* VERSION 3                      *
               :*                                *
               :**********************************
               :
0000 001E    = RCAR    EQU   R30
               :
0000 C6FD    = CAR     EQU   >C6FD
0000 C6FE    = POS     EQU   >C6FE
0000 C6FF    = RES     EQU   >C6FF
               :
```



```
0000 C700    = TABLE   EQU   >C700
               :
0000 8AC6FD            LDA   @CAR       : CAR CHERCHE
0003 D01E              MOV   A.RCAR
               :
0005 C5                CLR   B          : ON COMMENCE
0006 AAC700    CONT    LDA   @TABLE(B)  : AU DEBUT
0009 2000              CMP   %0.A
000B E208              JEQ   INEX
000D 4D001E            CMP   A.RCAR
0010 E20C              JEQ   TROUVE
0012 C3                INC   B          : CAR SUIVANT
0013 E0F1              JMP   CONT
0015 62        INEX    MOV   B.A        : SAUVE DER
0016 8BC6FE            STA   @POS       : POSITION
0019 B5                CLR   A          : INDIQUE
001A 8BC6FF            STA   @RES       : INCONNU
001D 0A                RETS
               :
001E 62        TROUVE  MOV   B.A        :SAUVE POSITION
001F 8BC6FE            STA   @POS       : DU CARACTERE
0022 2201              MOV   %1.A       : INDIQUE
0024 8BC6FF            STA   @RES       : TROUVE
0027 0A                RETS
               :
0028           *       END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

La version 3 propose une approche relativement plus simple : la fin de la table est indiquée par un caractère spécial (un marqueur, un délimiteur) qui est > 00 dans cet exemple. Il n'est donc plus nécessaire de connaître la taille de la table.

Il suffit de ne pas oublier ce caractère !

```
               :
               :**********************************
               :*                                *
               :* RECHERCHE D'1 CARACTERE        *
               :* DANS UNE LISTE                 *
               :* VERSION 4                      *
               :*                                *
               :**********************************
               :
0000 001E    = RCAR    EQU   R30
0000 001F    = PH      EQU   R31
0000 0020    = PL      EQU   R32
0000 0021    = CPTH    EQU   R33
0000 0022    = CPTL    EQU   R34
               :
```

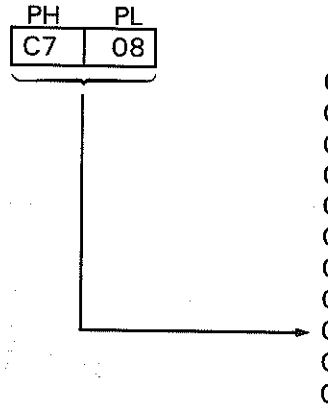
```
0000 C5FC      = CAR      EQU   >C5FC
0000 C5FD      = POSH     EQU   >C5FD
0000 C5FE      = POSL     EQU   >C5FE
0000 C5FF      = RES      EQU   >C5FF
                 :
0000 C600      = TABLE    EQU   >C600
                 :
0000 8AC5FC              LDA   @CAR        : CAR CHERCHE
0003 D01E                MOV   A,RCAR
                 :
0005 D522                CLR   CPTL        : 0 DANS
0007 D521                CLR   CPTH        : COMPTEUR
0009 88C60020            MOVD  %TABLE,PL   : INIT POINTEUR
000D 9A20      CONT      LDA   *PL         : REC CAR
000F 2D00                CMP   %0,A
0011 E211                JEQ   INEX        : TERMINE
0013 4D001E              CMP   A,RCAR
0016 E21B                JEQ   TROUVE      : C'EST BON
0018 D320                INC   PL          : CAR SUIVANT
001A 79001F              ADC   %0,PH
001D D322                INC   CPTL        : POS SUIVANTE
001F 790021              ADC   %0,CPTH
0022 E0E9                JMP   CONT
0024 982201    INEX      MOVD  CPTL,B      : SAUVE DER
0027 8BC5FD              STA   @POSH       : POSITION
002A B6                  XCHB  A
002B 8BC5FE              STA   @POSL
002E B5                  CLR   A           : INDIQUE
002F 8BC5FF              STA   @RES        : INCONNU
0032 0A                  RETS
                 :
0033 982201    TROUVE    MOVD  CPTL,B      :SAUVE POSITION
0036 8BC5FD              STA   @POSH       : DU CARACTERE
0039 B6                  XCHB  A
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

Cette nouvelle version du même programme permet une recherche dans une table de longueur supérieure à 255 caractères.
On utilise pour cela l'adressage indirect : l'adresse de l'élément à atteindre est contenu dans une paire de registres.
Pour parcourir la table, il suffit alors d'incrémenter la paire de registres, servant de pointeur.

Dans notre programme, nous utilisons une paire de registres (CPTH et CPTL) pour stocker la position en cours de recherche. Cette position peut être un nombre de 16 bits, ce qui nous impose de le stocker en mémoire en 2 temps le poids fort puis le faible. L'accès au caractère à tester est réalisé grâce à l'instruction LDA. Le caractère > 00 indique la fin de la table.

L'adressage indirect



Si (PH,PL contient > C708, alors LDA*PL charge le caractère '0' dans le registre A.

La version 5 réalise la même recherche que la version 4, mais, cette fois, on connaît uniquement l'adresse de début (DTBLE) et l'adresse de fin de la table (FTBLE). On a recours à une petite astuce pour simplifier l'écriture : on incrémente le pointeur d'adresse et le compteur de position AVANT de tester le caractère, ce qui nous oblige à leur soustraire 1 avant le premier passage.

```
                 :
                 :*****************************
                 :*                           *
                 :* RECHERCHE D'1 CARACTERE   *
                 :* DANS UNE LISTE            *
                 :* VERSION 5                 *
                 :*                           *
                 :*****************************
                 :
0000 001E      = RCAR     EQU   R30
0000 001F      = PH       EQU   R31
0000 0020      = PL       EQU   R32
0000 0021      = CPTH     EQU   R33
0000 0022      = CPTL     EQU   R34
0000 0023      = FTBH     EQU   R35
0000 0024      = FTBL     EQU   R36
                 :
0000 C5FC      = CAR      EQU   >C5FC
0000 C5FD      = POSH     EQU   >C5FD
0000 C5FE      = POSL     EQU   >C5FE
0000 C5FF      = RES      EQU   >C5FF
                 :
```

```
0000 C600      = DTBLE   EQU   >C600
0000 C77F      = FTBLE   EQU   >C77F
                 :
0000 8AC5FC             LDA   @CAR        : CAR CHERCHE
0003 D01E               MOV   A,RCAR
                 :
0005 88000022           MOVD  %>0000,CPTL :COMPTEUR = 0
                 :
0009 88C77F24           MOVD  %FTBLE,FTBL : FIN TABLE
                 :
000D 88C60020           MOVD  %DTBLE,PL   : INIT POINTEUR
0011 DB20               DECD  PL          :POUR COMMENCER
0013 DB22               DECD  CPTL        :COMPTEUR = -1
                 :
0015 D322       CONT    INC   CPTL
0017 790021             ADC   %00,CPTH    : INC COMPTEUR
001A D320               INC   PL
001C 79001F             ADC   %0,PH       : INC POINTEUR
                 :
001F 4D1F23             CMP   PH,FTBH     : TERMINE?
0022 E605               JNE   CONT1       : NON
0024 4D2024             CMP   PL,FTBL
0027 E217               JEQ   INEX        : TERMINE
                 :
0029 9A20       CONT1   LDA   *PL         : REC CAR
002B 4D001E             CMP   A,RCAR
002E E6E5               JNE   CONT        : CAR SUIVANT
                 :
0030 982201     TROUVE  MOVD  CPTL,B      :SAUVE POSITION
0033 8BC5FD             STA   @POSH       : DU CARACTERE
0036 B6                 XCHB  A
0037 8BC5FE             STA   @POSL
003A 2201               MOV   %1,A        : INDIQUE
003C 8BC5FF             STA   @RES        : TROUVE
003F 0A                 RETS
                 :
0040 982201     INEX    MOVD  CPTL,B      : SAUVE DER
0043 8BC5FD             STA   @POSH       : POSITION
0046 B6                 XCHB  A
0047 8BC5FE             STA   @POSL
004A B5                 CLR   A           : INDIQUE
004B 8BC5FF             STA   @RES        : INCONNU
004E 0A                 RETS
                 :
004F                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

COMPLÉMENTS

1) Le registre d'état ST

Le registre ST est un registre 8 bits, dont 4 nous concernent plus particulièrement : C,N,Z et I (les autres sont inutilisés).

C	N	Z	I	0	0	0	0

C'est dans le registre ST que le processeur indique son 'état interne', en positionnant ou non un ou plusieurs bits du quartet de poids fort.

Pratiquement toutes les opérations agissant sur des octets modifient ST. Après chaque opération (au sens large : ADD, SUB mais aussi CMP, MOV, LDA,...) chaque bit est mis à 1 ou à 0 selon le résultat. C'est au programmeur, par la suite, de tenir compte ou non de l'état de ces bits, au moyen des instructions de saut conditionnel ! (JEQ, JC, etc.).

A chaque bit est associé un état :

C: (CARRY). C'est le bit de retenue. Une retenue apparaît quand le résultat d'une opération ne peut être stocké sur un octet (100 + 200, par exemple). Certaines instructions positionnent C à 0 (CLRC, LDA,...) ou à 1 (SETC, RL dans certains cas,...).

N: (NEGATIVE). C'est la copie du bit de poids fort du résultat, qui correspond, en complément à 2, au signe du résultat (0 positif, 1 négatif).

Z: (ZERO). C'est le bit de 'zéro'. Il est mis à 1 si le résultat est nul.

Les instructions de saut conditionnel 'réagissent' à l'état de ces bits. JC, par exemple, réalise un saut si C est à 1 et uniquement dans ce cas, tandis que JP, par exemple, réalise un saut si N et Z sont à 0 en même temps. Le tableau suivant décrit ces instructions :

INSTRUCTION	MNÉMONIQUE	CODE	ÉTAT DES BITS POUR SAUT		
			C	N	Z
saut si retenue	JC	E3	1	×	×
saut si égal à zéro	JEQ	E2	×	×	1
saut si supérieur ou égal	JHS	E3	1	×	×
saut si inférieur	JL	E7	0	×	×
saut si négatif	JN	E3	×	1	×
saut si pas de retenue	JNC	E7	0	×	×
saut si non égal	JNE	E6	×	×	0
saut si différent de zéro	JNZ	E6	×	×	0
saut si positif	JP	E4	×	0	0
saut si positif ou nul	JPZ	E5	×	0	×
saut si nul	JZ	E2	×	×	1
saut si plus petit	JLT	E1	×	1	×
saut si plus grand	JGT	E4	×	0	0
saut si plus grand ou égal	JGE	E5	×	0	×

Exemple : LDA@> 8000
Si l'adresse > 8000 vaut 0, Z est mis à 1, N est mis à 0, C est mis à 0 de toute façon. Le saut aura lieu avec JEQ, JP, par exemple, mais n'aura pas lieu avec JC ou JN.

Attention :

★ De nombreuses instructions agissent sur ST. Vérifiez systématiquement les bits concernés dans les fiches descriptives des instructions, cela vous évitera de facheuses erreurs !

★ Suivant l'instruction utilisée, l'interprétation de l'état de ST peut changer. Exemples :

a. MOV %00,A positionne Z à 1, car la valeur est nulle.

b. MOV %05,A
CMP %05,A positionne Z à 1, car les nombres sont égaux (CMP effectue une soustraction de façon interne).

Dans les deux cas, Z est à 1, le résultat valant 0, mais cela ne signifie pas, dans le deuxième exemple, que les nombres sont nuls ! On peut, pour plus de clarté, utiliser JZ dans le premier cas et JEQ dans le second, si l'on veut tester le bit Z.
Il faut de toute façon appliquer le point 1 !

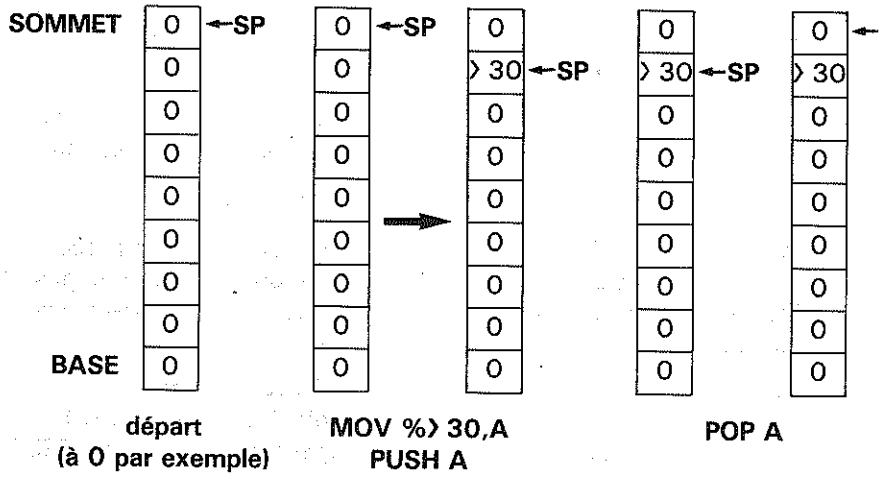
★ Le registre ST est très volatile : son contenu est modifié par chaque instruction (ou presque) du programme. Voici une erreur fréquente :


```
        MOV  %> F0,A
        ADD  %> 30,A
        MOV  A,B
        JC  ERREUR
OK      . . . . . .
        . . . . . .
        . . . . . .
```

On réalise une addition 8 bits, en prenant bien soin de sauver le résultat dans B, puis on teste la retenue pour savoir s'il n'y a pas eu débordement (ce qui est le cas). Mais c'est trop tard ! En effet, C est mise à 1 après ADD, mais MOV la remet à 0.
Alors, prudence ! et relire le premier point !

2. Gestion de la pile

Une pile est une zone mémoire accessible par des instructions spécifiques et d'une manière assez particulière. Les octets sont placés comme sur une pile d'assiettes : les uns sur les autres.
On empile lorsqu'on écrit dans la pile, l'octet se trouvant sur le dessus de la pile.
On dépile lorsqu'on lit dans la pile, l'octet lu étant toujours l'octet situé sur le dessus de la pile.
Le dessus de la pile est le sommet, le bas de la pile, la base.
Un pointeur, le pointeur de pile, est incrémenté ou décrémenté suivant l'opération, afin de toujours pointer vers le sommet de la pile.



LA PILE : Empiler et Dépiler

La pile du TMS 7040 est située en RAM CPU, dans la zone des registres. Sa base toujours en > 7F, son sommet, indiqué par SP, est variable.

LDSP permet d'initialiser le pointeur de pile SP : la valeur contenue dans B est stockée en SP.
STSP permet de récupérer la valeur courante du pointeur de pile : la valeur contenue dans SP est copiée dans B.

PUSH permet d'empiler un octet.
POP permet de dépiler un octet.

Certaines instructions utilisent la pile pour sauvegarder (CALL, TRAP) ou récupérer des données (RETS, RETI).

Exemple :

```
MOV  %> 60,B      : initialise B à > 60
LDSP              : copie dans SP: le sommet de la pile est
                    en > 60
MOV  %> 33,A      : initialise A
PUSH A            : empile A (> 33)
PUSH B            : empile B (> 60)
STSP              : copie SP dans B, qui vaut ici > 62
```

Cette pile est aussi accessible par LDA ou MOV, ce qui permet de récupérer des données sans modifier l'état de SP: LDA @> 0061 et MOV R97,A chargent > 60 dans A, après le programme précédent.

Attention :

★ Il faut bien vérifier l'adresse du sommet de la pile avant exécution de programmes car 2 problèmes peuvent survenir :

– les registres utilisés par le programme sont dans la pile. On risque de modifier des adresses de retour de sous-programmes, etc.

– la pile n'est pas assez grande : résultat imprévisible !

★ CALL empile l'adresse de retour avant exécution du sous-programme appelé et RETS la dépile. Il faut donc veiller à ne pas modifier la pile durant le sous programme ou à dépiler autant que nécessaire :

```
.....          MOV  %> 60,B     : initialise B à > 60
.....          LDSP             : copie B dans SP (=> 60)
> 1000         CALL @TEST       : appel sous-programme TEST
> 1003         .........
.....          .........
```



```
.....          .........
.....  TEST    MOV  %15,A
.....          PUSH A           : on empile (A).
.....          RETS
```

L'effet est désastreux car la pile contient :

```
                > 60:  xxx
                > 61:  > 10      MSB ADRESSE RETOUR
                > 62:  > 03      LSB ADRESSE RETOUR
      SP --     > 63:  > 33      VALEUR DE A
```

et REST dépile donc l'adresse > 0333 au lieu de > 1003.

Adresse Ecran Sous EXELMAX $0000

APPLICATIONS : L'ÉCRAN, LE CLAVIER, LE SYNTHÉTISEUR

Les programmes suivants sont des exemples d'application des techniques décrites aux périphériques de base de l'EXL100 ou de l'EXELTEL.

Ils sont destinés à vous montrer les possibilités offertes par ces deux appareils. Il vous faudra bien entendu les adapter ou les réécrire à votre façon pour les intégrer dans vos propres programmes.

1. Initialisation du VDP

Avant de pouvoir utiliser l'écran, il est indispensable de fournir au processeur chargé de sa gestion (le VDP) un certain nombre d'informations : adresse de la page écran, adresses du ou des générateurs, etc.

Vous trouverez une description complète du VDP au chapitre ''GESTION DE L'ECRAN'', qu'il est d'ailleurs préférable de lire avant d'essayer ces programmes.

Le programme suivant réalise une initialisation partielle du VDP : seuls deux générateurs sont définis, l'écran étant en mode TEXTE. La routine permettant d'effacer l'écran est décrite un peu plus loin.

La page écran est placée en > 0A00, le générateur BAGC0 en > 0000 et le générateur BAGC3 en > 0500 (le tout en VRAM, bien entendu). Cette initialisation est suffisante pour tester les programmes proposés par la suite.

```
                        :
                        ***************
                        *             *
                        * INIT VDP    *
                        *             *
                        ***************
                        :
                        :
                        :
0000 000D       = T1H   EQU  R13
0000 000E       = T1L   EQU  R14
0000 000F       = T2H   EQU  R15
0000 0010       = T2L   EQU  R16
                        :
0000 001B       = T8H   EQU  R27
0000 001C       = T8L   EQU  R28
                        :
0000 001E       = ECRH  EQU  R30
0000 001F       = ECRL  EQU  R31
0000 0020       = LIN   EQU  R32
0000 0021       = COL   EQU  R33
0000 0022       = ATTB  EQU  R34
0000 0023       = CAR   EQU  R35
                        :
0000 0000       = BAGC2 EQU  >0000
0000 0500       = BAGC3 EQU  >0500
0000 0A00       = BAPA  EQU  >0A00
                        :
                        :------------------------
                        :  CHARGEMENT DU
                        :  GENERATEUR INTEGRE
                        :  EN BAGC2 ET BAGC3
                        :------------------------
                        :
0000 05                 EINT
0001 8800001C           MOVD  %BAGC2.T8L    : BAGC2
0005 F2                 TRAP  13
0006 8805001C           MOVD  %BAGC3.T8L    : BAGC3
000A F2                 TRAP  13
                        :
                        :------------------------
                        :  EFFACE VRAM ECRAN
                        :------------------------
                        :
000B 880A001F           MOVD  %BAPA.ECRL    : AD ECRAN
000F 72E822             MOV   %>E8.ATTB     : W/B, CG=2
0012 720023             MOV   %00.CAR       : BLANC
                        :
0015 981F0E             MOVD  ECRL.T1L
0018 F6                 TRAP  9
0019 721920             MOV   %25.LIN
001C 722821     EFF1    MOV   %40.COL
001F 1222       EFF2    MOV   ATTB.A
0021 822E               WVDP  A
0023 1223               MOV   CAR.A
```



```
0025 822E               WVDP  A
0027 DA21F5             DJNZ  COL.EFF2
002A DA20EF             DJNZ  LIN.EFF1
                        :
                        :
                        :----------------------
                        : INIT REG CM1.CM2
                        :          CM3.CM4
                        :----------------------
                        :
                        : CM1:  TOUJOURS A >10
                        :
                        :
002D A2042D             MOVP  %>04.P45      : CM1
0030 A2102D             MOVP  %>10.P45      : FIXE
                        :
                        :
                        : CM2:  AFFICHAGE LIGNE 0
                        :       BACG3 ALPHAMOSAIQUE
                        :       PAS DE GRILLE
                        :
                        :
0033 A2052D             MOVP  %>05.P45
0036 A2C82D             MOVP  %>C8.P45
                        :
                        :
                        : CM3:  MODE TEXTE
                        :
                        :
0039 A2062D             MOVP  %>06.P45      : CM3
003C A2482D             MOVP  %>48.P45
                        :
                        :
                        : CM4:  BORD CYAN
                        :
                        :
003F A2072D             MOVP  %>07.P45      : CM4
0042 A2C02D             MOVP  %>C0.P45      : BORD CYAN
                        :
                        :
                        :----------------------
                        : BAPA (PAGE ECRAN)
                        : LE VDP AJOUTE 1
                        :----------------------
                        :
                        :
0045 880A0001           MOVD  %BAPA.B
0049 CB                 DECD  B
004A A2012D             MOVP  %>01.P45      : COL
004D 922D               MOVP  B.P45         : LSB
004F A2022D             MOVP  %>02.P45      : ROW
0052 822D               MOVP  A.P45         : MSB
0054 A20A2D             MOVP  %>0A.P45      : REG BAPA
0057 A2002D             MOVP  %>00.P45      : TRANS 1
005A A2002D             MOVP  %>00.P45      : TRANS 2
```

```
                        ;
                        ;
                        ;----------------------
                        ; BAGC2 (GENERATEUR 2)
                        ; LE VDP AJOUTE 2
                        ;----------------------
                        ;
                        ;
005D 88000001           MOVD  %BAGC2.B
0061 CB                 DECD  B
0062 CB                 DECD  B
0063 A2012D             MOVP  %>01.P45      : COL
0066 922D               MOVP  B.P45         : LSB
0068 A2022D             MOVP  %>02.P45      : ROW
006B 822D               MOVP  A.P45         : MSB
006D A2002D             MOVP  %>0D.P45      : BAGC2
0070 A2002D             MOVP  %>00.P45      : TRANS 1
0073 A2002D             MOVP  %>00.P45      : TRANS 2
                        ;
                        ;
                        ;
                        ;----------------------
                        ; BAGC3 (GENERATEUR 3)
                        ; LE VDP AJOUTE 2
                        ;----------------------
                        ;
                        ;
0076 88050001           MOVD  %BAGC3.B
007A CB                 DECD  B
007B CB                 DECD  B
007C A2012D             MOVP  %>01.P45      : COL
007F 922D               MOVP  B.P45         : LSB
0081 A2022D             MOVP  %>02.P45      : ROW
0084 822D               MOVP  A.P45         : MSB
0086 A20E2D             MOVP  %>0E.P45      : BAGC3
0089 A2002D             MOVP  %>00.P45      : TRANS 1
008C A2002D             MOVP  %>00.P45      : TRANS 2
                        ;
008F 0A                 RETS
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

2. Positionnement sur l'écran

Ce programme permet de placer un curseur au point de coordonnées (L,C) dans l'écran, avec L désignant la ligne et C la colonne. On obtient ainsi une instruction du type ''LOCATE (L,C)''.

A doit contenir la ligne L (de 0 et 24) et B la colonne (de 0 à 39). L'écran est supposé être en > 0A00.

On utilise TRAP 9 qui permet de placer le pointeur de lecture et d'écriture en VDP à la même adresse, celle contenue dans T1L (TEMP1). Le calcul est simple : le caractère visible en (L,C) sur l'écran occupe deux octets en VRAM : 1 pour l'attribut et 1 pour le code. La colonne doit donc être multipliée par 2 et, une ligne occupant 2 × 40 octets, le numéro de ligne est multiplié par 80.
La multiplication par 2 est réalisée par un décalage vers la gauche RL.

Au retour du programme, on ne voit évidemment rien : seuls les pointeurs de lecture et d'écriture du VDP sont positionnés.

```
                        ;
                        ;**********************
                        ;*                    *
                        ;*   CALCUL DE        *
                        ;*  COORDONNEES       *
                        ;*                    *
                        ;**********************
                        ;
                        ;
                        ; (ECRH.ECRL) = BAPA
                        ; A CONTIENT LE NUMERO DE LIGNE
                        ; B CONTIENT LE NUMERO DE COLONNE
                        ; MODE TEXT (80 OCTETS/LIGNE)

                        ;
                        ;
0000 000D       = T1H   EQU R13
0000 000E       = T1L   EQU R14
                        ;
0000 001E       = ECRH  EQU R30
0000 001F       = ECRL  EQU R31
                        ;
0000 880A001F               MOVD  %>0A00.ECRL
                        ;
0004 981F0E                 MOVD  ECRL.T1L    : INIT POINTEUR
0007 C8                     PUSH  B           : SAUVE COLONNE
0008 2C50                   MPY   %80.A       :1 LINE = 80 OC
000A 48010E                 ADD   B.T1L
000D 49000D                 ADC   A.T1H
0010 C9                     POP   B           : RECUP COLONNE
0011 CE                     RL    B           : x 2
0012 48010E                 ADD   B.T1L
0015 79000D                 ADC   %00.T1H
0018 F6                     TRAP  9           : POS VDP
0019 0A                     RETS              : TERMINE
                        ;
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

3. Affichage d'un caractère

On utilise le programme précédent comme sous-programme de calcul de l'adresse en VRAM associée à la position (L,C). Les numéros de ligne et de colonne sont cette fois stockés dans les registres LIN et COL (on ne veut pas les perdre !).

Deux registres sont utilisés pour stocker d'une part l'attribut du caractère (ATTB), d'autre part le code du caractère (CAR).

L'opération d'affichage est simple : on stocke, grâce à l'instruction WVDP, l'attribut et le code du caractère à l'adresse indiquée par le pointeur d'écriture VDP, après calcul par le sous-programme POSVDP.

L'attribut est obtenu en sachant que :

a. le générateur BAGC2 est toujours un générateur alphamosaique. L'attribut est donc défini ainsi :

BF	GF	RF	0	1	BB	GB	RB

(BF,GF,RF) définissent la couleur des points du caractère et (BB,GB,RB) la couleur du fond du caractère.

b. on obtient les couleurs grâce à la table :

	B	G	R
NOIR	0	0	0
ROUGE	0	0	1
VERT	0	1	0
JAUNE	0	1	1
BLEU	1	0	0
MAGENTA	1	0	1
CYAN	1	1	0
BLANC	1	1	1

Exemple :

Caractère de BAGC2 en jaune sur fond noir :
attribut = ?01101000, soit >68.


```
                    :
                    :**********************
                    :*                    *
                    :*  AFFICHAGE D'UN   *
                    :*  CARACTERE EN      *
                    :*  (L,C)             *
                    :*                    *
                    :**********************
                    :
                    :
                    :
                    :
                    :
0000 000D       = T1H    EQU R13
0000 000E       = T1L    EQU R14
                    :
0000 001E       = ECRH   EQU R30
0000 001F       = ECRL   EQU R31
0000 0020       = LIN    EQU R32
0000 0021       = COL    EQU R33
0000 0022       = ATTB   EQU R34
0000 0023       = CAR    EQU R35
                    :
0000 880A001F           MOVD  %>0A00,ECRL
                    :
0004 720520     PRNC    MOV   %05,LIN     : LIGNE 5
0007 721421             MOV   %20,COL     : COLONNE
                    :
000A 72E822             MOV   %>E8,ATTB   : B/W, CG=2
000D 724123             MOV   %'A',CAR    : CARAC 'A'
                    :
0010 8E001C             CALL  @POSVDP     : CALCUL POS
                    :
0013 1222               MOV   ATTB,A
0015 822E               WVDP  A           :  ATTRIB
0017 1223               MOV   CAR,A
0019 822E               WVDP  A           : PUIS CODE
                    :
001B 0A                 RETS              : TERMINE
                    :
001C 981F0E     POSVDP  MOVD  ECRL,T1L
001F 7C5020             MPY   %80,LIN
0022 48010E             ADD   B,T1L
0025 49000D             ADC   A,T1H
0028 7C0221             MPY   %02,COL
002B 48010E             ADD   B,T1L
002E 49000D             ADC   A,T1H
0031 F6                 TRAP  9
0032 0A                 RETS
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

4. Effacement d'une ligne

On utilise toujours le sous-programme POSVDP.
La méthode est simple : on place le pointeur d'écriture VDP sur la colonne 0 de la ligne voulue, puis on efface 40 caractères en écrivant l'attribut puis le caractère d'effacement > 00 (le caractère blanc > 20 est considéré comme un délimiteur).
On remarque l'utilisation de DJNZ et la simplicité d'écriture de la boucle liée au fait que les registres VDP sont auto-incrémentés après chaque accès à la VRAM.

```
                    :
                    :**********************
                    :*                    *
                    :*  EFFACEMENT       *
                    :*  DE LA LIGNE L     *
                    :*                    *
                    :**********************
                    :
                    :
                    :
                    :
                    :
0000 000D       =   T1H    EQU R13
0000 000E       =   T1L    EQU R14
                    :
0000 001E       =   ECRH   EQU R30
0000 001F       =   ECRL   EQU R31
0000 0020       =   LIN    EQU R32
0000 0021       =   COL    EQU R33
0000 0022       =   ATTB   EQU R34
0000 0023       =   CAR    EQU R35
                    :
0000 880A001F           MOVD  %>0A00.ECRL
                    :
0004 720520     PRNC    MOV   %05.LIN    : LIGNE 5
0007 720021             MOV   %00.COL    : COLONNE 0
                    :
000A 72E822             MOV   %>E8.ATTB  : W/B, CG=2
000D 720023             MOV   %00.CAR    : CARAC NUL
                    :
0010 8E0020             CALL  @POSVDP    : CALCUL POS
                    :
0013 5228               MOV   %40.B      : 40 CARAC
0015 1222       EFLN    MOV   ATTB.A
0017 822E               WVDP  A          :  ATTRIB
0019 1223               MOV   CAR.A
001B 822E               WVDP  A          : PUIS CODE
001D CAF6               DJNZ  B.EFLN
                    :
001F 0A                 RETS             : TERMINE
                    :
0020 981F0E     POSVDP  MOVD  ECRL.T1L
0023 7C5020             MPY   %80.LIN
0026 48010E             ADD   B.T1L
0029 49000D             ADC   A.T1H
002C 7C0221             MPY   %02.COL
002F 48010E             ADD   B.T1L
0032 49000D             ADC   A.T1H
0035 F6                 TRAP  9
0036 0A                 RETS
                    :
0037                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

5. Effacement de l'écran

Le programme utilise une double boucle : la première pour effacer une ligne, la seconde pour compter le nombre de lignes. Pas de difficulté dans ce programme.

```
                    :
                    :**********************
                    :*                    *
                    :*  EFFACEMENT       *
                    :*  DE L'ECRAN        *
                    :*                    *
                    :**********************
                    :
                    :
                    :
                    :
                    :
0000 000D       =   T1H    EQU R13
0000 000E       =   T1L    EQU R14
0000 000F       =   T2H    EQU R15
0000 0010       =   T2L    EQU R16
                    :
0000 001E       =   ECRH   EQU R30
0000 001F       =   ECRL   EQU R31
0000 0020       =   LIN    EQU R32
0000 0021       =   COL    EQU R33
0000 0022       =   ATTB   EQU R34
0000 0023       =   CAR    EQU R35
                    :
0000 880A001F           MOVD  %>0A00.ECRL
                    :
0004 720020     PRNC    MOV   %00.LIN    : LIGNE 0
0007 720021             MOV   %00.COL    : COLONNE 0
                    :
000A 72E822             MOV   %>E8.ATTB  : W/B, CG=2
000D 720023             MOV   %00.CAR    : CARAC NUL
                    :
```

```
0010 8E0025              CALL   @POSVDP    : CALCUL POS
                         :
0013 721910              MOV    %25.T2L    : 25 LIGNES
0016 5228        EFLN0   MOV    %40.B      : 40 CARAC
0018 1222        EFLN1   MOV    ATTB.A
001A 822E                WVDP   A          :  ATTRIB
001C 1223                MOV    CAR.A
001E 822E                WVDP   A          : PUIS CODE
0020 CAF6                DJNZ   B.EFLN1    : COL SUIVANTE
0022 DA10F1              DJNZ   T2L.EFLN0  : LIN SUIVANTE
                         :
0025 0A                  RETS               : TERMINE
                         :
0026 981F0E      POSVDP  MOVD   ECRL.T1L
0029 7C5020              MPY    %80.LIN
002C 48010E              ADD    B.T1L
002F 49000D              ADC    A.T1H
0032 7C0221              MPY    %02.COL
0035 48010E              ADD    B.T1L
0038 49000D              ADC    A.T1H
003B F6                  TRAP   9
003C 0A                  RETS
                         :
003D                     END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

6. Affichage d'une chaîne

Ce programme permet d'afficher une suite de caractères en ligne LIN et en colonnes COL. On accède au caractère de la chaîne par adressage indirect. Le code 0 signale la fin de la chaîne. On peut noter la transformation du caractère blanc > 20 en caractère nul > 00, indispensable de par la nature particulière du caractère > 20.

Ce programme est équivalent à un TRAP 20 (cf chapitre sur les TRAP).


```
                         :
                         :*********************
                         :*                   *
                         :*  AFFICHAGE        *
                         :*  D'UNE CHAINE     *
                         :*                   *
                         :*********************
                         :
                         :
                         :
                         :
                         :
0000 000D        = T1H   EQU R13
0000 000E        = T1L   EQU R14
0000 000F        = T2H   EQU R15
0000 0010        = T2L   EQU R16
                         :
0000 001E        = ECRH  EQU R30
0000 001F        = ECRL  EQU R31
0000 0020        = LIN   EQU R32
0000 0021        = COL   EQU R33
0000 0022        = ATTB  EQU R34
0000 0023        = CAR   EQU R35
0000 0024        = P1H   EQU R36
0000 0025        = P1L   EQU R37
                         :
0000 880A001F            MOVD   %>0A00.ECRL
                         :
0004 720320      PRNC    MOV    %03.LIN    : LIGNE 3
0007 720221              MOV    %02.COL    : COLONNE 2
                         :
000A 72E822              MOV    %>E8.ATTB  : W/B, CG=2
                         :
000D 8E002F              CALL   @POSVDP    : CALCUL POS
                         :
0010 88004625            MOVD   %ADTXT.P1L
0014 9A25        AF1     LDA    *P1L       : REC CARAC
0016 D023                MOV    A.CAR
0018 E214                JEQ    FIN        : SI 0 TERMINE
001A 1222                MOV    ATTB.A
001C 822E                WVDP   A          :  ATTRIB
001E 1223                MOV    CAR.A
0020 2D20                CMP    %' '.A     : BLANC =
0022 E601                JNE    AF2        : ASCII 0
0024 B5                  CLR    A
0025 822E        AF2     WVDP   A          : PUIS CODE
0027 D325                INC    P1L
0029 790024              ADC    %0.P1H
002C E0E6                JMP    AF1
                         :
002E 0A          FIN     RETS               : TERMINE
                         :
002F 981F0E      POSVDP  MOVD   ECRL.T1L
0032 7C5020              MPY    %80.LIN
0035 48010E              ADD    B.T1L
0038 49000D              ADC    A.T1H
```

```
003B 7C0221              MPY   %02.COL
003E 48010E              ADD   B.T1L
0041 49000D              ADC   A.T1H
0044 F6                  TRAP  9
0045 0A                  RETS
                    :
0046 4C49474E    ADTXT   BYTE  'LIGNE TEST',0
     45205445
     535400
                    :
0051             E       END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

7. Programme de SCROLL

Le terme 'SCROLL' désigne l'enroulement de l'écran par la montée d'une ligne de chacune de ses lignes, la ligne du haut disparaissant. La technique est simple : on recopie la ligne 1 dans la ligne 0 (qui disparaît donc), puis la ligne 2 dans la ligne 1, etc., jusqu'à la ligne 24 qui est recopiée en ligne 23. On efface alors la ligne 24.

Dans ce programme, on a profité du fait que l'écran est continu en mémoire : le premier octet de chaque ligne est distant de 80 octets du premier octet de la ligne suivante.
On décale donc d'un bloc les 1920 octets constituant les 24 lignes, en positionnant, grâce à TRAP 10, le pointeur d'écriture VDP et le pointeur de lecture VDP en deux adresses distantes de 80 octets.

```
                    :
                    :**********************
                    :*                    *
                    :*      SCROLL        *
                    :*                    *
                    :**********************
                    :
                    :
                    :
                    :
                    :
0000 000D       = T1H   EQU R13
0000 000E       = T1L   EQU R14
0000 000F       = T2H   EQU R15
0000 0010       = T2L   EQU R16
                    :
```



```
0000 001E       = ECRH  EQU R30
0000 001F       = ECRL  EQU R31
0000 0020       = LIN   EQU R32
0000 0021       = COL   EQU R33
0000 0022       = ATTB  EQU R34
0000 0023       = CAR   EQU R35
0000 0024       = P1H   EQU R36
0000 0025       = P1L   EQU R37
                    :
0000 880A001F            MOVD  %>0A00.ECRL
0004 72E822              MOV   %>E8.ATTB
0007 720023              MOV   %>00.CAR     : CAR BLANC
000A 981F0E      SCROL   MOVD  ECRL.T1L     : ADRESSE ECR
000D 981F10              MOVD  ECRL.T2L     : ADRESSE LEC
0010 785010              ADD   %80.T2L      : 1 LIGNE
0013 79000F              ADC   %00.T2H      : PLUS BAS
0016 F5                  TRAP  10
                    :
0017 88077F25            MOVD  %1919.P1L    : 1920 CARAC
001B D700        SCROL1  LVDP  A            : LIT
001D 822E                WVDP  A            : ECRIT
001F DB25                DECD  P1L
0021 E3F8                JC    SCROL1
                    :
0023 721820              MOV   %24.LIN      : LIGNE 24
0026 720021              MOV   %00.COL      : COLONNE 0
0029 8E0039              CALL  @POSVDP
                    :
002C 5228                MOV   %40.B        : EFFACE
002E 1222        SCROL2  MOV   ATTB.A       : DER LIGNE
0030 822E                WVDP  A            :  ATTRIB
0032 1223                MOV   CAR.A
0034 822E                WVDP  A            : PUIS CODE
0036 CAF6                DJNZ  B.SCROL2
                    :
0038 0A          FIN     RETS               : TERMINE
                    :
0039 981F0E      POSVDP  MOVD  ECRL.T1L
003C 7C5020              MPY   %80.LIN
003F 48010E              ADD   B.T1L
0042 49000D              ADC   A.T1H
0045 7C0221              MPY   %02.COL
0048 48010E              ADD   B.T1L
004B 49000D              ADC   A.T1H
004E F6                  TRAP  9
004F 0A                  RETS
                    :
0050             E       END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

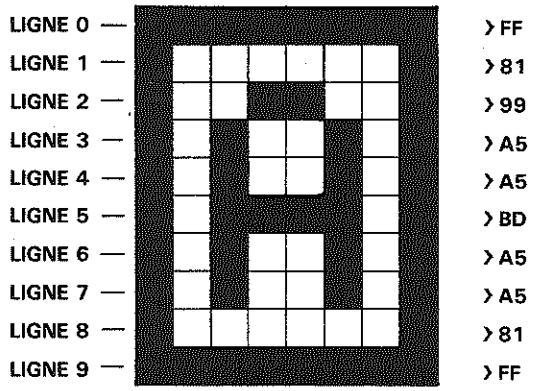
8. Modification de générateur

Ce programme permet de modifier le dessin associé à un caractère du générateur BAGC3.

Chaque caractère est constitué de 10 lignes de 8 points, soit 10 octets. Un générateur de caractère contient 128 caractères qui sont organisés en 10 pages de 128 octets. La page 0 réunit les lignes 9 des 128 caractères, la page 1 les lignes 8, et ainsi de suite jusqu'à la page 9 qui réunit les lignes 0 des 128 caractères (cf GESTION DE L'ECRAN7.

Le caractère à modifier est dans CAR, les 9 lignes constituant le dessin sont stockées à l'endroit. On note que la lecture des lignes en RAM est effectuée de la ligne 9 à la ligne 0, au moyen d'un DECD, tandis que l'écriture en VRAM est faite dans le sens habituel.

Le dessin correspondant à la table DESCAR est le suivant :




```
                    :
                    :**********************
                    :*                    *
                    :*  MODIFICATION      *
                    :*  D'UN CARACTERE    *
                    :*  DU GENERATEUR 2   *
                    :*                    *
                    :**********************
                    :
                    :
                    :
                    :
                    :
0000 000D       =   T1H     EQU R13
0000 000E       =   T1L     EQU R14
0000 000F       =   T2H     EQU R15
0000 0010       =   T2L     EQU R16
                    :
0000 001E       =   ECRH    EQU R30
0000 001F       =   ECRL    EQU R31
0000 0020       =   LIN     EQU R32
0000 0021       =   COL     EQU R33
0000 0022       =   ATTB    EQU R34
0000 0023       =   CAR     EQU R35
0000 0024       =   P1H     EQU R36
0000 0025       =   P1L     EQU R37
0000 0026       =   P2H     EQU R38
0000 0027       =   P2L     EQU R39
                    :
0000 0000       =   ADGEN2  EQU   >0000
                    :
0000 724123             MOV   %'A'.CAR
                    :
0003 88000027           MOVD  %ADGEN2.P2L : P2L = LIN 9
0007 482327             ADD   CAR.P2L     : ANCIEN CAR
                    :
000A 88002825           MOVD  %DESCAR.P1L : P1L = LIN 9
000E 780925             ADD   %09.P1L     : NOUVEAU CAR
0011 790024             ADC   %00.P1H
                    :
0014 720A20             MOV   %10.LIN     : 10 LIGNES
0017 98270E     MODIF   MOVD  P2L.T1L     : POSITIONNE
001A F7                 TRAP  8           : EN VRAM
001B 9A25               LDA   *P1L        : REC LIGNE
001D 822E               WVDP  A
001F DB25               DECD  P1L         : LIGNE SUIV
0021 788027             ADD   %128.P2L
0024 790026             ADC   %00.P2H
0027 DA20ED             DJNZ  LIN.MODIF
                    :
002A 0A                 RETS              : TERMINE
                    :
002B FF8199A5   DESCAR  BYTE  >FF.>81.>99.>A5.>A5
     A5
```

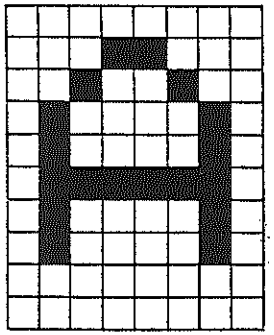
```
0030 BDA5A581            BYTE  >BD,>A5,>A5,>81,>FF
     FF
                  :
0035              *      END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

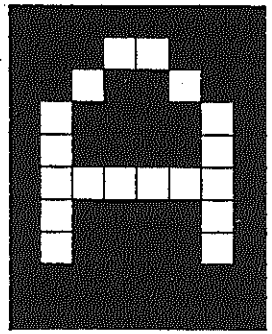
9. Inversion d'un caractère

Il s'agit cette fois d'inverser un caractère : un point allumé est éteint et un point éteint est allumé.
On utilise la possibilité donnée par TRAP 9 de positionner le pointeur de lecture et d'écriture à la même adresse. Les 10 octets constituant un caractère sont lus, inversés (INV) et réécrits les uns après les autres.

Caractère normal



Caractère inverse




```
                  :
                  :**********************
                  :*                    *
                  :*  INVERSION         *
                  :*  D'UN CARACTERE    *
                  :*  DU GENERATEUR 2   *
                  :*                    *
                  :**********************
                  :
                  :
                  :
                  :
                  :
0000 000D       = T1H    EQU R13
0000 000E       = T1L    EQU R14
0000 000F       = T2H    EQU R15
0000 0010       = T2L    EQU R16
                  :
0000 001E       = ECRH   EQU R30
0000 001F       = ECRL   EQU R31
0000 0020       = LIN    EQU R32
0000 0021       = COL    EQU R33
0000 0022       = ATTB   EQU R34
0000 0023       = CAR    EQU R35
0000 0024       = P1H    EQU R36
0000 0025       = P1L    EQU R37
0000 0026       = P2H    EQU R38
0000 0027       = P2L    EQU R39
                  :
0000 0000       = ADGEN2 EQU   >0000
                  :
0000 724123              MOV   %'A',CAR
                  :
0003 88000027            MOVD  %ADGEN2,P2L  : P2L = LIN 9
0007 482327              ADD   CAR,P2L      : ANCIEN CAR
                  :
                  :
000A 720A20              MOV   %10,LIN      : 10 LIGNES
000D 98270E      MODIF   MOVD  P2L,T1L      : POSITIONNE
0010 F6                  TRAP  9            : EN VRAM
0011 D700                LVDP  A            : REC LIGNE
0013 B4                  INV   A            : INVERSE BIT
0014 822E                WVDP  A
0016 788027              ADD   %128,P2L
0019 790026              ADC   %00,P2H
001C DA20EE              DJNZ  LIN,MODIF
                  :
001F 0A                  RETS               : TERMINE
                  :
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

10. Saisie et affichage d'une touche

Ce programme illustre le mode de fonctionnement du clavier : tout se passe, pour le programmeur, comme si le clavier était associé au registre R3, car le code de la dernière touche enfoncée y est systématiquement stocké.

Dans le programme, la touche enfoncée est affichée, sauf s'il s'agit de la touche ESC (code 27, qui permet de quitter le programme) ou d'une touche de code inférieur au blanc (code > 20).

Les codes renvoyés par le clavier sont donnés dans la table suivante :

```
                    ;
                    ;************************
                    ;*                      *
                    ;* SAISIE ET AFFICHAGE  *
                    ;*  D'UNE TOUCHE        *
                    ;*                      *
                    ;************************
                    ;
                    ;
                    ;
                    ;
0000 0003       =   VALUO  EQU  R03
                    ;
0000 000D       =   T1H    EQU  R13
0000 000E       =   T1L    EQU  R14
                    ;
0000 001E       =   ECRH   EQU  R30
0000 001F       =   ECRL   EQU  R31
0000 0020       =   LIN    EQU  R32
0000 0021       =   COL    EQU  R33
0000 0022       =   ATTB   EQU  R34
0000 0023       =   CAR    EQU  R35
                    ;
0000 880A001F           MOVD  %>0A00.ECRL
                    ;
0004 1203       SCRUTE  MOV   VALUO.A   : REC TOUCHE
0006 2D1B               CMP   %27.A     : ESC FIN PROG
0008 E20E               JEQ   FIN
000A 2D20               CMP   %' '.A
000C E7F6               JL    SCRUTE
000E D023               MOV   A.CAR
0010 72E822             MOV   %>E8.ATTB
0013 8E0019             CALL  @PRNC     : AFF CAR
0016 E0EC               JMP   SCRUTE
0018 0A         FIN     RETS
                    ;
0019 720520     PRNC    MOV   %05.LIN   : LIGNE 5
001C 721421             MOV   %20.COL   : COLONNE
001F 8E002B             CALL  @POSVDP   : CALCUL POS
0022 1222               MOV   ATTB.A
0024 822E               WVDP  A         :  ATTRIB
```



```
0026 1223               MOV   CAR.A
0028 822E               WVDP  A         : PUIS CODE
002A 0A                 RETS            : TERMINE
                    ;
002B 981F0E     POSVDP  MOVD  ECRL.T1L
002E 7C5020             MPY   %80_LIN
0031 48010E             ADD   B.T1L
0034 49000D             ADC   A.T1H
0037 7C0221             MPY   %02.COL
003A 48010E             ADD   B.T1L
003D 49000D             ADC   A.T1H
0040 F6                 TRAP  9
0041 0A                 RETS
                    ;
0042                    END
0000 Error(s)
```

Attention : les listings des programmes ayant été réduits pour être insérer dans le livre, les virgules peuvent apparaître comme des points. En règle générale, une virgule sépare les opérandes d'une instruction et un point-virgule indique un commentaire.

11. Utilisation du speech

L'utilisation du synthétiseur de parole est simple mais limitée : simple parce qu'il suffit de donner les codes associés à un son à la routine TRAP 4 (comme dans ce programme, où l'on fait prononcer le mot 'bonjour'), limitée car ces mêmes codes ne peuvent être définis que par une analyse du son, ce qui signifie que l'on est limité aux mots synthétisés par EXELVISION (très peu nombreux).

```
                    ;
                    ;********************
                    ;*                  *
                    ;* UTILISATION DU   *
                    ;*  SYNTHETISEUR    *
                    ;*     EXL100       *
                    ;*                  *
                    ;********************
                    ;
                    ;
0000 0003       =   VALUO  EQU  R03
0000 0009       =   PSONH  EQU  R09
0000 000A       =   PSONL  EQU  R10
                    ;
0000 8800100A           MOVD %BONJ.PSONL
0004 FB                 TRAP 4
                    ;
0005 1203       WAIT    MOV  VALUO.A
0007 2D04               CMP  %4.A
0009 E2FA               JEQ  WAIT
000B 2D1B               CMP  %27.A
```

```
000D E6F6               JNE  WAIT
000F 0A                 RETS
                :
0010 64E63ACD   BONJ    DATA >64E6,>3ACD,>A21A,>9141
     A21A9141
0018 EB50B558           DATA >EB50,>B558,>4506,>6342
     45066342
0020 447DA5E9           DATA >447D,>A5E9,>A336,>57D6
     A33657D6
0028 C6A98E36           DATA >C6A9,>8E36,>CA84,>6797
     CA846797
0030 32B81A56           DATA >32B8,>1A56
                :
0034 6E94B228           DATA >6E94,>B228,>7DC5,>D461
     7DC5D461
003C 4E6A52D2           DATA >4E6A,>52D2,>A8A3,>C924
     A8A3C924
0044 76CB72F0           DATA >76CB,>72F0,>2075,>BBBD
     2075BBBD
004C 6855351E           DATA >6855,>351E,>226C,>B7D7
     226CB7D7
0054 BD294D76           DATA >BD29,>4D76
                :
0058 611492B6           DATA >6114,>92B6,>B549,>A771
     B549A771
0060 4893D245           DATA >4893,>D245,>DFCC,>2E4B
     DFCC2E4B
0068 521F749B           DATA >521F,>749B,>3AAF,>8E83
     3AAF8E83
0070 19AD2B33           DATA >19AD,>2B33,>A547,>76D2
     A5477602
0078 9399BD2F           DATA >9399,>BD2F
                :
007C 15E35B6E           DATA >15E3,>5B6E,>8EB6,>4C8C
     8EB64C8C
0084 1FBA25F8           DATA >1FBA,>25F8,>764E,>5DE8
     764E5DE8
008C 9664A2D5           DATA >9664,>A2D5,>79A1,>5A8A
     79A15A8A
0094 94546145           DATA >9454,>6145,>662E,>562B
     662E562B
009C 4C149939           DATA >4C14,>9939
                :
00A0 E62CB6AC           DATA >E62C,>B6AC,>18B6,>9CD2
     18B69CD2
00A8 B9BCB43D           DATA >B9BC,>B43D,>D596,>AC29
     D596AC29
00B0 A759625B           DATA >A759,>625B,>D1AE,>E22E
     D1AEE22E
00B8 F1FF0100           DATA >F1FF,>0100
                :
00BC                    END
0000 Error(s)
```


ANNEXE

Caractères disponibles

a accent grave	2	>02			
a accent circonflexe	15	>0F			
e accent grave	16	>10			
e accent aigu	17	>11			
e accent circonflexe	18	>12			
c cédille	20	>14			
e tréma	22	>16			
o accent circonflexe	27	>1B			
u accent grave	28	>1C			
espace (blanc)	32	>20			
point d'exclamation	33	>21	!		
guillemets	34	>22	"		
dièse	35	>23	#		
symbole dollar	36	>24	$		
symbole pour cent	37	>25	%		
symbole du et	38	>26	&		
apostrophe	39	>27	'		
parenthèse ouvrante	40	>28	(		
parenthèse fermante	41	>29	)		
étoile	42	>2A	*		
signe plus	43	>2B	+		
virgule	44	>2C	,		
signe moins	45	>2D	–		
point	46	>2E	.		
signe de division	47	>2F	/		
chiffre 0	48	>30	0		
chiffre 1	49	>31	1		
chiffre 2	50	>32	2		
chiffre 3	51	>33	3		
chiffre 4	52	>34	4		
chiffre 5	53	>35	5		
chiffre 6	54	>36	6		
chiffre 7	55	>37	7		
chiffre 8	56	>38	8		
chiffre 9	57	>39	9		
deux points	58	>3A	:		
point virgule	59	>3B	;		
plus petit	60	>3C	<		
égal	61	>3D	=		
plus grand	62	>3E	>		
point d'interrogation	63	>3F	?		
a commercial	64	>40	@		
lettre A	65	>41	A		
lettre B	66	>42	B		
lettre C	67	>43	C		
lettre D	68	>44	D		
lettre E	69	>45	E		
lettre F	70	>46	F		
lettre G	71	>47	G		
lettre H	72	>48	H		
lettre I	73	>49	I		
lettre J	74	>4A	J		
lettre K	75	>4B	K		
lettre L	76	>4C	L		
lettre M	77	>4D	M		
lettre N	78	>4E	N		
lettre O	79	>4F	O		
lettre P	80	>50	P		
lettre Q	81	>51	Q		
lettre R	82	>52	R		
lettre S	83	>53	S		
lettre T	84	>54	T		
lettre U	85	>55	U		
lettre V	86	>56	V		
lettre W	87	>57	W		
lettre X	88	>58	X		
lettre Y	89	>59	Y		
lettre Z	90	>5A	Z		
crochet ouvrant	91	>5B	[		
back slash	92	>5C	\		

crochet fermant.....93 > 5D]
accent circonflexe....94 > 5E ^
blanc souligné.....95 > 5F _
quote.....96 > 60 `
lettre a.....97 > 61 a
lettre b.....98 > 62 b
lettre c.....99 > 63 c
lettre d.....100 > 64 d
lettre e.....101 > 65 e
lettre f.....102 > 66 f
lettre g.....103 > 67 g
lettre h.....104 > 68 h
lettre i.....105 > 69 i
lettre j.....106 > 6A j
lettre k.....107 > 6B k
lettre l.....108 > 6C l
lettre m.....109 > 6D m
lettre n.....110 > 6E n
lettre o.....111 > 6F o
lettre p.....112 > 70 p
lettre q.....113 > 71 q
lettre r.....114 > 72 r
lettre s.....115 > 73 s
lettre t.....116 > 74 t
lettre u.....117 > 75 u
lettre v.....118 > 76 v
lettre w.....119 > 77 w
lettre x.....120 > 78 x
lettre y.....121 > 79 y
lettre z.....122 > 7A z
accolade ouvrante..123 > 7B {
barre verticale.....124 > 7C |
accolade fermante..125 > 7D }
tilde.....126 > 7E ~
symbole copyright..127 > 7F (version EXELTEL)

La touche Ctl est permise (de manière consécutive) avec les touches @ , A à Z, [, \ , ^,], _, 0 à 9.

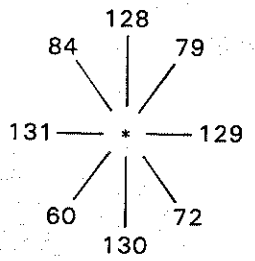
Touche	Code	Touche	Code	Touche	Code	Touche	Code
@, %	0	A, a	1	B, b	2	C, c	3
D, d	4	E, e	5	F, f	6	G, g	7
H, h	8	I, i	9	J, j	10	K, k	11
L, l	12	M, m	13	N, n	14	O, o	15
P, p	16	Q, q	17	R, r	18	S, s	19
T, t	20	U, u	21	V, v	22	W, w	23
X, x	24	Y, y	25	Z, z	26	[, (	27
\ , *	28	],)	29	^, ~	30	_, –	31
0	140	1	141	2	142	3	143
4	144	5	145	6	146	7	147
8	148	9	149				

Tableau des codes obtenus par appui Ctl + 1 Touche

Codification des touches spéciales du clavier

Curseur vers le haut.....128
Curseur vers la droite.....129
Curseur vers le bas.....130
Curseur vers la gauche.....131
Home.....133
Shift.....134
Caps lock.....135
Fonction.....136
Ctl.....137
Tab.....9
Del.....8
Escape.....27
Retour chariot.....13

Codes manettes de jeux



Le bouton de feu peut prendre les valeurs suivantes :

32, 95, 71, 78

La touche * donne le code 42.
La touche # donne le code 35.
Les touches 0 à 9 donnent des codes de 48 à 57.

Adresses utiles du Moniteur

```
************
* EQU 7020 *
************
*
* -----> DEFINITIONS DES PORTS
*
IOCTL  EQU  P0              ; I/O CONTROL
TDATA  EQU  P2              ; DATA TIMER
TCTL   EQU  P3              ; CONTROL TIMER
APORT  EQU  P4              ; PORT A
BPORT  EQU  P6              ; PORT B
RVRAM2 EQU  P36             ; DEUXIEME PORT LECTURE VDP
RVREG  EQU  P37             ; PORT LECTURE REGISTRE VDP
INIVDP EQU  P39
RVRAM1 EQU  P40             ; PREMIER PORT LECTURE VDP
WVREG  EQU  P45             ; PORT ECRITURE REGISTRES VDP
WVRAM  EQU  P46             ; PORT ECRITURE RAM VDP
MAIL   EQU  P48             ; PORT MAIL BOX
LMAIL  EQU  >0130           ; MAIL BOX POUR STA ET LDA
MODEMD EQU  P49             ;
MODEMC EQU  P50             ;
*
* -----> DEFINITIONS DES REGISTRES UTILISES
*
FLGCOM EQU  R2              ; FLAGS COMMUNICATION
VALEU0 EQU  R3              ; CODE RECU DU JOYSTICK 0
VALEU1 EQU  R4              ; CODE RECU DU JOYSTICK 1
SUICOM EQU  R6              ; ADRESSE TRAITEMENT SUITE COMMUNICATION
WRKSON EQU  R8              ; REGISTRE DE TRAVAIL POUR LE SPEECH
PTRSON EQU  R10             ; ADRESSE TABLE SON ALEATOIRE
SONFON EQU  R12             ; ADRESSE TABLE SON REPETITIF
TEMP1  EQU  R14             ; REGISTRE DE TRAVAIL
TEMP2  EQU  R16
TEMP3  EQU  R18
TEMP4  EQU  R20
TEMP5  EQU  R22
TEMP6  EQU  R24
TEMP7  EQU  R26
TEMP8  EQU  R28
TEMP9  EQU  R30
TEMP10 EQU  R32
RAMADR EQU  R34             ; ADRESSE XO YO ECRAN MAPPING
VDP06  EQU  R35
*
* -----> DEFINITIONS DES FLAGS
*
* COMMUNICATIONS <-----------------------------
*
COMM41 EQU  1               ; ATTENTE D'AKNOWLEDGE DU 7041
SPEECH EQU  2               ; FLAG SPEECH ON/OFF
COMDIR EQU  4               ; FLAG COMMUNICATION DIRECTE
CONVER EQU  8               ; FLAG COMMUNICATION EN COURS
*
* -----> ADRESSE UTILES
*
```



```
RAM    EQU  >C000           ; ADRESSE DEBUT RAM INTERNE (2K)
ROM    EQU  >0200           ; ADRESSE DE DEBUT MONITEUR
CRAM   EQU  >8000           ; ADRESSE PRISE EN MAIN COMMUNICATION 7041
REGIST EQU  >0002           ; DEBUT ZONE REGISTRE (SAUF A ET B)
BRJOY0 EQU  RAM             ; - TRAITEMENT INTERRUPT. JOY 0
BRJOY1 EQU  BRJOY0+2        ; - TRAITEMENT INTERRUPT. JOY 1
BRTIME EQU  BRJOY1+2        ; - TRAITEMENT INTERRUPT. TIMER
BRLANC EQU  BRTIME+2        ; - ADRESSE LANCEMENT PROGRAMME
BRTRAP EQU  BRLANC+3        ; - ADRESSE TRAITEMENT TRAP 23
BRINT3 EQU  BRTRAP+3        ; - ADRESSE TRAITEMENT INTERRUPT 3
SEED   EQU  BRINT3+3        ; PSEUDO ALEATOIRE
PROD   EQU  SEED+6          ; ZONE DE TRAVAIL
TEPNAM EQU  PROD+6          ; - SAUVE FILE NAME CASETTE
TEMADR EQU  TEPNAM+4        ; - SAUVE ADRESSE CASETTE
TEPCHK EQU  TEMADR+5        ; - SAUVE CHECKSUM CASETTE
FELNAM EQU  TEPCHK+1        ; - FILE NAME CASETTE
ADRSER EQU  FELNAM+4        ; ROUTINE SPECIAL INTERRUPTION SERIE
*-------------------------------------------------------------------
*---------> ADRESSES UTILISEES PAR LE MONITEUR CRAM <-------------*
*-------------------------------------------------------------------
SRECOR EQU  ADRSER+4        ; SAUVEGARDE RECORD
SLUNO  EQU  SRECOR+1        ; SAUVEGARDE LUNO
SCOMAN EQU  SRECOR+2        ; SAUVEGARDE COMAND
SDEVIC EQU  SRECOR+3        ; SAUVEGARDE DEVICE
SCURB  EQU  SRECOR+5        ; SAUVEGARDE CURBUF
SPABDR EQU  SRECOR+7        ; SAUVEGARDE PABADR
SFLG   EQU  SRECOR+9        ; SAUVEGARDE FLAG
STEM   EQU  SRECOR+10       ; SAUVEGARDE REGISTRE
SNAM   EQU  SRECOR+12       ; SAUVEGARDE
SCARCO EQU  SRECOR+13       ; ATTRIBUT CARACTERE VDP CRAM
SBORCO EQU  SRECOR+14       ; ATTRIBUT BORDER  VDP CRAM
SBAPPA EQU  SRECOR+16       ; ADRESSE BAPPA CRAM
STAPIN EQU  SRECOR+17       ; INCRUSTATION SI =4
SCM3   EQU  SRECOR+18       ; MIXE OU TEXTE MODE
OLDDEL EQU  SRECOR+19       ; DELAI POUR CLAVIER
OLDKEY EQU  SRECOR+20       ; DERNIERE TOUCHE
SBRY0  EQU  SRECOR+22       ; SAUV INTRPT VECTOR
SBRY1  EQU  SRECOR+24       ; SAUV INTRPT VECTOR
SBRTIM EQU  SRECOR+26       ; SAUV INTRPT VECTOR
SPETRP EQU  SRECOR+27           ; SPECIAL DEROUTAGE TRAP/INTERRUPT (DP)
SPESER EQU  SPETRP+3            ; 1ER OCTET LIBRE DANS RAM 2K
LIG80  EQU  SPESER+3
COL80  EQU  LIG80+1
CAR80  EQU  COL80+1
SSTAK  EQU  CAR80+20        ; Pointeur pile avant sauvegarde
SSPIL  EQU  SSTAK+1         ; Longueur de pile avant sauvegarde
MATLI1 EQU  SSPIL+60        ; Flags pour bibliotheque mathematique
MATLI2 EQU  MATLI1+1            "    "    "    "    "    "    "
MATLI3 EQU  MATLI2+1            "    "    "    "    "    "    "
S$ECRA EQU  MATLI3+1    Table definition d'ecran
S$XSX  EQU  S$ECRA+25   Curseur position colonne
S$YSY  EQU  S$XSX+1     Curseur position ligne
S$ATRB EQU  S$YSY+1     Atribut caractere
S$BAPA EQU  S$ATRB+2    ADRESSE BAPPA
S$CM3  EQU  S$BAPA+1    SAUVEGARDE VDP CM3
S$INCR EQU  S$CM3+1     SAUVEGARDE INCRUSTATION
```

```
S$BORD EQU  S$INCR+1    SAUVEGARDE BORDER COLOR
V$ACT  EQU  S$BORD+1     ; VECTEUR ACTUEL
V$INT1 EQU  V$ACT+1      ; VECTEUR INTERRUPT
V$INT2 EQU  V$INT1+1
V$INT3 EQU  V$INT2+1
*DISKST EQU  V$INT3+1    Saved disk status for DSKCTL latch
FRERAM EQU   V$INT3+2    ; DEBUT LIBRE
BUFIN2 EQU  >C100-20 DEBUT BUFFER D'ENTREE CRAM
BUFIN1 EQU  >COFE     FIN BUFFER
*
* -----> EQUIVALENCE DECODAGE IMAGES CODEES
*
CODINT EQU  >C0
VALBVR EQU  >07
VALWXY EQU  >38
*
* -----> FONCTIONS
*
*
*------>     ROUTINES DE LECTURE ET D'ECRITURE SUR
*                  BANDES MAGNETIQUES
*
*----->        DEFINITION DES REGISTRES UTISES
*
CHKSUM EQU  TEMP6        ======CHEKSUM
BUFPTR EQU  TEMP7        ======POINTEUR TAPE
DEBUFF EQU  TEMP8        ======DEBUT ZONE MEMOIRE
ENDBUF EQU  TEMP9        ======FIN ZONE MEMOIRE
CADRE  EQU  TEMP10-1     ======<FONCT K7(ENTREE)FLAG(SORTIE)
TAPFCT EQU  TEMP10       ======<LARGEUR CADRE
*
      PAGE
*
*----->        TABLE DES FONCTIONS TAPE
*
*
OUTRAM EQU  0    ENREGSTR BLOC (7020)
INRAMU EQU  2    LECT ET STOCK RAM (7020) ADR UTILI
INRAMK EQU  3    LECT ET STOCK RAM (7020) ADR ENTET
OUTVDP EQU  4    ENREGSTR BLOC (VDP)
INVDPU EQU  6    LECT ET STOCK RAM (VDP) ADR UTILIS
INVDPK EQU  7    LECT ET STOCK RAM (VDP) ADR ENTETE
*          Equates pour moniteur 7040 version EXL102
*
BRPAG  EQU  >F007        Branch to other page
CALPAG EQU  >F00A        Call an other page
LDAPAG EQU  >F00D        Load from an other page
SPHWRD EQU  >F010        Speek word
SCROLL EQU  >F013        Scroll 80 colons
GENE80 EQU  >F016        Build 80 colons generator
MODI80 EQU  >F019        Modify 80 colons generator
AFFI80 EQU  >F01C        Display a char 80 col
MESS80 EQU  >F01F        Display a message 80 col
CLR80  EQU  >F022        Clear scren 80 colons
STAPAG EQU  >F025        Store to an other page
SELPAG EQU  >F028        Select page
```

IMPRIMERIE TARDY QUERCY — CAHORS — 7295