5

SMAKY6

PROGRAMMATION DU Z80

Octobre 1980

EPSITEC-system sa

Le but de ce chapitre est de montrer en détail la structure et les instructions d'un processeur, le Z80. Seules les instructions indispensables au débutant sont étudiées en détail.

5.1 INTRODUCTION: LE PROCESSEUR ZILOG Z80

Le microprocesseur Zilog Z80 est une version améliorée, compatible software avec le 8080, avec davantage de registres et d'instructions.

Le Z80 est plus rapide que le 8080, et plus facile à interfacer; il contient des circuits de rafraîchissement des mémoires dynamiques et a des possibilités d'interruption intéressantes.

Pour toutes ces raisons, le Z80 a rapidement conquis le marché et son prix est compétitif en tenant compte de l'augmentation de performances et de l'économie dans les programmes dus à un répertoire d'instructions plus riche.

DIFFERENCES SOFTWARE ENTRE 8080 et Z80

Le Z80 est presqu'entièrement compatible avec le 8080; la différence concerne l'indicateur de dépassement de capacité, rarement utilisé.

Les avantages du Z80 sont les suivants:

- Un dédoublement de l'accumulateur A et du registre F et des registres B,C,D,E,H,L facilitant le changement de contexte lors de l'interruption ou de l'appel des sous-routines.
- Deux registres d'index IX et IY procurent les avantages de l'adressage indexé du 6800 avec pour différence le fait que le déplacement est signé, et qu'il y a deux registres. Beaucoup d'instructions utilisent ces deux registres et permettent d'effectuer les transferts nécessaires de données. Ces instructions ont toutefois 3 à 4 bytes de long et sont par conséquent plus lentes que d'autres.
- L'adressage indexé a été ajouté également pour la sélection des périphériques, l'adresse du périphérique étant mise dans le registre C.
- Des instructions de transfert de blocs de données et de recherche ont été ajoutées dans le Z80, de même que des instructions de décalage agissant sur les contenus de chaque registre et position mémoire indexées.
- L'adressage relatif existe pour quelques sauts conditionnels, mais non pour l'appel de sous-routines.
- Il est possible de mettre à "0" ou à "1" ou de tester un seul bit de n'importe quel registre ou position mémoire indexée.
- La plupart des instructions arithmétiques modifient les flags de façon plus complète que celles du 8080. Le flag de dépassement de capacité est très utile pour l'arithmétique signée, et les instructions d'ajustement décimal et de rotation 4 bits simplifient les opérations décimales en multiprécision.
- Les registres de rafraîchissement et de page d'interruption sont accessibles par software.

CARACTERISTIQUES HARDWARE

Les signaux du processeur Z80 sont simples et naturels (Fig. 1).
Il faut les coder, mais tous les signaux sont disponibles directement, sans multiplexage, à cause de l'économie de pins due à une alimentation unique (+5V) et à une entrée d'horloge unique. Tous les signaux de contrôle sont inversés, à trois états, directement compatibles pour une version multiprocesseur avec une charge du bus limitée.

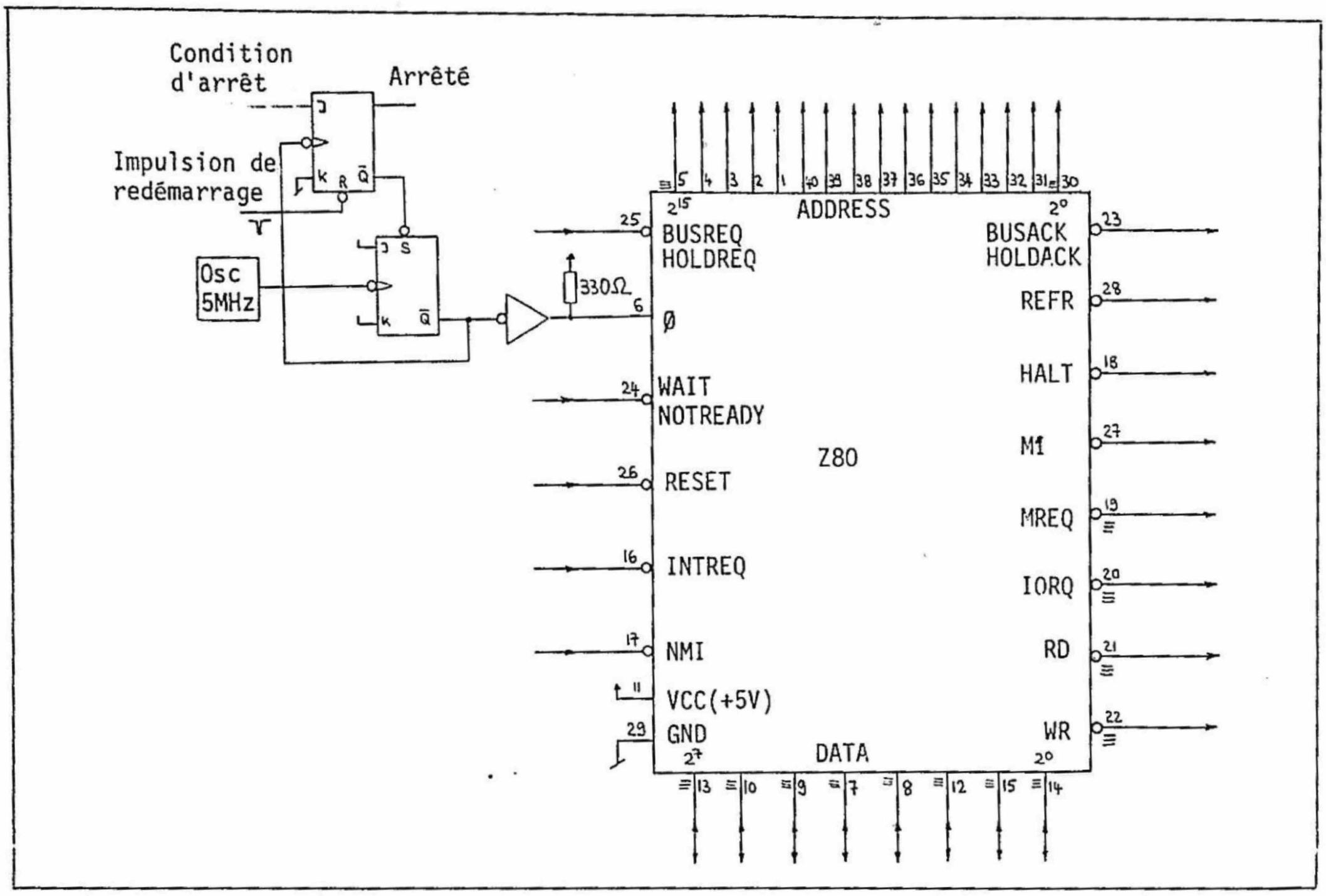


Fig. 1. Signaux du Z80 et horloge permettant d'exécuter du "clock step" lors du dépannage.

Les timings du Z80 ressemblent beaucoup à ceux du 8080, puisque ces deux circuits ont été conçus par la même équipe. Chaque instruction prend de 1 à 6 cycles, et chaque cycle de 1 à 3 états (impulsions d'horloge). Pour une instruction déterminée, le nombre d'états est généralement le même pour le Z80 et le 8080, mais la durée d'une impulsion d'horloge est respectivement 400 ns et 500ns.

Pour une description détaillée des signaux du Z80, le lecteur pourra se référer au "Z80-CPU technical Manual".

La fig. 2 donne les timings importants et montre les 6 cycles de base que le Z80 peut exécuter. Ces diagrammes montrent des signaux non inversés, qui correspondent aux signaux de contrôle inversés du processeur. Cette convention simplifie la conception des interfaces.

Il faut remarquer la différence de timing entre le premier cycle de transfert mémoire d'une instruction et les suivants. Le temps d'accès depuis l'instant de la sélection est de 450 ns lors du premier cycle de recherche et de 640 ns dans un cycle de lecture. Ces temps sont diminués par l'interface. Le temps d'accès du périphérique est aussi très court, mais il est augmenté de 800 ns durant

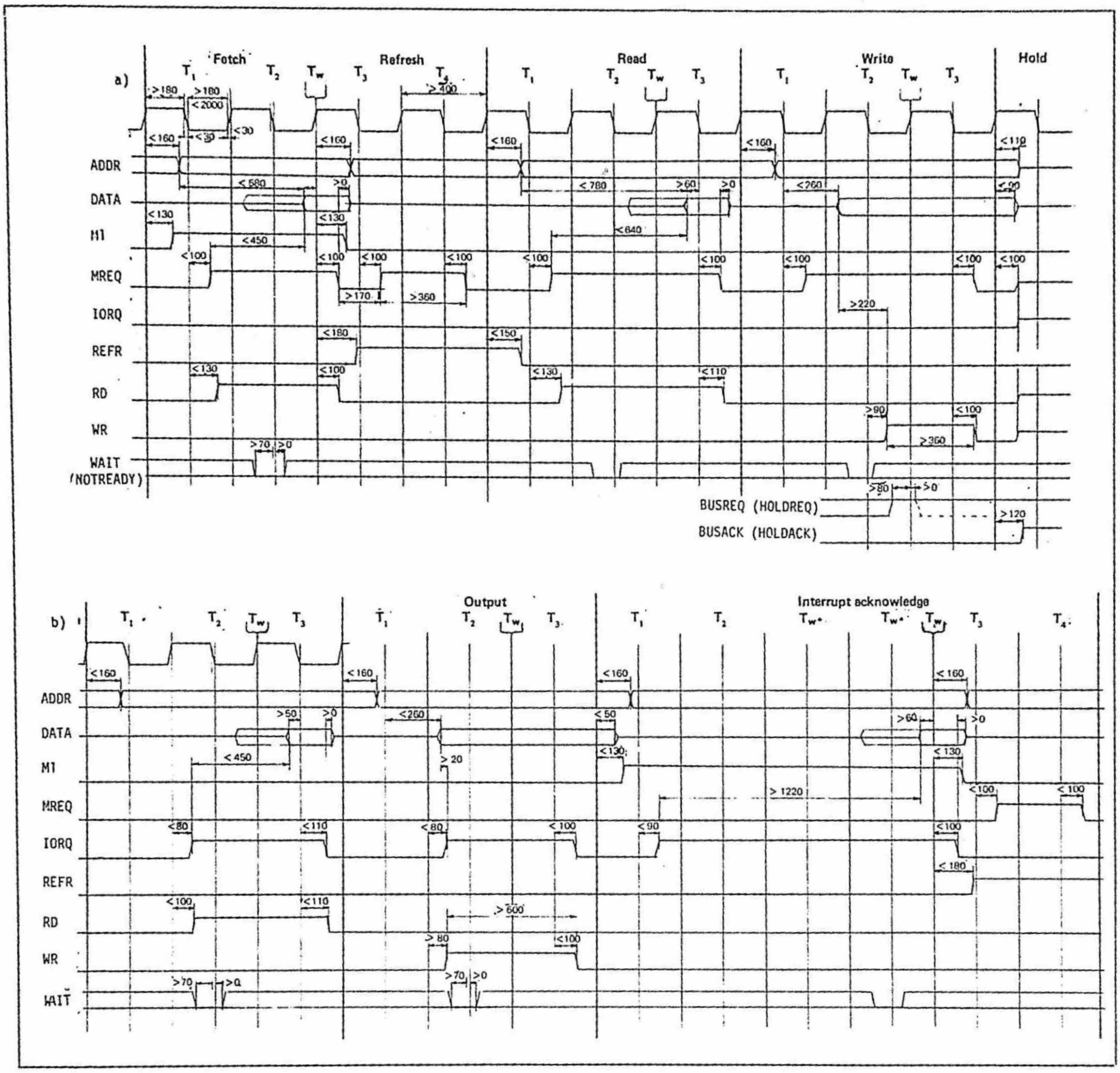


Fig. 2. Timings des 6 cycles fondamentaux du Z80.

l'"interrupt acknowledge", à cause du temps de propagation de la chaîne de priorité d'interruption utilisée par Zilog dans ses interfaces programmables. L'entrée WAIT (NOT READY) permet de ralentir le processeur par adjonction d'états Tw. Le temps accordé pour décider s'il faut attendre sur un périphérique est très bref. Il est aussi possible d'arrêter l'horloge dans son état 1, ce qui permet de contrôler le comportement du système en mode pas-à-pas (Fig. 1).

Pendant le cycle T1, le premier cycle de chaque instruction, et après l'"interrupt acknowledge", une adresse de rafraîchissement est mise sur le bus par un compteur de 7 bits, incrémenté après chaque cycle de rafraîchissement. Etant donné que certaines instructions comptent 20 états, c'est-à-dire s'exécutent en 8 µs, la fréquence de rafraîchissement est supérieure à 120 kHz , alors que la plupart des mémoires ne nécessitent que 64 kHz.

Lorsque le processeur est interrompu par le signal "Bus request", les lignes d'adresses et de données, ainsi que les lignes de contrôle à l'exception de "Bus acknowledge", sont flottantes. Les relations temporelles entre signaux

et adresses sont propres, ce qui simplifie le décodage et supprime les flips-flops de synchronisation. Le seul point regrettable est l'absence d'un signal avancé "write" dans les cycles d'écriture. Pour les instructions de sortie, ce défaut est corrigé par un timing approprié.

INTERFACE MEMOIRE

Dans un système minimal, il est facile de relier le Z80 à des circuits mémoire. Les signaux de contrôle pour des circuits mémoire ayant une pin "output enable" sont donnés dans la figure 3a. Le signal MREQ sélectionne la mémoire, l'impulsion d'écriture WR écrit en mémoire et l'impulsion de lecture RD permet de lire. Si l'on utilise une ROM sans pin "OE", il faut faire le ET logique de l'impulsion de lecture RD avec "MREQ", pour éviter tout court-circuit quand le programmeur essaie d'écrire dans la ROM.

La figure 3b montre l'interface avec une mémoire dépourvue de contrôle direct des sorties. Les signaux RD et WR doivent être combinés avec MREQ par quelques portes pour faire la sélection.

La figure 3c montre le principe de l'interfaçage avec une mémoire dynamique, rafraîchie par le microprocesseur. CE (chip enable) est actif lorsque la RAM est sélectionnée (CS actif) ou lorsque l'impulsion de rafraîchissement est décodée.

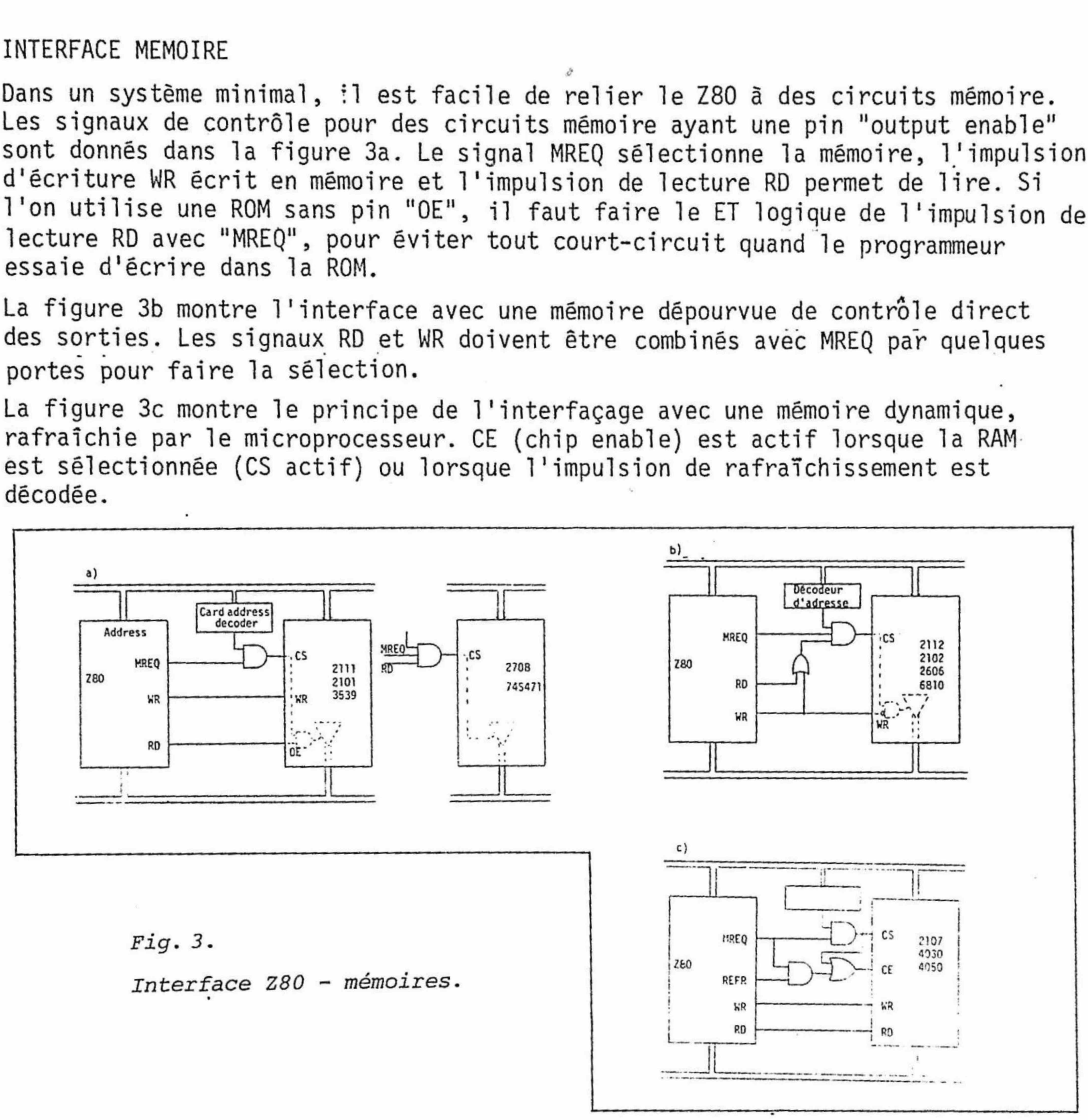


Fig. 3.

Interface Z80 - mémoires.

INTERFACES D'ENTREE/SORTIE

Zilog commercialise ou prépare une famille de chips interface faciles à utiliser dans une configuration minimale (fig. 4a). Les circuits interface d'autres fabrication peuvent également être utilisés. Les circuits INTEL ont des entrées lecture et écriture séparées et sont donc compatibles avec le Z80 (fig. 4b). Les circuits MOTOROLA ont une seule ligne de contrôle d'écriture, ce qui n'est en général pas un problème étant donné que le signal "write" du Z80 est actif aussi longtemps que le signal de sélection entrée/sortie (fig. 4c). Dans chaque cas, il faut vérifier les timings.

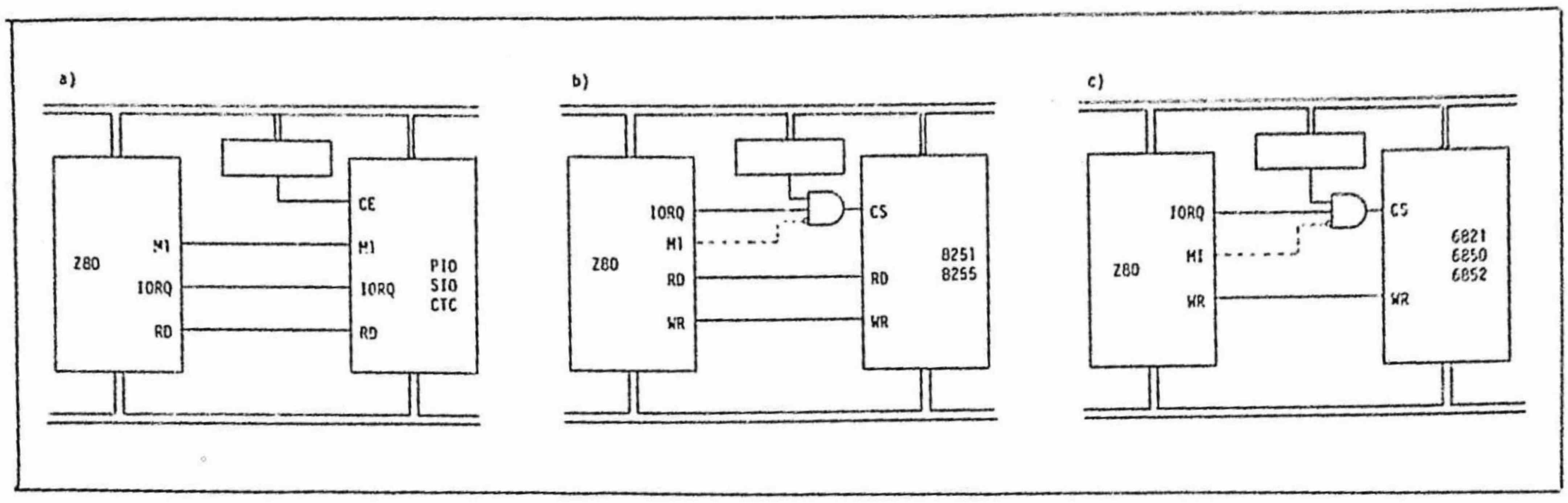


Fig. 4. Interface minimum Z80 - entrées/sorties.

INTERFACE MUBUS

Il est facile d'interfacer un processeur Z80 avec un système MUBUS· le schéma n'est toutefois pas aussi simple qu'on le voudrait, du fait que le Z80 n'a pas de signal "write" avancé.

La fig. 5 donne un schéma complet, tenant compte de toutes les contraintes, y compris la possibilité de faire un système multiprocesseur.

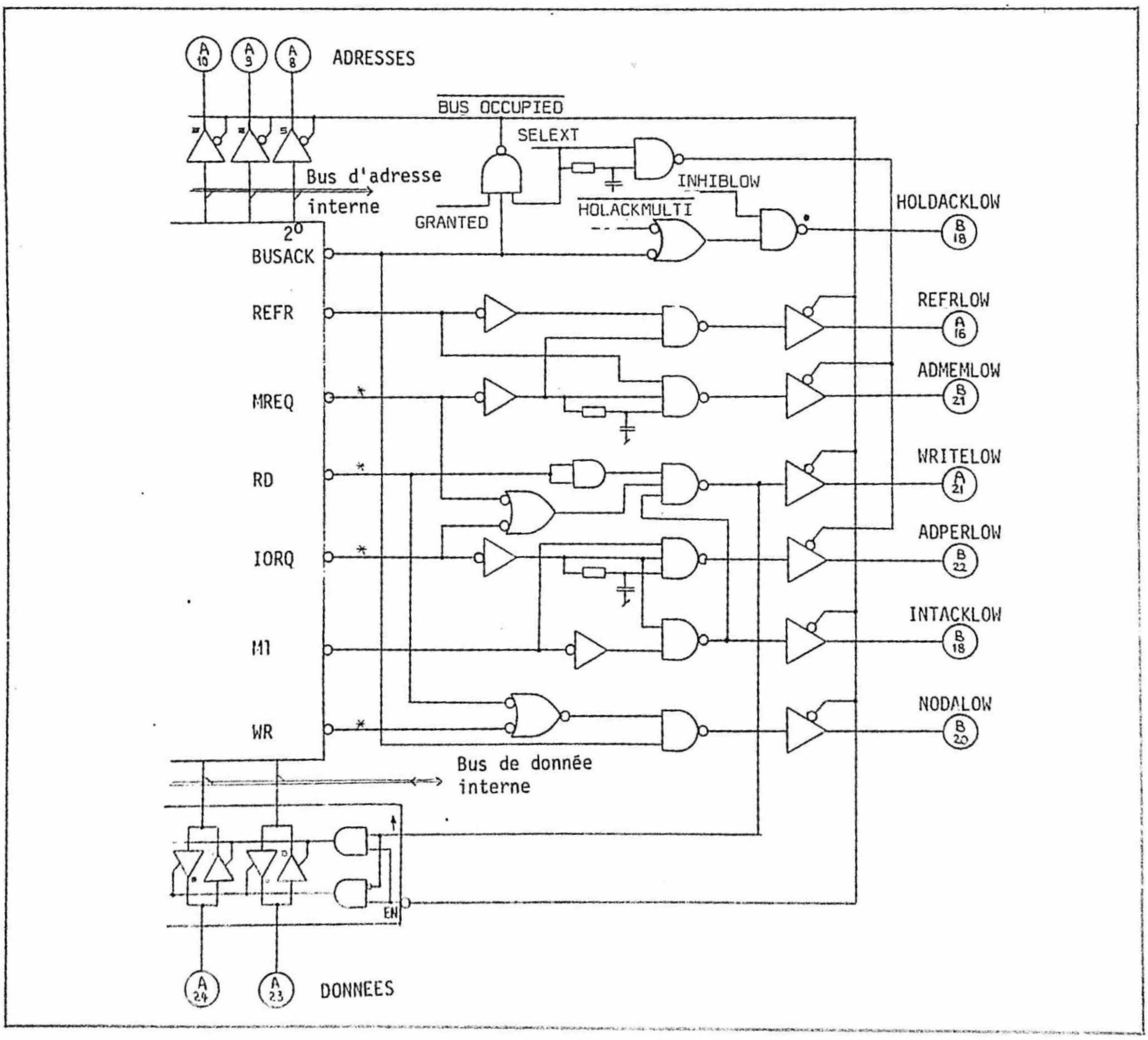
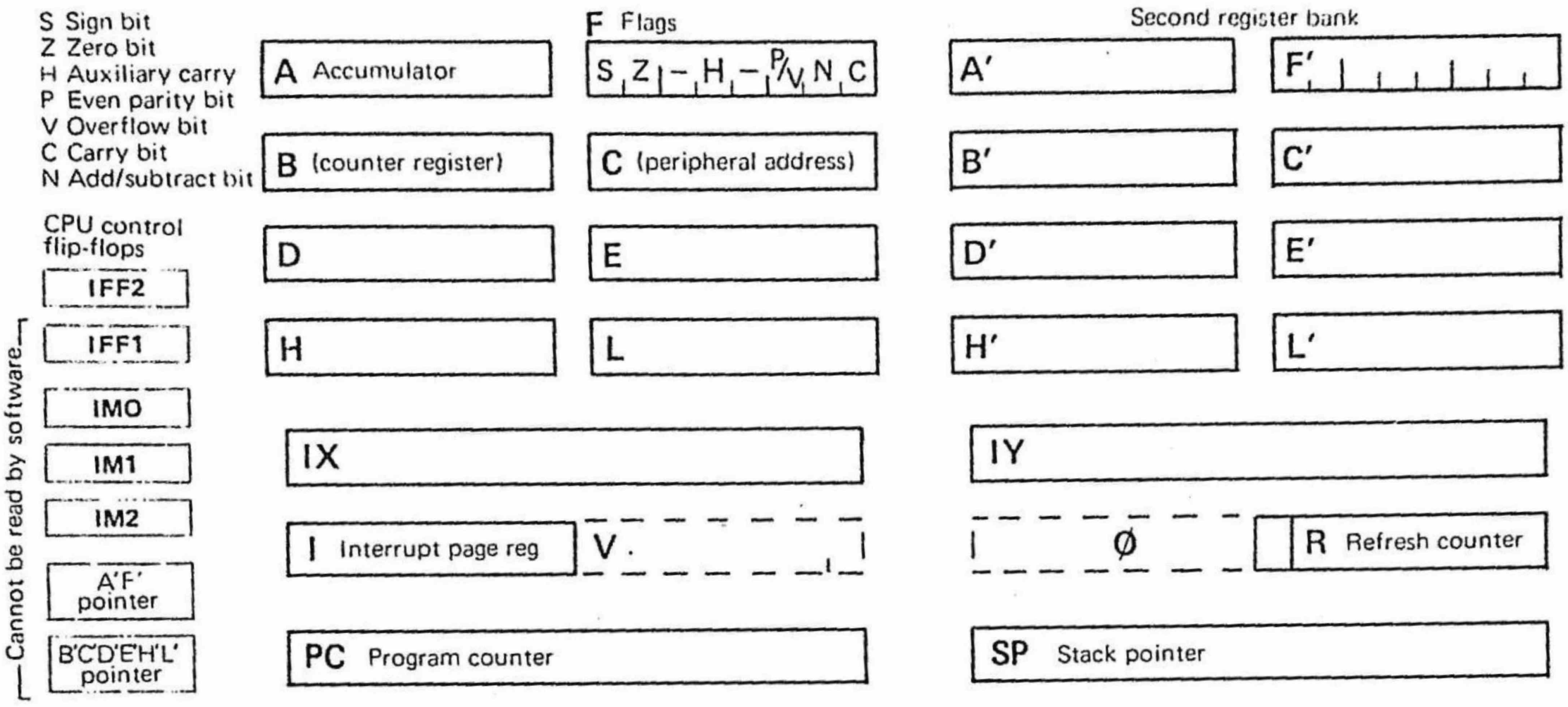


Fig. 5. Interface complet Z80 - MUBUS.

ORGANISATION DES REGISTRES, REPERTOIRE D'INSTRUCTIONS

Le processeur Zilog Z80 comporte 16 registres 8 bits, 4 registres 16 bits et deux registres spéciaux.



Les 8 registres du 8080 (Accumulateur A, registre de flags F et registres B,C, D,E,H,L) sont duplifiés. Il n'y a que deux instructions qui font l'échange des groupes AF et BCDEHL respectivement (le programmeur ne peut pas savoir quel groupe de registres il utilise).

Le registre de flags F est formé des indicateurs d'état habituels: S(signe), Z(égal à zéro), H (demi-report), C(report), qui sont modifiés selon le résultat de l'opération arithmétique précédente. Il existe aussi un flag N(soustraire) qui permet de corriger la valeur décimale après une instruction d'addition ou de soustraction. Ce flag vaut zéro, après les instructions logiques et d'incrémentation. Par contre, il vaut 1 après un ordre de soustraction, inversion, décrémentation ou comparaison. La correction décimale consiste à ajouter 6 lorsque N=0 si le résultat d'une opération binaire sur des digits décimaux est incorrect, ou de soustraire 6 si N=1. Le bit H de demi-report produit par le report sur les 4 derniers bits aide au décodage de cette condition.

Le flag le plus difficile à comprendre est le bit P/V de parité/dépassement de capacité. Ce bit est un étrange compromis entre la nécessité d'être compatible avec le flag de parité du 8080 et de désir d'ajouter un flag de dépassement de capacité, qui n'existe pas sur le 8080. Le flag P est modifié après une opération logique ou après un LOAD r,$(C) (transférer dans le registre r la donnée située dans le périphérique dont l'adresse est contenue dans C). Après les opérations arithmétiques, le même flag indique s'il y a dépassement de capacité (V=1 signale qu'il y a eu dépassement de capacité dans un calcul de complément à 2). Le fait d'utiliser le même flag pour la parité et le dépassement de capacité économise des instructions de saut conditionnel. Par exemple JUMP,PE (parity even: si résultat pair) est équivalent à JUMP,VS (oVerflow Set: dépassement de capacité). Ce même flag est encore utilisé dans l'instruction CPI, qui compare le contenu de A avec celui d'une position mémoire (S,Z,H modifiés) et décrémente le contenu des registres BC (V=0 si BC = 0). Il est utilisé également pour tester le flag d'interruption.

Le processeur contient encore quelques flip-flops de mode, qui ne sont pas accessibles au programmeur. Nous avons déjà mentionné les pointeurs pour A'F' et B'C'D'E'H'L'. Trois autres flip-flops définissant le mode d'interruption et sont mis à 1 par des instructions spéciales, qui mettent à zéro les deux autres flip-flops.

Le mode Ø est équivalent au 8080: l'instruction lue durant le signal "acknowledge" est exécutée; c'est d'habitude une instruction de restart (appel à l'une des adresses 0, 10, 20, 30, 40, 50, 60, 70 , instructions d'un byte), fournie par le périphérique demandant l'interruption.

Dans le mode 1, le processeur exécute un CALL 70, et dans le mode 2 un CALL indirect d'une adresse 16 bits, les 8 bits de poids faible étant donnés par le périphérique (valeur paire) et les 8 bits de poids fort étant contenus dans le registre I.

Le flip-flop d'interruption est dédoublé dans le processeur Z80 et la valeur de "IFF2" est transférée dans le flag P/V par l'instruction LOAD A,I. Ainsi, après une instruction LOAD A,I, JUMP,VS est équivalent à JUMP, ION et JUMP,VC à JUMP,IOF. Les deux flip-flops d'interruption IFF1 et IFF2 sont justifiés à cause de la présence dans le Z80 d'une entrée NMI (non maskable interrupt), utilisée en cas de rupture de courant.
Les interruptions mettent à zéro à la fois IFF1 et IFF2 et c'est au programmeur de réactiver l'interruption sitôt qu'il le désire..
NMI remet IFF1 à zéro, qui contrôle ION/IOF, mais ne modifie pas IFF2.
Il est donc possible de mémoriser l'état du système et de le retrouver inchangé à l'enclenchement.

Un registre spécial est le registre de rafraîchissement R. C'est un registre de 8 bits, qui peut être chargé par le contenu de A, mais c'est en fait un compteur de 7 bits, incrémenté à chaque instruction. Pendant le cycle de rafraîchissement, le bit de poids fort est transféré au bus d'adresse, mais n'est pas modifié par le comptage. Quelques applications spéciales sont possibles grâce au registre de rafraichissement; il peut, par exemple, être utilisé comme générateur de hasard 7 bits (bien que la probabilité soit faussée par le fait que certaines boucles comportent un nombre d'instructions non premier à 128).

Les 4 registres 16-bits du Z80 sont le compteur d'adresses PC et le pointeur d'adresses SP, ainsi que deux registres d'index (qui n'existent pas sur le 8080). Ainsi, avec le Z80, on peut utiliser simultanément jusqu'à 9 pointeurs (BC,DE,HL, B'C',D'E',H'L',SP,IX,IY) et les instructions utilisant les registres d'index IX et IY permettent d'ajouter un déplacement signé.

Les instructions spécifiques au Z80 (inexistantes avec le 8080) sont les instructions de saut relatif. Il en existe une de saut inconditionnel et quatre de saut conditionnel, ainsi qu'une instruction combinée de décrémentation et de saut relatif, qui permet de parcourir une boucle tant que le contenu du registre B n'est pas nul (DECJ,NE B,LOOP)

La convention pour l'adressage relatif est identique à celle qui est faite pour le 6800. Le second byte de l'instruction contient un déplacement signé de 8 bits, qui est ajouté au contenu du compteur d'adresses (lorsque ce dernier pointe déjà l'instruction suivante).

Lorsque le programmeur écrit une instruction, il considère comme adresse celle de début de l'instruction: il y a donc une différence de 2 unités avec ce que le microprocesseur prend comme adresse. JUMP .+ℓ' signifie JUMP (PC)+ℓ' avec PC pointant l'adresse de l'instruction suivante. Si le programmeur veut revenir à l'instruction précédente (et en supposant qu'elle comporte 2 bytes), ℓ' = -2 = 376 et le deuxième byte de l'instruction de saut relatif est -2-2 = -4 = 374.

Dans un programme, on n'écrira jamais JUMP .+4 ou JUMP .-23, mais par exemple JUMP NEXT ou JUMP ROUTINE. L'assembleur calculera la différence et tiendra compte de la correction de deux unités.

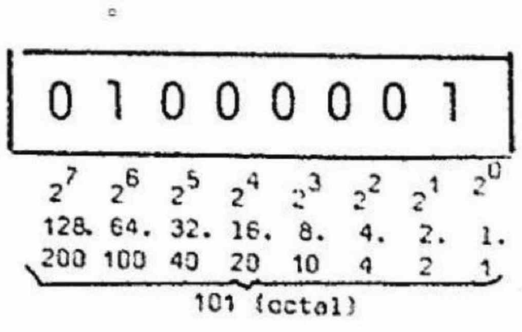
Pour commencer, nous ne considérerons que les registres A,B,C,D,E,F,H,L (8 bits), PC et SP (16 bits). Parmi les bits du registre F, nous nous intéresserons d'abord à S (signe du résultat), Z (vaut 1 lorsque le résultat est nul) et C (retenue produite par l'unité arithmétique).

5.2 INTRUCTIONS DE TRANSFERT

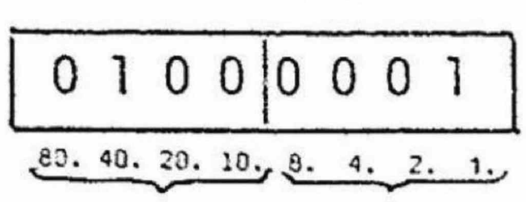
RAPPEL: CODAGE DE L'INFORMATION

Rappelons que le mot *bit*, utilisé aussi bien par l'anglais que le français représente un chiffre binaire, c'est à dire 0 ou 1.

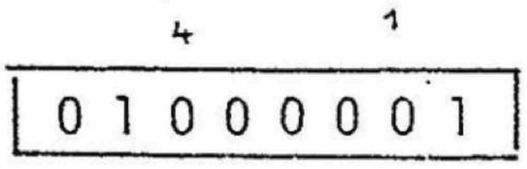
Considérons un mot de 8 bits, que les américains appellent *byte* et les français octet. Ce mot de 8 bits, par exemple 01 000 001, peut représenter



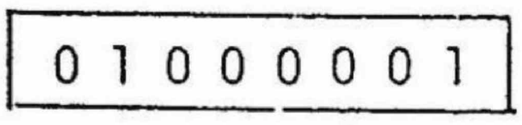
≙ un nombre binaire, chaque bit ayant un poids égal à 2^0, 2^1, 2^2, 2^3, ..., 2^7 (de droite à gauche).
Ce nombre binaire est équivalent à 101 octal ou 65. décimal (les nombres décimaux sont suivis d'un point).



≙ si l'on partage les 8 bits en 2 groupes de 4 bits et si l'on prend les équivalents décimaux des 2 nombres binaires de 4 bits, on obtient le nombre BCD (décimal codé binaire) 41.



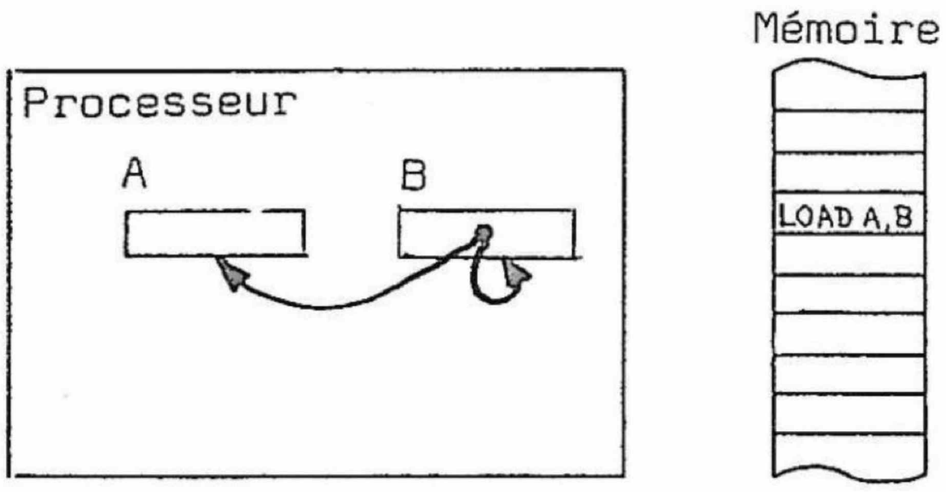
≙ représente aussi l'une des 256 instructions possibles d'un ordinateur 8 bits (pour le Z80, LOAD B,C qui copie le contenu du registre C dans le registre B).



≙ représente la lettre A dans le code ASCII.

INSTRUCTIONS DE TRANSFERT ENTRE REGISTRES ET MEMOIRE

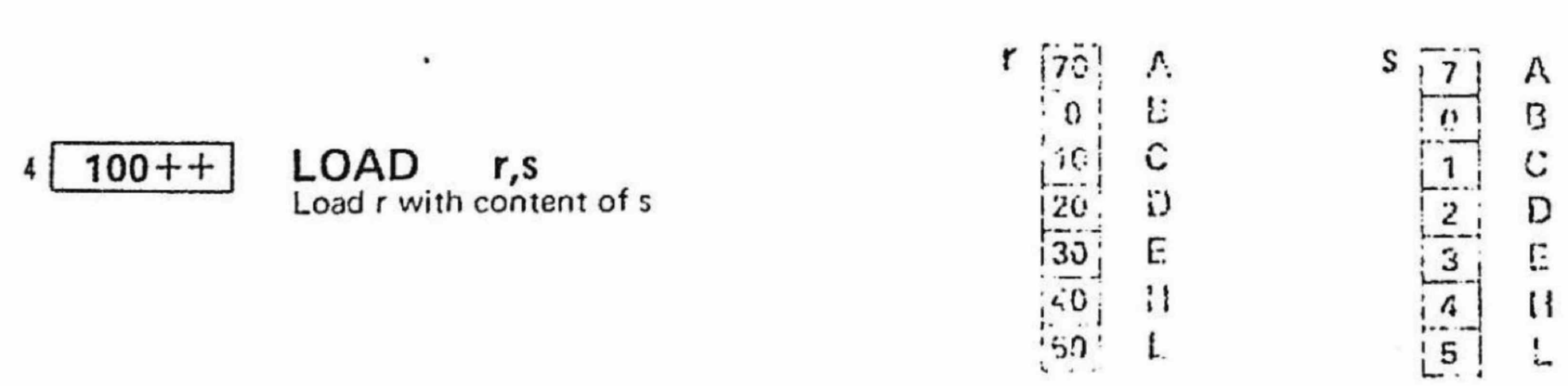
Les instructions de transfert sont toutes caractérisées par le code mnémonique LOAD.



Adressage entre registres

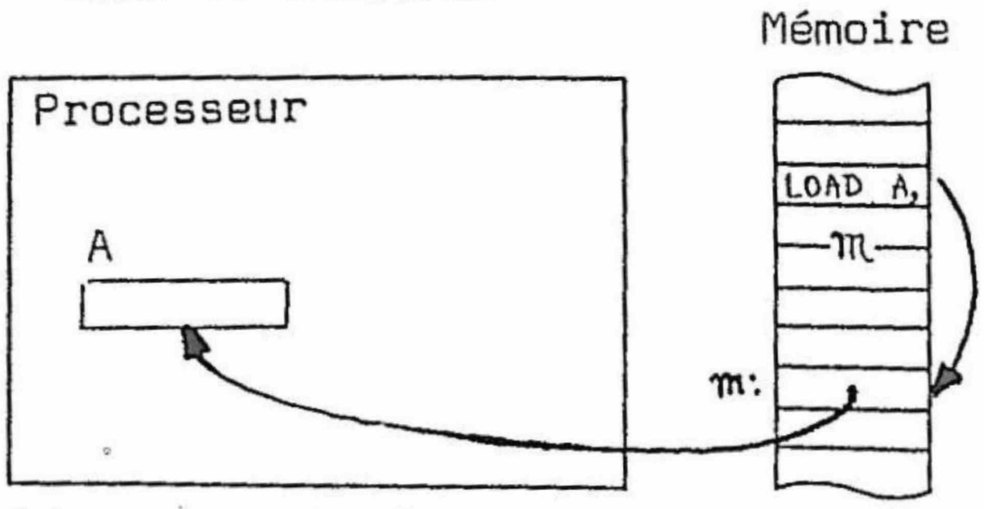
L'instruction LOAD A,B transfère le contenu du registre B dans le registre A. Le contenu de B n'est pas modifié.

Il y a 49 instructions de transfert entre les 7 registres de 8 bits A,B,C,D,E,H,L. Le tableau complet de ces instructions prendrait beaucoup trop de place et l'on écrit de façon condensée



Pour assembler une instruction déterminée, on effectue mentalement deux additions (en octal). Le ++ dans le code de base signifie qu'il faut ajouter deux valeurs, qui dépendent des opérandes (le tableau des valeurs à ajouter se trouve à proximité). Par exemple, le code octal correspondant à LOAD D,B est 100+20+0 = 120 .

Un autre type d'instruction de transfert est LOAD A,m, où m est une adresse dans la mémoire.

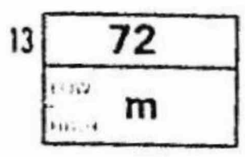


Adressage absolu

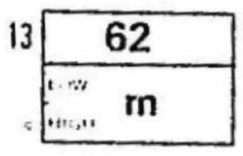
L'instruction LOAD A,m transfère dans A le contenu de la position mémoire d'adresse m.

L'avantage de cette instruction est que l'on peut modifier facilement le programme en modifiant le contenu de m.

Les transferts avec la mémoire ne peuvent se faire qu'avec le registre A, et sont codés



LOAD A,m
Load A absolute



LOAD m,A

m 16-bit number

Trois mots de 8 bits sont nécessaires pour coder une instruction de transfert entre un registre de 8 bits et une position mémoire. Le premier byte caractérise l'opération et le registre concerné, le deuxième byte contient les bits de poids faible de l'adresse, et le troisième byte les bits de poids fort. Par exemple, l'instruction LOAD A,PREMNB, destinée à transférer dans A le premier nombre d'une série de valeurs préalablement mises en mémoire (PREMNB est par exemple à l'adresse valant 2057 octal) se code:

72
57
4

adresse 57
page 4

adresse complète 2057 = 4·400 + 57

Le fractionnement de l'adresse complète en deux mots de 8 bits revient à considérer des pages de 256. positions, avec dans chaque page une adresse 8 bits valant 0 à 377. L'adresse dans la page se trouve dans le 2e byte, l'adresse de la page se trouve dans le 3e byte, et le tableau de correspondance suivant facilite le fractionnement:

adresse	0 - 377	page Ø
	400 - 777	1
	1000 - 1377	2
	1400 - 1777	3
	2000 - 2377	4
	2400 - 2777	5
	3000 - 3377	6
	3400 - 3777	7
	4000 - 4377	10
	...	
	40000 - 40377	100 (Ecran SMAKY 6)

On remarque une relation entre le nombre de milliers, multiplié par deux en octal et la page: par exemple l'adresse 6750 se code

```
      6 0 0 0  +  4 0 0  +  3 5 0
page    2x6    +    1
        14     +    1
         \_____/
          page 15     adresse  350
```

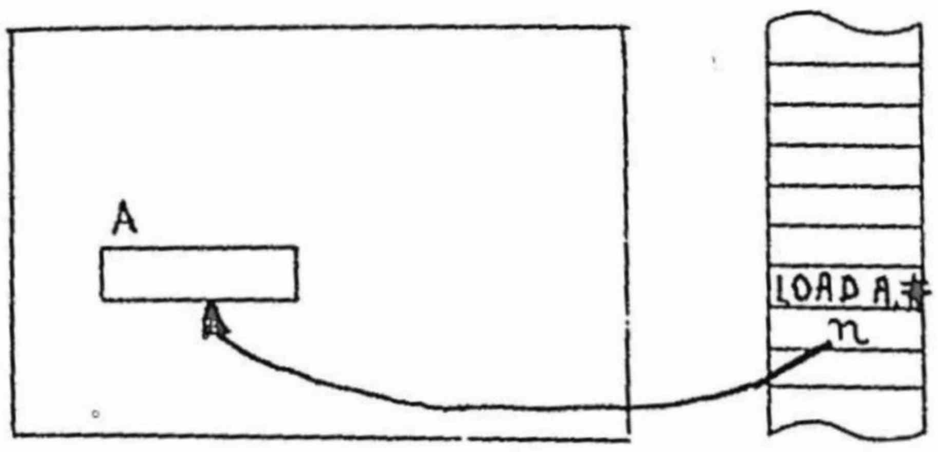
Pour coder l'instruction qui transfère dans le registre A le contenu de la position mémoire 6750, on utilise les codes suivants

72
350
15

LOAD A,6750

INSTRUCTIONS D'INITIALISATION DES REGISTRES

Un registre peut être initialisé à une certaine valeur prévue par le programme, comme nous l'avons vu dans la première partie.



Adressage immédiat

L'instruction LOAD A,#n (# se lit valeur) prend la valeur n et la met dans le registre A.

Les instructions d'initialisation des registres peuvent s'écrire

1	6+	LOAD r,#n
	n	Load r immediate

n 8-bit number

r		
	70	A
	0	B
	10	C
	20	D
	30	E
	40	H
	50	L

Ces instructions ont deux bytes.
n est une valeur numérique de 8 bits (0 à 377) représentée dans l'instruction par un symbole évoquant sa signification.

INSTRUCTIONS DE TRANSFERT ENTRE REGISTRES ET PERIPHERIQUES

Le Z80 permet de définir jusqu'à 256. adresses de périphériques, numérotées de 0 à 377, dans lesquelles on peut lire ou écrire comme en mémoire avec les instructions suivantes:

10	333	LOAD A,$n
	n	Load A with data from peripheral n (input)

11	323	LOAD $n,A
	n	Load peripheral n with content of A (output)

Les adresses de périphériques se distinguent des adresses mémoire grâce au signe $. Le transfert peut donc se faire directement avec le registre A. Souvent seule une partie des huit bits d'adresse est décodée, parce que l'on a rarement besoin de 256. périphériques. Le DAUPHIN, par exemple, ne décode que 6 bits d'adresse de périphériques. Les instructions LOAD A,$1, LOAD A,$101, LOAD A,$201 et LOAD A,$301 sont alors tout à fait équivalentes, mais il n'y a pas lieu d'utiliser les trois dernières. (Nombre maximum de périphériques distincts: 64).

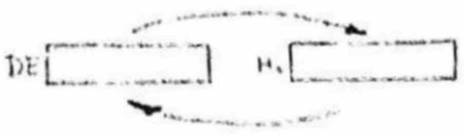
INSTRUCTIONS DE TRANSFERT AVEC LES REGISTRES 16 BITS

Les registres du Z80 sont associés par paires pour certaines instructions: H avec L, D avec E, B avec C. Les instructions de type LOAD HL,BC n'existent pas, car on peu les faire facilement avec deux instructions

```
LOAD H,B
LOAD L,C
```

Il existe en revanche une instruction qui permute les contenus des registres HL et DE (et ceux-là seulement)

4	353	EX DE,HL
		Exchange DE and HL



Le transfert de 16 bits en mémoire (aux adresses m et m+1) avec le registre HL est possible (adressage absolu)

16 | 52 | m (low) | LOAD HL,m

16 | 42 | m (low) | LOAD m,HL

Le Z80 autorise également le transfert des registres BC,DE,SP.

L'initialisation d'un registre 16 bits par une valeur donnée dans l'instruction (adressage immédiat) est également possible

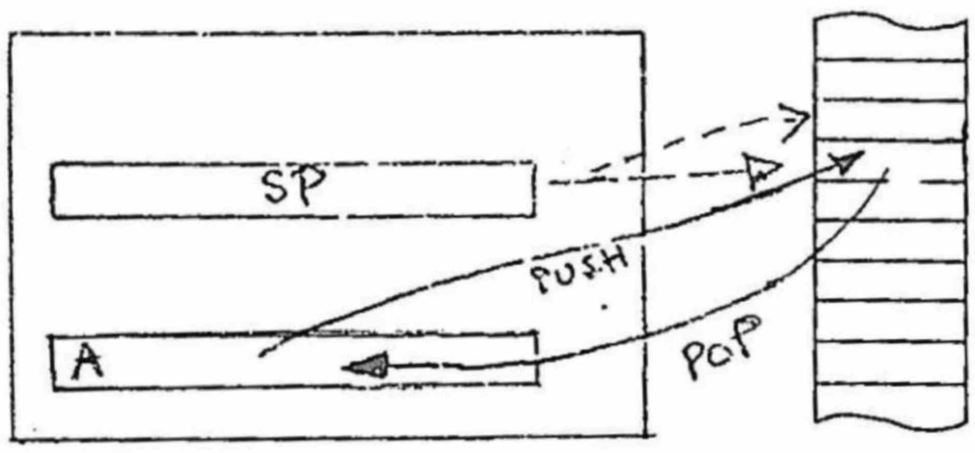
10 | 1+ | m (low, high) | LOAD p,#m

p	
0	BC
20	DE
40	HL
60	SP

On remarque comme avant que dans l'instruction la donnée de 16 bits est coupée en deux avec tout d'abord les 8 bits de poids faible (registre L,E ou C), puis les 8 bits de poids fort.

UTILISATION DE LA PILE

Une autre possibilité de transfert utilise le registre SP */Stack pointer/*, appelé pointeur de pile. C'est un registre 16 bits, dont le contenu représente l'adresse d'une certaine position mémoire.



L'instruction PUSH A transfère le contenu de A dans la position pointée par SP et, simultanément, décrémente le registre SP. En effectuant successivement PUSH A, PUSH A... on écrit le contenu de A dans les positions mémoire successives en dessus (si les lères adresses de la mémoire sont en haut du dessin) de la première.

Avec le Z80, on peut seulement sauver sur le stack des paires de registres: A et F, BC, DE, HL. En général on effectue PUSH AF, PUSH BC, PUSH DE ce qui mémorise les contenus des registres AF, BC, DE dans 6 positions mémoire successives (l'utilisateur ne se préoccupe pas de savoir lesquelles). Ensuite on reprend ces valeurs, dans l'ordre inverse, au moyen des instructions POP DE, POP BC, POP AF.

La pile */stack/* doit être initialisée pour pointer une zone mémoire disponible. Cette initialisation est faite par le programme moniteur. Lorsqu'on n'utilise pas ce programme moniteur, il faut définir la pile soi-même.

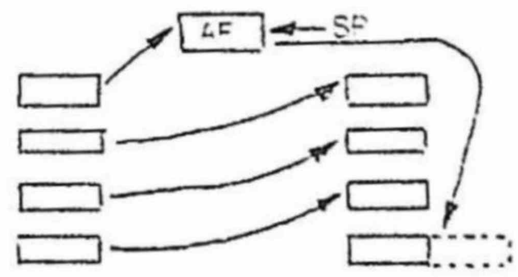
11 | 305+ | PUSH p

10 | 301+ | POP p
[POP AF modifies all flags]

p	
60	AF
0	BC
20	DE
40	HL

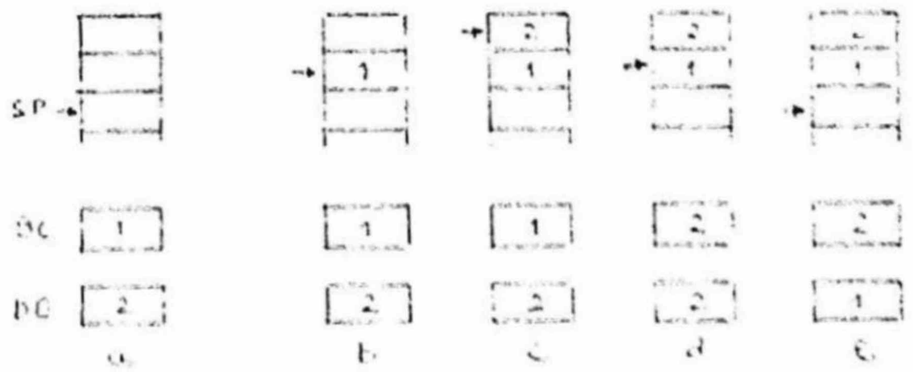
En utilisant ces nouvelles instructions, le programme permutant circulairement les contenus des registres A,B,C,D s'écrit simplment

```
LOAD    SP,#STACK    ;stack initial
PUSH    AF
LOAD    A,B
LOAD    B,C
LOAD    C,D
POP     DE           ;D contient alors la dernière valeur placée sur la
                     ;pile, soit le contenu primitif de A.
```



Comme l'instruction EX BC,DE n'existe pas, on peut la remplacer par:

```
PUSH    BC
PUSH    DE
POP     BC
POP     DE
```



5.3 COMPLEMENT A 1 ET A 2

Sans vouloir redéfinir de façon détaillée les notions de complément et de représentation des nombres négatifs, rappelons que le complément à 1 d'un mot binaire s'obtient en inversant tous ses bits, ou en effectuant la différence avec un mot de même longueur qui ne comporte que des 1.

Exemples: mot de 8 bits: 10 110 101 = 265 octal
complément à 1: 01 001 010 = 112 octal

Calcul du complément à 1 par soustraction:

```
  11 111 111      377
-10 110 101     - 265
----------      -----
  01 001 010      112
```

Mot de 16 bits: 1 101 011 100 111 110 = 153476 octal
Complément à 1: 0 010 100 011 000 001 = 024 301

Calcul du complément à 1 par soustraction:

```
  1 111 111 111 111 111      177 777
-1 101 011 100 111 110     - 153 476
-----------------------    ---------
  0 010 100 011 000 001      024 301
```

Le calcul en octal directement évite de devoir écrire des lignes de Ø et de 1: il est très facile, car les règles sont ici identiques à celles du système décimal.

Le complément à 2 s'obtient en faisant la différence avec un nombre égal à 2^n, n étant le nombre de bits. Ce nombre a partout des 0, avec un 1 pour le poids immédiatement supérieur.

Exemples: mot de 8 bits: 10 110 100 = 264 octal

Calcul du complément à 2:

```
  100 000 000      400
-  10 110 100    - 264
---------------    -----
   01 001 100      114
```

En octal 8 bits le calcul se ramène à des différences à 8, puis à 7, avec pour chiffre de poids le plus fort une différence à 4 ou à 3, car ce chiffre ne code que deux bits.

Si on a des mots de 16 bits, il faut faire la différence à 200 000. Dans la pratique on fait la différence à 177 777 et on rajoute 1.

Le complément à 2 d'un nombre est généralement utilisé pour représenter l'opposé de ce nombre. Il est important de connaître la longueur du mot binaire et être sûr que les nombres positifs ou négatifs à représenter ne dépassent pas la grandeur permise.

Exemples: mots de 8 bits: +45 représenté 00 100 101
-45 représenté par son complément à 2^8, égal à
400 - 45 = 333 11 011 011
↑
bit de signe
+243 ne peut pas être représenté (valeur maximum +177)
-303 ne peut pas être représenté (valeur minimum -200 représenté par 20

Autres exemples

Nombre	Complément à 2
2	376 représente -2
14	364 représente -14
321 (-57)	57

Les nombres représentés ainsi sont appelés nombres arithmétiques. Le bit de poids le plus fort est le bit de signe. Lorsqu'il vaut 1, le nombre est négatif et représenté sous forme de complément à 2. Pour en connaître la valeur absolue, il faut en reprendre la complément à 2.

Exemple: Quel est l'équivalent décimal du nombre arithmétique octal 8 bits 323 ?

- Ce nombre est négatif car le 8e bit vaut 1 (> 200 base 8)
- La valeur absolue, égale au complément à 2, vaut 400 - 323 =055
  (calcul mental: 0-3, c'est-à-dire 8-3 = 5 avec emprunt
  0-emprunt-2, càd 7-2=5 avec emprunt
  4-emprunt-3 = 0)
- La valeur décimale équivalente est 55 (octal) = 5·8+5 = 45. (décimal)
- Le nombre arithmétique 8 bits 323 donné représente le nombre décimal -45.

Il faut remarquer dans la donnée de cet exemple que les 3 termes arithmétique octal 8 bits sont importants. Le nombre pourrait être un nombre positif octal 8 bits (certains parlent alors de nombre logique). Dans ce cas, les 8 bits sont utilisés pour les nombres de 0 à 377, et il n'est plus possible de représenter les nombres négatifs.

Le nombre pourrait être un nombre arithmétique octal 9 bits. Dans ce cas, 323 serait un nombre positif, car le 9e bit vaut 0.

Si le nombre était un nombre arithmétique décimal, il ne faudrait pas préciser le nombre de bits, mais le nombre de digits, et définir clairement comment le signe est représenté.

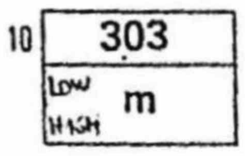
La représentation des nombres négatifs sous forme de nombres arithmétiques en complément à 2 facilite considérablement les opérations arithmétiques. Il suffit d'ajouter les compléments au lieu de soustraire. Par exemple si l'on doit effectuer l'opération 8 bits 124+31-173 (octal), et que l'on s'est assuré que chaque nombre et chaque résultat partiel ne dépasse pas la capacité de la machine (7 bits plus un 8e bit de signe), il suffit de convertir -173 en son complément à deux 8 bits (400-173=205) et d'additionner

```
  124        155
+  31      +205      ──→ Le résultat est négatif (> 200) et peut être conservé
 ----      ----           tel quel pour les calculs ultérieurs.
  155        362
```

Ces quelques exemples montrent bien la diversité des représentations des nombres et les problèmes et erreurs qui peuvent apparaître lorsque la représentation utilisée n'a pas bien été définie et comprise.

5.4 INSTRUCTIONS DE SAUT

Les instructions de saut permettent de revenir en arrière, passer par-dessus une zone réservée, etc.



JUMP m
Jump absolute

} adresse du saut: poids faible (positio
puis poids fort (page) avec le Z80

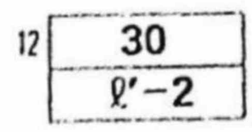
Comme exemple d'application d'un saut, citons le raccommodage */patch/* d'un programme dans lequel on doit insérer une ou plusieurs instructions oubliées. Plutôt que de réassembler et recharger toute la suite du programme, on peut supprimer quelques instructions et les remettre avec l'instruction oubliée dans la zone de raccommodage.

```
EXEMPLE:  PROG:   100        LOAD    D,# Ø
                    1
                    2        LOAD    B,# 15
                    3                      OUBLI:  ADD   A,#  10
                    4        LOAD    TRUC,A
                    5
                    6
                    7        ADD     A,# 10
```

Programme "racommodé"

```
          PROG:   100        LOAD    D,# 0
                    1
                    2  303   JUMP    PATCH
                    3  200
                    4    0
                    5        plus utilisé
          NEXT:     6
                    7        ADD     A,# 10

          PATCH:  200        LOAD    B,# 15
                    1
                    2        ADD     A,# 10
                    3
                    4        LOAD    TRUC,A
                    5
                    6
                    7  303   JUMP    NEXT
                  210  106
                    1    0
```

En plus du saut absolu, il existe une instruction de saut relatif. L'adressage relatif permet de définir une adresse par rapport à l'adresse de l'instruction qui s'y réfère, de même que l'on dit dans le langage courant "il habite deux maisons plus loin que moi". L'avantage est que les nombres sont plus petits et prennent moins de place, donc les programmes plus courts et moins coûteux en mémoire.



JUMP .+ℓ'
Jump relative

Le déplacement ℓ permet de sauter de 1 à 177 positions mémoire ou de revenir en arrière de 177 positions au maximum (dans ce cas ℓ vaut de 377 à 200). Le programmeur, s'il fait l'assemblage à la main, doit calculer le déplacement en faisant la différence entre l'adresse de l'instruction à laquelle il veut sauter et l'adresse de l'instruction qui suit l'instruction de saut relatif. Si le saut revient en arrière (déplacement négatif), le complément à 2 est utilisé.

EXEMPLE: *Le programme vu précédemment devient*

```
PROG:   100          LOAD    A,# 0
          1
          2   30     JUMP.   PATCH
          3   74                          200-104 = 74
NEXT:     4          LOAD    TRUC,A
          5
          6          ADD     A,# 10
          7

PATCH:  200          LOAD    B,# 15
          1
          2          ADD     A,# 10
          3
          4   30     JUMP    NEXT
          5  276                          104-206 = -102 = 377-102+1 =
                                                = 275 + 1 = 276
```

Une autre instruction qui utilise l'adressage relatif est

8	20	**DECJ,NE B,.+ℓ'**	13 states if B = Ø
13	ℓ'−2	Decrement B and jump relative if result non equal to zero (no flags modified)	

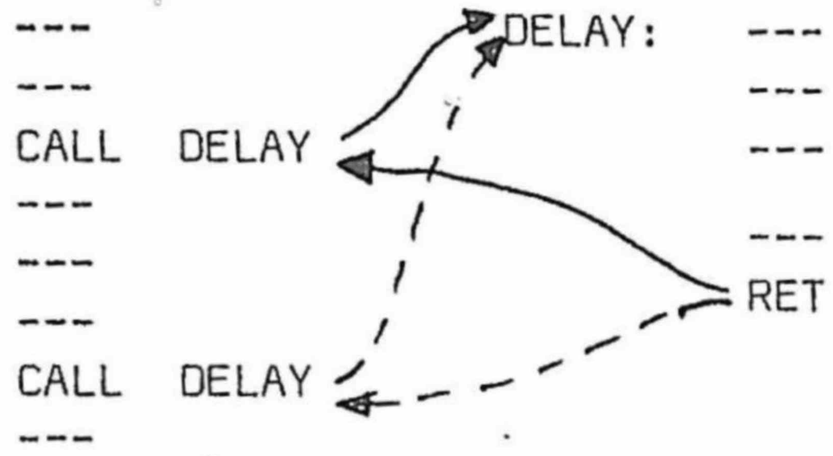
qui permet de faire facilement des boucles d'attente. DECJ,NE B,.+ℓ' décrémente le registre B jusqu'à ce que son contenu soit Ø. Si la valeur initiale de B est nulle, la boucle est effectuée 400 fois (octal) = 256. , valeur maximum possible.

Ainsi l'instruction | 20 / 376 | DECJ,NE B,. qui demande 8 cycles pour être exécutée prend un temps total de 256.x8.x0,4 µs = 820 µs si la valeur initiale de B est Ø et si le processeur tourne à vitesse maximale. Le symbole . (point) veut dire PC (compteur d'adresses). JUMP . est un saut sur place (seul un RESET permet d'en sortir) que l'on préfère écrire en général STOP: JUMP STOP .

5.5 APPELS DE SOUS-PROGRAMMES

Lorsque la même partie de programme doit apparaître à plusieurs endroits du programme (par exemple une attente de quelques millisecondes), il est stupide de la recopier, sauf si c'est plus court ou plus rapide.

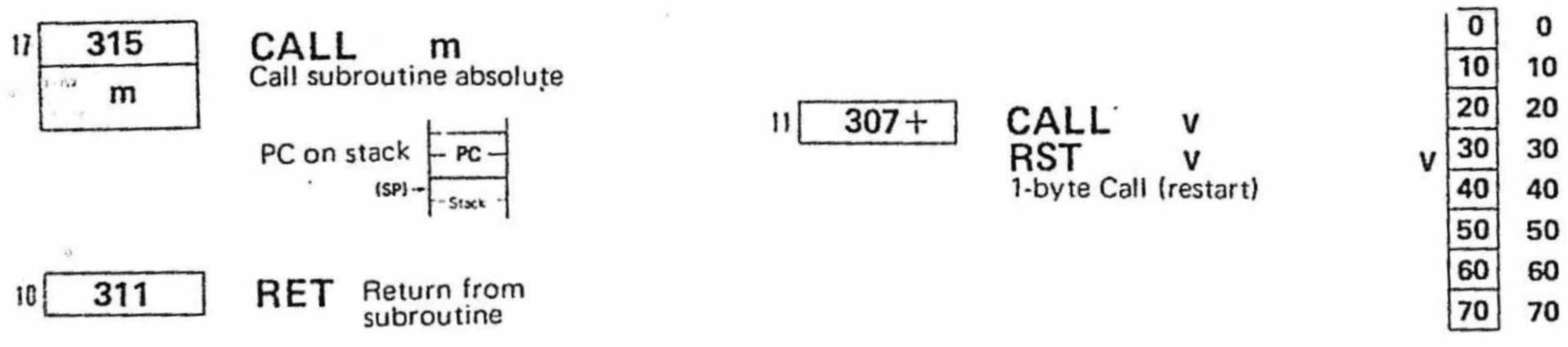
L'instruction CALL permet de sauter à la copie unique de cette partie de programme appelée alors sous-programme */sub-routine/*, avec retour automatique à la suite du programme à la fin du sous-programme, lorsque l'instruction RET */return/* est exécutée.



On ne peut pas mettre une instruction JUMP à la fin du sous-programme, car on retournerait toujours à la même adresse. L'instruction RET va chercher une adresse, mémorisée dans le registre SP */Stack Pointer/* du processeur lors du CALL, qui est l'adresse de l'instruction suivante dans le programme principal.

L'instruction CALL est identique à un JUMP avec sauvetage de l'adresse de l'instruction suivante (qui se trouve dans le PC */Program Counter/*) sur la pile. On peut dire que CALL X est équivalent à PUSH PC suivi de JUMP X.

On remarque dans le Z80 l'existence d'une instruction CALL à 1 byte, appelée parfois RST, qui permet d'appeler des routines placées aux positions 0, 10, 20, ..., 70 de la mémoire.

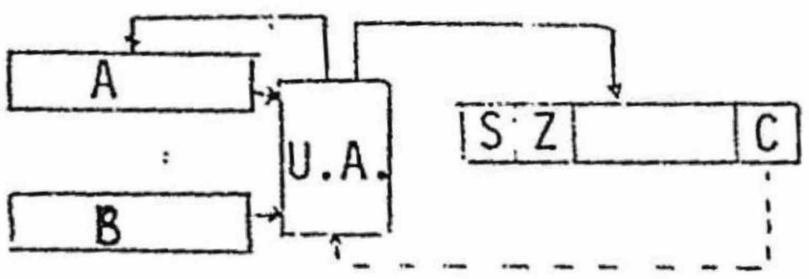


5.6 INSTRUCTIONS ARITHMETIQUES

INSTRUCTIONS D'ADDITION

Après les instructions de transfert et de saut, nous abordons les instructions qui effectuent des opérations.(addition, soustraction, comparaison, décalage,..). Nous avons aperçu dans un exemple précédent l'instruction ADD A,#10, mais nous allons d'abord étudier les opérations entre deux registres.

L'instruction ADD A,B , par exemple, additionne les contenus des registres A et B et met le résultat dans A. Elle modifie donc A, ainsi que le registre de flags.



S bit 2^7 = 200 bit de poids fort du résultat (signe (S=0 positif; S=1 négatif)

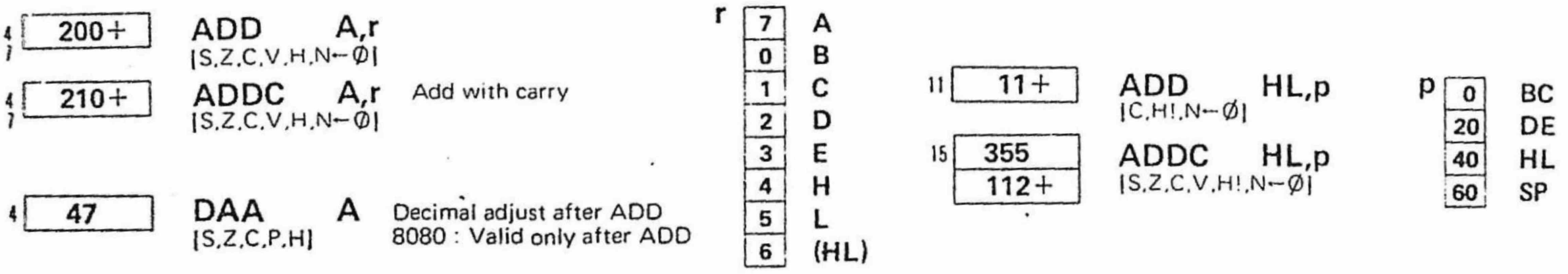
Z bit 2^6 = 100 vaut 1 si résultat = Ø

C bit 2^0 = 1 Carry vaut 1 si l'addition produit un dépassement de capacité

L'instruction ADDC A,B additionne les contenus des registres A et B et le conteu du Carry

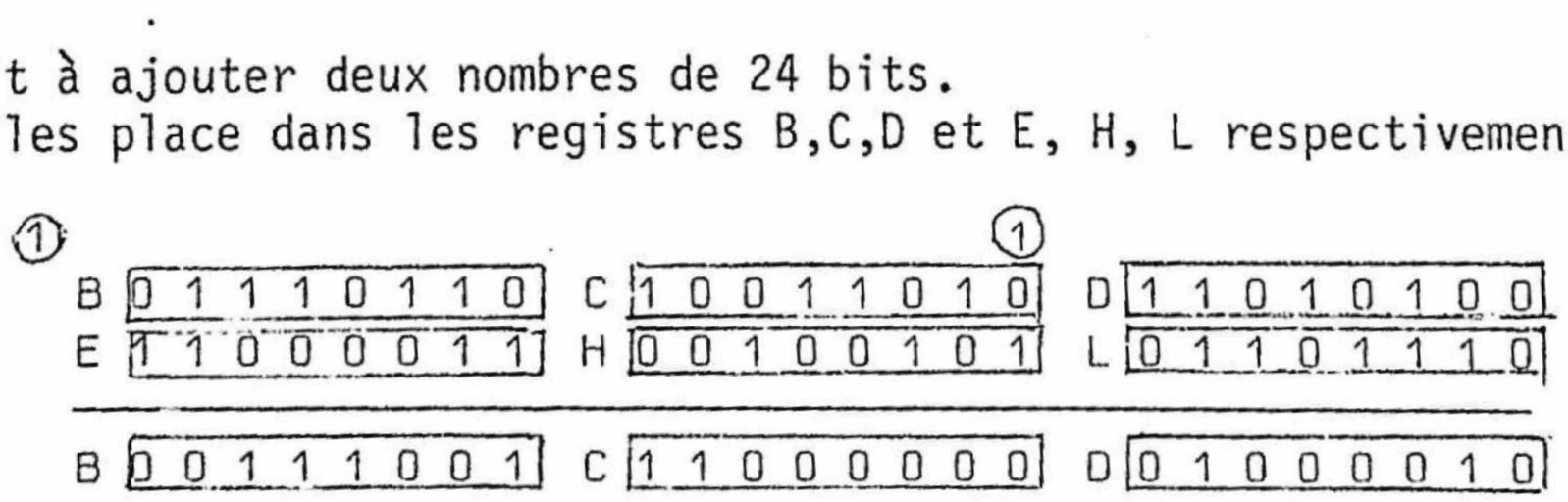
Lorsque les additions ne se font pas sur des nombres octaux, mais sur des nombres BCD */Binary Coded Decimal/*, il faut faire suivre ADD ou ADDC de DAA A */decimal Adjust/*, qui corrige le résultat binaire intermédiaire.

Les additions peuvent également être effectuées entre registres 16 bits et nous résumons toutes les additions entre registres en écrivant



Lorsqu'il faut faire des additions avec des nombres de plus de 8 ou 16 bits, on décompose en tranches.

EXEMPLES Soit à ajouter deux nombres de 24 bits.
On les place dans les registres B,C,D et E, H, L respectivement



Le plus simple serait de pouvoir écrire

```
ADD   D,L
ADDC  C,H
ADDC  B,E
```

qui tient compte du Carry qui peut se propager d'un groupe à l'autre.

Ces instructions n'étant pas disponibles, il faut compliquer le programme:

```
LOAD    A,D  }
ADD     A,L  }  ADD   D,C
LOAD    D,A  }
LOAD    A,C  }
ADDC    A,H  }  ADDC  C,H
LOAD    C,A  }
LOAD    A,B  }
ADDC    A,E  }  ADDC  B,E
LOAD    B,A  }
```

Pour additionner deux nombres décimaux contenus dans HL et DE, on ne peut utiliser l'instruction ADD HL,DE . Il faut passer par l'accumulateur A et faire le programme suivant:

```
DADD:  LOAD    A,L
       ADD     A,E
       DAA     A
       LOAD    L,A
       LOAD    A,H
       ADDC    A,D
       DAA     A
       LOAD    H,A
```

INSTRUCTIONS DE SOUSTRACTION

Les instructions de soustraction entre registres sont les suivantes:

4/7 | 220+ | **SUB A,r** |S,Z,C,V,H,N←1|

4/7 | 230+ | **SUBC A,r** |S,Z,C,V,H,N←1| Subtract and subtract carry

15 | 355 / 102+ | **SUBC HL,p** |S,Z,C,V,H,N←1|

4 | 47 | **DA A** |S,Z,C,P,H| Z80 : Valid after ADD (N=O) and SUB (N=1)

Les soustractions en BCD doivent être suivies de DA A.

L'instruction SUB HL,DE n'existe pas sur le Zilog Z8Ø. Souvent, il faut la remplacer par un groupe d'instructions que l'on peut appeler par un sous-programme

```
;---    CALL    RSUB     ;routine calculant  SUB HL,DE
                         ;cette routine modifie A
RSUB:   LOAD    A,L      ; le flag C est initialisé correctement
        SUB     A,E      ;soustrait les bytes de poids faible
        LOAD    L,A
        LOAD    A,H
        SUBC    A,D      ;soustrait les bytes de poids fort et la retenue
        LOAD    H,A
        RET
```

Remarque: il est plus simple d'utiliser d'utiliser les instructions:

```
OR      A,A
SUBC    HL,DE
```

Le calcul des compléments à 1 et à 2 est un cas particulier de soustraction, et se fait au moyen des instructions suivantes.

4	57	CPL	A	1's complement (invert)
		[H←1,N←1]		
8	355	NEG	A	Negate (2's complement)
	104	[S,Z,C,V,H,N←Ø]		Subtract Ø−A

INSTRUCTIONS D'INCREMENTATION ET DE DECREMENTATION

Ces opérations permettent d'ajouter ou de soustraire 1 au contenu des registres. Ces sont les suivantes:

				r	
4	4+	INC	r	7	A
11		[S,Z,V,H,N←Ø]		0	B
4	5+	DEC	r	1	C
11		[S,Z,V,H,N←1]		2	D
				3	E
				4	H
				5	L
				6	(HL)

11 states if (HL)

				p	
6	3+	INC	p	0	BC
				20	DE
6	13+	DEC	p	40	HL
				60	SP

Il faut toutefois remarquer que les instructions d'incrémentation ou de décrémentation de paires de registres ne modifient pas les flags.

Pour faire un décompteur 16 bits, dans BC par exemple, et tester son passage par zéro, on peut écrire

```
        DEC   BC
        LOAD  A,B
        OR    A,C    ;calcule le OU logique des 2 registres B et C
                     ;le résultat est nul seulement si les deux
                     ;registres, donc BC, est nul
```

L'instruction DA A */Decimal Adjust/* permet de faire des compteurs ou décompteurs décimaux. Par exemple, si BCD est un compteurs 6 digits, il faut effectuer les instructions suivantes chaque fois que l'on doit additionner 1.

```
COUNT:  INC     D      ;unités et dizaines
        LOAD    A,D
        DA      A
        LOAD    D,A    ;ne modifie pas les flags
        JUMP,NE FIN
        INC     C      ;centaines et milliers
        LOAD    A,C
        DA      A
        LOAD    C,A
        JUMP,NE FIN
        INC     B      ;dizaines et centaines de millers
        LOAD    A,B
        DA      A
        LOAD    B,A
FIN:    JUMP,EQ DEPCAP  ;dépassement de capacité
```

Ce même programme peut s'écrire

```
COUNT:  INC     D
        LOAD    A,D
        DA      A
        LOAD    D,A
        LOAD    A,# 0
        ADDC    A,C
        DA      A
        LOAD    C,A
        LOAD    A,# 0
        ADDC    A,B
        DA      A
        LOAD    B,A
        JUMP,CS DEPCAP                  ;dépassement de capacité
```

Ce nouveau programme est plus court et plus rapide que le précédent (sauf s'il n'y a pas de report sur les centaines).

Il est aussi possible d'ajouter ou soustraire un nombre 8 bits au contenu du registre A (adressage immédiat)

7	100+Op	Op	A,#n
	n	[flags modified as above]	

Op	206	ADD
	216	ADDC
	226	SUB
	236	SUBC

Par exemple `ADD A,# 2` est équivalent à

```
INC  A
INC  A
```

sauf que le carry est modifié lorsque l'on dépasse 377.

5.7 OPERATIONS LOGIQUES

Quatre opérations logiques agissent sur les contenus des registres 8 bits du Z80: le "ET logique" AND, le "OU logique" OR, le "OU exclusif" XOR et la comparaison COMP. Le codage de ces instructions est le suivant:

4/7	240+	**AND** A,r	|S,Z,C←Ø,P,H,N←Ø|
4/7	260+	**OR** A,r	|S,Z,C←Ø,P,H,N←Ø|
4/7	250+	**XOR** A,r	|S,Z,C←Ø,P,H,N←Ø|
4/7	270+	**COMP** A,r	|S,Z,C,V,H,N←1| Compare

7 states if (HL)

r	
7	A
0	B
1	C
2	D
3	E
4	H
5	L
6	(HL)

7 | 100+Op | Op A,#n
n

[flags modified as above]

Op	
246	AND
266	OR
256	XOR
276	COMP

L'instruction AND est souvent utilisée pour masquer, enlever, c'est-à-dire mettre à zéro une partie d'un mot.Par exemple, si seuls les 4 bits de poids faible d'un nombre dans A doivent être additionnés à un autre nombre dans B, on définira un masque égal à $00001111_2 = 17_8$ et on écrira

```
MASK=   17
...
AND     A,#MASK         A     10101010
ADD     A,B             MASK  00001111
                              --------
                              00001010   à ajouter à B
```

Le AND est une sorte de passoire qui ne laisse des bits à l'état 1 que là où se trouvent des 1 dans le second opérande. Souvent on ne laisse passer qu'un seul bit et le flag EQ (equal) permet de savoir si ce bit est égal à 0 ou 1.

L'instruction OR permet de faire le OU de deux mots binaires. Elle est souvent utilisée pour forcer à 1 certains bits, et s'appelle BIS */Bit set/* sur certains processeurs.(PDP 11,...).

```
OR    A,#SBIT         A     01011010
                      SBIT  11000000
                            --------
                            11011010
```

L'instruction XOR est utile dans certaines opérations de contrôle (elle inverse les bits sélectionnés par un masque), et pour remettre à zéro les registres.

```
XOR   A,A             A     01100101
                      A     01100101
                            --------
                            00000000
```

Lors de l'emploi des instructions logiques, il est utile de pouvoir modifier le carry par les deux instructions

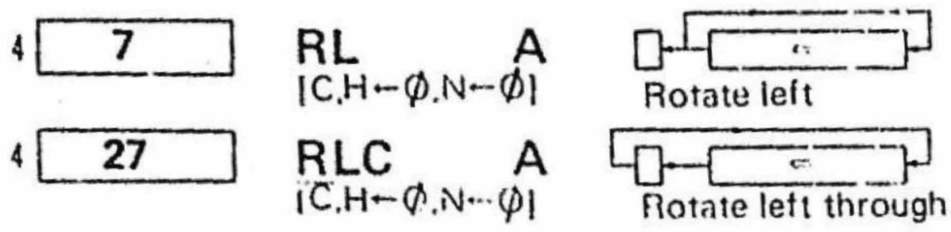
4	67	**SETC** Set carry |C←1,H←Ø,N←Ø|
4	77	**CPLC** Complement carry |C←$\overline{C}$,H!,N←Ø|

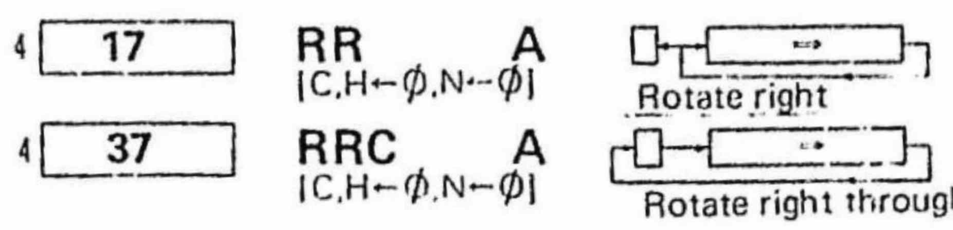
Clear Carry with e.g. OR A,A instruction

L'instruction *CLRC /Clear Carry/* n'existe pas car les instructions OR A,A et AND A,A ont cet effet, en agissant toutefois simultanément sur d'autres flags.

5.8 INSTRUCTIONS DE DECALAGE

Quatre instructions d'un byte permettent de décaler à gauche ou à droite le contenu du registre A, en passant par le Carry ou non. Ce sont:



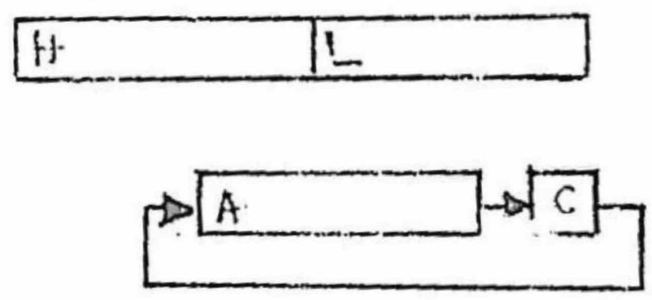


Rappelons que ces instructions sont utiles, par exemple, pour diviser ou multiplier par 2.

EXEMPLE: Ecrire un programme décalant HL à droite (divise par 2).

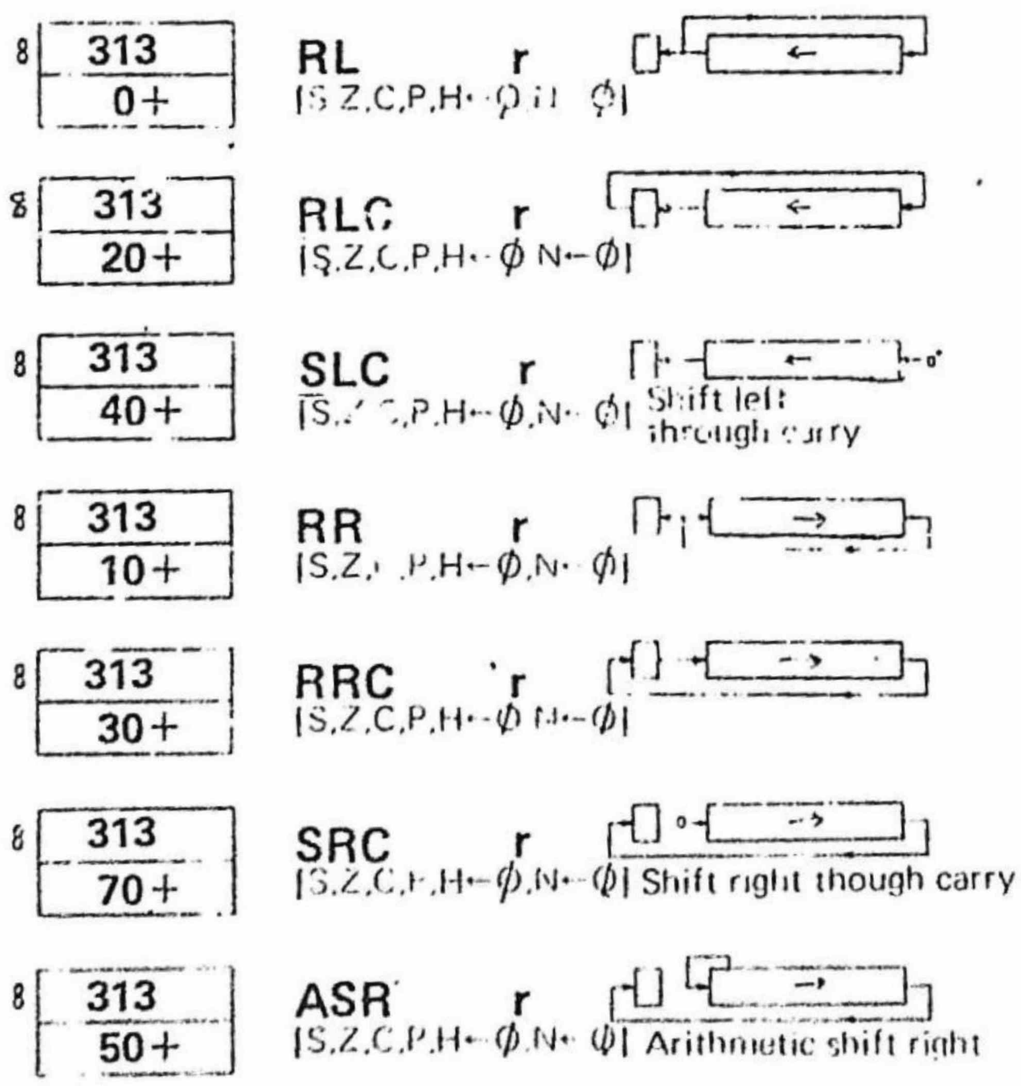
Le programme transfère successivement H et L dans le registre A, dans lequel le décalage est effectué. Le bit de droite de H est sauvé temporairement dans le Carry au cours de l'opération.

```
LOAD    A,H
RRC     A
LOAD    H,A
LOAD    A,L
RRC     A
LOAD    L,A
```



Il faut noter qu'avec ces instructions, la valeur préalable du Carry C se trouve dans le bit de poids fort de H.
En général, au cours d'un décalage à droite, on impose que le bit introduit dans le poids fort soit Ø. L'instruction OR A,A qui ne modifie pas A, mais force la carry à Ø, doit précéder les 6 instructions ci-dessus.

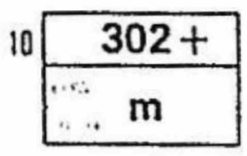
Dans le Z80 des instructions de décalage ont été ajoutées pour tous les registres Ce sont:



r	
7	A
0	B
1	C
2	D
3	F
4	H
5	L

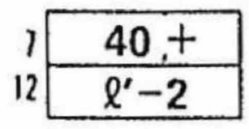
5.9 INSTRUCTIONS DE SAUTS CONDITIONNELS

En complément des instructions JUMP m, JUMP .+ℓ' et DECJ,NE B,.+ℓ' étudiées précédemment, on trouve les instructions de saut conditionnel suivantes



JUMP,t m
Jump if test t true

t		
0	NE	Non equal [Z=0]
	ZC	Zero bit clear
10	EQ	Equal [Z=1]
	ZS	Zero bit set
20	HS	Higher or same
	CC	Carry clear
30	LO	Lower
	CS	Carry set
60	PL	Plus (positive)
	SC	Sign bit clear
70	MI	Minus (negative)
	SS	Sign bit set
40	VC	Overflow bit clear
	PO	Parity odd [P=0]
50	VS	Overflow bit set
	PE	Parity even [P=1]



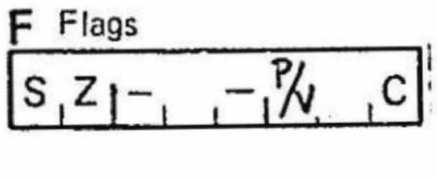
JUMP,t' .+ℓ'

(relatif)

t'		
0	NE	Non equal [Z=0]
	ZC	Zero bit clear
10	EQ	Equal [Z=1]
	ZS	Zero bit set
20	HS	Higher or same
	CC	Carry clear
30	LO	Lower
	CS	Carry set

ℓ' 8-bit displacement in jumps relative to first byte of the instruction
0 to 201 positive, 377 (−1) to 202 (−176) negative

F Flags



S Sign bit
Z Zero bit
P Even parity bit
V Overflow bit
C Carry bit

Les instructions de saut conditionnel donnent toute la puissance à la programmation. Selon l'état des flags, le saut s'effectue ou non (si la condition n'est pas satisfaite, on continue à l'instruction suivante), avec les possibilités suivantes:

JUMP,NE ,ZC — saut si le flag Z n'est pas nul */non equal/*, c'est à dire tant que le contenu du registre modifié précédemment par une instruction arithmétique ou logique n'est pas nul.

JUMP,EQ ,ZS — saut si le flag Z est nul */equal/*, c'est à dire si le contenu du registre modifié dans l'instruction précédente est nul.

JUMP,CC ,HS — saut si le bit de poids faible du registre F (flag C) est nul */carry clear/*

JUMP,CS ,LO — saut si le bit de poids faible du registre F est 1 */carry set/*

EXEMPLES Le programme suivant calcule la somme des n premiers nombres entiers.
Le nombre n est dans le registre A, la somme chargée dans A également.
Le programme signale s'il y a dépassement de capacité

```
SOM:        LOAD    B,A         ;B contient alors n
            LOAD    A,#Ø        ;A contient la somme partielle

SO2:        ADD     A,B
            JUMP,CS DEPCAP

            DEC     B           ;décompter
            JUMP,NE SO2         ;test de fin      } DECJ,NE B,SO2

            TRAP
DEPCAP:     TRAP                ;A contient la somme
```

Le programme suivant additionne le contenu de HL avec un nombre positif donné (égal à 12345, par exemple).
L'opération s'effectue en deux temps, en passant par l'accumulateur.
Le nombre 12345 doit être fractionné en deux bytes au moment de l'addition 012345 = 024 - 345.

```
SOMME:      LOAD    A,L
            ADD     A,#NBRE&377 ;prend les 8 bits de poids faible
                  345
            LOAD    L,A
            LOAD    A,H
            ADDC    A,#NBRE/400 ;prend les 8 bits de poids fort
                   24
            LOAD    H,A
            JUMP,CS DEPCAP
```

La multiplication de deux nombres peut se faire d'innombrables façons, selon l'algorithme utilisé, la précision voulue et les registres utilisés. L'algorithme le plus simple consiste à remplacer la multiplication par une suite d'additions (valable pour des nombres entiers seulement).

Par exemple, pour multiplier deux nombres 8 bits dans A et B, il y a avantage à faire apparaître le produit dans HL et à utiliser l'instruction ADD HL,DE. Le deuxième nombre est utilisé comme compteur de cycles et il ne peut pas y avoir de dépassement de capacité.

```
MULT:   LOAD    E,A
        LOAD    D,#0        ;DE contient le premier nombre
        LOAD    HL,#0       ;HL accumule les sommes partielles

MUL1:   ADD     HL,DE
        DEC     B
        JUMP,NE MUL1        ;recommence tant qu'il n'y a
                            ;pas eu B additions
        TRAP
```

La routine suivante multiplie un nombre 8 bits par un nombre 16 bits et génère un résultat 24 bits, donc sans dépassement de capacité. Cet algorithme est l'algorithme rapide d'additions et de décalages, les opérations élémentaires se faisant par mots de 16 bits entre HL et DE, avec sauvetage des bits de dépassement de capacité dans A.

```
MUL:    LOAD    B,#8.       ;compteur des 8.=10 cycles de décalage
        EX      DE,HL       ;échange pour avoir le 2e nombre dans DE
        LOAD    HL,#0       ;initialisation du produit partiel
MUL1:   ADD     HL,HL       ;décalage de HL
        RLC     A           ;overflow dans A et transfert du bit de poids
                            ;fort de A dans le Carry
        JUMP,CC MUL2        ;saut si le bit de poids fort est nul
        ADD     HL,DE       ;addition
        ADDC    A,#0        ;transfert de l'overflow éventuel dans A
MUL2:   DECJ,NE B,MUL1      ;si pas terminé, recommence un cycle
        TRAP
```

Il existe d'innombrables algorithmes de division, dont l'algorithme par soustractions successives, qui est laissé comme exercice.

L'algorithme ci-dessous utilise des comparaisons et divise un mot de 16 bits par un mot de 8 bits, avec un résultat de 16 bits et un reste de 8 bits.

```
DIV:    LOAD    C,#16.      ;compteur de cycles
        XOR     A,A         ;clear de A
DIV1:   ADD     HL,HL       ;décale HL et
        RLC     A           ;transfère le poids fort dans A
        COMP    A,B         ;compare: peut-on soustraire B ?
        JUMP,CS DIV2
        INC     L           ;oui placer 1 dans le poids faible
        SUB     A,B         ;effectuer la soustraction
DIV2:   DEC     C           ;terminé ?
        JUMP,NE DIV1        ;sinon, recommence un cycle
        TRAP                ;fin
```

Voici encore un exemple de programme, qui effectue la conversion binaire-décimale par comptage-décomptage.

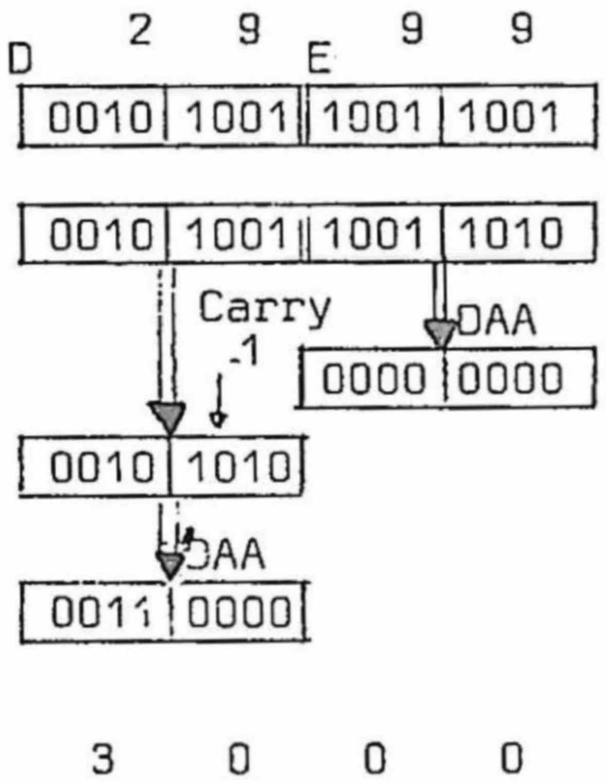
HL : Nombre binaire donné
DE : Nombre décimal calculé

Algorithme : décompter dans HL en binaire, avec test de fin;
compter dans DE en décimal.

```
CONV:     DEC      HL
          INC      DE
          LOAD     A,E
          DAA      A
          LOAD     E,A
          LOAD     A,D
          ADDC     A,#Ø
          DAA      A
          LOAD     D,A
          JUMP,CS  ERROR
          LOAD     A,H
          OR       A,L
          JUMP,NE  CONV
ERROR:    TRAP
```

Correction de DE

Exemple :



Comme l'instruction OR A,L met le carry à Ø, on est sûr que le carry final est à Ø si le résultat est correct, et à 1 si le résultat est faux (nombre binaire donné ≥ 23420)

Les quatre autres catégories de sauts conditionnels n'existent que pour des adresses absolues.

JUMP,SC saut si le résultat précédent est positif */sign clear/*
,PL */plus/*

JUMP,SS saut si le résultat précédent est négatif */sign set/*
,MI */minus/*

JUMP,PO saut si la parité du résultat précédent est impaire */parity odd/*
,VC saut s'il n'y a pas eu dépassement de capacité sur les nombres arithmétiques */overflow bit clear/*

JUMP,PE saut si la parité du résultat précédent est paire */parity even/*
,VS saut s'il y a eu dépassement de capacité sur les nombres arithmétiques */overflow bit set/*.

EXEMPLE: pour convertir un nombre arithmétique (en complément à 2 s'il est négatif) en une valeur absolue (dans A) et la valeur 0 ou 1 selon le signe (dans B), on peut écrire:

```
CONV:    LOAD     B,#Ø
         OR       A,A        ;charge les flags selon A
         JUMP,PL  CONV2
         NEG      A          ;complément à 2
         INC      B          ;charge le signe
CONV2:
```

Comme autre exemple, considérons un programme qui lit une ligne genre télex. Toutes les 20 ms, un "un" ou un "zéro" est transmis sur la ligne. Des "zéro" remplissent les silences, et un premier "un" */start bit/* marque la fin d'un silence et le début d'un mot de 5 bits correspondant à un caractère. Chaque mot est suivi d'un silence (zéro) de 1 bit au moins.

Exemple de message:

0 0 0 0 0 0 [1] 1 0 1 0 1 0 [1] 1 1 1 1 1 0 0 0 0 0 [1] 0 0 1 0 0 0 [1] 0 0 0 0 1 0 0

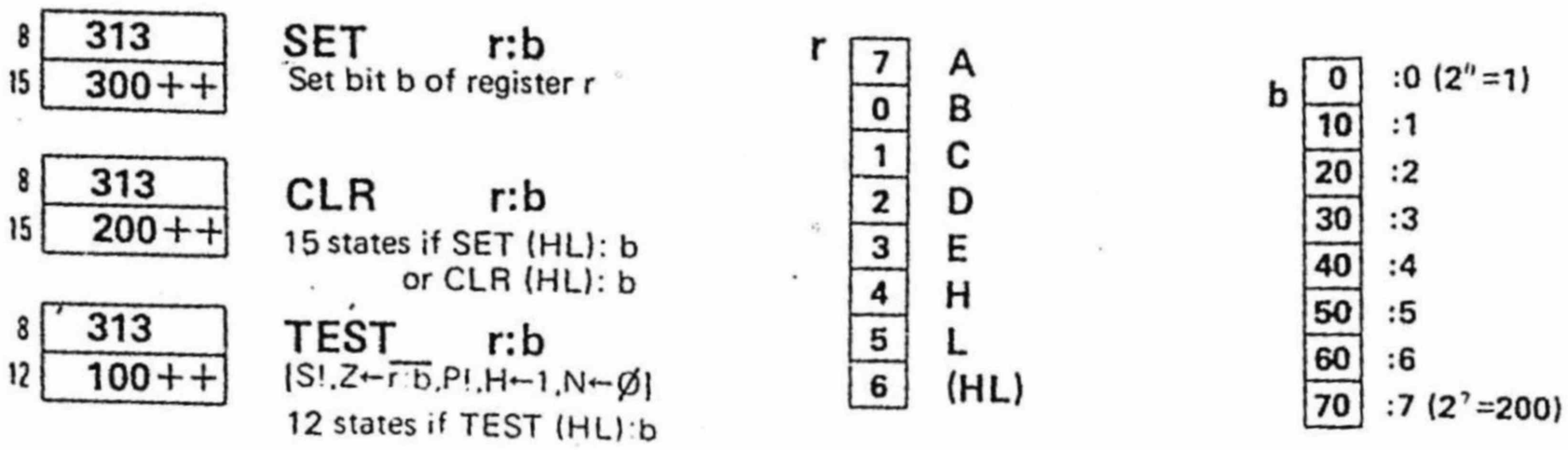
1er caractère 2e car. 3e car. 4e car.

Une routine BIT, non écrite ici car elle dépend de l'interface avec le télex, lit l'état de la ligne et met le CARRY à un si la ligne est à "un" et à zéro si la ligne est à "zéro". Une routine CARACTERE forme un mot de 5 bits une fois que le premier bit (start) a été reconnu. Le programme principal donné en exemple passe par-dessus les silences et attend un caractère spécial de début, par exemple 10101 = 25 octal.

```
.TITLE TELEX    ;attend un caractère de début
LONG= 5         ;nombre de bits par caractère
CARDEB=25       ;caractère de début de message

;---   sous programmes

BIT:   ---      ;selon l'interface
                ;cette routine ne peut modifier que A et B
       RET      ;retour avec CS (carry set) si le bit vaut 1

CARACTERE:
       LOAD     C,# Ø     ;registre caractère
       LOAD     D,# LONG  ;compteur de bits par caractère
CAR2:  CALL     BIT
       RLC      C         ;décalage de Carry dans C
       DEC      D
       JUMP,NE  CAR2
       RET               ;retour avec le caractère dans C

;~--   PROGRAMME

TELEX:
       CALL     BIT
       JUMP,CC  TELEX     ;saut si la retenue est nulle

;le bit vaut 1, c'est le début d'un caractère

       CALL     CARACTERE
       CALL     BIT
       JUMP,CS  ERROR     ;erreur si le bit lu vaut 1
       LOAD     A,C
       COMP     A,# CARDEB
       JUMP,NE  TELEX     ;retour pour attendre le caractère suivant

;suite du programme lorsque le caractère de début est trouvé
       ---

;action en cas d'erreur (pas de silence à la fin d'un caractère)

ERROR: ---
```

5.10 ACTION SUR UN BIT DETERMINE D'UN REGISTRE

Il est possible de tester, mettre à 0 ou à 1 le bit b d'un registre r par les opérations suivantes



Par exemple, si lorsque le bit de poids fort (bit 2^7) d'un nombre est à 1, on veut forcer le dernier bit à zéro et l'avant dernier à 1, on peut écrire

```
        TEST    A:7
        JUMP,EQ NEXT  ;si le bit est Ø, on ne change rien
        SET     A:1
        CLR     A:Ø
NEXT:   ---
```

Le bit $2^2 = 4$ du registre de Flag F peut avoir deux significations selon l'instructior précédente. Dans certains cas (AND, OR, XOR, TEST), c'est la parité du résultat (P vaut 1 si le résultat est pair); pour les autres instructions modifiant les flags c'est l'overflow"V", c'est-à-dire le dépassement de capacité dans une opération arithmétique sur des nombres en complément à 2.

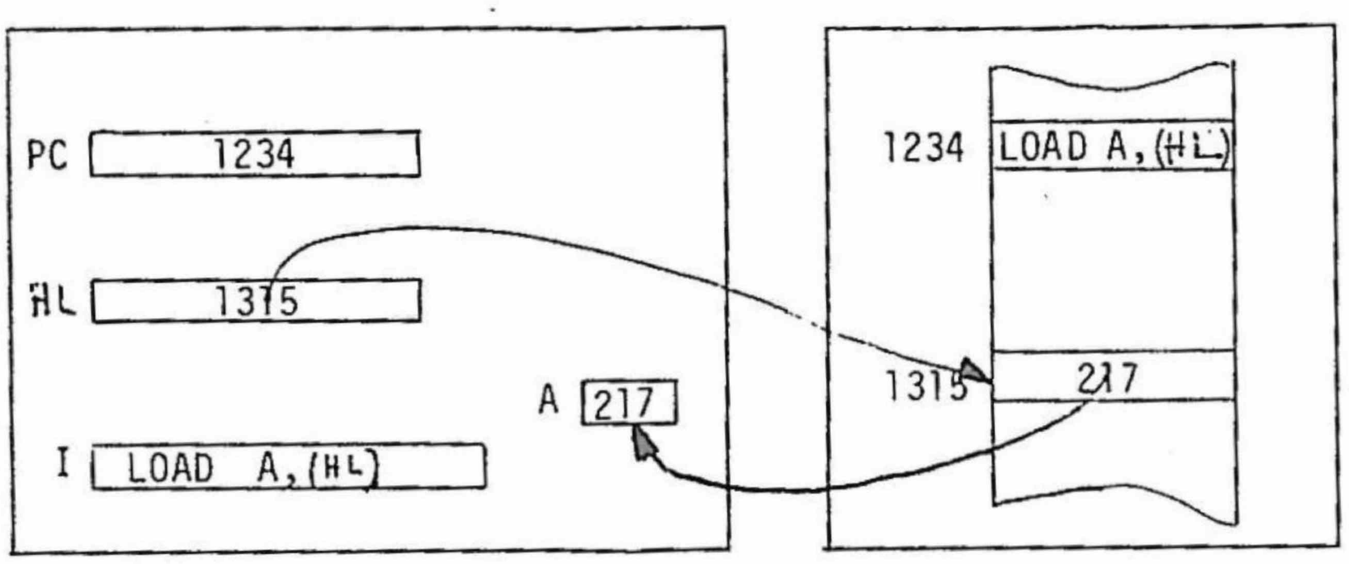
Par exemple, si l'on additionne 123 (positif) et 76 (positif), le résultat de l'additionneur n'est pas valable et le flag V passe à 1.

Ainsi, lorsque l'on travaille avec des nombres positifs uniquement (nombres logiques) JUMP,CS */jump if carry set/* signale un dépassement de capacité.

Avec les nombres arithmétiques, c'est JUMP,VS */jump if oVerflow bit Set/* qui signale un dépassement de capacité.

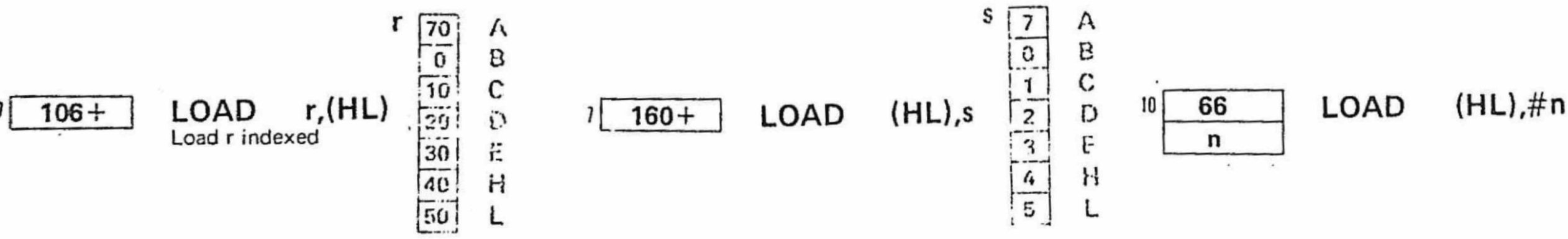
5.11 ADRESSAGE INDEXE

Quatre modes d'adressage, c'est-à-dire façons de spécifier où se trouve l'information considérée, ont été rencontrés dans les précédentes parties. Dans le mode registre, le numéro du registre qui contient l'information (70 pour A, 0 pour B, ...) est spécifié dans le code même de l'instruction. Dans le mode immédiat, l'information se trouve dans le 2e byte de l'instruction (où dans les 2e et 3e). Dans le mode absolu, l'adresse complète de la position mémoire contenant l'information figure dans l'instruction. Dans le mode relatif, c'est la différence entre l'adresse courante (adresse contenue dans le compteur d'adresses PC) et l'adresse de la position mémoire contenant l'information qui est donnée dans l'instruction.
Dans l'adressage indexé, l'adresse de l'information se trouve dans le registre HL et l'on écrit LOAD A,(HL) pour dire que le contenu de la position mémoire dont l'adresse est dans HL est transféré dans A. La parenthèse autour de HL indique que le contenu de HL est une adresse; LOAD A,HL voudrait dire que le contenu de HL est transféré dans A, sans passer par une référence mémoire (cette instruction est par ailleurs absurde car A a 8 bits et HL 16 bits).



Adressage indexé

Les instructions disponibles sont les suivantes:



Exemple:

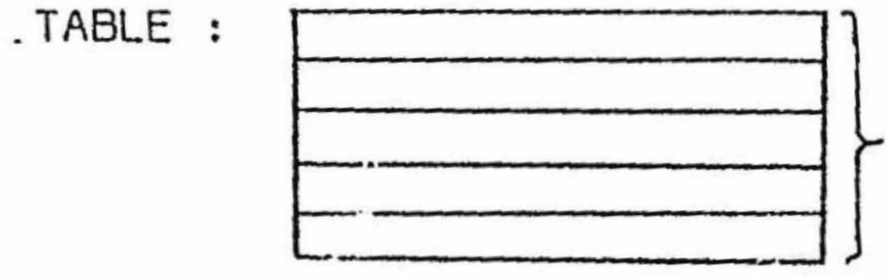
Le programme suivant remplit toute la mémoire de caractères 40 (espace en code ASCII). Le principe utilisé est d'écrire dans les positions mémoire successives en relisant chaque fois. Si la comparaison montre une différence, c'est que l'adresse ne correspond plus à de la mémoire (ou que la mémoire est défectueuse !). En général une mémoire inexistante apparaît comme contenant uniquement des 000 ou des 377.

```
INIM:   LOAD     A,# SPACE
        LOAD     HL,DEBMEM
INI2:   LOAD     (HL),A
        LOAD     B,A          ;sauve A dans B pour comparaison ultérieure
        LOAD     A,(HL)
        INC      HL
        COMP     A,B
        JUMP,EQ  INI2         ;retourne initialiser la position suivante
        TRAP                  ;si égalité, sinon retourne au moniteur
```

Le sous-programme suivant est équivalent, car INC HL ne modifie pas les flags.

```
INI2:   LOAD     (HL),A
        COMP     A,(HL)
        INC      HL
        JUMP,EQ  INI2
```

Etudions en détail un dernier programme, qui additionne n nombres dans une table.



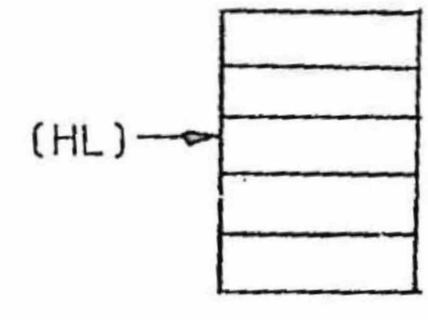
n nombres de 8 bits consécutifs dans une zone mémoire à partir de l'adresse TABLE

résultat 8 bits dans le mot d'adresse RESULT

mot mémoire indiquant un dépassement de capacité

si (DEP) = 0 résultat correct
si (DEP) = 1 résultat incorrect

Principe : L'adresse TABLE est placée dans le registre HL qui pointe successivement chacun des nombres de la table lors des additions.

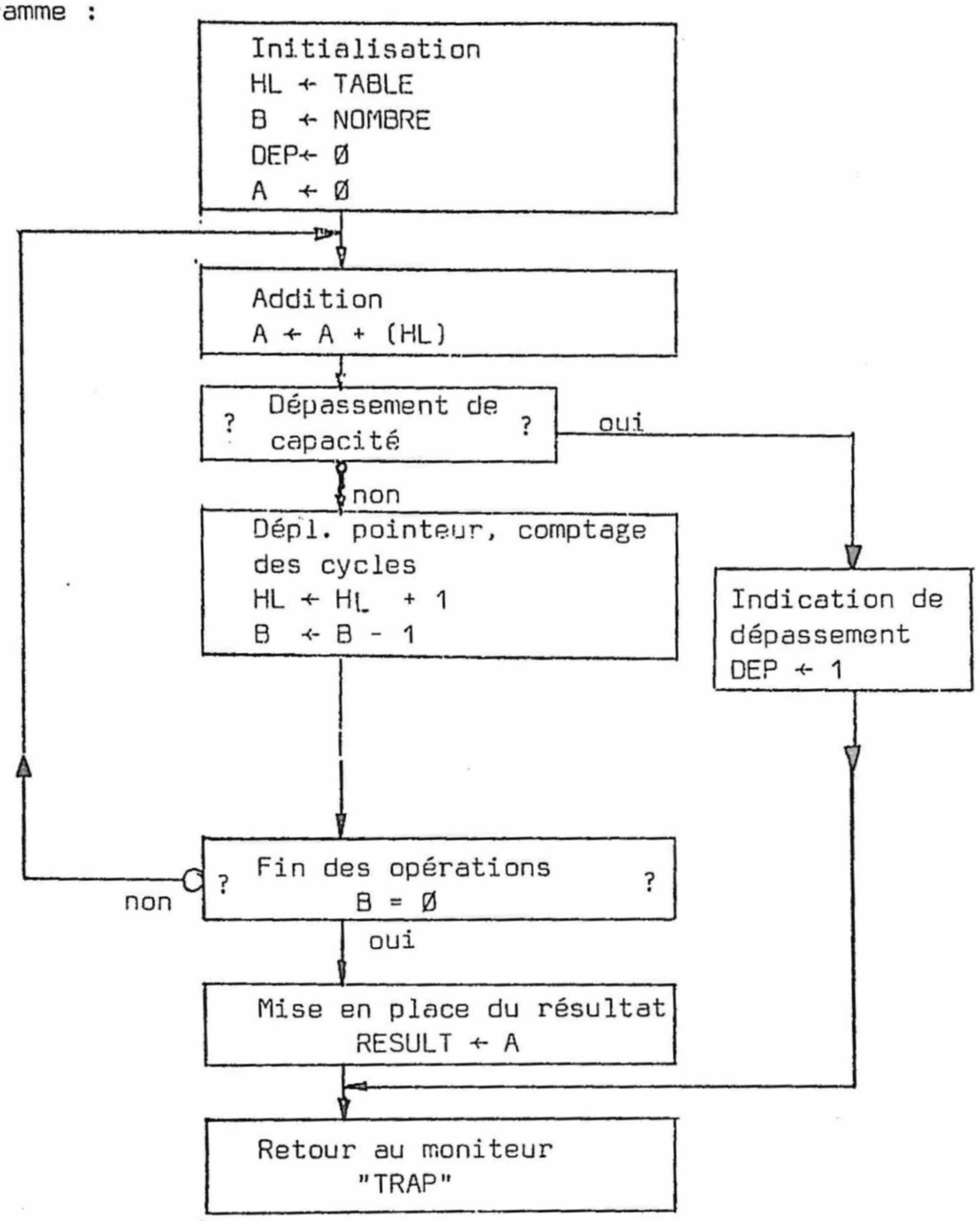



```
Coeur du programme :   ADD      A,(HL)
                       JUMP,CS  ERREUR   arrêt dès dépassement de capacité
                       INC      HL       déplace le pointeur
```

Il faut initialiser au début du programme le pointeur, un compteur de cycle égal au nombre de nombres dans la table, le contenu initial de l'accumulateur et le mot de contrôle du dépassement de capacité.

```
NOMBRE = 6
ADDTABLE: LOAD     HL,# TABLE
          LOAD     B,# NOMBRE
          LOAD     A,# 0
          LOAD     DEP,A
ADDT2:    ADD      A,(HL)
          JUMP,CS  ERROR
          INC      HL
          DEC      B
          JUMP,NE  ADDT2
          LOAD     RESULT,A
ERROR:    LOAD     A,# 1
          LOAD     DEP,A
          TRAP

TABLE:    .BYTE    3,4,5,10,12,14

RESULT:   .BLKB    1
DEP:      .BLKB    1

.END
```

Organigramme :



5.12 AUTRES OPERATIONS

NON OPERATION

L'instruction NOP ne produit aucun effet si ce n'est une attente de quelques microsecondes. Elle permet d'"effacer" des instructions superflues lors de la mise au point d'un programme ou de prévoir un espace pour un saut ou une trappe

POUR LES AUTRES OPERATIONS:

CONSULTER LA FEUILLE DE MNEMONICS CALM - Z80

5.13 EXEMPLES DE PROGRAMMES

SIMULATEUR DE TELETYPE SIMPLE

```
                                .TITLE  TTY1.SR
                                .PROC   Z80
                                .REF    SM6
100000                          .LOC    100000

                        ; Simulateur de TTY ultra-simplifié


                    TTY:
100000  016 024             LOAD    C,#NLI
100002  347 126             .W      ?IDIS           ; init écran
                    TTY1:
100004  347 015             .W      ?IFCAR          ; un car du clavier ?
100005  107                 LOAD    B,A             ; B <-- ce caractère
100007  332 027 200         JUMP,CS TTY4            ; non => lit usart
                    TTY2:
100012  170                 LOAD    A,B
100013  347 046             .W      ?IFWPP          ; essaye d'envoyer le car tout
100015  060 365             JUMP,CC TTY1            ; en faisant l'écho de ce qui
100017  347 044             .W      ?IFRPR          ; peut être lu par l'usart
100021  070 357             JUMP,CS TTY2            ; jusqu'à ce que l'envoi soit
100023  347 000             .W      ?DICAR          ; fait.
100025  030 363             JUMP    TTY2

                    TTY4:
100027  347 044             .W      ?IFRPR          ; un car de l'usart ?
100031  070 351             JUMP,CS TTY1            ; non => lit clavier
100033  347 000             .W      ?DICAR
100035  030 345             JUMP    TTY1


                100000      .END    TTY
```

```
80/10/01 17:20:15  CROSS REFERENCE MAP                          02-01


000020 references
Source file 000025 usefull lines long
Binary file 000037 bytes long
Assembly time: 0008 seconds     0187 lines/min

?DICAR= 000347  01-0026 01-0032
?IDIS = 053347  01-0015
?IFCAR= 006747  01-0017
?IFRPR= 022347  01-0024 01-0030
?IFWPP= 023347  01-0022
ILI   = 002024  01-0014
TTY   : 100000  01-0013 01-0037
TTY1  : 100004  01-0016 01-0023 01-0031 01-0033
TTY2  : 100012  01-0020 01-0025 01-0027
TTY4  : 100027  01-0019 01-0029
```

SIMULATEUR DE TELETYPE AVEC OU SANS ECHO

```
                        .TITLE  TTY2.SR ; dir. JDN:SHAKY  et  PERIPH                    80/10/01 17:22:10  CROSS REFERENCE MAP

                        .PROC   Z80
                        .REF    SHG

100000                  .LOC    100000


                    ; Simulateur de TTY avec ou sans echo.
                    ; --------------------------------------


                    ;--------\
                    ; TTY     >
                    ;========/

                    ; Programme principal. Commence par demander si on
                    ; veut ou pas l'echo.

                    TTY:
100000  016 024             LOAD    C,#LINES
100002  347 126             .W      ?IDIS
100004  347 123 204 200     .W      ?TEXT.ECHO?     ; Affiche texte
100010  347 001             .W      ?GETCAR         ; Attend la reponse
100012  347 000             .W      ?DICAR
100014  346 003             AND     A,#3            ; Masque
100016  117                 LOAD    C,A
100017  347 030 347 043     .W      ?CALPHA,?RETURN
100021  076 040             LOAD    A,#SPACE
100023  315 155 200         CALL    PDICAR
                    TTY1:
100030  347 015             .W      ?IFCAR          ; Clavier ?
100032  107                 LOAD    B,A             ; Sauve ev. car dans B
100033  332 117 200         JUMP,CS TTY4            ; Go to TTY4 si pas clavier
100036  376 037             COMP    A,#DEFINE
100040  312 142 200         JUMP,EQ HOME
100043  313 111             TEST    C:1             ; Echo ?
100045  302 053 200         JUMP,NE TTY2            ; Non, go to TTY2
100050  315 155 200         CALL    PDICAR          ; Oui, carac sur l'ecran
                    TTY2:
100053  170                 LOAD    A,B             ; Essaye d'envoyer le carac
100054  347 016             .W      ?IFWPP
100056  322 104 200         JUMP,CC TTY3            ; Si c'etait possible, go to TTY3
100061  347 044             .W      ?IFRPR          ; Si on ne peut pas envoyer, il
100063  070 366             JUMP,CS TTY2            ; faut attendre en lisant ce qui
100065  365                 PUSH    AF
100066  076 005             LOAD    A,#INV
100070  347 000             .W      ?DICAR
100072  361                 POP     AF
100073  315 155 200         CALL    PDICAR          ; pouvait ev. arriver.
100076  076 004             LOAD    A,#NORM
100100  347 000             .W      ?DICAR
100102  030 347             JUMP    TTY2

                    TTY3:
100104  313 111             TEST    C:1
100106  312 117 200         JUMP,EQ TTY4
100111  006 012             LOAD    B,#LF           ; Si on vient d'envoyer un CR,
100113  376 015             COMP    A,#CR           ; faut aussi envoyer un LF.
100115  050 334             JUMP,EQ TTY2

                    TTY4:
100117  347 044             .W      ?IFRPR          ; Un carac arrive ?
100121  070 305             JUMP,CS TTY1            ; Non, go to TTY1
100123  365                 PUSH    AF
100124  076 005             LOAD    A,#INV
100126  347 000             .W      ?DICAR
100130  361                 POP     AF
100131  315 155 200         CALL    PDICAR          ; Oui, affiche carac sur l'ecran
100134  076 004             LOAD    A,#NORM
100136  347 000             .W      ?DICAR
100140  030 265             JUMP    TTY1

                    HOME:
100142  347 016             .W      ?GETFCN
100144  376 120             COMP    A,#KILL+CURSOR
100146  040 260             JUMP,NE TTY1
100150  357 032             .W      ?RTN
100152  303 000 000         JUMP    0


                    ;--------\
                    ; PDICAR >
                    ;========/

                    ; Equivalent a ?DICAR avec en plus l'affichage du pointeur.

                    ; in    A       caractère à afficher
                    ; out   -
                    ; mod   AF,DE,HL

                    PDICAR:
100155  127                 LOAD    D,A
100156  347 041             .W      ?GETCURSOR      ; HL := position du curseur
100160  076 040             LOAD    A,#SPACE        ; clear curseur
100162  347 000             .W      ?DICAR
100164  347 040             .W      ?SETCURSOR      ; position du curseur := HL
100166  172                 LOAD    A,D
100167  346 177             AND     A,#177          ; suppr la parité
100171  347 000             .W      ?DICAR
100173  347 041             .W      ?GETCURSOR      ; HL := position du curseur
100175  076 240             LOAD    A,#SPACE+200    ; Set curseur
100177  347 000             .W      ?DICAR
100201  347 040             .W      ?SETCURSOR      ; position du curseur := HL
100203  311                 RET


100204  124 145 154 145 ECHO7:  .ASCII  "Teletype simulator, type<CR><CR>"
100236  011 060 011 156 ECHO:   .ASCII  "       0       no echo, LF added to CR<CR>"
100271  011 061 011 156 ECHO1:  .ASCII  "       1       no echo (NOVA)<CR>"
100313  011 062 011 145 ECHO2:  .ASCII  "       2       echo, LF added to CR (PDP)<CR>"
100351  011 063 011 145 ECHO3:  .ASCIZ  "       3       echo (CDC)<TAB>"


100000                  .END    TTY

000076 references
Source file 000031 usefull lines long
Binary file 000370 bytes long
Assembly time: 0013 seconds     0373 lines/min

?CALPH= 014347  01-0030
?DICAR= 000347  01-0027 01-0050 01-0054 01-0069 01-0073 01-0093 01-0102 01-0103
?GETCA= 000747  01-0026
?GETCU= 020747  01-0056 01-0103
?GETFO= 007347  01-0077
?IDIS = 053347  01-0024
?IFCAR= 006747  01-0034
?IFRPR= 022347  01-0046 01-0065
?IFWPP= 023347  01-0044
?RETUR= 021747  01-0030
?RTN  = 015357  01-0080
?SETCU= 020347  01-0099 01-0106
?TEXT = 051747  01-0025
CR    = 000015  01-0061 01-0110 01-0111 01-0112 01-0113
CURSOR= 000100  01-0078
DEFINE= 000037  01-0037
ECHO  : 100236  01-0111
ECHO1 : 100271  01-0112
ECHO2 : 100313  01-0113
ECHO3 : 100351  01-0114
ECHO7 : 100204  01-0025 01-0110
HOME  : 100142  01-0039 01-0076
INV   = 000005  01-0049 01-0068
KILL  = 000020  01-0078
LF    = 000012  01-0060
LINES = 000024  01-0023
NORM  = 000004  01-0353 01-0072
PDICAR  100155  01-0032 01-0041 01-0052 01-0071 01-0094
SPACE = 000040  01-0031 01-0097 01-0104
TAB   = 000011  01-0114
TTY   : 100000  01-0022 01-0118
TTY1  : 100030  01-0033 01-0066 01-0074 01-0079
TTY2  : 100053  01-0040 01-0042 01-0047 01-0055 01-0062
TTY3  : 100104  01-0045 01-0057
TTY4  : 100117  01-0036 01-0059 01-0064
```

```
                .TITLE  SORT_ALLIST

                .PROC   Z80
                .REF    SMG

                .LOC    100000


START:
                LOAD    C,#NLI
                .U      ?IDIS
START1:
                LOAD    HL,#0
                .U      ?SETCURS
                .U      ?TEXT,TEXT,?GETLINE
                LOAD    D,#1
PASS:
                LOAD    A,D             ;INITIAL VALUE OF A IS 1
                OR      A,A             ;LOAD ZERO FLAG ACCORDING A
                JUMP,EQ START1          ;FINISHED AFTER 256 PASS
                LOAD    D,#0            ;PREPARE FIRST COMPARIZON
                LOAD    HL,#ALPHA+NOBR
                LOAD    BC,#ENDALPHA-ALPHA-NOBR
COMPA:
                INC     BC
                LOAD    A,B
                OR      A,C
                JUMP,EQ PASS            ;GO TO NEXT-PASS
                LOAD    A,(HL)          ;ARRAY VALUE IN A
                INC     HL
                COMP    A,(HL)          ;COMPARE TO NEXT VALUE
                JUMP,CS COMPA           ;RETURN IF A >=
                JUMP,EQ COMPA
                LOAD    E,(HL)          ;EXCHANGE IF SMALLER
                DEC     HL              ;
                LOAD    (HL),E          ;
                INC     HL              ;
                LOAD    (HL),A          ;
                LOAD    D,#1            ;NOT THE LAST PASS
                JUMP    COMPA           ;GO COMPARE THE NEXT PAIR


TEXT:   .ASCIZ  /Type a text .....  <BELL><CR>/

                .END    START
```

CHRONOMETRE


```
                        .TITLE  CHRONO.SR       JIN:SMAKY et DEMO

                        .PROC   Z80
                        .REF    SMG

100000                  .LOC    100000


                ; Daniel ROUX 78
                ;
                ;       C H R O N O M E T R E
                ;       ---------------------
                ;
                ; Ce programme simule un chronometre avec la possibilite
                ; de prendre des temps intermediares.
                ; Les fonctions possibles sont les suivantes :
                ;
                ; -START        demarre
                ; -STOP         arret
                ; -RESET        remise a zero
                ; -LAP          prise d'un temps intermediere
                ;
                ; Les fonctions START/STOP se font avec la touche "S".
                ; Ces deux fonctions sont effectuees alternativement.
                ; C'est le bit 2^0 de FONCTION qui vaut 0 si le chrono
                ; est arrete et 1 dans le cas contraire.
                ; Les fonctions RESET/LAP se font avec la touche "R".
                ; La fonction RESET est effectuee si le chrono est arrete
                ; et la fonction LAP est effectuee si le chrono tourne.


                CHRONO:
100000  257            XOR     A,A             ; Init STOP
100001  062 204 200     LOAD    FONCTION,A
100004  001 024 000     LOAD    BC,#LINES+400*0 ; Init l'ecran alpha
100007  347 030         .W      ?IDICAR
100011  347 030         .W      ?CALPHA
100013  347 021         .W      ?IALPHA
100015  041 225 200     LOAD    HL,#TINIT       ; Affiche petit texte aide-
100020  347 006         .W      ?DITEX          ; memoire
100022  001 030 004     LOAD    BC,#LINES-4+400*4 ; Place pour LAP
100025  347 020         .W      ?IDICAR
100027  041 000 023     LOAD    HL,#0+400*(LINES-1) ; Place le pointeur tout en
100032  347 040         .W      ?SETCURSOR          ; bas a gauche
100034  041 142 200     LOAD    HL,#AFTEMPS     ; Pour routine d'affichage du
100037  347 004         .W      ?IRTC           ; temps
                CHRON0:
100041  016 000         LOAD    C,#0            ; Clear du compteur de LAP
                CHRON1:
100043  347 021         .W      ?GETCAR         ; Attente d'un caractere
100045  376 122         COMP    A,#'R
100047  312 077 200     JUMP.EQ RESLAP          ; RESET/LAP ?
100052  376 123         COMP    A,#'S
100054  312 066 200     JUMP.EQ STASTO          ; START/STOP ?
100057  041 211 200     LOAD    HL,#TERROR      ; Si non c'est une erreur
100062  347 006         .W      ?DITEX
100064  030 355         JUMP    CHRON1

                STASTO:
100066  041 204 200     LOAD    HL,#FONCTION    ; Inverse le flag START
100071  176             LOAD    A,(HL)
100072  356 001         XOR     A,#1
100074  167             LOAD    (HL),A
100075  030 344         JUMP    CHRON1

                RESLAP:
100077  041 204 200     LOAD    HL,#FONCTION    ; Teste le flag START
100102  313 106         TEST    (HL):0
100104  312 126 200     JUMP.EQ RESET           ; Si START=0 go to RESET
100107  041 220 200     LOAD    HL,#TLAP        ; Effectue la fonction LAP
100112  347 006         .W      ?DITEX          ; Affiche le mot LAP
100114  014             INC     C               ; Inc le compteur du nb de LAP
100115  171             LOAD    A,C
100116  347 072         .W      ?AFOCA          ; Affiche l'etat de ce compteur
100120  076 015         LOAD    A,#CR           ; Effectue un ROLL sans effacer
100122  347 000         .W      ?DICAR          ; la fin de la ligne.
100124  030 315         JUMP    CHRON1

                RESET:
100126  041 205 200     LOAD    HL,#TEMPS       ; Remise a zero des centiemes,
100131  006 004         LOAD    B,#4            ; des secondes et des minutes.
                RESE1:
100133  066 000         LOAD    (HL),#0
100135  043             INC     HL
100136  020 373         DECJ.NE B,RESE1
100140  030 277         JUMP    CHRON0


                ; Routine d'incrementation et d'affichage du temps.

                AFTEMPS:
100142  041 204 200     LOAD    HL,#FONCTION    ; Test le flag START
100145  313 106         TEST    (HL):0
100147  312 151 200     JUMP.EQ AFT1            ; Si START=0, il ne faut pas
100152  041 205 200     LOAD    HL,#TEMPS       ; incrementer le temps car le
100155  347 111 347 112 .W      ?INCSEC,?INCHOUR ; chrono est arrete.
                AFT1:
100161  347 041         .W      ?GETCURSOR      ; Sauve le curseur
100163  345             PUSH    HL
100164  041 012 023     LOAD    HL,#10.+400*(LINES-1) ; Place le pointeur en bas
100167  347 040         .W      ?SETCURSOR            ; et 10. a gauche
                000002  .RDX    2
100171  076 060         LOAD    A,#00110000     ; Masque pour le :
                000010  .RDX    8.
100173  041 210 200     LOAD    HL,#TEMPS+3     ; Affichage du temps
100176  347 115         .W      ?AFTIME
100200  341             POP     HL              ; Restitution du pointeur
100201  347 040         .W      ?SETCURSOR
100203  311             RET


100204                  000001 FONCTION: .BLKB  1               ; Flag START

100205  000           TEMPS:    .B      0       ; Centiemes
100206  000                     .B      0       ; Secondes
100207  000                     .B      0       ; Minutes
100210  000                     .B      0       ; Heures

100211  105 122 122 117 TERROR: .ASCIZ  "ERROR(CR)"
100220  114 101 120 040 TLAP:   .ASCIZ  "LAP "
100225  103 110 122 117 TINIT:  .ASCII  "CHRONOMETRE(CR)"
100241  123 072 040 123        .ASCII  "S  START/STOP(CR)"
100257  122 072 040 122        .ASCIZ  "R: RESET/LAP"


                100000          .END    CHRON0
```



```
000055 references
Source file 000036 usefull lines long
Binary file 000274 bytes long
Assembly time: 0015 seconds     0344 lines/min


?AFOCA= 035347  01-0075
?AFTIM= 046747  01-0107
?CALPH= 014347  01-0038
?DICAR= 000347  01-0077
?DITEX= 003347  01-0041 01-0057 01-0072
?GETCA= 000747  01-0051
?GETCU= 020747  01-0099
?IALPH= 010747  01-0039
?IDICA= 010347  01-0037 01-0043
?INCHO= 045347  01-0097
?INCSE= 044747  01-0097
?IRTC = 002347  01-0047
?SETCU= 020347  01-0045 01-0102 01-0109
AFT1  : 100161  01-0095 01-0098
AFTEMP: 100142  01-0046 01-0092
CHRON0: 100041  01-0048 01-0087
CHRON1: 100043  01-0050 01-0058 01-0065 01-0078
CHRONO: 100000  01-0033 01-0128
CR    = 000015  01-0076 01-0120 01-0122 01-0123
FONCTI  100204  01-0035 01-0061 01-0068 01-0093 01-0113
LINES = 000024  01-0036 01-0042 01-0044 01-0101
RESE1   100133  01-0083 01-0085
RESET   100126  01-0070 01-0080
RESLAP  100077  01-0053 01-0067
STASTO  100066  01-0055 01-0060
TEMPS   100205  01-0081 01-0093 01-0106 01-0115
TERROR  100211  01-0056 01-0120
TINIT   100225  01-0040 01-0122
TLAP    100220  01-0071 01-0121
```

TEST MEMOIRE VIVE


```
                        .TITLE  TRAMS.SR        ; JDN:SMAKY et SM
                        .PROC   Z80
                        .REF    SMG

                ; Test mémoire utilisant le système avec jolie mise en page,
                ; comptage des passes, erreurs et affichage de la durée du test
                ; SMAKY 6 et DATAS

                ; Definitions du programme

001000 DEPL     = 1000          ; Adresse du programme (RAM ou ROM)
                                ; par rapport à l'écran
043000 DEBRAM   = 43000         ; Début zone testable


041000                  .LOC    SALPHA+DEPL     ; Dans l'écran
                                                ; (augmente la zone testable)


                ; On peut executer le programme depuis le moniteur en tapant 4100

                DEB:
041000  303 224 102             JUMP    TRAM

041003  124 145 163 164 TEX1:   .ASCII  "Test memoire SMAKY-DATAS"
041033  015 124 145 163 TEX11:  .ASCIZ  "(CR)Test de ( )=43000) "
041060  040 141 040 050 TEX2:   .ASCIZ  " a ((')=140000) "
041077  015 116 142 040 TEX3:   .ASCIZ  "(CR)Nb de passes: "
041117  011 011 105 162 TEX4:   .ASCIZ  "(TAB)(TAB)Erreurs: "
041136  011 040 040 040 TEX5:   .ASCIZ  "(TAB)     Duree: "
041150  015 101 144 162 TEX6:   .ASCIZ  "(CR)Adresse  Relu   Ecrit"
041177  101 144 162 145 TEX7:   .ASCIZ  "Adresse incorrecte !"


                ;       Debut du programme

                TRAM:
041224  061 000 105             LOAD    SP,#ENDALPHA
041227  001 010 000             LOAD    BC,#8.          ; 8 lignes en haut de l'écran
041232  347 020                 .W      ?IDICAR
041234  076 002                 LOAD    A,#CLEARW       ; Effacement de cette zone
041236  347 000                 .W      ?DICAR
041240  041 003 102             LOAD    HL,#TEX1        ; Copie premier texte
                TRA2:
041243  347 006                 .W      ?DITEX
041245  347 100                 .W      ?INMON          ; Donnée premier argument
041247  021 000 106             LOAD    DE,#DEBRAM      ; Acceptable?
041252  347 056                 .W      ?COMPHLDE
041254  332 150 103             JUMP,LO ERLEC           ; Si non, retour
041257  345                     PUSH    HL              ; Adresse de debut
041260  041 060 102             LOAD    HL,#TEX2
041263  347 006                 .W      ?DITEX
041265  347 100                 .W      ?INMON          ; Donnée 2e argument
041267  321                     POP     DE
041270  325                     PUSH    DE
041271  267                     OR      A,A
041272  355 122                 SUBC    HL,DE           ; Calcul longueur à tester
041274  332 150 103             JUMP,LO ERLEC           ; Erreur si négatif
041277  345                     PUSH    HL              ; Longueur a tester sur la pile
041300  041 077 102             LOAD    HL,#TEX3
041303  347 006                 .W      ?DITEX
041305  347 041                 .W      ?GETCURSOR
041307  042 216 103             LOAD    PASSE,HL        ; Coordonnée nombre de passes
041312  041 117 102             LOAD    HL,#TEX4
041315  347 006                 .W      ?DITEX
041317  347 041                 .W      ?GETCURSOR
041321  042 220 103             LOAD    ERREUR,HL       ; Coordonnée nombre d'erreurs
041324  041 133 102             LOAD    HL,#TEX5
041327  347 006                 .W      ?DITEX
041331  347 041                 .W      ?GETCURSOR
041333  042 222 103             LOAD    DUREE,HL        ; Coordonnée durée du test
041336  041 150 102             LOAD    HL,#TEX6
041341  347 006                 .W      ?DITEX

041343  041 224 103             LOAD    HL,#TEMPS       ; Table sec-min-heures
041346  006 006                 LOAD    B,#LONGTEMPS    ; Longueur de la table
                TRA4:
041350  066 000                 LOAD    (HL),#0         ; Temps initial nul
041352  043                     INC     HL
041353  020 373                 DECJ,NE B,TRA4
041355  041 174 103             LOAD    HL,#HORLOGE     ; Adresse routine mesure temps
041360  347 004                 .W      ?IRTC
041362  373                     ION
041363  331                     EX      BL
041364  041 000 000             LOAD    HL,#0           ; Compteur de passes
041367  021 000 000             LOAD    DE,#0           ; Compteur d'erreurs
041372  331                     EX      BL

041373  021 004 004             LOAD    BC,#4+4*400     ; 4 lignes en 4e pos pour
041376  347 020                 .W      ?IDICAR         ; affichage des erreurs

                ;       Affichage compteurs et temps

                AGAIN:
041400  347 041                 .W      ?GETCURSOR
041402  042 214 103             LOAD    SAVPOINT,HL     ; Sauvetage du pointeur courant
041405  052 216 103             LOAD    HL,PASSE
041410  347 040                 .W      ?SETCURSOR      ; Déplacement du pointeur
041412  331                     EX      BL
041413  043                     INC     HL
041414  347 102                 .W      ?AFMOHL         ; Affichage compteur de passes
041416  325                     PUSH    DE
041417  331                     EX      BL
041420  052 220 103             LOAD    HL,ERREUR
041423  347 040                 .W      ?SETCURSOR
041425  341                     POP     HL
041426  347 102                 .W      ?AFMOHL         ; Affichage compteur d'erreur
041430  052 222 103             LOAD    HL,DUREE
041433  347 040                 .W      ?SETCURSOR
041435  041 220 103             LOAD    HL,#TEMPS+LONGTEMPS-2
                        000002  .RDX     2
041440  076 040                 LOAD    A,#00100000     ; JJ hh:mm
                        000010  .RDX     8.
041442  347 115                 .W      ?AFTIME         ; Affichage du temps
041444  052 214 103             LOAD    HL,SAVPOINT
041447  347 040                 .W      ?SETCURSOR      ; Rétablissement pointeur
                ION
041451  373                     POP     DE              ; Récuperation adresse et
041452  321                     POP     HL              ; longueur à tester
041453  341                     PUSH    HL
041454  345                     PUSH    DE
041455  325                     LOAD    B,C             ; Premier caractère
041456  101
```

```
                ;       Ecriture d'un motif

                STORE:
041457  160                     LOAD    (HL),B
041460  042 210 103             LOAD    ECRIT,HL
041463  043                     INC     HL
041464  004                     INC     B
041465  033                     DEC     DE
041466  172                     LOAD    A,D
041467  263                     OR      A,E
041470  040 365                 JUMP,NE STORE

                RENIT:
041472  321                     POP     DE              ; Recupération adr. et long.
041473  341                     POP     HL
041474  345                     PUSH    HL
041475  325                     PUSH    DE
041476  101                     LOAD    B,C

                ;       Relecture et comparaison

                CHECK:
041477  176                     LOAD    A,(HL)
041500  042 212 103             LOAD    RELU,HL
041503  270                     COMP    A,B
041504  302 121 103             JUMP,NE ERROR
                CONTI:
041507  043                     INC     HL
041510  004                     INC     B
041511  033                     DEC     DE
041512  172                     LOAD    A,D
041513  263                     OR      A,E
041514  040 361                 JUMP,NE CHECK
041516  014                     INC     C
041517  030 257                 JUMP    AGAIN

                ;       Erreur mémoire

                ERROR:
041521  305                     PUSH    BC              ; En cas d'erreur, affichage
041522  347 043                 .W      ?RETURN
041524  347 102                 .W      ?AFMOHL         ; adresse
041526  347 042                 .W      ?SPACE
041530  347 070                 .W      ?AFBIN          ; contenu théorique
041532  347 042                 .W      ?SPACE
041534  170                     LOAD    A,B
041535  347 070                 .W      ?AFBIN          ; contenu effectif
041537  323 003                 LOAD    SHP,A
041541  373                     ION
041542  301                     POP     BC
041543  331                     EX      BL
041544  023                     INC     DE
041545  331                     EX      BL
041546  030 337                 JUMP    CONTI


                ;       Erreur de donnee

                ERLEC:
041550  001 010 000             LOAD    BC,#8.          ; 8 lignes depuis le haut
041553  347 020                 .W      ?IDICAR
041555  076 002                 LOAD    A,#CLEARW
041557  347 000                 .W      ?DICAR
041561  041 177 102             LOAD    HL,#TEX7
041564  347 006                 .W      ?DITEX
041566  041 033 102             LOAD    HL,#TEX11
041571  303 243 102             JUMP    TRA2

                ;       Routine d'interruptionexecutée 50 fois par seconde

                HORLOGE:
041574  041 224 103             LOAD    HL,#TEMPS
041577  347 111                 .W      ?INCSEC
041601  320                     RET,CC
041602  347 112                 .W      ?INCHOUR
041604  320                     RET,CC
041605  347 113                 .W      ?INCDAY
041607  311                     RET

                        ; Paramètres en mémoire vive

041610  000 000          ECRIT:  .W      0               ; Pointeur ecriture
041612  000 000          RELU:   .W      0               ; Pointeur relecture
041614  000 000          SAVPOINT:.W     0               ; Pointeur courant
041616  000 000          PASSE:  .W      0               ; Pointeur aff. nb de passes
041620  000 000          ERREUR: .W      0               ; Pointeur aff. nb d'erreurs
041622  000 000          DUREE:  .W      0               ; Pointeur aff. durée
041624                   TEMPS:  .BLKW   3               ; Cent-Sec,Min-Heures,Jour-Mois
                000006   LONGTEMPS= .-TEMPS


                041224          .END    TRAM
```

```
000120 references
Source file 000168 usefull lines long
Binary file 000624 bytes long
Assembly time: 0020 seconds     0504 lines/min


?AFBIN= 034347  01-0173 01-0176
?AFMOH= 041347  01-0107 01-0113 01-0171
?AFTIM= 046747  01-0120
?COMPH= 027347  01-0054
?DICAR= 000347  01-0048 01-0195
?DITEX= 003347  01-0051 01-0058 01-0067 01-0071 01-0075 01-0079 01-0197
?GETCU= 020747  01-0068 01-0072 01-0076 01-0101
?IDICA= 010347  01-0046 01-0096 01-0193
?INCDA= 045747  01-0209
?INCHO= 045347  01-0207
?INCSE= 044747  01-0205
?INMON= 040347  01-0052 01-0059
?IRTC = 002347  01-0088
?RETUR= 021747  01-0170
?SETCU= 020347  01-0104 01-0111 01-0115 01-0122
?SPACE= 021347  01-0172 01-0174
AGAIN : 041400  01-0100 01-0164
CHECK : 041477  01-0151 01-0162
CLEARW= 000002  01-0047 01-0194
CONTI : 041507  01-0156 01-0184
CR    = 000015  01-0032 01-0034 01-0037
DEB   : 041000  01-0028
DEBRAM= 043000  01-0016 01-0053
DEPL  = 001000  01-0014 01-0020
DUREE : 041622  01-0077 01-0114 01-0219
ECRIT : 041610  01-0134 01-0214
ENDALP= 042400  01-0044
ERLEC : 041550  01-0055 01-0064 01-0191
ERREUR: 041620  01-0073 01-0110 01-0218
ERROR : 041521  01-0155 01-0168
HORLOG  041571  01-0087 01-0203
HP    = 000003  01-0178
LONGTE= 000006  01-0082 01-0116 01-0221
PASSE : 041616  01-0069 01-0103 01-0217
RELU  : 041612  01-0153 01-0215
RENIT : 041472  01-0142
SALPHA= 040000  01-0020
SAVPOI  041614  01-0102 01-0121 01-0216
STORE : 041457  01-0132 01-0140
TAB   = 000011  01-0035 01-0038
TEMPS : 041624  01-0081 01-0116 01-0204 01-0220 01-0221
TEX1  : 041003  01-0031 01-0049
TEX11 : 041033  01-0033 01-0198
TEX2  : 041060  01-0043 01-0060
TEX3  : 041077  01-0064 01-0066
TEX4  : 041117  01-0075 01-0070
TEX5  : 041136  01-0076 01-0074
TEX6  : 041150  01-0047 01-0078
TEX7  : 041177  01-0038 01-0190
TRA2  : 041243  01-0058 01-0199
TRA4  : 041350  01-0080 01-0084
TRAM  : 041224  01-0029 01-0043 01-0223
```

FILE COMPRESS

```
80/03/17 08:59:37  FILE_COMPRESS

    1                           .TITLE FILE_COMPRESS            ;RF:790504
    2
    3                           .PROC   Z80     ; Z80 processor
    4                           .REF    SNG     ; SNAKY 6 system
    5
    6
    7                   ;       this program is designed to compress source files
    8                   ;       by putting the carriage return <15> just after the
    9                   ;       last significant character of each line.
   10                   ;       input and output files are different enabling a
   11                   ;       easy comparison after work as been done.
   12
   13
```

```
80/03/17 08:59:37  FILE_COMPRESS                PROGRAM

    1                           .SBTTL  PROGRAM
    2
    3 053000                    .LOC    53000
    4
    5                   START:
    6 053000  001 024 000               LOAD    BC,#20.
    7 053003  347 020                   .W      ?IDICAR         ; init a full window on
    8 053005  347 122                   .W      ?CLEAR          ; screen
    9 053007  347 021                   .W      ?IALPHA
   10 053011  347 123 225 126           .W      ?TEXT,TBEG      ; display the little text
   11 053015  347 123 201 126           .W      ?TEXT,TINFIL
   12 053021  347 005                   .W      ?GETLINE        ; and get input filename
   13 053023  353                       EX      HL,DE
   14 053024  357 005                   .W      ?OPEN           ; open file for read.ng
   15 053026  332 173 126               JUMP,CS ERROR
   16 053031  062 004 127               LOAD    INCH,A          ; and save channel number
   17 053034  347 123 243 126           .W      ?TEXT,TOUTFIL
   18 053040  347 005                   .W      ?GETLINE        ; get output filename
   19 053042  353                       EX      HL,DE
   20 053043  357 003                   .W      ?CREATE         ; open for writing
   21 053045  332 173 126               JUMP,CS ERROR
   22 053050  062 005 127               LOAD    OUTCH,A         ; and save channel number
   23                   LOOP:
   24 053053  072 004 127               LOAD    A,INCH
   25 053056  021 006 127               LOAD    DE,#BUFFER      ; read one line from file
   26 053061  357 012                   .W      ?RDLINE         ; into buffer
   27 053063  332 146 126               JUMP,CS LEND
   28 053066  041 006 127               LOAD    HL,#BUFFER      ; search the CR in the buffer
   29 053072  001 200 000               LOAD    BC,#200
   30 053074  076 015                   LOAD    A,#CR
   31 053076  355 261                   CPIR
   32 053100  053                       DEC     HL              ; and adjust pointer to last
   33 053101  053                       DEC     HL              ; supposed valid char.
   34                   LOP1:
   35 053102  176                       LOAD    A,(HL)          ; fetch the character
   36 053103  053                       DEC     HL
   37 053104  376 011                   COMP    A,#TAB          ; if either a tab or a space
   38 053106  050 372                   JUMP,EQ LOP1            ; we skip over
   39 053110  376 040                   COMP    A,#SPACE
   40 053112  050 366                   JUMP,EQ LOP1
   41 053114  043                       INC     HL              ; adjust pointer to cr new
   42 053115  043                       INC     HL              ; position
   43 053116  066 015                   LOAD    (HL),#CR        ; and write new CR
   44 053120  041 006 127               LOAD    HL,#BUFFER
   45 053123  006 015                   LOAD    B,#CR           ; display line on screen for
   46 053125  347 013                   .W      ?DITED          ; operator's agreement
   47 053127  347 043                   .W      ?RETURN
   48 053131  021 006 127               LOAD    DE,#BUFFER      ; and write the new line to
   49 053134  072 005 127               LOAD    A,OUTCH         ; the output file.
   50 053137  357 014                   .W      ?WRLINE
   51 053141  332 173 126               JUMP,CS ERROR
   52 053144  030 305                   JUMP    LOOP            ; this will be donne a lot
   53                   LEND:
   54 053146  376 026                   COMP    A,#EREOF        ; if not an EOF it's an error
   55 053150  302 173 126               JUMP,NE ERROR
   56 053153  072 004 127               LOAD    A,INCH          ; job is ended, close input
   57 053156  357 007                   .W      ?CLOSE          ; and output files
   58 053160  072 005 127               LOAD    A,OUTCH
   59 053163  357 007                   .W      ?CLOSE
   60 053165  347 123 344 126           .W      ?TEXT,TEND      ; display terminal text
   61 053171  357 032                   .W      ?RTN            ; and get back to CLI
   62
   63                   ERROR:
   64 053173  357 023                   .W      ?ERROR          ; print error massage
   65 053175  357 032                   .W      ?RTN            ; and exit
   66 053177  030 372                   JUMP    ERROR
   67
   68
   69 053201  015 015 015 107 TINFIL: .ASCIZ /<CR><CR><CR>Give input file name: /
   70 053233  015 015 107 151 TOUTFIL:.ASCIZ /<CR><CR>Give output file name: /
   71 053263  002 011 011 123 TBEG    .ASCII /<CLEAR,U><TAB><TAB>Source compactification/
   72 053317  040 103 102 157         .ASCIZ / program version 0-0/
   73 053344  015 015 015 015 TEND:   .ASCIZ /<CR><CR><CR><CR>Program successfully ended<CR>/
   74 053404                  INCH:   .BLKB   1               ; input channel number
   75 053405                  OUTCH:  .BLKB   1               ; output channel number
   76 053406                  BUFFER: .BLKB   200             ; line buffer
   77
   78                         053000  .END    START
```

```
80/03/17 08:53:40  CROSS REFERENCE MAP


000087 references
Source file 000075 usefull lines long
Binary file 000404 bytes long
Assembly time: 0010 seconds     0450 lines/min
```

```
?CLEAR= 053047  02-0005
?CLOSE= 002757  02-0057 02-0059
?CREAT= 003357  02-0020
?DITE = 0.?47   02-0046
?ERROR= 011757  02-0064
?GETL = 002747  02-0012 02-0018
?IALPH= 010?47  02-0009
?IDICA= 010217  02-0007
?OPEN = 002757  02-0014
?RDLIN= 003757  02-0023
?RETUR= 021?47  02-0047
?RTN  = 015757  02-0061 02-0065
?TEXT = 053?47  02-0010 02-0011 02-0017 02-0060
?WRLIN= 00?357  02-0050
BUFFER= 053406  02-0025 02-0029 02-0044 02-0046 02-0076
CLEAR,= 000002  02-0071
CR    = 000015  02-0030 02-0043 02-0045 02-0069 02-0070 02-0073
EREOF = 000026  02-0054
ERROR : 053173  02-0015 02-0021 02-0051 02-0055 02-0063 02-0066
INCH  : 053404  02-0016 02-0024 02-0056 02-0074
LEND  : 053146  02-0027 02-0053
LOOP  : 053053  02-0023 02-0052
LOP1  : 053102  02-0034 02-0038 02-0040
OUTCH . 053405  02-0022 02-0049 02-0058 02-0075
SPACE = 000040  02-0039
START : 053000  02-0005 02-0078
01-   TAB   = 000011  02-0037 02-0071
TBEG  : 053265  02-0010 02-0071
TEND  . 053344  02-0060 02-0073
TINFIL: 053201  02-0011 02-0069
TOUTFI: 053233  02-0017 02-0070
```

COMPARE DEUX CHAINES DE CAPACTERES

```
                        .TITLE  COMPSTRING.SR
                        .PROC   Z80
                        .REF    SMB

                        ;Daniel ROUX  15.2.80

000000 BCARRY   =       0               ; flag carry de F
000006 BZERO    =       6               ; flag zero de F
000003 BANYST   =       3               ; caractere '- trouve
000005 BFINI    =       5               ; comparaison terminee


                ;-------\
                ; COMPST >
                ;-------/

                ;Compare deux chaines de caracteres, l'une pouvant contenir une
                ;information incomplete, a savoir:
                ;   '- remplace une chaine quelconque (eventuellement nulle).
                ;   '% remplace un caractere quelconque.
                ;Les majuscules sont equivalentes aux minuscules dans la
                ;comparaison.
                ;
                ;in     HL pointeur chaine contenant '- et '%
                ;       DE pointeur autre chaine
                ;
                ;out    EQ  (DE) strictement egal a (HL)
                ;       CC  (DE) inclu dans (HL)
                ;       CS  (DE) different DE (HL)
                ;
                ;mod    F

                COMPST:
000000  305            PUSH    BC
000001  325            PUSH    DE
000002  345            PUSH    HL
000003  365            PUSH    AF
000004  016 000        LOAD    C,#0            ; clear tous les flags
000006  313 361        SET     C:BZERO         ; setz (EQ)
                CPCH0:
000010  313 131        TEST    C:BANYST        ; '- trouve ?
000012  302 034 000    JUMP,NE CPCH1
000015  176            LOAD    A,(HL)          ; non => y a-t-il un '- en (HL)
000016  376 055        COMP    A,#'-
000020  302 034 000    JUMP,NE CPCH1
000023  032            LOAD    A,(DE)          ; aussi un '- en (DE) ?
000024  376 055        COMP    A,#'-
000026  312 034 000    JUMP,EQ CPCH1
000031  043            INC     HL              ; si oui, HL pointe suivant et
000032  313 331        SET     C:BANYST        ; indique un '- trouve.
                CPCH1:
000034  325            PUSH    DE              ; sauve HL et DE
000035  345            PUSH    HL
                CPCH2:
000036  176            LOAD    A,(HL)          ; fin chaine pointee par HL ?
000037  315 236 000    CALL    TESTFIN
000042  312 161 000    JUMP,EQ CPCH5
000045  376 055        COMP    A,#'-           ; fin comparaison partielle ?
000047  302 065 000    JUMP,NE CPCH21          ; non => cpch21
000052  032            LOAD    A,(DE)          ; '- en (DE) ?
000053  376 055        COMP    A,#'-
000055  312 120 000    JUMP,EQ CPCH23          ; oui => caracteres egaux
000060  313 261        CLR     C:BZERO         ; non => chaines pas egales
000062  303 171 000    JUMP    CPCH6
                CPCH21:
000065  032            LOAD    A,(DE)          ; fin chaine pointee par DE ?
000066  315 236 000    CALL    TESTFIN
000071  312 132 000    JUMP,EQ CPCH3
000074  315 225 000    CALL    MINMAJ
000077  107            LOAD    B,A             ; B <-- maj. pointee par DE
000100  176            LOAD    A,(HL)          ; A <-- car. pointee par HL
000101  376 052        COMP    A,#'*           ; '* ?
000103  302 122 000    JUMP,NE CPCH22
000106  170            LOAD    A,B
000107  376 052        COMP    A,#'*
000111  312 125 000    JUMP,EQ CPCH23
000114  313 251        CLR     C:BZERO         ; clrz (NE) si HL pointe '* et
000116  257            XOR     A,A             ; que DE ne pointe pas '*.
000117  303 125 000    JUMP    CPCH23
                CPCH22:
000122  315 225 000    CALL    MINMAJ          ; A <-- maj. pointee par DE
000125  270            COMP    A,B             ; compare les deux caracteres
                CPCH23:
000126  043            INC     HL              ; avance les pointeurs
000127  023            INC     DE
000130  050 304        JUMP,EQ CPCH2           ; caracteres egaux ?
                CPCH3:                          ; non
000132  313 131        TEST    C:BANYST        ; '- trouve ?
000134  312 146 000    JUMP,EQ CPCH32          ; non => set bfini
000137  032            LOAD    A,(DE)          ; oui => chaine (DE) finie ?
000140  315 236 000    CALL    TESTFIN
000143  302 150 000    JUMP,NE CPCH33
                CPCH32:
000146  313 351        SET     C:BFINI         ; set bfini si (DE) fini ou '-
                CPCH33:                         ; trouve.
000150  341            POP     HL              ; chaines partielles pas egales
000151  321            POP     DE              ; restitue HL et DE,
000152  313 301        SET     C:BCARRY        ; setc (CS) et
000154  313 251        CLR     C:BZERO         ; clrz (NE).
000156  303 202 000    JUMP    CPCH8
                CPCH5:                          ; (H.) fini
000161  032            LOAD    A,(DE)          ; (DE) aussi fini ?
000162  315 236 000    CALL    TESTFIN
000165  043 343        JUMP,NE CPCH3           ; non => chaines differentes
000167  313 351        SET     C:BFINI         ; (DE) et (HL) fini => set bfini
                CPCH8:                          ; chaines partielles egales.
000171  361            POP     AF              ; pops bidon pour compenser les
000172  361            POP     AF              ; pops HL et DE.
000173  313 201        CLR     C:BCARRY        ; clrc (CC)
000175  313 231        CLR     C:BANYST        ; annule le '- trouve
000177  303 210 000    JUMP    CPCH9
                CPCH8
000202  313 131        TEST    C:BANYST        ; si '- trouve et chaines pas
000204  312 210 000    JUMP,EQ CPCH9           ; egales => INC DE (cherche
000207  023            INC     DE              ; plus loin).
                CPCH9:
000210  313 151        TEST    C:BFINI         ; recommence jusqu'a ce que ce
000212  312 010 000    JUMP,EQ CPCH0           ; soit fini'
000215  361            POP     AF              ; restitue F
000216  107            LOAD    B,A
000217  341            POP     HL              ; restitue HL et DE
000220  321            POP     DE
000221  305            PUSH    BC              ; push A et flags actuellement
000222  361            POP     AF              ; dans BC et pop cf.
000223  301            POP     BC              ; restitue BC
000224  311            RET

                ;Convertit le caractere dans A en majuscule.
                MINMAJ:
000225  376 141        CMP     A,#'a
000227  310            RET,LO
000230  376 173        CMP     A,#'z+1
000232  330            RET,HI
000233  326 040        SUB     A,#('a-'A)
000235  311            RET

                ;Test si le caractere dans A est un terminateur.
                TESTFIN:
000236  376 015        COMP    A,#CR
000240  310            RET,EQ
000241  267            OR      A,A
000242  311            RET



                ;                       o
                ;                      /|\
                ;---------------------- X ----------------------------------
                ;                      /|\
                ;                       o
                ;
                ;               coupez ici pour pouvoir utiliser COMPST !

                ;Programme de test pour COPMST.
                TEST:
000243  016 024        LOAD    C,#NLI
000245  347 126        .W      ?IDIS
                LOOP:
000247  347 043        .W      ?RETURN
000251  347 136        .W      ?TEXTIM
000253  103 150 141 151        .ASCIZ  /Chaine sans '-' :/
000275  347 005        .W      ?GETLINE
000277  021 122 001        LOAD    DE,#BUFFER
000302  001 100 000        LOAD    BC,#64.
000305  355 260            LDIR
000307  347 136            .W      ?TEXTIM
000311  103 150 141 151        .ASCIZ  /Chaine avec '-' :/
000333  347 005            .W      ?GETLINE
000335  021 122 001        LOAD    DE,#BUFFER
000340  315 000 000        CALL    COMPST
000343  312 053 001        JUMP,EQ AEQ
000346  322 020 001        JUMP,CC ACC
                ACS:
000351  347 136            .W      ?TEXTIM
000353  005 143 150 141        .ASCIZ  /<INV>chaines strictement differentes<NORM><CR>/
000416  030 227            JUMP    LOOP
                ACC:
000420  347 136            .W      ?TEXTIM
000422  005 143 150 141        .ASCIZ  /<INV>chaines semblables<NORM><CR>/
000450  303 247 000        JUMP    LOOP
                AEQ:
000453  347 136            .W      ?TEXTIM
000455  005 143 150 141        .ASCIZ  /<INV>chaines strictement semblables<NORM><CR>/
000517  303 247 000        JUMP    LOOP

000522          000100 BUFFER:  .BLKB   64.

                000243 .END     TEST
```

```
000115 references
Source file 000145 usefull lines long
Binary file 000522 bytes long
Assembly time: 0013 seconds     0603 lines/min
```

SUPPRESSION DES TABULATEURS


```
                                .TITLE  SUPTAB
                                .PROC   Z80
                                .REF    SM3
100000                          .LOC    100000

                        ;       Daniel ROUX     25.2.80
                        ;
                        ;Cette routine remplace les tabulateurs par des espaces dans
                        ;une ligne de commande commencant par (DE) et terminee par un
                        ;caractere terminateur quelconque.

000010 TABCOL   =       8.

                        ;--------\
                        ; SUPTAB >
                        ;--------/

                        ;in     DE pointeur a une chaine de caracteres
                        ;out    (DE)+... chaine modifiee (longueur egale ou plus grande)
                        ;mod    -

                        SUPTAB:
100000  365                     PUSH    AF
100001  305                     PUSH    BC
100002  325                     PUSH    DE
100003  345                     PUSH    HL
100004  142                     LOAD    H,D
100005  153                     LOAD    L,E
                        SUPT1:
100006  032                     LOAD    A,(DE)          ; cherche la fin de la chaine
100007  023                     INC     DE
100010  315 125 200             CALL    TESTFIN
100013  040 371                 JUMP.NE SUPT1
100015  033                     DEC     DE
100016  016 000                 LOAD    C,#0            ; init compteur de position
                        SUPT2:
100020  345                     PUSH    HL              ; fin de la ligne ?
100021  267                     OR      A,A
100022  355 122                 SUBC    HL,DE
100024  341                     POP     HL
100025  312 120 200             JUMP.EQ SUPT9
100030  176                     LOAD    A,(HL)          ; tabulateur ?
100031  376 011                 COMP    A,#TAB
100033  312 042 200             JUMP.EQ SUPT3
100036  043                     INC     HL              ; non => cherche car. suivant
100037  014                     INC     C
100040  030 356                 JUMP    SUPT2
                        SUPT3:
100042  171                     LOAD    A,C             ; calcul nb d'espaces a mettre
100043  306 010                 ADD     A,#TABCOL
100045  346 370                 AND     A,#377-(TABCOL-1)
100047  221                     SUB     A,C
100050  365                     PUSH    AF
100051  305                     PUSH    BC
100052  075                     DEC     A
100053  312 104 200             JUMP.EQ SUPT4
100056  345                     PUSH    HL
100057  325                     PUSH    DE
100060  353                     EX      HL,DE
100061  267                     OR      A,A
100062  355 122                 SUBC    HL,DE
100064  104                     LOAD    B,H
100065  115                     LOAD    C,L
100066  321                     POP     DE
100067  325                     PUSH    DE
100070  203                     ADD     A,E
100071  137                     LOAD    E,A
100072  076 000                 LOAD    A,#0
100074  212                     ADDC    A,D
100075  127                     LOAD    D,A
100076  341                     POP     HL
100077  325                     PUSH    DE
100100  355 270                 LDDR
100102  321                     POP     DE
100103  341                     POP     HL
                        SUPT4:
100104  301                     POP     BC
100105  361                     POP     AF
100106  107                     LOAD    B,A
100107  201                     ADD     A,C
100110  117                     LOAD    C,A
                        SUPT5:
100111  066 040                 LOAD    (HL),#SPACE     ; remplit zone vide par des espace
100113  043                     INC     HL
100114  020 373                 DECJ.NE B,SUPT5
100116  030 300                 JUMP    SUPT2
                        SUPT9:
100120  341                     POP     HL
100121  321                     POP     DE
100122  301                     POP     BC
100123  361                     POP     AF
100124  311                     RET

                        ;--------\
                        ; TESTFIN >
                        ;--------/

                        ;in     A caractere quelconque
                        ;out    EQ si terminateur
                        ;       NE autrement
                        ;mod    F

                        TESTFIN:
100125  376 015                 COMP    A,#CR
100127  310                     RET.EQ
100130  267                     OR      A,A
100131  311                     RET

                        ;*******************************************************

                        TEST:
100132  016 024                 LOAD    C,#LI
100134  347 120                 .W      ?PIDIS
                        TEST0:
100136  347 005                 .W      ?GETLINE
100140  353                     EX      HL,DE
100141  315 000 200             CALL    SUPTAB
100144  353                     EX      HL,DE
100145  347 000 347 043         .W      ?DITEX,?RETURN
100151  030 303                 JUMP    TEST0

                        100132  .END    TEST
```

00035 references
Source file [illegible] useful lines long
Binary file [illegible] bytes long
Assembly time [illegible] seconds [illegible] lines/min

ADJONCTION DE TABULATEURS

```
                                .TITLE  AJTAB
                                .PROC   Z80
                                .REF    SHS
100000                          .LOC    100000

                        ;       Daniel ROUX     25.2.80
                        ;
                        ;Cette routine remplace les espaces par des tabulateurs dans
                        ;une ligne de commande commencant par (DE) et terminee par un
                        ;caractere terminateur quelconque.
                        ;La ligne ne doit pas contenir de tabulateurs !! Au cas ou vous
                        ;n'etes pas sur, appelez SUPTAB avant AJTAB :
                        ;       CALL    SUPTAB
                        ;       CALL    AJTAB

000010 TABCOL   =       8.

                        ;---------\
                        ; AJTAB   >
                        ;=========/

                        ;in     DE pointeur a une chaine de caracteres
                        ;out    (DE)+... chaine modifiee (longueur egale ou plus petite)
                        ;mod    -

                        AJTAB:
100000  365                             PUSH    AF
100001  305                             PUSH    BC
100002  325                             PUSH    DE
100003  345                             PUSH    HL
100004  142                             LOAD    H,D
100005  153                             LOAD    L,E
100006  016 000                         LOAD    C,#0
                        AJTA1:
100010  176                             LOAD    A,(HL)
100011  315 125 200                     CALL    TESTFIN
100014  312 040 200                     JUMP,EQ AJTA2
100017  043                             INC     HL
100020  014                             INC     C
100021  376 040                         COMP    A,#SPACE
100023  040 363                         JUMP,NE AJTA1
100025  171                             LOAD    A,C
100026  346 007                         AND     A,#TABCOL-1
100030  040 356                         JUMP,NE AJTA1
100032  053                             DEC     HL
100033  066 011                         LOAD    (HL),#TAB
100035  043                             INC     HL
100036  030 350                         JUMP    AJTA1
                        AJTA2:
100040  001 000 000                     LOAD    BC,#0
                        AJTA4:
100043  345                             PUSH    HL
100044  267                             OR      A,A
100045  355 122                         SUBC    HL,DE
100047  341                             POP     HL
100050  312 120 200                     JUMP,EQ AJTA9
100053  176                             LOAD    A,(HL)
100054  053                             DEC     HL
100055  003                             INC     BC
100056  376 011                         COMP    A,#TAB
100060  040 361                         JUMP,NE AJTA4
                        AJTA5:
100062  345                             PUSH    HL
100063  267                             OR      A,A
100064  355 122                         SUBC    HL,DE
100066  341                             POP     HL
100067  312 120 200                     JUMP,EQ AJTA9
100072  176                             LOAD    A,(HL)
100073  376 040                         COMP    A,#SPACE
100075  040 344                         JUMP,NE AJTA4
100077  053                             DEC     HL
100100  003                             INC     BC
100101  305                             PUSH    BC
100102  325                             PUSH    DE
100103  345                             PUSH    HL
100104  043                             INC     HL
100105  124                             LOAD    D,H
100106  135                             LOAD    E,L
100107  043                             INC     HL
100110  355 260                         LDIR
100112  341                             POP     HL
100113  321                             POP     DE
100114  301                             POP     BC
100115  033                             DEC     DE
100116  030 342                         JUMP    AJTA5
                        AJTA9:
100120  341                             POP     HL
100121  321                             POP     DE
100122  301                             POP     BC
100123  361                             POP     AF
100124  311                             RET

                        ;---------\
                        ; TESTFIN >
                        ;=========/

                        ;in     A caractere quelconque
                        ;out    EQ si terminateur
                        ;       NE autrement
                        ;mod    F

                        TESTFIN:
100125  376 015                         COMP    A,#CR
100127  310                             RET,EQ
100130  267                             OR      A,A
100131  311                             RET

                        ;=======================================================

                        TEST:
100132  016 024                         LOAD    C,#4+1
100134  347 120                         .W      ?IDIS
                        TEST0
100136  347 005                         .W      ?GETLINE
100140  353                             EX      HL,DE
100141  315 000 200                     CALL    AJTAB
100144  353                             EX      HL,DE
100145  347 000 347 043                 .W      ?DITEX,?RETURN
100151  030 352                         JUMP    TEST0

                        100132          .END    TEST
```

000037 references
Source file 000002 useful lines long
Binary file 000120 bytes long
Assembly time 0000 seconds   0000 lines/min

TEST IF


```
                        .TITLE  TESTIF.SR
                        .PROC   Z80
                        .REF    COB

                        ; Daniel ROUX  24.4.80

; Ce programme met en évidence les structures comportants des
; .IF  .ELSE  et des  .ENDIF
; Pour cela, le programme donné en entrée est copié sur un
; autre en sortie qui se verra ajouté la structure.

;--------\
; START  >
;--------/

.Début du programme principal.

START:
000000  257                     XOR     A,A             ; niveau zero au depart
000001  062 062 001             LOAD    LEVEL,A
STRU1:
000004  347 136                 .W      ?TEXTIM         ; ouvre le fichier en
000006  015 111 156 160         .ASCIZ  /(CR)Input file name: / ; entrée.
000011  347 005                 .W      ?GETLINE
000013  353                     EX      HL,DE
000014  357 005                 .W      ?OPEN
000016  070 344                 JUMP,CS STRU1
000020  062 063 001             LOAD    INCH,A          ; sauve le no de canal
STRU2:
000023  347 136                 .W      ?TEXTIM         ; crée le fichier en
000025  015 117 165 164         .ASCIZ  /(CR)Output file name: /; sortie.
000051  347 005                 .W      ?GETLINE
000053  353                     EX      HL,DE
000054  001 000 000             LOAD    BC,#0
000057  357 000                 .W      ?CREATE
000061  070 340                 JUMP,CS STRU2
000063  062 064 001             LOAD    OUTCH,A         ; sauve le no de canal
000066  347 043                 .W      ?RETURN
LOOP:
000070  072 063 001             LOAD    A,INCH          ; lit une ligne du
000073  021 105 001             LOAD    DE,#NBUF        ; fichier en entrée.
000076  357 012                 .W      ?RDLINE
000100  352 144 000             JUMP,CS FIN             ; si erreur => end of file
000103  315 172 000             CALL    WRDEBUT         ; calcul la structure
000106  072 064 001             LOAD    A,OUTCH         ; écrit la ligne avec
000111  021 025 001             LOAD    DE,#DBUF        ; la structure sur le
000114  357 014                 .W      ?WRLINE         ; fichier en sortie.
000116  076 056                 LOAD    A,#'.
000120  347 000                 .W      ?DICAR
000122  030 344                 JUMP    LOOP            ; continue jusqu'à la
                                                        ; fin du fichier en
                                                        ; entrée.
FIN:
000124  072 063 001             LOAD    A,INCH          ; ferme le fichier en
000127  001 000 000             LOAD    BC,#0           ; entrée.
000132  357 007                 .W      ?CLOSE
000134  072 064 001             LOAD    A,OUTCH         ; ferme le fichier en
000137  001 000 000             LOAD    BC,#0           ; sortie.
000142  357 007                 .W      ?CLOSE
000144  347 043 347 043         .W      ?RETURN,?RETURN
000150  357 002                 .W      ?RTN            ; retourne au CLI.


                        ;--------\
                        ; WRDEBUT >
                        ;--------/

                        ; Analyse la ligne lue dans le fichier en entrée et génère la
                        ; structure.
                        ; in    HL pointe à la ligne lue en entrée
                        ;       LEVEL   niveau actuel
                        ; out   LEVEL   nouveau niveau
                        ;       DBUF    structure
                        ; mod   LEVEL, AF, BC, DE, HL

                        WRDEBUT:
000153  315 012 001             CALL    SKPSP           ; saute les espaces et les tabs
000156  176                     LOAD    A,(HL)
000157  376 056                 COMP    A,#'.           ; pseudo ? (commence par un . )
000161  312 207 000             JUMP,EQ WRDE1
000164  315 333 000             CALL    BVERT           ; non => génère la structure
000167  311                     RET                     ; verticale.
                        WRDE1:
000170  043                     INC     HL
000171  021 044 001             LOAD    DE,#IF          ; .IF ?
000174  315 026 001             CALL    COMPSTR
000177  312 247 000             JUMP,EQ PIF
000202  021 047 001             LOAD    DE,#ELSE        ; .ELSE ?
000205  315 026 001             CALL    COMPSTR
000210  312 256 000             JUMP,EQ PELSE
000213  021 054 001             LOAD    DE,#ENDIF       ; .ENDIF ?
000216  315 026 001             CALL    COMPSTR
000221  312 270 000             JUMP,EQ PENDIF
000224  315 333 000             CALL    BVERT           ; si autre pseudo => comme une
000227  311                     RET                     ; instruction normale.

                        ;--------\
                        ; PIF    >
                        ;--------/

                        ; Augmente le level et calcul les structures horizontale et
                        ; verticale.
                        ; in    LEVEL   niveau actuel
                        ; out   LEVEL   nouveau niveau
                        ;       DBUF    structure
                        ; mod   LEVEL, DBUF, AF, BC, DE, HL

                        PIF
000230  072 062 001             LOAD    A,LEVEL
000233  074                     INC     A               ; augmente le level
000234  062 062 001             LOAD    LEVEL,A

                        ;--------\
                        ; PELSE  >
                        ;--------/

                        ; Calcul les structures horizontale et verticale.
                        ; in    LEVEL   niveau actuel
                        ; out   DBUF    structure
                        ; mod   DBUF, AF, BC, DE, HL.

                        PELSE:
000237  315 333 000             CALL    BVERT
000242  315 001 001             CALL    NUMERO
000245  315 366 000             CALL    BHORI
000250  311                     RET

                        ;--------\
                        ; PENDIF >
                        ;--------/

                        ; Calcul les structures horizontale et verticale puis
                        ; diminue le level.
                        ; in    LEVEL   niveau actuel
                        ; out   LEVEL   nouveau niveau
                        ;       DBUF    structure
                        ; mod   LEVEL, DBUF, AF, BC, DE, HL

                        PENDIF:
000251  072 062 001             LOAD    A,LEVEL         ; level = zero ?
000254  267                     OR      A,A
000255  312 320 000             JUMP,EQ PEND1           ; oui => erreur
000260  315 333 000             CALL    BVERT
000263  315 001 001             CALL    NUMERO
000266  315 366 000             CALL    BHORI
000271  072 062 001             LOAD    A,LEVEL
000274  075                     DEC     A               ; diminue le level
000275  062 062 001             LOAD    LEVEL,A
000300  311                     RET

                        PEND1:
000301  041 065 001             LOAD    HL,#DBUF        ; met beaucoup de ">"
000304  006 020                 LOAD    B,#LBUF
                        PEND2:
000306  066 076                 LOAD    (HL),#'>
000310  043                     INC     HL
000311  020 373                 DECJ,NE B,PEND2
000313  311                     RET


                        ;--------\
                        ; BVERT  >
                        ;--------/

                        ; Calcul la structure verticale.
                        ; in    LEVEL
                        ; out   HL      pointeur après le dernier "!"
                        ;       DBUF
                        ; mod   AF, BC, HL

                        BVERT:
000314  041 065 001             LOAD    HL,#DBUF
000317  006 020                 LOAD    B,#LBUF
                        BVER1:
000321  066 040                 LOAD    (HL),#SPACE     ; remplit le buffer pour la
000323  043                     INC     HL              ; structure avec des espaces.
000324  020 373                 DECJ,NE B,BVER1
000326  072 062 001             LOAD    A,LEVEL         ; level = zero ?
000331  267                     OR      A,A
000332  310                     RET,EQ                  ; oui => ne génère aucune structu
000333  041 065 001             LOAD    HL,#DBUF
                        BVER2:
000336  066 041                 LOAD    (HL),#'!        ; met "! " dans le buffer de
000340  043                     INC     HL              ; structures autant de fois que
000341  043                     INC     HL              ; vaut level.
000342  075                     DEC     A
000343  040 371                 JUMP,NE BVER2
000345  053                     DEC     HL
000346  311                     RET

                        ;--------\
                        ; BHORI  >
                        ;--------/

                        ; Calcul la structure horizontale.
                        ; in    HL      pointeur où mettre le premier '-'
                        ; out   DBUF
                        ; mod   AF, BC, DE, HL

                        BHORI:
000347  021 105 001             LOAD    DE,#MBUF
                        BHOR1:
000352  066 055                 LOAD    (HL),#'-        ; remplit depuis le dernier
000354  043                     INC     HL              ; "!" avec des '-'
000355  347 036                 .W      ?COMPHLDE
000357  040 371                 JUMP,NE BHOR1
000361  311                     RET

                        ;--------\
                        ; NUMERO >
                        ;--------/

                        ; Insert le numero du niveau sur le dernier "!".
                        ; in    HL      pointeur apres le dernier "!"
                        ;       LEVEL   niveau courant
                        ; out   -
                        ; mod   A

                        NUMERO:
000362  053                     DEC     HL
000363  072 062 001             LOAD    A,LEVEL
000366  306 060                 ADD     A,#'0           ; converti en ascii
000370  167                     LOAD    (HL),A
000371  043                     INC     HL
000372  311                     RET

                        ;--------\
                        ; SKPSP  >
                        ;--------/

                        ; Saute les espaces et les tabulateurs.
                        ; in    HL pointeur au debut de la ligne à analyser
                        ; out   HL pointe le premier caractère qui n'est pas un séparateu
                        ; mod   AF, HL

                        SKPSP:
000373  176                     LOAD    A,(HL)
000374  043                     INC     HL
000375  376 040                 COMP    A,#SPACE
000377  050 372                 JUMP,EQ SKPSP
000401  376 011                 COMP    A,#TAB
000403  050 366                 JUMP,EQ SKPSP
000405  053                     DEC     HL
000406  311                     RET

                        ;--------\
                        ; COMPSTR >
                        ;--------/

                        ; Compare deux chaines de caractères.
                        ; in    DE pointeur à la chaine de référence
                        ;       HL pointeur à la chaine cherchée
                        ; out   CC si les deux chaines sont égales.
                        ; mod   AF, DE

                        COMPSTR:
000407  345                     PUSH    HL
                        COMP1:
000410  032                     LOAD    A,(DE)
000411  267                     OR      A,A
000412  312 042 001             JUMP,EQ COMP3
000415  276                     COMP    A,(HL)
000416  043                     INC     HL
000417  023                     INC     DE
000420  050 366                 JUMP,EQ COMP1
000422  067                     SETC
                        COMP3:
000423  341                     POP     HL
000424  311                     RET


                        ; Pseudos opérations décodées.

000425  111 106 000     IF:     .ASCIZ  /IF/
000430  105 114 123 105 ELSE:   .ASCIZ  /ELSE/
000435  105 116 104 111 ENDIF:  .ASCIZ  /ENDIF/


                        ; Variables.

                000020 LBUF     =       80.
000443          000001 LEVEL:   .BLKB   1               ; niveau
000444          000001 INCH:    .BLKB   1               ; canal en entrée
000445          000001 OUTCH:   .BLKB   1               ; canal en sortie
000446          000020 DBUF:    .BLKB   LBUF            ; buffer contenant la structure
000466          000080 MBUF:    .BLKB   200             ; buffer contenant la ligne


                000000          .END    START
```

BANNIERE

```
                        .TITLE  BANDS.SR         ;.dir JDN:SMAKY et LIB
                        ;B00103
                        ;Forme une bannière dans un buffer

                        ;Remplacer PUTB par l'appel APEX  PUTEX et revoir la def des vart

                        .REF    COB
                        .PROC   Z80
100000                  .LOC    100000

                 000014 LBAN    = 12.            ;Longueur max par ligne

100000  016 024         TEST:   LOAD    C,#NLI
100002  347 126                 .W      ?IDIS
100004  347 136                 .W      ?TEXTIN
100006  101 146 146 151         .ASCIZ  "Affiche une banniere. Tapez la ligne<CR>"
100014  347 005                 .W      ?GETLINE
100016  076 014         TE1     LOAD    A,#LBAN
100020  021 000 130             LOAD    DE,#BUFFER
100023  315 113 200             CALL    BAN3S
100026  365                     PUSH    AF                      ;sauve le Carry, indic fin
100027  353                     EX      HL,DE
100030  006 015                 LOAD    B,#CR
100032  176             TE2:    LOAD    A,(HL)
100033  376 000                 COMP    A,#0
100035  312 106 200             JUMP,EQ FIN
100040  347 036 347 043         .W      ?DITEB ?RETURN
100044  030 364                 JUMP    TE2

100046  353             FIN:    EX      HL,DE
100047  361                     POP     AF
100050  070 344                 JUMP,CS TE1     ;Trop long
100052  166                     TRAP

                        ;
                        ;;---   BAN3S   Banière
                        ;;--- BAN3SB    idem terminateur dans B
                        ;
                        ;;in    HL      pointeur au texte termine par 0 (ou B)
                        ;;      A       long max banniere   (B terminateur)
                        ;;      DE      pointe buffer destination
                        ;;out   HL      pointeur à la fin du texte
                        ;;      buffer termine par 0
                        ;;      BC      longueur buffer
                        ;;      CS      banniere incomplete CC terminateur atteint
                        ;;mod   F A BC HL

                        ;Variables dans le programme
                        ;HL pointeur texte
                        ;DE matrice de points 3*5
                        ;C  code lettre
                        ;B  compteur de points par caractère CV
                        ;A  compteur de décalages et reg temporaire
                        ;PBUFF pointeur buffer
                        ;LBUFF longueur buffer
                        ;CC compteur de caractères par ligne
                        ;CL compteur de lignes par ligne de texte

100053  006 000         BAN3S:  LOAD    B,#0
100055  325             BAN3SB: PUSH    DE
100056  062 176 201            LOAD    ICC,A
100061  170                    LOAD    A,B
100062  062 201 201            LOAD    TERM,A
100065  355 123 172 201        LOAD    PBUFF,DE
100071  076 015                LOAD    A,#CR
100073  315 346 200            CALL    PUTB

100076  345                    PUSH    HL
100077  076 005                LOAD    A,#5
100101  062 200 201            LOAD    CL,A
                                                ;5 lignes par ligne de texte
100104  072 176 201     BA2:   LOAD    A,ICC
100107  062 177 201            LOAD    CC,A
100112  341                    POP     HL
100113  345                    PUSH    HL

100114  116             BA4:   LOAD    C,(HL)
100115  043                    INC     HL
100116  072 201 201            LOAD    A,TERM
100121  271                    COMP    A,C
100122  006 001                LOAD    B,#1    ;Flag
100124  050 077                JUMP.,EQ BA3
100126  171                    LOAD    A,C
100127  376 015                COMP    A,#CR
100131  050 072                JUMP.,EQ BA2
100133  346 077                AND     A,#77   ;gen car de 64. caractères

100135  345                    PUSH    HL
100136  041 370 200            LOAD    HL,#TAB3S
100141  207                    ADD     A,A     ,double:2 bytes par car
100142  137                    LOAD    E,A
100143  026 000                LOAD    D,#0
100145  031                    ADD     HL,DE
100146  136                    LOAD    E,(HL)
100147  043                    INC     HL
100150  126                    LOAD    D,(HL)
100151  341                    POP     HL

100152  072 200 201            LOAD    A,CL
100155  075                    DEC     A
100156  315 302 200            CALL    ROTDE
100161  006 003                LOAD    B,#3
                                                ;3 points par car
100163  313 172         BA6.   TEST    D:7     ;poids fort de DE
100165  076 040                LOAD    A,#'
100167  050 001                JUMP.,EQ BA7
100171  171                    LOAD    A,C
100172  315 345 200     BA7:   CALL    PUTB

100175  076 005                LOAD    A,#5
100177  315 302 200            CALL    ROTDE
100202  020 357                DECJ,NE D,BA6

100204  076 040                LOAD    A,#'
100206  315 345 200            CALL    PUTB
100211  315 345 200            CALL    PUTB
100214  072 177 201            LOAD    A,CC
100217  075                    DEC     A
100220  062 177 201            LOAD    CC,A
100223  040 227                JUMP,NE BA1     ;D=0

                        BA8:
100225  076 015                LOAD    A,#CR
100227  315 345 200            CALL    PUTB
100232  072 200 201            LOAD    A,CL
100235  075                    DEC     A
100236  062 200 201            LOAD    CL,A
100241  040 241                JUMP,NE BA5

100243  016 015                LOAD    C,#NLI
100245  315 345 200            CALL    PUTB
100250  076 000                LOAD    A,#0    ;Terminateur
100252  315 345 200            CALL    PUTB
100255  321                    POP     DE      ;Correction de pile
100256  267                    OR      A,A     ,CLRC
100257  003                    DEC     B

100320  312 324 200                    JUMP,EQ BA8
100323  067                            SETC
100324  221                    BA8:    POP     DE
100325  355 113 174 201                LOAD    DC,LBUFF
100331  311                            RET

                                ;--     ROTDE   décalage circulaire de DE
                                ;in     A amplitude du décalage  DE registre décalé
                                ;out    DE
                                ;mod    F A DE

100332  074                    ROTDE:  INC     A
100333  075                    RO2:    DEC     A
100334  310                            RET,EQ
100335  313 043                        SLC     E
100337  313 022                        RLC     D
100341  060 370                        JUMP,CC RO2
100343  034                            INC     E
100344  030 365                        JUMP    RO2

                                ;--     PUTB    Copie dans zone mémoire pour transf ult
                                ;in     A car  PBUFF pointeur
                                ;out    -
                                ;mod    F A

100346  345                    PUTB:   PUSH    HL
100347  052 172 201                    LOAD    HL,PBUFF
100352  167                            LOAD    (HL),A
100353  043                            INC     HL
100354  042 172 201                    LOAD    PBUFF,HL
100357  052 174 201                    LOAD    HL,LBUFF
100362  043                            INC     HL
100363  042 174 201                    LOAD    LBUFF,HL
100366  341                            POP     HL
100367  311                            RET

                        000002 .RDX     2
                               ;        ! low-high  matrice 3*5
100370  176 357                TAB3S:  .W 1110111101111110     ;@
100372  174 362                        .W 1111001001111100     ;A
100374  124 375                        .W 1111110101010100     ;B
100376  142 164                        .W 0111010001100010     ;C
100400  134 374                        .W 1111110001011100     ;D
100402  142 375                        .W 1111110101100010     ;E
100404  102 371                        .W 1111100101000010     ;F
100406  172 164                        .W 0111010001111010     ;G
100410  076 371                        .W 1111100100111110     ;H
100412  342 217                        .W 1000111111100010     ;I
100414  136 214                        .W 1000110001011110     ;J
100416  242 372                        .W 1111010101000010     ;K
100420  040 374                        .W 1111110000100000     ;L
100422  276 370                        .W 1111100010111110     ;M
100424  276 373                        .W 1111101110111110     ;N
100426  134 164                        .W 0111010001011100     ;O
100430  104 371                        .W 1111100101000100     ;P
100432  174 164                        .W 0111010001111100     ;Q
100434  144 373                        .W 1111101101100100     ;R
100436  122 225                        .W 1001010101010010     ;S
100440  302 017                        .W 0000111111000010     ;T
100442  076 374                        .W 1111110000111110     ;U
100444  036 174                        .W 0111110000011110     ;V
100446  076 373                        .W 1111101100111110     ;W
100450  066 331                        .W 1101100100110110     ;X
100452  006 037                        .W 0000111100000110     ;Y
100454  146 315                        .W 1100110101100110     ;Z

100456  100 374                        .W 1111110001000000     ;[
100460  020 021                        .W 0001000100010000     ;\
100462  176 004                        .W 0000010001111110     ;]
100464  304 027                        .W 0001011111000100     ;^
100466  040 204                        .W 1000010000100000     ;_

100470  000 000                        .W 0000000000000000     ;SP
100472  300 005                        .W 0000101111000000     ;!
100474  300 030                        .W 0001100011000000     ;"
100476  034 343                        .W 1110001100011100     ;#
100500  322 227                        .W 1001011111010010     ;$
100502  064 131                        .W 0101100100110100     ;%
100504  370 355                        .W 1110110111111000     ;&
100506  300 000                        .W 0000000011000000     ;'
100510  100 104                        .W 0111010001000010     ;(
100712  101 004                        .W 0000100010111000     ;)
100514  234 163                        .W 0111001110011100     ;*
100516  210 043                        .W 0010001110001000     ;+
100520  000 006                        .W 0000011000000000     ;,
100522  010 041                        .W 0010000100001000     ;-
100524  000 002                        .W 0000001000000000     ;.
100526  004 101                        .W 0100000100000100     ;/

100530  176 374                        .W 1111110001111110     ;0
100532  340 227                        .W 1001011111100000     ;1
100534  144 315                        .W 1100110101100100     ;2
100536  124 215                        .W 1000110101010100     ;3
100540  020 177                        .W 0111111100010000     ;4
100542  132 275                        .W 1011110101011010     ;5
100544  122 165                        .W 0111010101010010     ;6
100546  106 311                        .W 1100100101000110     ;7
100550  124 125                        .W 0101010101010100     ;8
100552  131 225                        .W 1001010101011100     ;9

100554  200 002                        .W 0000001010000000     ;:
100556  200 006                        .W 0000011010000000     ;;
100560  242 042                        .W 0010001010100010     ;<
100562  224 122                        .W 0101001010010100     ;=
100564  210 212                        .W 1000101010001000     ;>
100566  104 015                        .W 0000110101000100     ;?

                        000010 .RDX     8.

100570  000                    CANAL:  .B      0       ;Canal fichier
100571  000                    LPT:    .B      0       ;Canal imprimante

100572  000 000                PBUFF:  .W      0       ;Pointeur dans le buffer
100574  000 000                LBUFF:  .W      0       ;Longueur du buffer

100576  000                    ICC:    .B      0       ;long max des lignes
100577  000                    CC:     .B      0       ;compteur de caractères par ligne
100600  000                    CL:     .B      0       ;compteur de lignes par ligne de texte
100601  000                    TERM:   .B      0

034000                          .LOC    34000
                        BUFFER:                         ;buffer banière et texte

                                100000 .END     TEST

000280 references
Source file 00193 useful lines long
Binary file 00302 bytes long
Assembly time  0015 seconds     0752 lines/min
```

PISTON

```
                        .TITLE PISTON

                        .PROC   Z80
                        .REF    SMB


040000          BASE    = 40000
000100          WIDTH   = 100
000022          CTE     = 22
000024          HEIGHT  = 24
000777          XMVE    = 1*400+377
040300          YMVE    = 100*400+300
000010          TIMER   = 10


                START:
000000  257             XOR     A,A
000001  041 000 100     LOAD    HL,#BASE
000004  001 000 373     LOAD    BC,#-WIDTH*HEIGHT
                CLMEM:
000007  066 040         LOAD    (HL),#SPACE
000011  043             INC     HL
000012  003             INC     BC
000013  270             COMP    A,B
000014  040 371         JUMP,NE CLMEM
000016  041 323 000     LOAD    HL,#FIXLST
000021  303 046 000     JUMP    LOOP

                CURRENT:
000024  000 000         .W      0
                OLD:
000026  000 000         .W      0
                NEW:
000030  000 000         .W      0

                CYCLE:
000032  076 010         LOAD    A,#TIMER
000034  127             LOAD    D,A
                WAIT:
000035  075             DEC     A
000036  041 335 000     LOAD    HL,#OBJLST
000041  040 372         JUMP,ZC WAIT
000043  025             DEC     D
000044  040 367         JUMP,ZC WAIT
                LOOP:
000046  176             LOAD    A,(HL)
000047  267             OR      A,A
000050  050 360         JUMP,ZS CYCLE           ; end of list of objects
000052  137             LOAD    E,A
000053  043             INC     HL
000054  126             LOAD    D,(HL)
000055  043             INC     HL
000056  345             PUSH    HL
000057  353             EX      DE,HL
000060  315 066 000     CALL    MOVE
000063  341             POP     HL
000064  030 360         JUMP    LOOP

                ; subroutine move

                ; current : pointer of object

                MOVE:
000066  042 024 000             LOAD    CURRENT,HL
000071  065                     DEC     (HL)            ; count
000072  314 107 000             CALL,ZS MOVEX           ; overflow, time to move object
000075  052 024 000             LOAD    HL,CURRENT
000100  043                     INC     HL
000101  043                     INC     HL              ; point to y
000102  065                     DEC     (HL)            ; same for y
000103  314 115 000             CALL,ZS MOVEY
000106  311                     RET


                MOVEX:
000107  001 377 001             LOAD    BC,#XMVE
000112  303 120 000             JUMP    COMM

                MOVEY:
000115  001 300 100             LOAD    BC,#YMVE
                COMM:
000120  043                     INC     HL
000121  126                     LOAD    D,(HL)          ; txi
000122  036 377                 LOAD    E,#-1
000124  257                     XOR     A,A
000125  222                     SUB     A,D
000126  362 135 000             JUMP,PL NEG             ; negativ?
000131  110                     LOAD    C,B             ; use positiv move
000132  172                     LOAD    A,D
000133  036 000                 LOAD    E,#0
                NEG:
000135  103                     LOAD    B,E
000136  305                     PUSH    BC              ; remember for later

000137  053                     DEC     HL
000140  167                     LOAD    (HL),A          ; reset counter
000141  315 265 000             CALL    OBJECT

000144  026 000                 LOAD    D,#0*
                RUBOUT:
000146  066 040                 LOAD    (HL),#SPACE     ; clear screen
000150  003                     INC     BC
000151  012                     LOAD    A,(BC)          ; move increment
000152  267                     OR      A,A             ; end of table ?
000153  312 163 000             JUMP,ZS ENDRUB
000156  137                     LOAD    E,A             ; put in low byte
000157  031                     ADD     HL,DE           ; next point of object
000160  003                     INC     BC
000161  030 363                 JUMP    RUBOUT


                ENDRUB:
000163  315 265 000             CALL    OBJECT
000166  042 023 000             LOAD    OLD,HL
000171  321                     POP     DE
000172  031                     ADD     HL,DE
000173  042 033 000             LOAD    NEW,HL

000176  026 000                 LOAD    D,#0
                LOOK:
000200  176                     LOAD    A,(HL)
000201  376 040                 COMP    A,#SPACE        ; look out for obstacles
000203  302 221 000             JUMP,NE OBSTCL
000206  003                     INC     BC
000207  012                     LOAD    A,(BC)          ; get next increment
000210  267                     OR      A,A
000211  312 232 000             JUMP,ZS ENDLK
000214  137                     LOAD    E,A
000215  031                     ADD     HL,DE
000216  003                     INC     BC
000217  030 357                 JUMP    LOOK

000221  315 302 000     OBSTCL: CALL    SPEED
000224  052 026 000             LOAD    HL,OLD
000227  303 235 000             JUMP    UPDATE
                ENDLK:
000232  052 030 000             LOAD    HL,NEW
                UPDATE:
000235  353                     EX      DE,HL
000236  052 024 000             LOAD    HL,CURRENT
000241  053                     DEC     HL
000242  162                     LOAD    (HL),D
000243  053                     DEC     HL
000244  163                     LOAD    (HL),E
000245  315 265 000             CALL    OBJECT

000250  026 000                 LOAD    D,#0
                DRAW:
000252  012                     LOAD    A,(BC)
000253  167                     LOAD    (HL),A
000254  003                     INC     BC
000255  012                     LOAD    A,(BC)
000256  267                     OR      A,A
000257  310                     RET,ZS
000260  137                     LOAD    E,A
000261  031                     ADD     HL,DE
000262  003                     INC     BC
000263  030 365                 JUMP    DRAW

                ; subroutine object
                ; puts current object's postion in hl
                ; and description pointer in bc

                OBJECT:
000265  052 024 000             LOAD    HL,CURRENT
000270  053                     DEC     HL
000271  126                     LOAD    D,(HL)
000272  053                     DEC     HL
000273  136                     LOAD    E,(HL)
000274  053                     DEC     HL
000275  106                     LOAD    B,(HL)
000276  053                     DEC     HL
000277  116                     LOAD    C,(HL)
000300  353                     EX      DE,HL
000301  311                     RET

                SPEED:
000302  052 024 000             LOAD    HL,CURRENT
000305  043                     INC     HL
000306  006 111                 LOAD    B,#'I
000310  270                     COMP    A,B
000311  312 316 000             JUMP,EQ X
000314  043                     INC     HL
000315  043                     INC     HL
                X:
000316  106                     LOAD    B,(HL)
000317  257                     XOR     A,A
000320  220                     SUB     A,B
000321  167                     LOAD    (HL),A
000322  311                     RET

000323  346 001 356 001 FIXLST: .W     BAS+4 VERTG+4 VERTD+4 HAUT+4 0

000335  015 001 026 001 OBJLST: .W     HAHA+4 BALL1+4 BALL2+4 BALL3+4 BALL4+4 BALL5+4
000351  076 001 106 001         .W     BALL6+4 BALL7+4 BALL8+4 BALL9+4 BALL10+4
000363  146 001 156 001         .W     BALL11+4 BALL12+4 BALL13+4 BALL14+4 BALL15+4
000375  216 001 226 001         .W     BALL16+4 BALL17+4 BALL18+4 BALL19+4 BALL20+4 0

000411  014 002 305 100 HAHA:   .WBBB HORDES+CTE BASE+3*WIDTH+5 1 5 2 7

                        040101 BGN      = BASE+WIDTH+1

000422  262 001 102 100 BALL1:  .WBBBB OA BGN+1   1  1  1  1
000432  264 001 103 100 BALL2:  .WBBBB OB BGN+2   1  1  1 -1
000442  266 001 104 100 BALL3:  .WBBBB OC BGN+3   1 -1  1  1
000452  270 001 105 100 BALL4:  .WBBBB OD BGN+4   1 -1  1 -1
000462  272 001 106 100 BALL5:  .WBBBB OE BGN+5   1  2  1  1
000472  274 001 107 100 BALL6:  .WBBBB OF BGN+6   1  2  1 -1
000502  276 001 110 100 BALL7:  .WBBBB OG BGN+7   1  2  1 -1
000512  300 001 111 100 BALL8:  .WBBBB OH BGN+8.  1  2  1  2
000522  302 001 112 100 BALL9:  .WBBBB OI BGN+9.  1  2  1 -2
000532  304 001 111 100 BALL10: .WBBBB OJ BGN+10  1 -2  1  1
000542  306 001 112 100 BALL11: .WBBBB OK BGN+11  1 -2  1 -1
000552  310 001 113 100 BALL12: .WBBBB OL BGN+12  1 -2  1  2
000562  312 001 114 100 BALL13: .WBBBB OM BGN+13  1 -2  1 -2
000572  314 001 115 100 BALL14: .WBBBB ON BGN+14  1 -3  1  1
000602  316 001 116 100 BALL15: .WBBBB OO BGN+15  1  3  1 -1
000612  320 001 117 100 BALL16: .WBBBB OP BGN+16  1  3  1  2
000622  322 001 120 100 BALL17: .WBBBB OQ BGN+17  1 -3  1 -1
000632  324 001 123 100 BALL18: .WBBBB OR BGN+18. 1  3  1 -2
000642  326 001 124 100 BALL19: .WBBBB OS BGN+19. 1 -3  1 -2
000652  330 001 125 100 BALL20: .WBBBB OT BGN+20. 1  2  1  6


000662  101 000         OA:     .B      'A 0
000664  102 000         OB:     .B      'B 0
000666  103 000         OC:     .B      'C 0
000670  104 000         OD:     .B      'D 0
000672  105 000         OE:     .B      'E 0
000674  106 000         OF:     .B      'F 0
000676  107 000         OG:     .B      'G 0
000700  110 000         OH:     .B      'H 0
000702  111 000         OI:     .B      'I 0
000704  112 000         OJ:     .B      'J 0
000706  113 000         OK:     .B      'K 0
000710  114 000         OL:     .B      'L 0
000712  115 000         OM:     .B      'M 0
000714  116 000         ON:     .B      'N 0
000716  117 000         OO:     .B      'O 0
000720  120 000         OP:     .B      'P 0
000722  121 000         OQ:     .B      'Q 0
000724  122 000         OR:     .B      'R 0
000726  123 000         OS:     .B      'S 0
000730  124 000         OT:     .B      'T 0


000732  372 001 377 077 HAUT:   .WBBBB HORDES BASE-1 1 177 0 1
000742  372 001 277 104 BAS:    .WBBBB HORDES BASE+WIDTH*(HEIGHT-1)-1 1 177 0 1
000752  172 002 077 100 VERTG:  .WBBBB VERDES BASE+WIDTH-1 1 177 0 1
000762  172 002 177 100 VERTD:  .WBBBB VERDES BASE+2*WIDTH-1 1 -200 0 177


000772  055 001 055 001 HORDES: .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001012  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001032  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001052  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001072  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001112  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001132  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1
001152  055 001 055 001         .B      '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 1 '- 0

001172  111 100 111 100 VERDES: .B      'I 100 'I 100 'I 100 'I 100 'I 100
001214  111 100 111 100         .B      'I 100 'I 100 'I 100 'I 100 'I 100
001236  111 100 111 100         .B      'I 100 'I 100 'I 100 'I 100 'I 100
001256  111 100 111 100         .B      'I 100 'I 100 'I 0


                        000000          .END START
```

LIFE (= JEU DE LA VIE)

```
                        .TITLE LIFAUCON.SR

                        .PROC   Z80
                        .REF    SYS

100000                  .LOC    100000


                ; Daniel Roux   30.01.78
                ; idem         01.10.80  (presque 3 ans plus tard!)


                ; Jeu de life sur faucon.
                ;
                ;       *       *   ****    *****
                ;       *       *   *       *
                ;       *       *   ****    ****
                ;       *       *   *       *
                ;       *****   *   *       *****
                ;
                ;
                ; Le jeu de life permet une simulation d'une forme
                ; de vie très simple rappelant l'évolution des
                ; bactéries.
                ;
                ; Le programme comprend deux parties distinctes:
                ;
                ; 1)    Un programme permettant de dessiner sur
                  01-01
```

```
  01    LIFAUCON.SR
  02    CROSS REFERENCE MAP
```

```
80/10-01 16:14:53  LIFAUCON.SR                                  01-01

  1                     .TITLE LIFAUCON.SR
  ;
                ;
                ; Règles du jeu de life
                ; ---------------------
                ;
                ; Ces règles vont decider de l'extinction ou de la
                ; creation d'une bactérie en fonction de son
                ; nombre de voisines. Une bactérie peut avoir 8
                ; voisines au maximum.
                ;
                ;               1   2   3
                ;
                ;               4   *   5
                ;
                ;               6   7   8
                ;
                ;
                ;       -si un emplacement sur l'écran à deux
                ;        bactéries voisines, il ne sera pas
                ;        modifié.
                ;       -si un emplacement sur l'écran à trois
                ;        bactéries voisines, il s'y crée une
                ;        nouvelle bactérie.
                ;       -pour tout autre nombre de voisines, l'
                ;        emplacement se vide. (la bactérie meurt)
                ;
                ; Exemple
                ; -------
                ;
                ; 1)    *               2)      *
                ;       ***                     **
                ;                                *
                ;
                ; 3)    **              4)      **
                ;       **                      * *
                ;       **                       **
                ;
                ;
                ;
                ; Les touches "R","D","F" et "C" permettent
                ; de se déplacer dans toutes les directions.
                ; si on appuie simultanément sur l'une de
                ; ces quatre touches et sur "CTRL", on crée de
                ; nouvelles cellules.
                ;  "Z"  place le pointeur au milieu de l'écran
                ;  " "  tue la cellule située sous le pointeur
                ;  "S"  calcule une seule génération
                ;  "A"  calcule à vitesse maximale
                ;       pressez sur "z" pour arreter et remettre
                ;       le pointeur au milieu..
                ;  "K"  effacce l'écran.
                ;
                ; Pour charger un motif depuis une bande papier,
                ; tirer simplement la bande à travers le microleru.
                ;
                ;
                ; Calcul de la génération suivante
                ; ********************************
                ;
                ;
                ; On dispose au départ de 2 fois NLIXNCAR positions
                ; mémoire (ce qui correspond à 2 écrans alpha-
                ; numériques complets).
                ; Les NLIXNCAR premières commencent à "VIEW" et correspondent
                ; à l'écran.
                ; Les NLIXNCAR suivantes commencent à "VOISIN" et ne sont pas
                ; visibles.
                ;
                ; "VIEW" peut contenir soit des espaces positifs
                ; ou négatifs (40 ou 240), soit des "o" positifs
                ; ou négatifs (117 ou 317).
                ; "VOISIN" contient des valeurs qui ne peuvent
                ; pas etre vues.
                ; Ces valeurs correspondent au nombre de voisines
                ; de chaques cellules dans "VIEW".
                ; Si le bit 2^7 est à 1, cela veut dire qu'à cet
                ; emplacement, il n'y a pas de bactérie sur "VIEW".
                ;
                ; Si l'on décide de créer une cellule, il faut
                ; donc incrémenter dans "VOISIN" les 8 bytes
                ; situes autour de cette cellule.
                ;
```

```
                ; Algorithme général:
                ;
                ; On suppose au départ que "VOISIN" contient deja
                ; le nombre de cellules situees autour de chaque
                ; emplacement dans "VIEW".
                ; Il suffit alors le lire tous les bytes de
                ; "VOISIN", et, en fonction de ce nombre, de
                ; créer ou de tuer la cellule correspondante
                ; dans "VIEW".
                ; Si, par exemple, on devait tuer une cellule, il
                ; faudrait décrémenter les 8, bytes situes autour
                ; de la dite cellule. Mais cela ne peut pas etre
                ; fait, car, lorsque l'on arriverait en dessous
                ; de cette cellule, la situation serait faussée,
                ; on va donc mettre sur le stack une information
                ; disant qu'après avoir finit de balayer "VOISIN",
                ; il faut décrémenter tout ce qui se trouve autour.
                ; on va donc faire:
                ;
                ;       PUSH    DE
                ;       PUSH    AF
                ;
                ; "DE" contient l'adresse de la position.
                ; "A" est négatif s'il faut décrémenter et
                ; est positif dans le cas contraire.
                ; "F" a le carry à zero.
                ;
                ; Avant le mettre sur le stack toutes ces
                ; valeurs, on va pusher "AF" avec un carry
                ; à un, ce qui permettra de voir quelle est
                ; la premiere valeur qui a été mise sur le
                ; stack.
                ;
                ;
                ; STACK:
                ;       "AF"    ;c=1
                ;       "DE"    ;adresse dans "VOISIN"
                ;       "AF"    ;c=0
                ;       "DE"    ; ...
                ;       "AF"    ;c=0
                ;       "DE"    ; ...
                ;       ...
                ;
                ;


                ; Constantes :

002400 DIM      = NLIXNCAR              ; dimension d'un écran

040000 VIEW     = SALPHA                ; écran visible
043100 VOISIN   = SGRA+NCAR             ; écran invisible
                                        ; une ligne (NCAR byte) de part
                                        ; et d'autre de "VOISIN"
                                        ; peuvent être modifiées !


                ;--------\
                ; LIFE    >
                ;--------/

                LIFE:

                ; Rempli "VIEW" (2400 bytes) par des espaces
                ; (40) et "VOISIN" (2400 bytes) par des inv (200).

100000  041 100 106             LOAD    HL,#VOISIN
100003  021 000 100             LOAD    DE,#VIEW
100006  001 377 004             LOAD    BC,#DIM-1
100011  076 040                 LOAD    A,#SPACE
                LIF1:
100013  066 200                 LOAD    (HL),#200
100015  043                     INC     HL
100016  022                     LOAD    (DE),A
100017  023                     INC     DE
100020  013                     DEC     BC
100021  005                     DEC     B
100022  004                     INC     B
100023  362 013 200             JUMP,PL LIF1            ; end ?

                ; Les nliXncar positions situées à partir de
                ; "VOISIN" doivent se terminer par un 100, ce qui
                ; permettra, lors du calcul de la génération, de
                ; voir qu'on est arrivé à la fin du balayage
                ; de "VOISIN".

100026  076 100                 LOAD    A,#100
100030  062 100 113             LOAD    VOISIN+DIM,A


                ; Place le pointeur de dessin au milieu de
                ; l'écran.

                LIF11:
100033  041 137 102             LOAD    HL,#VIEW+1137
100036  042 152 201             LOAD    POINT,HL
100041  066 240                 LOAD    (HL),#SPACE!200


                ; Attente d'un caractère soit du clavier, soit
                ; de l'usart 0 .

                LIF2:
100043  347 044                 .W      ?IFRPR
100045  322 054 200             JUMP,CC LIF4
100050  347 015                 .W      ?IFOAR
100052  070 367                 JUMP,CS LIF2


                ; Test et débranchement selon la touche pressée.

                LIF4:
                        COMP    A,#SPACE        ; space ?
                        JUMP,EQ RUBOUT
                        COMP    A,#'Z           ; Z ?
                        JUMP,EQ LIF11
                        COMP    A,#'K           ; K ?
                        JUMP,EQ LIFE
                        LOAD    DE,#-100
                        COMP    A,#'R-100       ; CTRL R ?
                        JUMP,EQ CELL
                        LOAD    DE,#-1
                        COMP    A,#'D-100       ; ctrl D ?
                        JUMP,EQ CELL
                        LOAD    DE,#1
                        COMP    A,#'F-100       ; CTRL F ?
                        JUMP,EQ CELL
                        LOAD    DE,#100
                        COMP    A,#'C-100       ; CTRL C ?
                        JUMP,EQ CELL

                        COMP    A,#'A           ; A ?
                        JUMP,EQ GO
                        COMP    A,#'S           ; S ?
                        JUMP,EQ STEP
                        LOAD    DE,#-1
                        COMP    A,#'D           ; D ?
                        JUMP,EQ MOVE
                        LOAD    DE,#1
                        COMP    A,#'F           ; F ?
                        JUMP,EQ MOVE
                        LOAD    DE,#100
                        COMP    A,#'C           ; C ?
                        JUMP,EQ MOVE
                        LOAD    DE,#-100
                        COMP    A,#'R           ; R ?
                        JUMP,EQ MOVE
```

```
                        ; Si aucune de ces touches n'est pressée, on
                        ; en attend une autre.

100233  030 236                 JUMP    LIF2


                        ; Déplacement du pointeur de dessin.

                        MOVE:
100205  052 152 201             LOAD    HL,POINT
100210  313 276                 CLR     (HL),7
100212  031                     ADD     HL,DE           ; déplacement

                        ; Est-ce que l'on sort de l'écran ?

                        IFMOVE:
100215  174                     LOAD    A,H
100214  376 100                 COMP    A,HVIEW/400
100216  332 235 200             JUMP,LO NOMOVE
100221  376 105                 COMP    A,H(VIEW+DIM)/400
100223  322 235 200             JUMP,HS NOMOVE
                        IFMOV1:
100226  313 376                 SET     (HL),7
100230  042 152 201             LOAD    POINT,HL
100233  030 206                 JUMP    LIF2

                        ; Si l'on est sortit de l'écran, il faut revenir
                        ; en arrière.

                        NOMOVE:
100235  267                     OR      A,A             ; clrc
100236  355 122                 SUBC    HL,DE
100240  030 364                 JUMP    IFMOV1


                        ; Effacement d'une cellule. il faut décrémenter
                        ; toutes les positions voisines dans "VOISIN".

                        RUBOUT:
100242  052 152 201             LOAD    HL,POINT
100245  176                     LOAD    A,(HL)

                        ; s'il n'y a deja plus de cellule à cet endroit,
                        ; il ne faut rien faire.
100246  376 240                 COMP    A,HSPACE!200
100250  312 043 200             JUMP,EQ LIF2
100253  066 240                 LOAD    (HL),HSPACE!200
100255  021 100 006             LOAD    DE,HVOISIN-VIEW
100260  031                     ADD     HL,DE
100261  313 376                 SET     (HL),7
100263  315 140 201             CALL    DEAD
100266  303 043 200             JUMP    LIF2


                        ; crée une nouvelle cellule. il faut incrémenter
                        ; toutes les positions voisines dans "VOISIN".

                        CELL:
100271  325                     PUSH    DE
100272  052 152 201             LOAD    HL,POINT
100275  176                     LOAD    A,(HL)

                        ; s'il y a deja une cellule à cet endroit,
                        ; il ne faut rien faire.

100276  376 317                 COMP    A,#'0!200
100300  312 316 200             JUMP,EQ CEL2
100303  066 317                 LOAD    (HL),#'0!200
100305  021 100 006             LOAD    DE,HVOISIN-VIEW
100310  031                     ADD     HL,DE
100311  313 276                 CLR     (HL),7
100313  315 126 201             CALL    ALIVE
                        CEL2:
100316  321                     POP     DE
100317  030 264                 JUMP    MOVE


                        ; calcul d'une seule génération, replace le
                        ; pointeur de dessin au meme endroit.

                        STEP:
100321  315 346 200             CALL    GEN
100324  052 152 201             LOAD    HL,POINT
100327  313 376                 SET     (HL),7
100331  303 043 200             JUMP    LIF2


                        ; calcul à vitesse maximum des générations
                        ; de life

                        GO:
100334  315 346 200             CALL    GEN
100337  347 015                 .W      ?IFCAR
100341  070 371                 JUMP,CS GO
100343  303 654 200             JUMP    LIF4



                        ; Calcul de la génération suivante
                        ; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                        ; Push sur le stack "AF" avec le carry à un.

                        GEN:
100346  067                     SETC
100347  365                     PUSH    AF

                        ;       LOAD    B,#3!200
                        ;       LOAD    C,#SPACE
100350  001 040 203             LOAD    BC,#(3!200)*400+SPACE


                        ; "DE" pointe dans "VOISIN".
                        ; "HL" pointe dans "VIEW".

100353  021 100 106             LOAD    DE,HVOISIN
100356  041 000 100             LOAD    HL,HVIEW
100361  303 002 201             JUMP    GEN1

                        ; Est-ce qu'on est arrivé à la fin de "VOISIN" ?
                        ; Un 100 a été placé à la fin des 20.*64. bytes
                        ; de "VOISIN". Si on lit une valeur plus grande
                        ; que 30 et positive (plus petit que 200), c'est
                        ; qu'on a fini.

                        GEN2:
100364  376 030                 CMP     A,#30
100366  362 033 201             JUMP,PL GEN3
100371  032                     LOAD    A,(DE)
100372  366 200                 OR      A,#200          ; C <-- 0
100374  022                     LOAD    (DE),A

                        ; Met sur le stack du travail pour plus tard.
                        ; "A" est plus petit que zero, il faudra donc
                        ; (plus tard) décrémenter les 8 bytes situes
                        ; autour de l'adresse donnée dans "DE".

100375  325                     PUSH    DE
100376  365                     PUSH    AF


                        ; Tue une cellule dans "VIEW"

                        GEN4:
100377  161                     LOAD    (HL),C          ; LOAD (HL),#SPACE
```

```
                        ; Pointe la valeur suivante dans "VIEW" et
                        ; dans "VOISIN".

                        GEN5:
100400  023                     INC     DE
100401  043                     INC     HL

                        GEN1:
100402  032                     LOAD    A,(DE)
100403  267                     OR      A,A
100404  362 023 201             JUMP,PL GEN5
100407  270                     COMP    A,B             ; COMP  A,#'200
100410  040 365                 JUMP,NE GEN4
100412  346 177                 AND     A,#200-1        ; C <-- 0
100414  022                     LOAD    (DE),A

                        ; Met sur le stack du travail pour plus tard.
                        ; "A" est plus grand que zero, il faudra donc
                        ; (plus tard) incrémenter les 8 bytes situes
                        ; autour de l'adresse donnée dans "DE".

100415  325                     PUSH    DE
100416  365                     PUSH    AF

                        ; Crée une cellule dans "VIEW"

100417  066 117                 LOAD    (HL),#'O
100421  030 355                 JUMP    GEN6


                        ; Est-ce que "a" est egal à 2 ou à 3 ?

                        GEN6:
100423  037                     RRC     A
100424  075                     DEC     A
100425  040 335                 JUMP,NE GEN2

                        ; Crée une cellule dans "VIEW"

100427  066 117                 LOAD    (HL),#'O
100431  030 345                 JUMP    GEN5


                        ; Effectue le travail stocke sur le stack.

                        GEN3:
100433  001 276 377             LOAD    BC,#-(NCAR+2)
100436  021 176 000             LOAD    DE,#(2*NCAR)-2
100441  303 047 201             JUMP    GEN8

                        GEN7:
100444  315 104 201             CALL    DEAD1
                        GEN8:
100447  361                     POP     AF
100450  330                     RET.CS                  ; end ?
100451  341                     POP     HL
100452  372 044 201             JUMP,MI GEN7
100455  315 062 201             CALL    ALIVE1
100460  030 365                 JUMP    GEN8


                        ;       1   2   3
                        ;
                        ;       4  (hl)  5
                        ;
                        ;       6   7   8

                        ALIVE1:
100462  053                     DEC     HL              ; déplacement en 4
100463  064                     INC     (HL)
100464  043                     INC     HL              ; déplacement en 5
100465  043                     INC     HL
100466  064                     INC     (HL)
100467  011                     ADD     HL,BC           ; déplacement en 1
100470  064                     INC     (HL)
100471  043                     INC     HL              ; déplacement en 2
100472  064                     INC     (HL)
100473  043                     INC     HL              ; déplacement en 3
100474  064                     INC     (HL)
100475  031                     ADD     HL,DE           ; déplacement en 6
100476  064                     INC     (HL)
100477  043                     INC     HL              ; déplacement en 7
100500  064                     INC     (HL)
100501  043                     INC     HL              ; déplacement en 8
100502  064                     INC     (HL)
100503  311                     RET


                        DEAD1:
100504  053                     DEC     HL              ; déplacement en 4
100505  065                     DEC     (HL)
100506  043                     INC     HL              ; déplacement en 5
100507  043                     INC     HL
100510  065                     DEC     (HL)
100511  011                     ADD     HL,BC           ; déplacement en 1
100512  065                     DEC     (HL)
100513  043                     INC     HL              ; déplacement en 2
100514  065                     DEC     (HL)
100515  043                     INC     HL              ; déplacement en 3
100516  065                     DEC     (HL)
100517  031                     ADD     HL,DE           ; déplacement en 6
100520  065                     DEC     (HL)
100521  043                     INC     HL              ; déplacement en 7
100522  065                     DEC     (HL)
100523  043                     INC     HL              ; déplacement en 8
100524  065                     DEC     (HL)
100525  311                     RET


                        ;  ATTENTION !!!
                        ;
                        ; Les routines "ALIVE" et "DEAD" peuvent détruire
                        ; une ligne (NCAR bytes) avant ou après les NLI*NCAR
                        ; bytes de "VOISIN".

                        ; Incrémente tous les bytes situes autour du byte
                        ; dont la coordonnée (dans les NLI*NCAR positions
                        ; de "VOISIN") est donnée dans HL.

                        ALIVE:
100526  001 276 377             LOAD    BC,#-(NCAR+2)
100531  021 176 000             LOAD    DE,#(2*NCAR)-2
100534  315 062 201             CALL    ALIVE1
100537  311                     RET

                        ; Décrémente tous les bytes situes autour du byte
                        ; dont la coordonnée (dans les NLI*NCAR positions
                        ; de "VOISIN") est donnée dans HL.

                        DEAD:
100540  001 276 377             LOAD    BC,#-(NCAR+2)
100543  021 176 000             LOAD    DE,#(2*NCAR)-2
100546  315 104 201             CALL    DEAD1
100551  311                     RET



                        ; Variables en RSX :

                        POINT:
100552  000000                .BLKW   1



100000                  .END    LIFE
```