JDN

6

SMAKY6

INTRODUCTION A LA PROGRAMMATION AVEC SMILE

Octobre 1980

EPSITEC-system sa

INTRODUCTION A LA PROGRAMMATION DU SYSTEME SMAKY AVEC SMILE

Octobre 1980

Le but de cette notice est d'apprendre à programmer en langage d'assemblage et de se familiariser aux caractéristiques des microprocesseurs, en particulier du Z80 utilisé comme processeur du SMAKY6. Elle contient des exemples qui illustrent les explications, et des exercices qui sont à exécuter directement sur un SMAKY.

Un SMAKY avec un système 1.6 ou ultérieur est nécessaire.
La variante SMILE 2.5 est utilisable, mais les révisions 3.4 et ultérieures sont préférables.

1ère partie: INTRODUCTION GENERALE

1. ARCHITECTURE

Le SMAKY6 contient, en plus du processeur, une mémoire morte (ROM) contenant un programme de dialogue initial et de nombreux sous-programmes facilitant la programmation. Des interfaces permettent de lire un clavier, d'afficher des caractères sur un écran et de communiquer avec des périphériques extérieurs (COBUS, lecteur, perforateur, etc.). La mémoire vive est chargée par les programmes utilisateur.

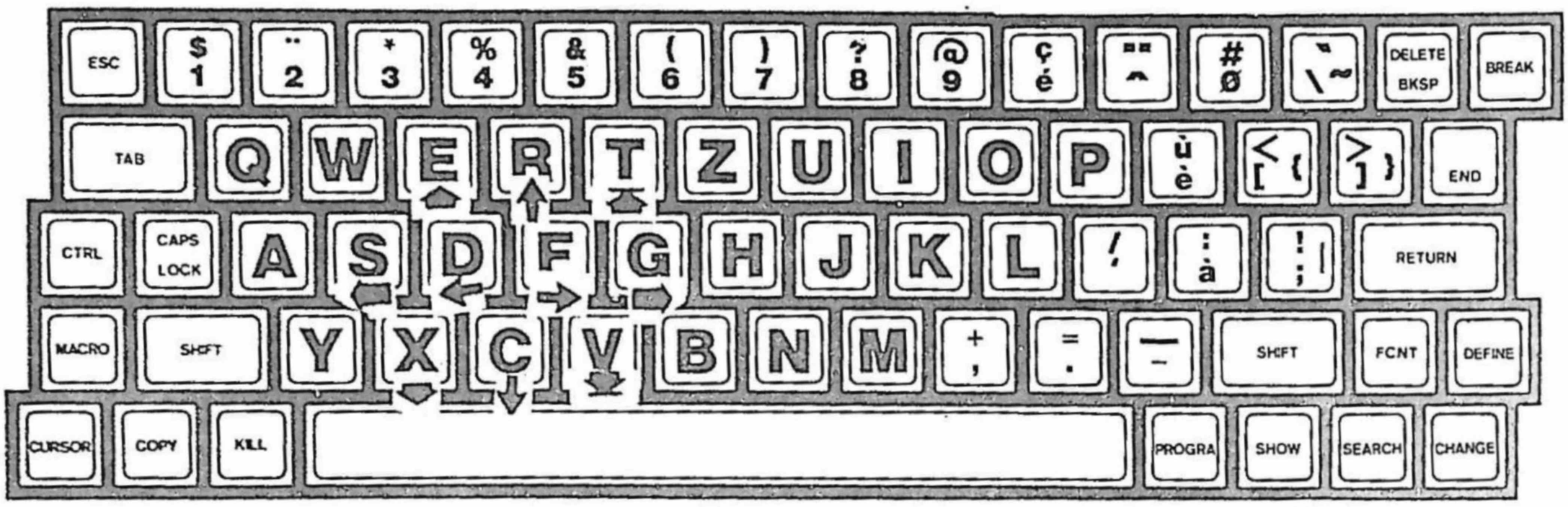
Pour pouvoir faire les exemples et exercices qui suivent, il faut tout d'abord charger SMILE dans un SMAKY; il faut procéder de différentes façons selon la configuration à disposition.

1) A l'EPFL, après un RESET ou un LOGOUT, taper Clé d'accès↵ puis SMILE↵
   La clé d'accès est un nom, un surnom (connu du système) ou un nom de répertoire.

2) Sur un SMAKY avec floppy, taper SMILE↵ après avoir bootstrapé.

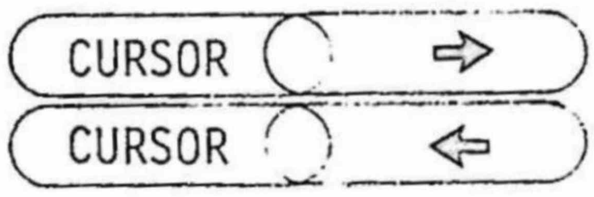
2. SMILE

Le programme SMILE, une fois chargé en mémoire, permet d'éditer, d'assembler et de démarrer des programmes. Dans le mode d'édition, le clavier-écran se comporte comme une machine à écrire, avec des possibilités supplémentaires de correction grâce aux touches fonction supplémentaires (Maintenir la touche CAPS LOCK enfoncée pour taper les programmes). Par exemple, pour déplacer le curseur dans un texte tapé sur l'écran, on maintient CURSOR pressé et on donne la direction et l'amplitude du déplacement avec l'une de 10 touches repérées par des flèches.



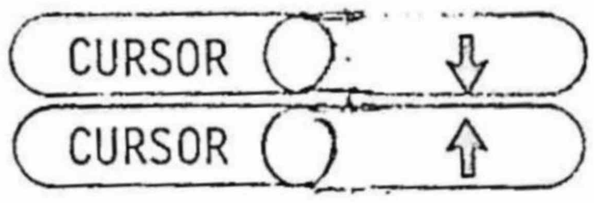
avance le pointeur d'un caractère

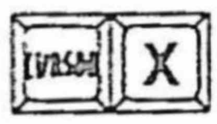
recule le pointeur d'un caractère



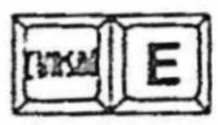
descend le pointeur d'une ligne

remonte le curseur d'une ligne

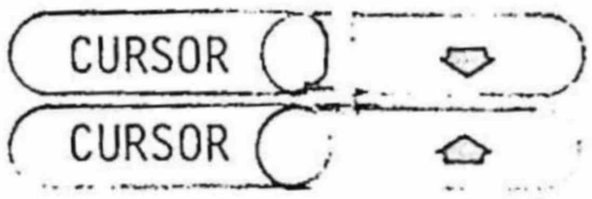




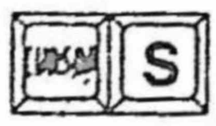
descend le pointeur de 4 lignes



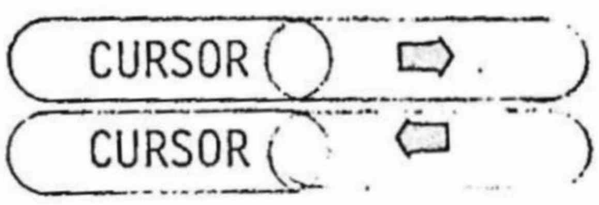
remonte le pointeur de 4 lignes

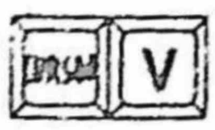


avance le pointeur d'un mot

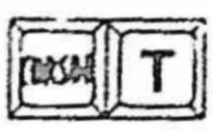


recule le pointeur d'un mot

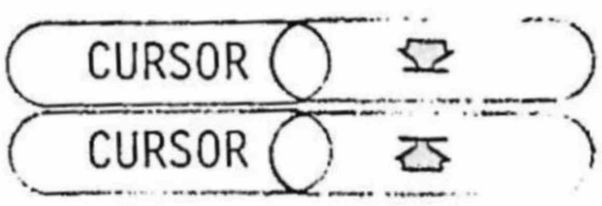




place le pointeur à la fin du texte



place le pointeur au début du texte



EXERCICE 2.1 *Taper le modèle ci-contre. Utiliser la touche TAB pour avancer d'une colonne, la touche BS. pour effacer le caractère précédent, la touche DEL pour effacer le caractère se trouvant sous le curseur*

EXERCICE D'EDITION

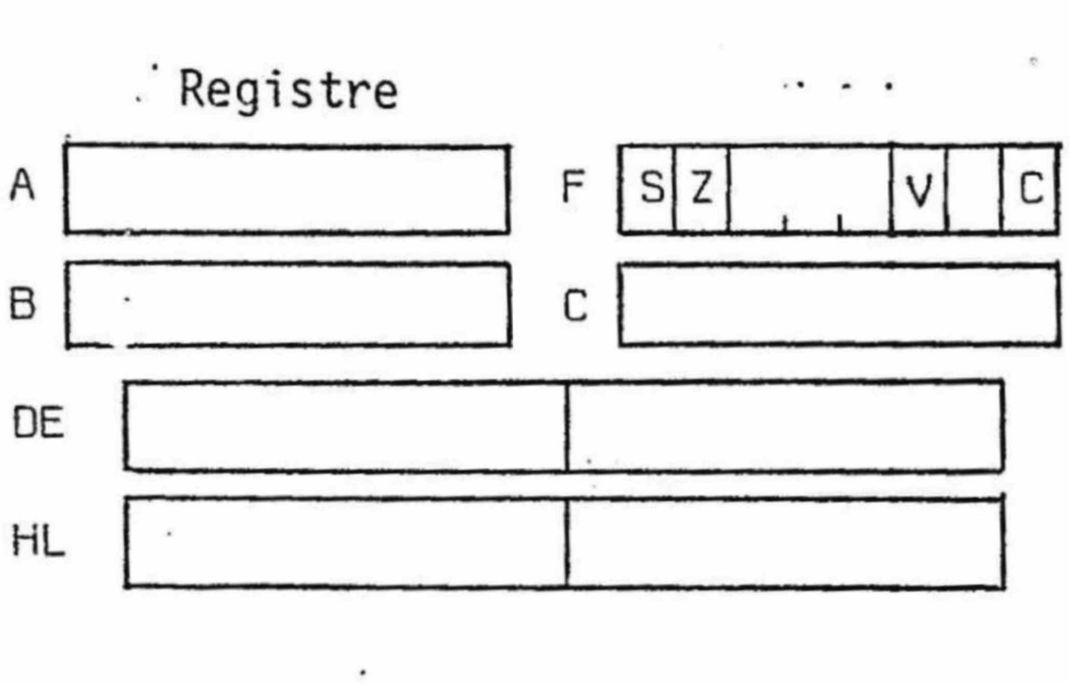
A = 2
B = 3
C = 4

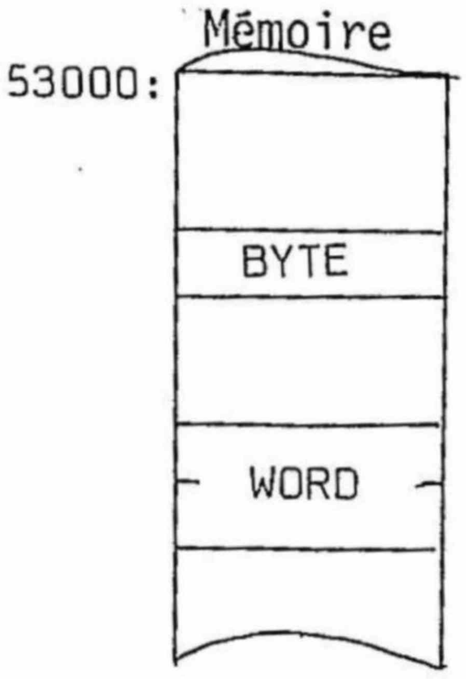
(Taper (A) (TAB) (=) (SPACE) (2) (RETURN)...)

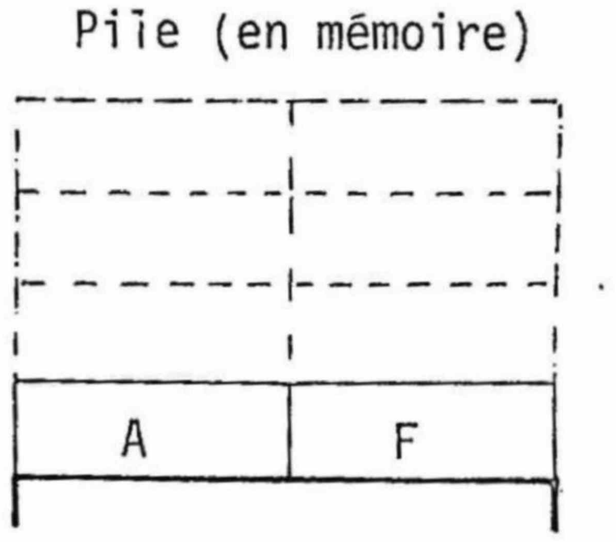
3. MODELE SIMPLIFIE DU Z80

Le SMAKY6 comporte l'équivalent de 300 000 transistors; nous nous contenterons donc en première étape d'un modèle très simplifié, ce qui est possible grâce aux sous-programmes (appels) contenus dans la ROM.

Le modèle simplifié du processeur Z80 est le suivant: il a trois registres A, B et C de 8 bits, 2 registres DE et HL de 16 bits (considérés parfois comme 4 registres D E H L de 8 bits), et un registre de fanions (flags) F, contenant les indicateurs C (carry), Z(equal, equal to zero), S(sign) et V(arithmetic overflow).

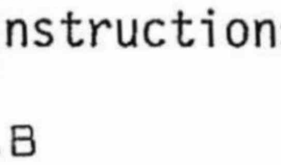






La mémoire contient le programme et les données, qui sont soit des octets (byte), soit des sedets (word). Une zone mémoire particulière joue le rôle d'une pile, sur laquelle on peut "poser" ou "pousser" (push) le contenu d'une paire de registres, pour le reprendre (pop) plus tard.

Quelques instructions typiques sont:

LOAD	A,B	chargement de A avec Ie contenu de B (ne modifie pas B)
LOAD	A,# 5	chargement de A avec la valeur 5 (octale)
LOAD	A,5	chargement de A avec la valeur contenue dans la 5e cellule mémoire (cette adresse est en ROM; elle contient 17)
ADD	A,C	addition de A et C, résultat dans A. Carry mis à 1 si dépassement (nombres logiques) (CS) Z mis à 1 si le résultat est nul (EQ) S mis à 1 si le signe (bit de poids fort) est à 1 (SS) V mis à 1 si dépassement (nombres arithmétiques) (VS)
COMP	A,# 2	compare le contenu de A et la valeur 2. Z mis à 1 si A=2 (EQ) C mis à 1 si A<2 (LO) (A contient le nombre logique) S mis à 1 si A<2 (SS) (A contient un nombre arithmétique)
RRC	A	 rotation de A à travers le Carry (modifie C,Z,S)
INC	A	ajoute 1 à A (modifie C,Z,S)
JUMP	NEXT	saut à l'instruction d'étiquette NEXT
JUMP,EQ	LOOP	saut à l'instruction d'étiquette LOOP si le fanion Z vaut 1 (résultat précédent égal à zéro ou comparaison égale)
DECJ,NE	B,LOOP	est équivalent à { DEC B ; JUMP,NE LOOP Décompte B et saute à LOOP si le résultat est ≠ 0 (Non Equal), continue si B=0

PUSH	AF	sauvetage des 2 registres A et F sur la pile
POP	BC	transfert du sommet de la pile dans BC
EX	HL,DE	échange des contenus des registres HL et DE

4. SRUCTURE D'UN PROGRAMME

Les instructions seront alignées dans un programme dont la structure générale est la suivante:

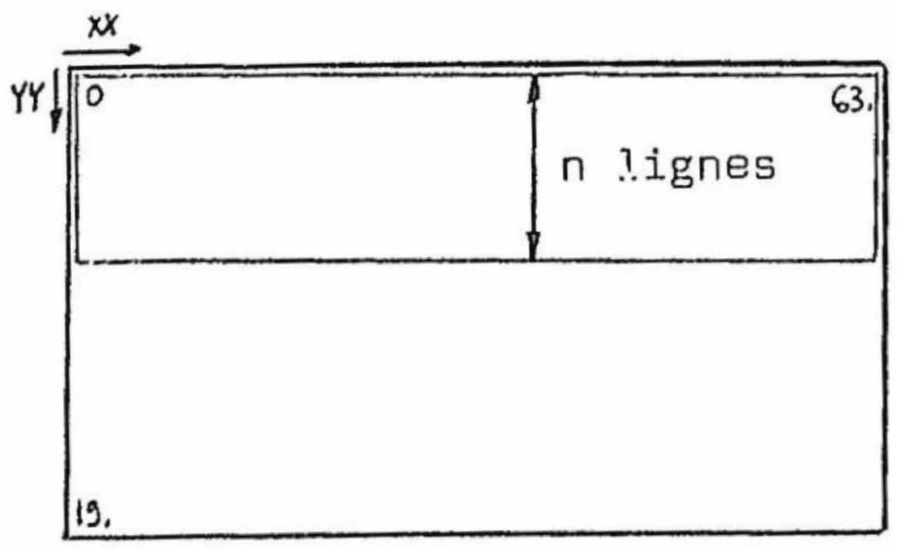
	TITRE Indicatdions pour l'assembleur		
Commentaires			
Etiquette:	Code (mnémonic)	Opérandes	;commentaires
Indication de fin pour l'assembleur			

Exemple 4.1.

```
          .TITLE  TEST              ;KP 29.5.80
          .PROC   Z80
          .REF    SM6
;boucle infinie
DEB:      LOAD    A,#1              ;A contient successivement 1,
DEB2:     ADD     A,A               ;2,3,,6,7,...,176,177,376,377,376,377, etc
          INC     A                 ;mais on ne voit pas son contenu
          JUMP    DEB2

          .END    DEB
```

5. UTILISATION DE L'ECRAN ET DU CLAVIER

L'écran est une grille de 20. x 64. caractères que l'on doit initialiser et sur laquelle on peut déposer des caractères de façon séquentielle ou aléatoire



L'appel .W ?DICAR permet d'afficher le caractère dont le code a été préparé dans A.

Par exemple, pour afficher l'alphabet sur la première ligne de l'écran, le programme est le suivant:

Exemple 5.1.

```
        .TITLE  ALPHA
        .PROC   Z80
        .REF    SM6

;affiche l'alphabet
NBLIGNES = 10.                  ;10. lignes pour la fenêtre d'affichage

ALPHA:  LOAD    C,# NBLIGNES
        .W      ?IDIS           ;initialisation de la fenêtre d'affichage
        LOAD    A,# 'A          ;'A est le code ASCII de A (101 octal)
ALP2:   .W      ?DICAR          ;affichage sur l'écran
        INC     A
        COMP    A,#'Z+1         ;alphabet fini?
        JUMP,NE ALP2
        TRAP                    ;reprendre le contrôle avec une
                                ;touche quelconque
        .END    ALPHA
```

Le modèle du clavier est simple: chaque touche pressée est mémorisée. On peut lire son code avec l'appel ?GETCAR, qui donne dans A le code ASCII de la touche.

Par exemple, pour simuler une machine à écrire, il suffit d'écrire:

Exemple 5.2.

```
        .TITLE  MACH
        .PROC   Z80
        .REF    SM6

;machine à écrire simple

(NLI    = 20.           ;hauteur de l'écran, déjà défini dans .REF SM6)

MACH:   LOAD    C,# NLI
        .W      ?IDIS
MA2:    .W      ?GETCAR, ?DICAR ;équivalent à {.W ?GETCAR
        JUMP    MA2             {.W ?DICAR

        .END    MACH    ATTENTION: toujours taper un RETURN pour
                                   finir la dernière ligne
```

REMARQUE: dans cette notice, on écrira NLI pour le nombre de lignes (=20.). On peut aussi écrire LINES .

6. ASSEMBLAGE ET EXECUTION

Les programmes précédents peuvent être tapés sous le contrôle de SMILE, puis assemblés par l'ordre

PROGRA( ) Z (maintenir PROGRA, agir sur Z).

Après assemblage, taper un espace pour reprendre le contrôle et si l'assemblage a été correct, exécuter avec l'ordre

PROGRA( ) V

Reprendre le contrôle avec NMI si c'est une boucle infinie, suivi d'un espace.

EXERCICE 6.1 *Taper les deux exemples précédents et vérifier leur exécution.*

Faire SHOW( ) *i* — SHOW( ) *i* *pour travailler dans les zones d'édition* $i=0,1,2,\ldots$ *et conserver dans la mémoire tous les programmes tapés.*

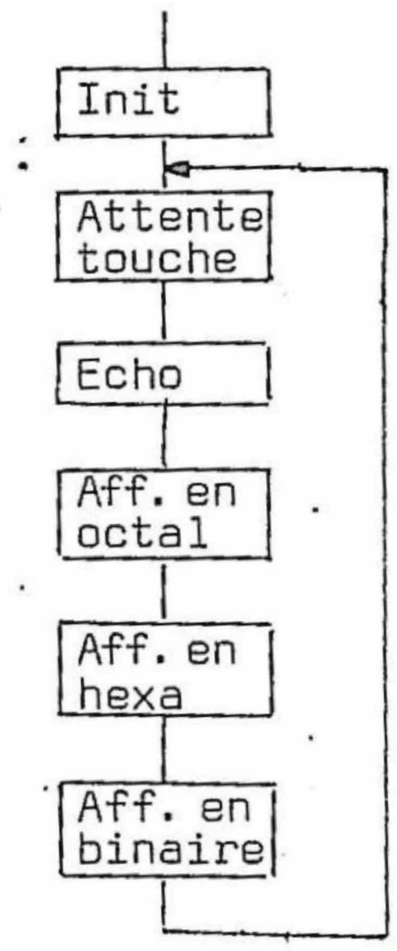
EXERCICE 6.2 *Modifier le programme de machine à écrire pour que la touche = efface l'écran.*

2ème partie: APPELS PRINCIPAUX DU SYSTEME SMAKY6

7. APPELS DE MISE EN PAGE ET AFFICHAGE DE NOMBRES

?SPACE	= 21347	"Affiche" un espace (ne modifie pas A)
?TAB	= 50747	"Affiche" un tabulateur (passe en colonne modulo 8.)
?RETURN	= 21747	Passe au début de la ligne suivante (ne modifie pas A)
?DEL	= 54347	Efface le précédent caractère (comme la touche DELETE)
?AFBIN	= 34347	Affiche le contenu de A en binaire
?AFOCA	= 35347	Affiche le contenu de A en octal (en fait le Carry et A càd 9 bits)
?AFHEX	= 34747	Affiche le contenu de A en BCD ou hexadécimal
?AFOHL	= 35747	Affiche le contenu de HL en octal
?AFXHL	= 54747	Affiche le contenu de HL en BCD ou hexa

Exemple 7.1




```
        .TITLE  AFCODE  ;AZ 23 mai 1980
        .PROC   Z80
        .REF    SM6


AFCODE: LOAD    C,# NLI/2     ;moitié de l'écran initialisée
        .W      ?IDIS

AF2:    .W      ?GETCAR
        .W      ?DICAR, ?SPACE, ?AFOCA,  ?SPACE, ?AFHEX, ?SPACE, ?AFBIN
        JUMP    AF2
        . END   AFCODE
```

EXERCICE 7.1 Ecrire en assembleur SM6 le programme qui s'écrit

en BASIC:

```
10 FOR N=1 TO 10
20 PRINT N, 2*N, 3*N
30 NEXT N
40 STOP
```

en PASCAL:

```
PROGRAM TABLEAU (ECRAN,OUTPUT)
VAR NBRE: INTEGER;
BEGIN
  FOR NBR1:=1 TO 10
    DO WRITELN (NBRE,2*NBRE,3*NBRE);
END.
```

8. APPELS DE LECTURE DE NOMBRES

?IN0C = 33747 — Attend un nombre octal et construit sa valeur binaire dans HL. Une touche non chiffre termine l'appel, qui revient avec le nombre dans HL, et la dernière touche (terminateur) dans A.

?INDEC = 32747 — Attend un nombre décimal et construit sa valeur BCD dans HL.
Seuls les 4 derniers chiffres sont conservés.

?INHEX = 33347 — Attend un nombre hexadécimal et construit sa valeur binaire dans HL. Les touches A...F sont considérées comme des chiffres

EXEMPLE 8.1 Afficher le caractère ASCII correspondant à un code tapé en octal au clavier :

```
        .TITLE  AFCODE  ;MA 17.3.80
        .PROC   Z80
        .REF    SM6

;code tapé en octal


AFCODE: LOAD    C,# NLI
        .W      ?IDIS
AFCO2:  .W      ?INOC,  ?TAB        ;INOC donne le nombre dans HL
                                    ;H n'a pas de signification
        LOAD    A,L                 ;DICAR veut le caractère dans A
        .W      ?DICAR, ?RETURN
        JUMP    AFCO2
        . END   AFCODE
```

Exercice 8.1. *Ecrire le programme qui attend un nombre décimal de 4 digits et l'affiche en binaire 16 bits.*

Exercice 8.2. *Ecrire le même programme qu'avant, mais séparer chaque groupe de 4 bits par un espace.*
Indication: décaler le contenu de HL bit par bit dans le Carry avec l'instruction ADD HL,HL. Si le Carry vaut Ø, afficher 'Ø; s'il vaut 1, afficher '1.
Tous les 4 bits, afficher un espace.

9. REMARQUE IMPORTANTE

Une instruction, un appel, une partie de programme ont un effet sur les registres, positions mémoire et périphériques qui doivent être présents à l'esprit, et qu'il faut bien documenter dès que la fonction n'est pas courante.

Exemple 9.1.

```
        Instruction  ADD A,B

;in     A,B opérandes (8 bits)
;out    A résultat (8 bits)
;       CS (C=1) carry set si dépassement de capacité
;       EQ (Z=1) si résultat nul
;       SS (S=1) si bit de poids fort à un
;mod    F,A
```

Exemple 9.2.

```
        Appel   ?INOC

;in     - (touches du clavier)
;out    HL nombre tapé
;       A  dernière touche non chiffre
;       C  nombre de chiffres tapés
;       EQ si le nombre est vide (pas de touche chiffre tapée)
;mod    F, A, B, C, HL, écran
```

Exemple 9.3.

```
        Programme AFCODE

;in     - (touches du clavier)
;out    écran effacé, codes tapés et caractères correspondants alignés
;mod    F, A, B, C, HL, écran
```

Exercice 9.1. Documenter avec in/out/mod la partie de programme qui affiche HL en binaire avec des tranches de 4 bits, telle qu'elle a été écrite dans l'exercice 8.1.

10. APPELS D'AFFICHAGE DE TEXTE

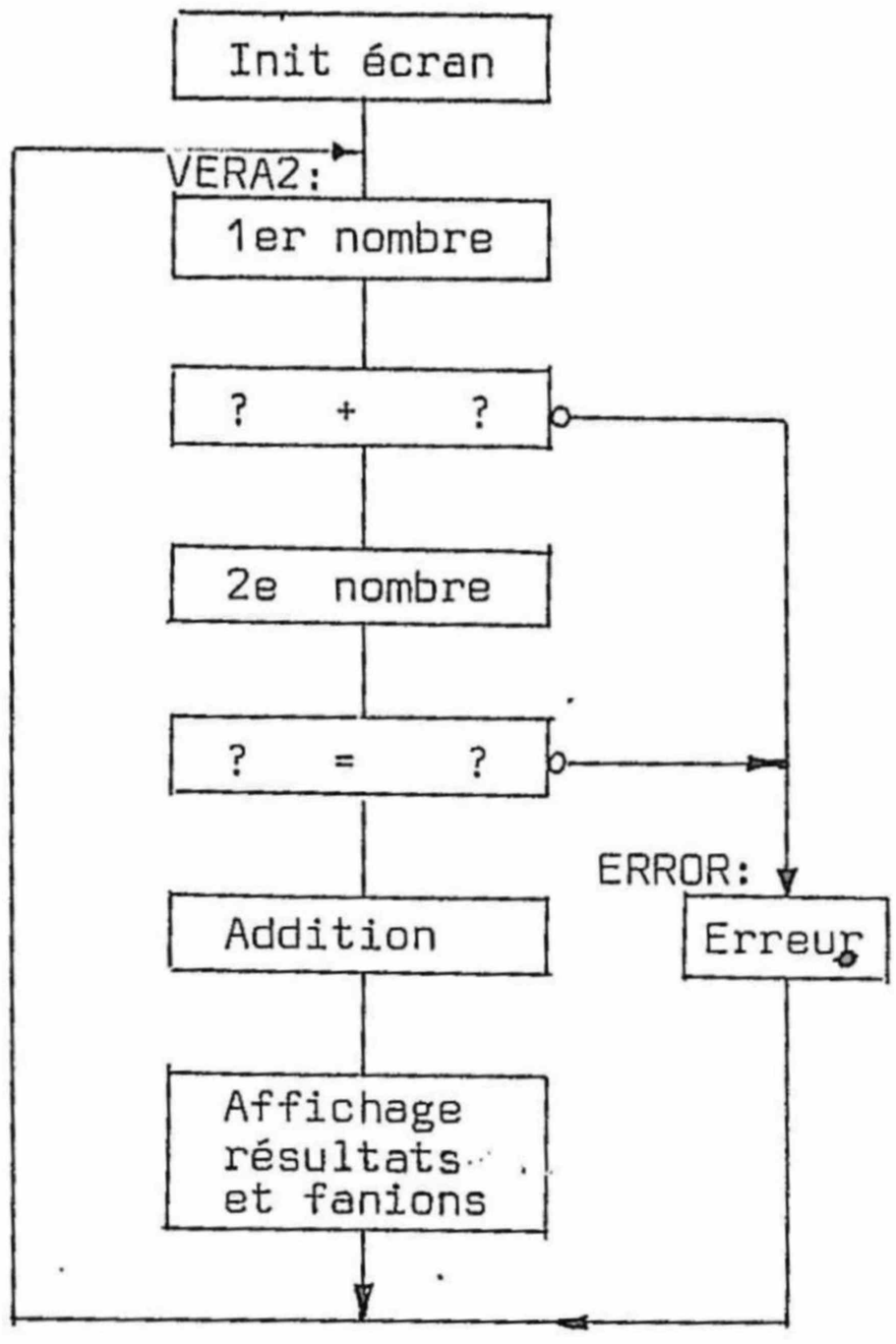
On peut afficher un texte préparé en mémoire par l'appel ?TEXT, suivi de l'adresse (étiquette) du texte en mémoire. Les textes sont déclarés soit au début, soit à la fin du programme.

Exemple 10.1

```
TX:     LOAD    C,# NLI
        .W      ?IDIS
        .W      ?TEXT,TEX1
        .W      ?INOC
        .W      ?TEXT,TEX2
        TRAP
TEX1:   .ASCIZ  "DONNEZ MOI UN NOMBRE:"
TEX2:   .ASCIZ  "<CR>MERCI<CR>"
```

Exemple 10.2 On veut vérifier l'instruction d'addition
ADD A,D

Organigramme;




```
        .TITLE  VERADD
        .PROC   Z80
        .REF    SM6

;vérification de l'addition

VERADD: LOAD    C,# NLI
        .W      ?IDIS

VERA2:  .W      ?TEXT, ADTEX1, ?INOC
        COMP    A,# '+          ;seul le + est reconnu comme termi-
        JUMP,NE ERROR           ;nateur du 1er nombre
        LOAD    D,L             ;le 1er opérande est sauvé dans D,
        .W      ?INOC           ;registre non détruit par les appels
        COMP    A,# '=          ;suivants
        JUMP,NE ERROR           ;seul = est accepté comme terminateur
                                ;du 2e  nombre
        LOAD    A,L             ;2ème opérande dans A
        ADD     A,D             ;addition
        .W      ?AFOCA          ;affichage du résultat dans A

        PUSH    AF              ;l'instruction LOAD  A,F n'existant pas
        POP     BC              ;il faut faire un détour par la pile en
        LOAD    A,C              mémoire
        .W      ?TEXT,ADTEX2, ?AFBIN, ?RETURN ;affichage des flags en binaire
        JUMP    VERA2

ERROR:  .W      ?TEXT,ADTEX3
        JUMP    VERA2

ADTEX1: .ASCIZ  " Tapez l'opération : "
ADTEX2: .ASCIZ  " Fanions SZ-----C : "
ADTEX3: .ASCIZ  " Tapez le bon terminateur! <CR>"

        .END    VERADD
```

Remarque

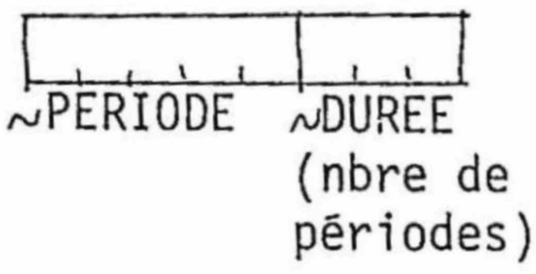
L'appel ?DITEX peut aussi être utilisé pour afficher des textes, mais il est moins pratique pour un texte simple, car il faut préparer l'adresse du texte dans HL, et le registre HL est modifié.

```
PUSH    HL
LOAD    HL,# ADTEX1     } est équivalent à    .W    ?TEXT.,ADTEX1
.W      ?DITEX
POP     HL
```

Exercice 10.1. Ecrire un programme qui attend une séquence de 3 touches prédéfinies, par exemple PTQ et affiche dans ce cas seulement "votre coffre est ouvert", et dans les autres cas "j'appelle la police".

11. BRUITAGES

L'appel ?PLAY est semblable à l'appel ?DITEX, mais au lieu d'afficher le texte pointé par HL, il joue des codes, les mots mémoire étant interprétés de façon simplifiée :



Exemple 11.1

```
        .TITLE  MUS
        .PROC   Z80
        .REF    SM6

MUS:    LOAD    HL,# MOR            ;adresse du morceau
        .W      ?PLAY
MU2:    JUMP    MU2                 ;attente: taper NMI pour retourner à SMILE
MOR:    .B      DO, RE, DO, SIL, MI, SOL
        .B      Ø                   ;un zéro doit terminer le morceau

        .END    MUS
```

Remarques

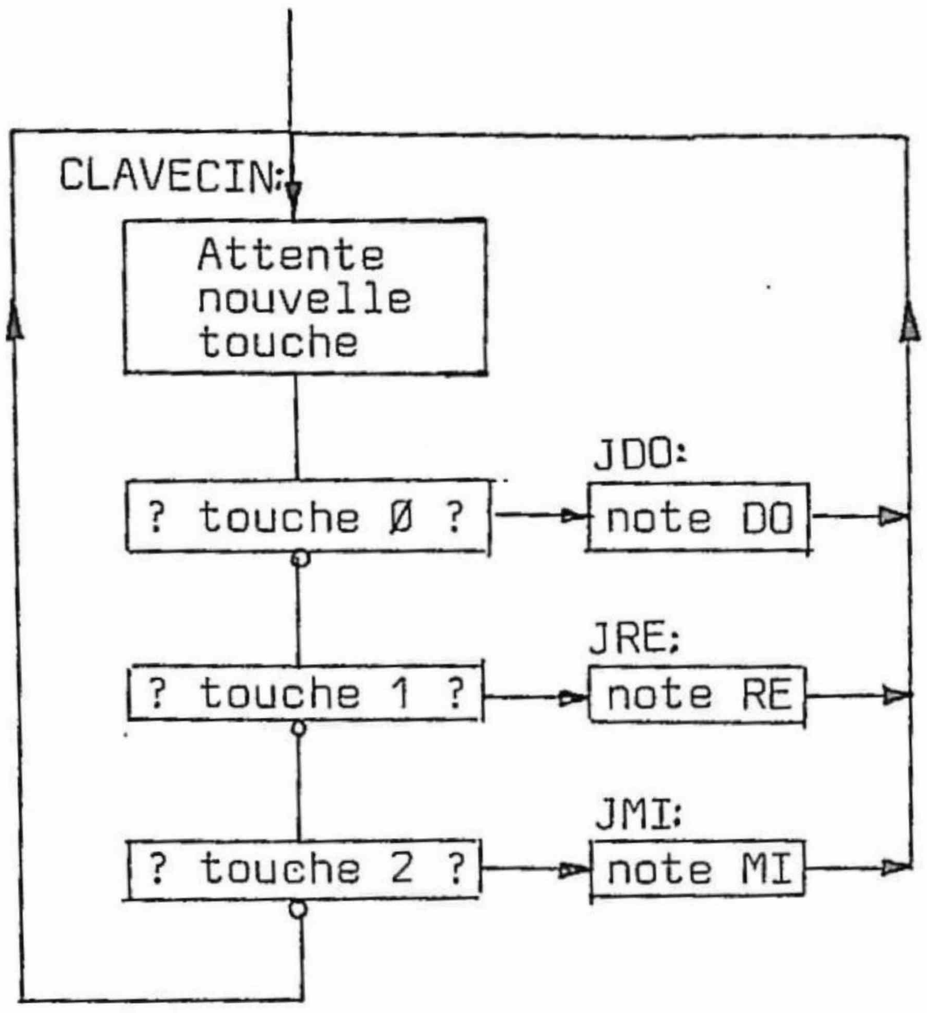
Les appels sont des mots de 16 bits qui sont interprétés par le processeur comme des appels de sous-programme. .W doit être utilisé pour mettre dans le programme un mot de 16 bits. Les notes sont par contre des mots de 8 bits, .B doit être utilisé dans ce cas.

```
.ASCIZ  "TRUC"  est une facilité pour écrire:
.B      'T, 'R, 'U, 'C, Ø           (un zéro termine le mot, aucune lettre
                                    n'a le code Ø)
```

L'appel ?BEEP permet de ne jouer qu'une note, dont le code a été préparé dans A.

Exemple 11.2. Simulation d'un clavecin; les touches 0-1-2 jouent DO-RE-MI.

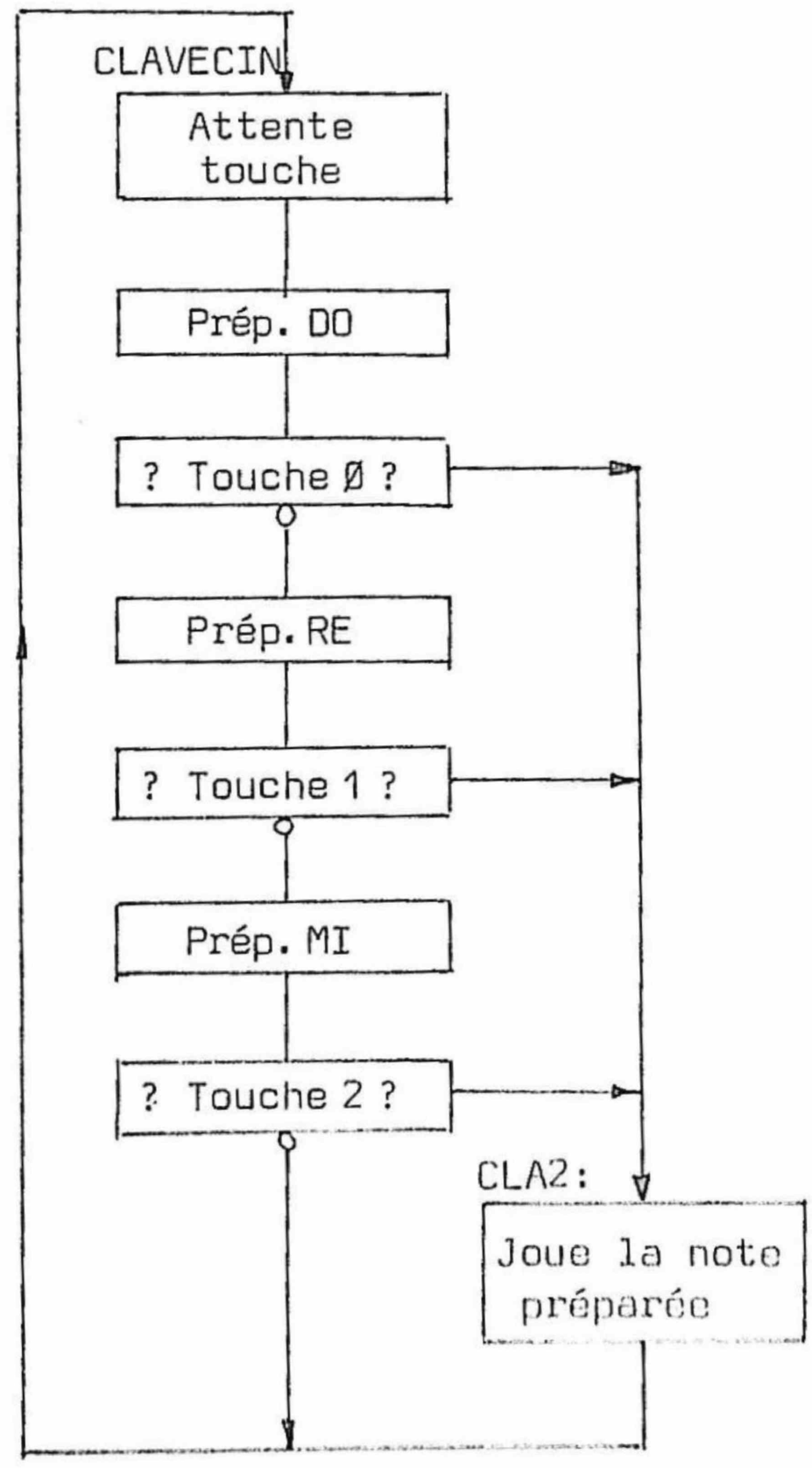
```
        .TITLE  CLAVECIN
        .PROC   Z80
        .REF    SM6

;clavecin très simplifié

CLAVECIN: .W        ?GETCAR
          COMP      A,# 'Ø
          JUMP,EQ   JDO
          COMP      A,# '1
          JUMP,EQ   JRE
          COMP      A,# '2
          JUMP,EQ   JMI
          JUMP      CLAVECIN

JDO:      LOAD      A,# DO
          .W        ?BEEP
          JUMP      CLAVECIN

JRE:      LOAD      A,# RE
          .W        ?BEEP
          JUMP      CLAVECIN

JMI:      LOAD      A,# MI
          .W        ?BEEP
          JUMP      CLAVECIN

          . END     CLAVECIN
```



Ce programme peut être allégé

Exemple 11.3.

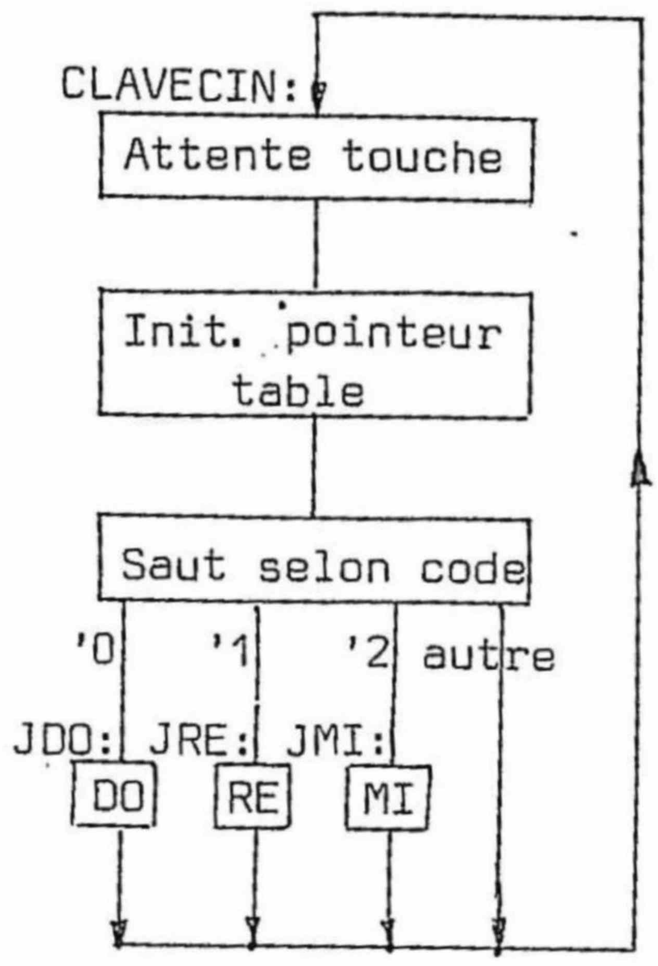
```
CLAVECIN: .W        ?GETCAR
          LOAD      B,# DO
          COMP      A,# 'Ø
          JUMP,EQ   CLA2
          LOAD      B,# RE
          COMP      A,# '1
          JUMP,EQ   CLA2
          LOAD      B,# MI
          COMP      A,# '2
          JUMP,NE   CLAVECIN

CLA2:     LOAD      A,B
          .W        ?BEEP
          JUMP      CLAVECIN
```



Ce programme est encore très lourd. Il peut être simplifié en utilisant l'appel ?JUMPCAR = 30747, qui permet de préparer une table de correspondance entre les touches et les adresses de programme à exécuter chaque fois.

Exemple 11.4.

```
CLAVECIN: .W      ?GETCAR
          LOAD    DE,# TABLE
          .W      ?JUMPCAR
          JUMP    CLAVECIN   ;autres touches
                             ;ignorées
JDO:      LOAD    A,# DO
JD2:      .W      ?BEEP
          JUMP    CLAVECIN

JRE:      LOAD    A,# RE
          .W      ?BEEP      } JUMP  JD2
          JUMP    CLAVECIN   }

JMI:      LOAD    A,# MI
          .W      ?BEEP      } JUMP  JD2
          JUMP    CLAVECIN   }

TABLE:    .BW     'Ø,JDO  ;on écrit .BW car la table contient successivement
          .BW     '1,JRE  ;un byte (le code de la touche) et un word (l'a-
          .BW     '2,JMI  ;dresse de la routine)
          .B      Ø
```



L'appel ?BUZZ fait un buzz unique et ne modifie aucun registre. Le code ASCII BEL=7 a le même effet. On peut donc l'introduire dans un texte, par exemple pour attirer l'attention de l'utilisateur au moment de lui demander un nombre:

Exemple 11.5

```
          ...                                          ...
          .W      ?TEXT,TEX1,?BUZZ,?INOC    ou        .W      ?TEXT,TEX1,?INOC
          ...                                          ...
TEX1:     .ASCIZ  "1er NOMBRE?"              TEX1:     .ASCIZ  "1er NOMBRE?<BELL>"
```

12. APPELS ARITHMETIQUES

Quelques opérations non disponibles sous forme d'instructions du Z80 ont été ajoutées comme appels.

```
?BINBCD     ;in   HL
= 52347
            ;out  AHL

            ;mod  F A HL
```

HL 0000100000000000 nbre binaire (2^{10})

AHL 0000 0000 0001 0000 0010 0100 (1024)

```
?BCDBIN     ;in   HL
= 52747
            ;out  HL

            ;mod  HL
```

HL 1001 1001 1001 1001 nbre BCD (9999)

HL 0010011100001111 nbre binaire (23417)

```
?COMPHLDE   ;in   HL DE
 = 27347
            ;out  Carry,EQ,Sign

            ;mod  F
```

HL [ ] DE [ ]

F [ZS . . . C]

CS=LO si HL < DE } (logical)
CC=HS si HL ≥ DE }
EQ=ZS si HL = DE
SS si HL < DE (arith)

```
?MUL        ;in   HL
= 27747
            ;out  HLDE

            ;mod  F A DE HL
```

HL [Terme additif] + DE [Multiplicande] *BC [Multiplicateur]

HLDE [Produit] A [= Ø]

```
?DIV        ;in   HL DE
= 30347
            ;out

            ;mod  F A DE HL
```

[Dividende] :BC [Diviseur]

DE [quotient] HL [reste] F [C] A [=0 si carry=0]

CS si erreur (dépassement de capacité)

Exemple 12.1. Vérification des routines de conversion.

```
            .TITLE  VERCONV
            .PROC   Z80
            .REF    SM6

;convertit un nombre

VERCONV:LOAD    C,# NLI
            .W      ?IDIS

VER2:       .W      ?INDEC, ?TAB, ?BCDBIN, ?AFOHL, ?TAB, ?BINBCD, ?AFXHL, ?RETURN
            JUMP    VER2
            . END   VERCONV
```

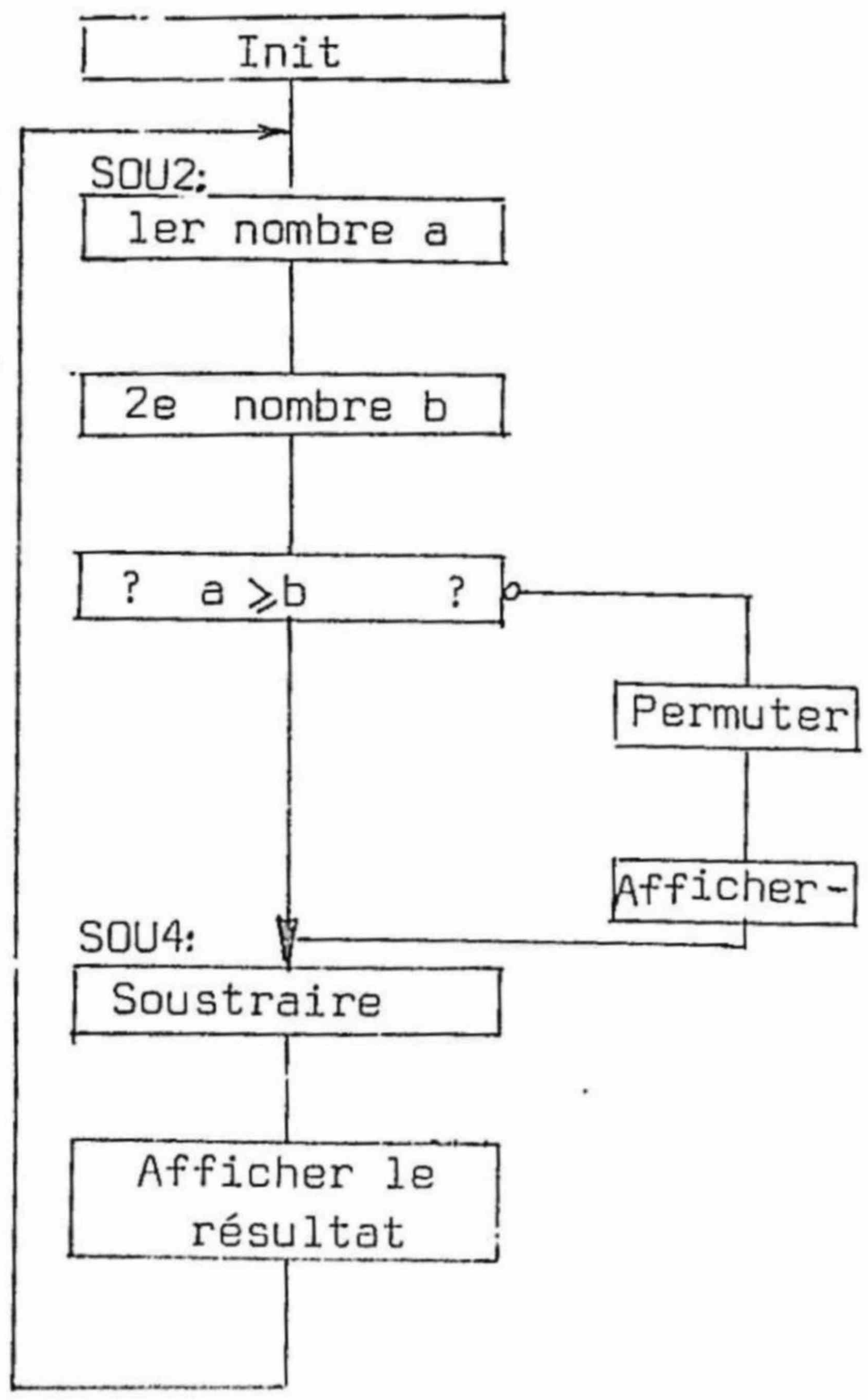
Exemple 12.2. Soustraction de deux nombres positifs, résultat donné en complément à 2.

```
          .TITLE SOUSTRACTION
          .PROC  Z80
          .REF   SM6

SOUS:     LOAD    C,# NLI
          .W      ?IDIS
SOU2:     .W      ?INOC
          EX      HL,DE                ;sauvetage 1er opérande dans DE
          .W      ?INOC
          EX      HL,DE                ;échange pour avoir le 1er opérande dans HL
          OR      A,A          }SUB HL,DE (instruction qui n'existe malheureusement pas)
          SUBC    HL,DE        }
          .W      ?SPACE, ?AFOHL, ?RETURN
          JUMP    SOU2
          .END    SOUS
```

Exécuter le programme avec une dizaine de paires de valeurs très différentes.

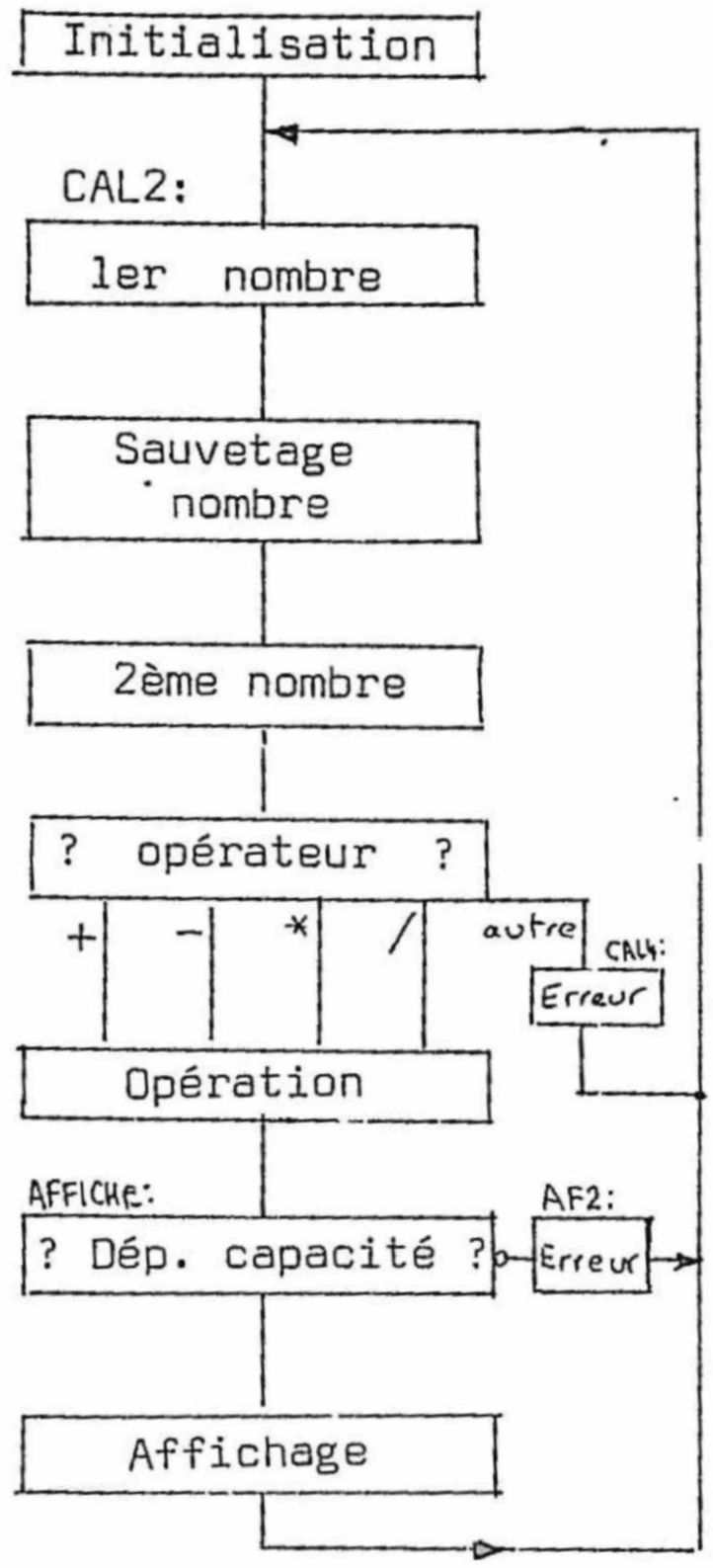
Exemple 12.3. Soustraction de 2 nombres positifs, a-b, résultat signé

```
SOUS:     LOAD    C,# NLI
          .W      ?IDIS

SOU2:     .W      ?INOC
          EX      HL,DE              ;1er opérateur DE
          .W      ?INOC, ?SPACE      ;2e opérateur HL
          EX      HL,DE              ;permuter
          .W      ?COMPHLDE          ;1er op. < 2e op. ?
          JUMP,HS SOU4
          EX      HL,DE              ;si oui, permuter
          LOAD    A,# '-             ;afficher le -
          .W      ?DICAR

SOU4:     OR      A,A                ;si non
          SUBC    HL,DE
          .W      ?AFOHL, ?RETURN
          JUMP    SOU2
```



Exécuter le programme avec une dizaine de valeurs très différentes.

Exemple 12.4. Calculatrice octale 4 opérations, nombre entiers, en notation polonaise: a b +




```
        .TITLE  CALCO   ;PP 790913
        .PROC   Z80
        .REF    SM6

CALC:   LOAD    C,# NLI
        .W      ?IDIS

CAL2:   .W      ?INOC, ?SPACE   ;1er nombre
        PUSH    HL              ;sauvetage sur la pile
        .W      ?INOC           ;2e nombre  et opérateur
        LOAD    DE,# TABLE
        .W      ?JUMPCAR

CAL4:   .W      ?TEXT, TX1
        JUMP    CAL2

TABLE:  .BW     '+, DOPLUS
        .BW     '-, DOMINUS
        .BW     '*, DOMUL
        .BW     '/, DODIV
        .B      Ø

;on entre avec le 1er argument sur la pile, le 2e dans HL
;on sort avec le résultat dans HL et CS s'il y a dépassement de capacité

DOPLUS: POP     DE
        ADD     HL,DE
        JUMP    AFFICHE

DOMINUS:POP     DE
        EX      HL,DE   ;permuter pour soustraire 1er - 2ème
        OR      A,A
        SUBC    HL,DE   ;remplace l'instruction manquante  SUB HL,DE
        JUMP    AFFICHE

DOMUL:  LOAD    B,H
        LOAD    C,L     ;remplace l'instruction  LOAD BC,HL
        POP     DE
        LOAD    HL,# 0
        .W      ?MUL
        EX      HL,DE
        LOAD    A,D
        OR      A,E     ;remplace l'instruction TEST DE et met le carry à Ø
        JUMP,EQ AFFICHE
        SETC            ;met de carry à 1 (dépassement de capacité)
        JUMP    AFFICHE
```

```
DODIV:  LOAD    B,H         ;diviseur
        LOAD    C,L
        LOAD    HL,# 0
        POP     DE          ;dividende
        .W      ?DIV
        EX      HL,DE       ;quotient dans HL pour affichage ultérieur


AFFICHE: JUMP,CS AFF2
        .W      ?AFOHL, ?RETURN
        JUMP    CAL2

AFF2:   .W      ?TEXT,TX2
        JUMP    CAL2

TX1:    .ASCIZ  * Je ne connais pas cette opération <CR>"

TX2:    .ASCIZ  " Dépassement de capacité <CR>"

        .END    CALC
```

Exercice 12.1. *Modifier le programme précédent pour afficher le reste de la division précédé de "Reste:"*

Exercice 12.2. *Modifier le programme précédent pour taper l'opération en notation algébrique usuelle a + b =*

Exercice 12.2. *Ecrire le programme calculatrice 2 ou 4 opérations en décimal (nombres entiers).*
Indication: convertir en binaire pour effectuer les calculs.

13. APPELS DE MISE EN PAGE

Avec ?IDIS, les textes commencent toujours au haut de l'écran. Il est possible de commencer n'importe où en préparant les coordonnées de début dans HL et en utilisant l'appel ?SETCUR.

L'appel ?GETCUR permet de savoir où est le pointeur (curseur).

Exemple 13.1. On veut afficher un * au centre de l'écran.

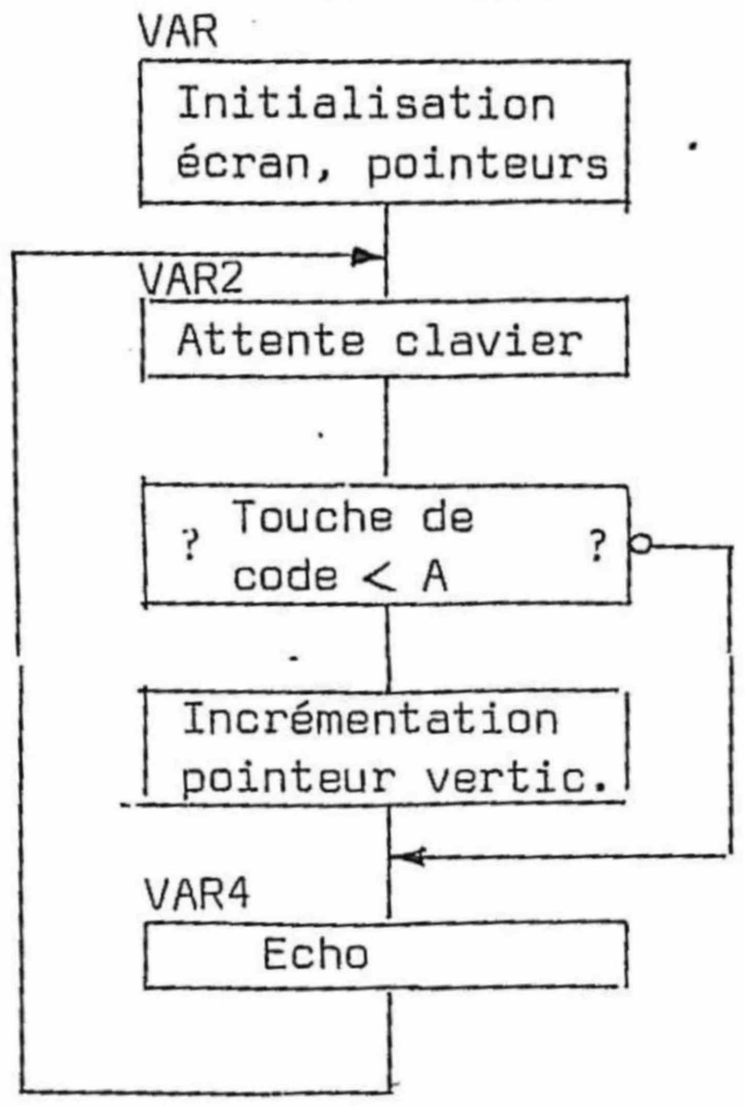
```
        .TITLE  ETOILE
        .PROC   Z80
        .REF    SM6


CENTRE  = (NCAR/2)* XX + (NLI/2)* YY

ETOILE: LOAD    C,# NLI
        .W      ?IDIS
        LOAD    HL,# CENTRE
        .W      ?SETCUR
        LOAD    A,# '*
        .W      ?DICAR
        TRAP                    ;retourne à SMILE, qui reprend le contrôle
        .END    ETOILE          ;à la prochaine touche tapée
```

Exemple 13.2. On veut que le texte tapé au clavier soit affiché verticalement.

```
        .TITLE  VERT
        .PROC   Z80
        .REF    SM6

;paramètres de mise en page
PREMCAR= 1*XX + 4*YY            ;2e colonne, 5e ligne

VERT:   LOAD    C,# NLI
        .W      ?IDIS
        LOAD    HL,# PREMCAR ;pointeur caractère
VER1:   .W      ?SETCUR

VER2:   .W      ?GETCAR, ?DICAR ;attente clavier et écho
        INC     H
        .W      ?SETCUR         ;JUMP VER1  plus court
        JUMP    VER2

        . END   VERT
```

Exemple 13.3. Variante du programme précédent, dans laquelle seuls les chiffres doivent être affichés verticalement (en fait, pour simplifier, tous les caractères dont le code ASCII est inférieur à celui de la lettre A).

```
VAR:    LOAD    C,# NLI
        .W      ?IDIS
        LOAD    HL,# PREMCAR
        .W      ?SETCUR

VAR2:   .W      ?GETCAR
        COMP    A,# 'A          ;chiffre ou lettre ?
        JUMP,LO VAR4
        .W      ?DICAR          ;lettre
        JUMP    VAR2

VAR4:   PUSH    AF              ;chiffre
        .W      ?GETCUR
        POP     AF
        INC     H
        .W      ?SETCUR,?DICAR
        JUMP    VAR2
```



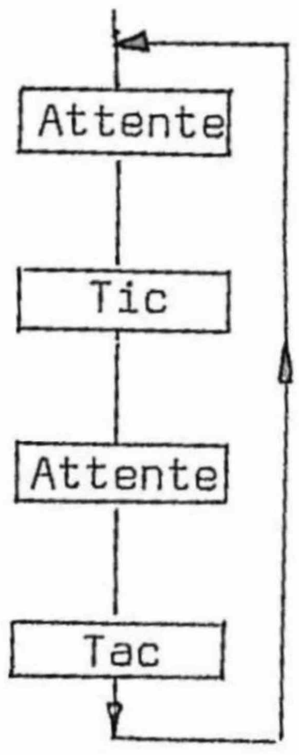
Question: les nombres s'affichent en fait en oblique. Pourquoi? Corriger le programme.

14. ATTENTE

L'appel .W ?DELAY,ATT permet de ralentir l'exécution en créant une attente de ATT millisecondes (ATT < 65 000.)

Eexmple 14.1. Le programme doit imiter le tic-tac d'une pendule.

```
        .TITLE  TICTAC
        .PROC   Z80
        .REF    SM6

TIC     = 11
TAC     = 21
AT1     = 500
AT2     = 540

TICTAC: .W      ?DELAY,AT1
        LOAD    A,# TIC
        .W      ?BEEP
        .W      ?DELAY,AT2
        LOAD    A,# TAC
        .W      ?BEEP
        JUMP    TICTAC

        .END    TICTAC
```



Exercice 14.1. Ecrire le programme qui déplace un caractère horizontalement sur l'écran, en le faisant rebondir sur les bords (utiliser ?SETCURSOR, ?SPACE, ?DICAR, ?DELAY).

3ème partie. INSTRUCTIONS PRINCIPALES DU PROCESSEUR Z80

15. INSTRUCTIONS DE TRANSFERT

Ces instructions ne modifient pas les fanions.

LOAD { A / B / C / D / E / H / L / (HL) } , { # VAL / A / B / C / D / E / H / L / (HL) }

LOAD A , { (BC) / (DE) / (HL) }

LOAD { (BC) / (DE) / (HL) } , A

LOAD A , { ADVAL / $ADPER }

LOAD { ADVAL / $ADPER } , A

LOAD { BC / DE / HL } , { # VAL / ADVAL }

LOAD ADVAL , { BC / DE / HL }

EX { HL,DE }

PUSH { AF / BC / DE / HL } POP { AF / BC / DE / HL }

Exemples:

LOAD	E,# NBRE1	charge dans E la valeur NBRE1 déclarée au début du programme
LOAD	D,A	chargement de D par le contenu de A
LOAD	(HL),C	chargement de la position mémoire dont l'adresse est dans HL par le contenu de C
LOAD	A,COUNT	chargement de A par le contenu de la position mémoire d'étiquette COUNT, déclarée par COUNT: .BLKB 1 ou COUNT = 42000 (position mémoire permise, visible sur l'écran)
LOAD	A,$CLA	lecture directe du périph. clavier
LOAD	SAVEHL,HL	transfert dans la position SAVEHL du contenu de HL. A la fin du programme, on a par exemple: SAVEHL: .BLKW 1

Instruction d'échange de HL de DE

Sauvetage d'une paire de registres sur la pile.
Récupération dans une paire de registres du sommet de la pile.

Succédanés à des instructions manquantes:

LOAD HL,DE

```
LOAD  H,D
LOAD  L,E
```

LOAD A,F

```
PUSH  AF
POP   BC
LOAD  A,C
```

modifie B et C en plus de A

```
PUSH  BC
PUSH  AF
POP   BC
LOAD  A,C
POP   BC
```

Ne modifie que A

EX HL,BC

```
PUSH  HL
PUSH  BC
POP   HL
POP   BC
```

16,8 µs de temps d'exécution

```
PUSH  HL
LOAD  H,B
LOAD  L,C
POP   BC
```

11,6 µs

```
PUSH  BC
EX    (SP),HL
POP   BC
```

16. INSTRUCTIONS ARITHMETIQUES 8 BITS

Ces instructions modifient les fanions C,Z,S,V.

ADD, ADDC, SUB, SUBC, COMP, AND, OR, XOR — A , { #VAL, A, B, C, D, E, H, L, (HL) }

Exemples:

ADDC	A,#Ø	ajoute à A Ø ou 1 selon la valeur du Carry.
COMP	A,(HL)	compare le contenu de A avec le contenu de la position mémoire dont l'adresse est dans HL

CPL	A	Complément à 1 de A
NEG	A	Complément à 2 de A
DA	A	Correction BCD après une opération 8 bits sur A (Utilise le Carry et le Carry auxiliaire généré lors de l'opération 8 bits)

INC, DEC — { A, B, C, D, E, H, L, (HL) }

INC	(HL)	ajoute 1 au contenu de la position mémoire dont l'adresse est dans HL

RL	Rotation à gauche
RLC	Rotation à gauche à travers le Carry
SLC	Décalage à gauche
RR	Rotation à droite
RRC	Rotation à droite à travers le Carry
SRC	Décalage à droite (:2 nombre logique)
ASR	Décalage à droite (:2 nombre arithmétique)

RL, RLC, SLC, RR, RRC, SRC, ASR — { A, B, C, D, E, H, L, (HL) }

Succédanés à des instructions manquantes:

CPL B

```
LOAD  A,B
CPL   A
LOAD  B,A
```

```
LOAD  A,#-1
SUB   A,B
```

NEG B

```
LOAD  A,B
NEG   A
LOAD  B,A
```

```
XOR   A,A
SUB   A,B
LOAD  B,A
```

modifient A et F en plus de B

```
PUSH  AF
XOR   A,A
SUB   A,B
LOAD  B,A
POP   AF
```

ne modifie que B

17. INSTRUCTIONS ARITHMETIQUES 16 BITS

Ces instructions modifient les fanions C,Z,S,V d'une manière qui n'est pas toujours logique.

ADD, ADDC, SUBC } HL, { BC, DE, HL }

Exemples:

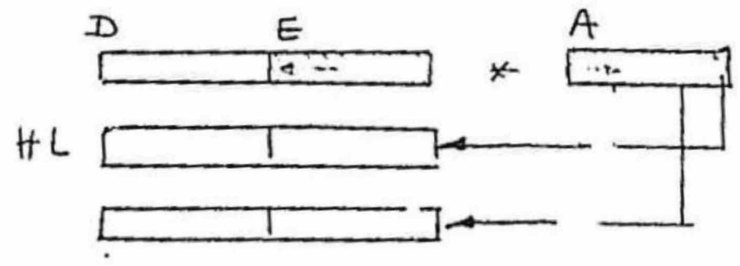
ADD HL,HL	double le contenu de HL (modifie C, mais pas Z,S,V)
ADDC HL,HL	double le contenu de HL et ajoute le Carry (modifie C,Z,S,V)

INC, DEC { BC, DE, HL } ne modifient pas du tout les fanions

Succédanés à des instructions manquantes:

SUB	*HL,DE*	OR A,A SUBC HL,DE	soustraction de HL et DE
SLC	*HL*	ADD HL,HL	décalage à gauche de HL (double)
SRC	*HL*	SRC H RRC L	décalage à droite de HL (divise par 2 les nombres logiques)
ASR	*HL*	ASR H RRC L	décalage à droite de HL (divise par 2 les nombres arithmétiques)
NEG	*HL*	EX HL,DE LOAD HL,#0 OR A,A SUBC HL,DE EX HL,DE modifie F	LOAD A,L CPL A LOAD L,A LOAD A,H CPL A LOAD H,A INC HL complément à 2 de HL
DA	*HL*	impossible: il faut faire toutes les opérations dans A et corriger au fur à à mesure.	

Exemple 17.1. Routine de multiplication de E(8 bits) par A(8 bits), résultat dans HL (16 bits).




```
;in     E multiplicande, A multiplicateur
;out    HL produit
;mod.   F,A,B,D,E,HL

MULEA:  LOAD    D,#Ø    ;DE multiplicande 16 bits
        LOAD    HL,#Ø   ;produit partiel initial
        LOAD    B,#8.

MUL2:   TEST    A:Ø     ;bit de poids faible de A
        JUMP,EQ MUL4
        ADD     HL,DE   ;add. du multiplicande au produit partiel

MUL4:   SLC     E       ;SLC DE  décalage de DE (double)
        RLC     D
        RR      A       ;décalage de A
        DECJ,NE B,MUL2
        RET
```

Variante: les instructions

```
TEST     A:Ø
JUMP,EQ  MUL4
...
RR       A
```

sont en général remplacées par

```
RR       A
JUMP,CC  MUL4
```

Programme complet permettant le test de la routine:

```
        .TITLE  MULT
        .PROC   Z80
        .REF    SM6

;routine de multiplication et prog. de test
TEST:   LOAD    C,# NLI
        .W      ?IDIS

TE2:    .W      ?INOC, ?SPACE
        LOAD    E,L                 ;multiplicande
        .W      ?INOC, ?SPACE
        LOAD    A,L                 ;multiplicateur
        CALL    MULEA
        .W      ?AFOHL, ?RETURN     ;résultat dans HL
        JUMP    TE2

;routine ...

MULEA:  ...

        . END   TEST
```

Exercice 17.1. *Modifier l'exemple 17.1 pour effectuer le produit en commençant par les poids forts de A, avec décalage à droite de DE.*

Exemple 17.2. Routine de multiplication de HL (16 bits) par A (8 bits), résultat dans AHL (24 bits).

```
;??????????
;;   ?MULHLA    MUL     HL,A,AHL
;??????????
;;
;;in     HL multiplicande, A multiplicateur
;;out    AHL produit 24 bits
;;mod    F, A, HL
;;
MULHLA: PUSH    BC
        PUSH    DE
        LOAD    B,#8.       ;décompteur de cycles de multiplication
        EX      HL',DE      ;multiplicande dans DE
        LOAD    HL,#Ø       ;produit partiel initial
MUL1:   ADD     HL,HL       ;décalage du produit partiel
        RLC     A           ;décalage du poids fort du multiplicateur dans le Carry
        JUMP.,CC MUL2
        ADD     HL,DE       ;addition si ce bit vaut 1
        ADDC    A,#Ø        ;débordement éventuel sauvé dans A
MUL2:   DECJ,NE B,MUL1      ;répéter 8 fois
        POP     DE
        POP     BC
        RET
```

Exercice 17.2. *Ecrire et tester la routine de division HL:A*

18. INSTRUCTIONS DE TEST

Elles permettent de savoir si un bit, un octet (byte), un sedet (word) est égal à zéro ou non.

Test de bit:

TEST { A, B, C, D, E, H, L, (HL) } : RANG

Exemples:

```
BSIGNE = 7
...
TEST   H:BSIGNE
```

charge le fanion Z avec la valeur du bit de poids fort de H (si BSIGNE = 7)

```
TEST   (HL):Ø
```

charge Z avec la valeur du bit de poids faible de la position mémoire dont l'adresse est dans HL

Succédanés aux instructions de test d'octets et sedets:

TEST A { OR A,A | { ADD A,#Ø — Z=1 (EQ) si A est nul

TEST HL { LOAD A,H / OR A,L — Z=1 (EQ) si HL est nul (le mot (ou les deux moitiés du mot) ne peut être nul que si le nombre total est nul)

modifie A

19. INSTRUCTIONS DE SAUT ET D'APPEL DE ROUTINES

JUMP / CALL ETIQ

JUMP / CALL , { EQ, NE, CS LO, CC HS, SS MI, SC PL, VS, VC } ETIQ

Exemple:

```
ADD      A,B          A,B positifs (nombres logiques)
JUMP,CS  OVERFLOW     saut si  A+B ⩾ 2^8

ADD      A,B          A,B arithmétiques
JUMP,VS  OVERFLOW     saut si  |A|+|B| ⩾ 2^7

COMP     A,B          A,B positifs
JUMP,LO  PLUPETIT     saut si  A < B  (nbres logiques)

COMP     A,B          A,B arithmétiques
JUMP,MI  PLUSPETIT    saut si  A < B
```

```
DEC      BC
LOAD     A,B
OR       A,C
JUMP,NE  LOOP
```

DECJ,NE BC,LOOP (modifie A!)

Exemple 19.1.

```
;sous programme pour ajouter 1 dans le registre 24 bits CBA
;in  CBA
;out CBA, EQ si CBA=0
;mod F,CBA

INCCBA: INC      A
        RET,NE
        INC      B
        RET,NE
        INC      C
        RET
```

20. INSTRUCTIONS DE MODIFICATION DE BITS

SET / CLR { A, B, C, D, E, (HL) } : RANG

Exemple:

FLAG1 = 3

SET D:FLAG1 met à zéro le bit 2^3 de D si FLAG1=3

pourrait s'écrire LOAD D:FLAG1,#1

SETC Met le Carry à 1 (SET C voudrait dire met à 1 le registre C)

CPLC Complémente (inverse) le Carry

Succédanés aux instructions manquantes:

CLRC { OR A,A clear le Carry (modifie S,Z,V en plus) } { SETC / CPLC } ne modifient que le Carry

On peut modifier un ou plusieurs bits avec les opérations logiques ET, OU.

OR A,#200 est équivalent à *SET A:7* OR A,#360 est équivalent à *SET A:(7,6,5,4)*

AND A,#177 est équivalent à *CLR A:7* AND A,#17 est équivalent à *CLR A:(7,6,5,4)*

Certains processeurs (PDP11 en particulier) ont les instructions

BIS (bit set) et BIC (bit clear)

BIS A,B { OR A,B les bits de A correspondant aux bits de B qui valent 1 sont mis à 1

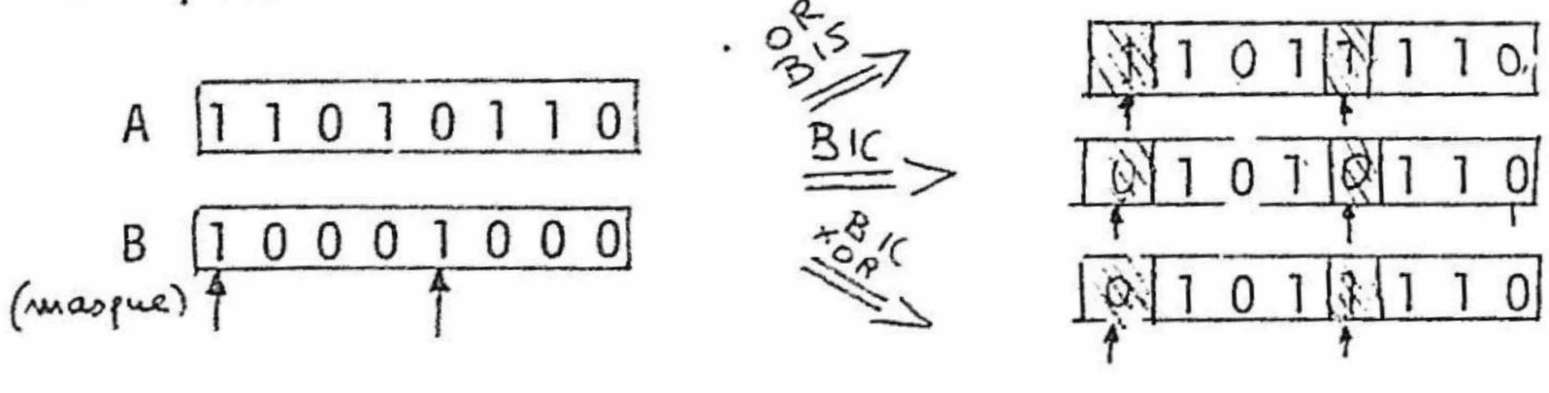
BIC A,B

{ LOAD C,A; LOAD A,B; CPL A; AND A,C }

{ CPL A; OR A,B; CPL A } les bits de A correspondant aux bits de B qui valent 1 sont mis à zéro

L'instruction BII (bit inverse) est réalisée par l'instruction XOR (ou exclusif)

Exemple:



L'instruction *CLR* peut être réalisée avec un XOR A,A (1 byte) ou LOAD A,#Ø (2 bytes).

4e partie: TECHNIQUE DE PROGRAMMATION

21. REGISTRES EN MEMOIRE

Lorsque le nombre de registres à disposition est insuffisant, des positions mémoire sont utilisées comme registres supplémentaires. Le Z80 est toutefois un processeur très pauvre pour travailler directement avec la mémoire. Les transferts 8 bits ne peuvent se faire qu'avec A, les transferts 16 bits avec HL, DE, BC. Ces instructions de transfert prennent beaucoup de place.
Les positions mémoire utilisées comme registres sont caractérisées par une étiquette et sont définies par un .B ou .W.
Toutes les opérations doivent être exécutées dans les registres, et les positions mémoire ne peuvent être utilisées que pour sauver provisoirement une valeur. Par exemple, les compteurs ou décompteurs de boucle sont souvent réalisés en mémoire.

```
        LOAD    A,# CINITIAL
        LOAD    COUNT,A         ;initialisation du compteur
LOOP:   ...
        ...
        ...
        LOAD    A,COUNT     }
        DEC     A           }   ;DEC COUNT
        LOAD    COUNT,A     }   ! modifie A
        JUMP,NE LOOP
        ...

COUNT:  .B      Ø               ;réserve un byte en mémoire
                                ;(à la fin du programme)
                                ;et l'initialise à la valeur Ø
```

La pile est une zone mémoire particulière très pratique pour sauver les registres. L'adresse est implicite; il n'est pas nécessaire de la déclarer.

L'exemple précédent peut s'écrire comme suit si le compteur est sur la pile.

```
        LOAD    A,# CINITIAL
        PUSH    AF

LOOP:   ...
        POP     AF      }
        DEC     A       }   !modifie A
        PUSH    AF      }
        JUMP,NE LOOP
```

L'utilisation de la pile nécessite des précautions, car il ne faut pas se tromper dans l'ordre de remplissage et vidage, et surtout ne jamais avoir dans une routine ou une boucle plus de PUSH que de POP: la pile remplit alors toute la mémoire, jusqu'à destruction du programme, si l'on appelle souvent la routine.

22. TABLEAUX EN MEMOIRE

Une zone mémoire de m bytes à partir de l'adresse TABL peut être réservée avec l'instruction

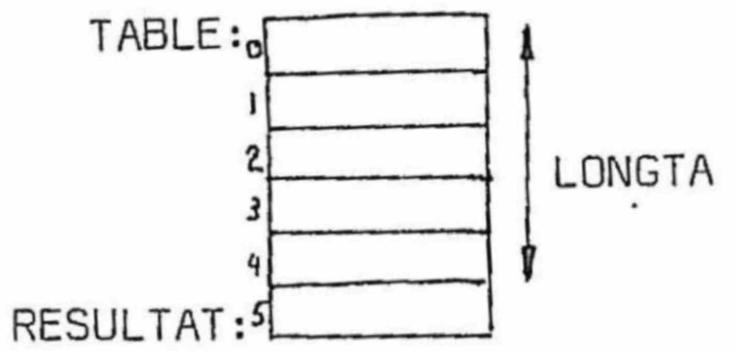
```
TABL:    .BLKB  m                (.Block Byte)
```

Si cette zone doit recevoir des mots de 16 bits, on utilise la pseudo-instruction .BLKW.

Pour traiter des tableaux de nombres, les registres 16 bits du Z80 sont utilisés comme pointeurs dans ces tableaux (ils contiennent des adresses de nombres dans les tableaux) et le registre A est utilisé pour les ~~adjonctions~~. opérations

Exemple 22.1. Addition de 5 nombres entiers 8 bits préparés dans un tableau. Le résultat doit être placé en 6e position du tableau.

```
ADTAB:  LOAD     HL,# TABLE      ;init. pointeur table
        LOAD     B,# LONGTA      ;init. décompteur boucle
        XOR      A,A             ;A=0, carry = Ø

ADT2:   ADD      A,(HL)          ;add. la valeur pointée
        INC      HL
        DECJ,NE  B,ADT2

        LOAD     (HL),A          ;sauvetage du résultat
        ...

LONGTA  = 5

TABLE:  .BLKB    LONGTA
                                 } ou  TABLE: .BLKB LONGTA+1
RESULTAT:.BLKB   1
```



Le contenu du tableau a été préparé ou construit par un autre programme.

Le tableau peut être initialisé n'importe où dans la mémoire. L'habitude est de regrouper tous les tableaux dans une zone commune, en fin de programme. Avec le SMAKY6, il est intéressant de placer ces tableaux dans l'écran, qui est une zone mémoire affichée en permanence. Si les 8. premières lignes sont réservées pour les dialogues avec l'utilisateur, la mémoire à disposition pour les tableaux va de ~~41000 (début~~ de la 8ème ligne) à 42377 (coin inférieur droit de l'écran). Il y a 100=64. bytes par ligne d'écran. L'adresse des débuts de table doit être définie par un = . SALPHA + 200 à SALPHA + 4FF

Exemple 22.2. Programme complet pour la préparation d'un tableau de nombres, le calcul du total et l'affichage du résultat.

```
        .TITLE  TABLEAU
        .PROC   Z80
        .REF    SM6
;définitions propres au programme
LIGNE   = 8.
TABLE   = SALPHA+16.*NCAR     ;tableau au début de la 16ème ligne écran
LONGTA  = 5                   ;tableau de 5 bytes
;programme
TABLEAU:~~LOAD    C,# NLI~~
        ~~.W      ?IDIS~~     ;efface tout l'écran
        LOAD    C,# LIGNE
        .W      ?IDIS         ;initialise 8. lignes seulement pour la suite
```

```
;remplissage tableau

        LOAD    DE,#TABLE           ;ne pas utiliser HL, qui va être modifié
        LOAD    B,# LONGTA          ;par ?INOC

TAB2:   PUSH    BC
        .W      ?INOC, ?SPACE
        POP     BC
        LOAD    A,L
        LOAD    (DE),A
        INC     DE
        DECJ,NE B,TAB2

;calcul total

        LOAD    HL,# TABLE
        LOAD    B,# LONGTA
        XOR     A,A

TAB4:   ADD.    A,(HL)
        INC     HL
        DECJ,NE B,TAB4
        LOAD    (HL),A

;affichage du résultat

        LOAD    HL,# TABLE+LONGTA
        LOAD    A,(HL)
        .W      ?RETURN, ?AFOCA, ?RETURN

        .END    TABLEAU
```

Exercice 22.1. *Modifier l'exemple précédent pour détecter un dépassement de capacité et/ou afficher un résultat correct 16 bits.*

Exercice 22.2. *Définir 2 tableaux TAB1 =41000 et TAB2 = 41200. Remplir ces tableaux avec des codes ASCII (appel ?GETCAR). Calculer la somme byte à byte de ces 2 tableaux dans TAB3 = 41400.*

L'"organigramme" pour cet exercice peut s'écrire

Initialisation écran
Init. 1er tableau et compteur
Remplissage avec des codes ASCII
Init 2e tableau et compteur
Remplissage 2e tableau
Init. pointeur 1er, 2e tableau et tableau des résultats
Addition
Affichage éventuel en clair des nombres contenus dans le tableau des résultats

23. TABLE DE CONVERSION

A une valeur connue correspond souvent une autre valeur que le programme doit déterminer. Si le calcul est trop long ou trop compliqué, une table à deux entrées permet d'obtenir les valeurs cherchées.

Le cas le plus simple se trouve lorsque les valeurs connues sont comprises entre 0 et 255. (entiers 8 bits). Le tableau des valeurs correspondantes est défini en mémoire avec un .B (byte) ou .W (word).

Exemple 23.1. Dessiner sur l'écran des caractères faisant apparaître une demi-sinusoïde.

Les lignes 0, 1, ... 17. correspondent aux valeurs des sinus de 10^o en 10^o. La table de sinus est à définir de 10^o en 10^o également.

```
          .TITLE  SINUS
          .PROC   Z80
          .REF    SM6

SINUS:    LOAD    C,# NLI
          .W      ?IDIS

          LOAD    C,# 18.          ;compteur de ligne
          LOAD    D,# Ø            ;angle initial
SIN2:     LOAD    A,D              ;angle dans A
          LOAD    HL,# TASIN       ;pointeur au début de la table de sinus
          ADD     A,L
          LOAD    L,A
          LOAD    A,H              ;ADD HL,A     calcul du point d'entrée
          ADDC    A,# Ø                          dans la table
          LOAD    H,A

          LOAD    B,(HL)           ;valeur du sinus dans B

          LOAD    A,# '*

          INC     B
SIN4:     DEC     B
          JUMP,EQ SIN6
          .W      ?DICAR
          JUMP    SIN4
SIN6:     .W      ?RETURN

          INC     D                ;angle suivant
          DEC     C                ;décompteur
          JUMP,NE SIN2

          TRAP
;table prop. au sinus:  sin 90° = 50.

TASIN:    .B      0
          .B      9.               ;50·sin 10° = 8,68
          .B      17.              ;50·sin 20° = 17,1
          .B      25.              ;50·sin 30° = 25
          .B      32., 38., 43., 47., 49., 50.                    ;etc.
          .B      49., 47., 43., 38., 32., 25., 17., 9., 0

          .END    SINUS
```

Exemple 23.2. *On veut jouer des notes aussi correctement que possible. La rangée de touches A S D F G H J K L doit permettre de jouer la gamme. A chacune de ces touches correspond une fréquence et une durée, exprimée en nombre de périodes. Les touches sont reconnues par leur code ASCII, qui n'est malheureusement pas dans l'ordre des touches.*

1ère solution: construire une grande table, dans l'ordre des codes ASCII.

Note	Touche	Code ASCII	No d'ordre	Période	Nbre périodes
DO	A	101	0	277	103
	B	102	1	0	0
	C	103	2	0	0
MI	D	104	3	230	125
	E	105	4	0	0
FA	F	106	5	217	132

...

Après avoir lu le clavier, on soustrait la valeur 'A=101 pour avoir le numéro d'ordre dans la table, on double ce n° d'ordre étant donné qu'il y a deux bytes pour chaque note, et on prélève dans la table la fréquence et le nombre de périodes qui permettent de jouer cette note.

2ème solution: pour éviter la place perdue dans la table précédente, une première table est balayée pour trouver quel est le No d'ordre de la touche pressée.

No d'ordre	Touche
0	A
1	S
2	D
3	F
...	...

Ce No d'ordre permet ensuite d'accéder à la table des périodes et durées, qui sont dans l'ordre des notes.

Cette deuxième solution est plus flexible, mais son temps d'exécution est plus grand. On entendra donc plus le raccord entre deux notes.

24. MISE AU POINT DE PROGRAMMES

La mise au point d'un programme peut être rapide ou très longue et pénible. Ceci dépend essentiellement de la méthode de travail utilisée.

Un premier principe à appliquer est de bien réfléchir au programme avant de le taper, de bien le structurer en le tapant, de le sauver avant de commencer la mise au point.

Une bonne méthode est le test sur papier. Une fois que le programme est écrit sur brouillon, on vérifie son déroulement en simulant soi-même le processeur, copiant l'état des registres sur une feuille de papier, dessinant l'écran, etc.

Un deuxième principe est de décomposer le programme en modules, de si possible tester chaque module séparément, et en tous cas de rajouter pendant la mise au point des instructions d'affichage de résultats partiels entre chaque module. En particulier, il est conseillé d'afficher les variables dans l'écran et, si un usage fréquent de la pile est fait, d'y placer également la pile avec l'instruction LOAD SP,#42377.

L'interprétation du contenu de la pile est délicat à cause de l'interruption, mais les erreurs éventuelles de PUSH/POP apparaissent clairement.

L'instruction TRAP permet d'arrêter l'exécution et d'examiner les états des registres (sous SMILE, ordre (PROGRA) R). Il est possible de continuer l'exécution à partir de ce point d'arrêt avec (PROGRA) X; . On peut donc avoir plusieurs TRAP dans un programme.

Il est également possible de définir une routine de test qui affiche des registres, des positions mémoire, ou joue des notes, et d'appeler ensuite cette routine à certains points du programme où on a de la peine à imaginer ce qui se passe.

De façon générale, tester un programme n'est pas vérifier qu'il fonctionne une fois. C'est être sûr que, quelles que soient les conditions d'utilisation de ce programme, il fonctionnera correctement. Cette philosophie doit s'appliquer d'abord aux modules et ensuite au tout.

25. PROBLEMES

25.1. Calculer les nombres premiers inférieurs à 2^{16}.
Afficher leur valeur en décimal sur l'écran, puis afficher le crible d'Eratostène pour les 1280. premières valeurs (l'écran a 64.x20. caractères).

25.2. Faire une statistique des codes trouvés dans 2048. positions consécutives en mémoire (à partir d'une adresse variable).
Représenter un histogramme des valeurs, calculer la moyenne et l'écran type.

25.3 Générer sur le haut-parleur le code morse des caractères tapés au clavier. Mémoriser le message dans une zone mémoire, pour pouvoir le réexpédier automatiquement une 2e fois.

25.4. Ecrire les routines ?AFOHL, ?AFXHL en supprimant les zéros non significatifs au début du nombre. Généraliser ces routines pour afficher des nombres multiprécision en mémoire.

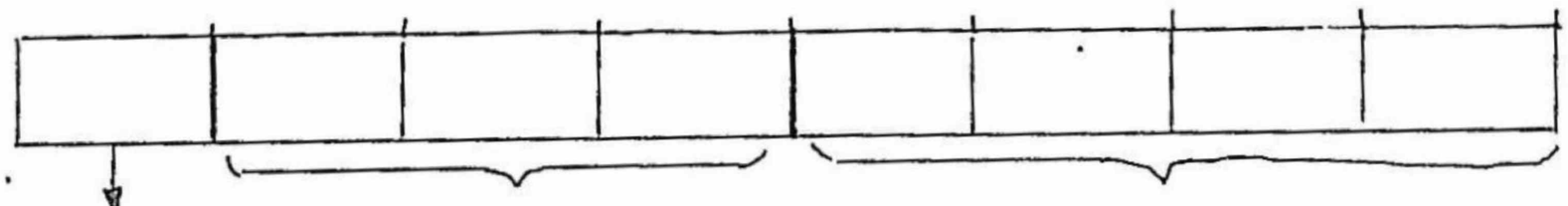
25.5. Ecrire des routines RND A, RND HL qui génèrent dans A ou HL un nombre entier aléatoire.
Ecrire le programme qui vérifie la nature aléatoire de ces nombres.

25.6. Ecrire un programme qui permet de générer sur l'écran un "show" publicitaire avec des textes préparés à l'avance, des motifs décoratifs, des mélodies, etc.

25.7. Ecrire un programme qui permet d'afficher sur tout l'écran une horloge (affichage 7 segments ou analogique). Les routines de base doivent permettre de générer des segments de longueur et inclination donnée, ou de relier par un segment deux points donnés.

25.8, Ecrire un programme permettant de remplir un formulaire affiché sur l'écran en caractères inversés, et de mémoriser ou perforer au fur et à mesure la partie significative du questionnaire.

25.9. Ecrire une routine BEEP jouant des notes aussi correctes que possible.
Format proposé:



Octave

0: bas
1: aigu

TRIPLE CROCHE	= 0
DOUBLE CROCHE	= 1
CROCHE	= 2
NOIRE	= 3
NOIRE POINTEE	= 4
BLANCHE	= 5
BLANCHE POINTEE	= 6
RONDE	= 7

SILENCE	= 0
DO	= 1
DOD	= 2
RE	= 3
MIB	= 4
MI	= 5

etc.