Note from the editor:

This bilingual transcription was created in September 2025 by Milli using the original French scanned manual and ChatGPT for OCR and translation. While care has been taken to ensure accuracy, errors may remain. Readers are encouraged to refer to the original French manual if any discrepancies or mistakes are found.

MacADAM Manual

English Version

CONTENTS

Simply insert the cassette into the drive and press "RESET". The system loads and launches the program.

On the first screen displayed, by pressing the "Up Arrow" and "Down Arrow" keys, you can change the background and text color.

II-1 THE SELECTION MENU

The first menu is simply a menu for selecting other menus:

1 - the editing mode
2 - the input/output control menu
3 - the assembly menu
4 - the initialization menu
5 - the environment parameters menu

To access one of these menus, simply type 1, 2, 3, 4, or 5.

This menu allows you to choose the configuration in which you will work.

This choice allows you to select for memory paging, the read and write units, and for the printer: the choice of paper, the number of lines printed, and the number of lines per page.

It also lets you introduce a title to display at the beginning of your listings. To modify it, position yourself with the arrow keys on the option. Then type the value for your option. Press “RETURN” to go back to the main menu.

When you enter an item, you can use
alphabetic characters as well as the
"left" and "right" arrow keys.

II-2 THE INITIALIZATION MENU

This is the simplest: it only asks for confirmation
before clearing the work buffer. Its purpose is
to avoid accidentally erasing the program due
to a mistaken operation.

II-3 THE EDITING MENU

II-3-1 OPERATION

The principle is simple.

The screen is a window that allows you to
enter, modify, or delete text to be assembled.
All text is written into the workspace and
displayed on the screen. The cursor indicates
the position where the next character will be
introduced.

II-3-2 SPECIAL CHARACTERS

- The space bar: space
- The RETURN key: line break
- The TABULATION key: tab
- The "Up Arrow" key: move cursor up
- The "Down Arrow" key: move cursor down
- The "Left Arrow" key: move cursor left
- The "Right Arrow" key: move cursor right
- The "DEL" key: delete a character
- The "INS" key: insert a space
- The "CTRL" key combined with others: special functions
- The "RETURN" key: end of input for the line and move to the next one

When you enter text, you may use the alphabetic
keys on the keyboard as well as the special
keys listed above. "RETURN" marks the end
of a line. "CTRL-F" starts a text search within
the current file. "CTRL-L" reformats the line.

All of the system’s special characters are grouped
and accessible from the main menu.

II-3-3 RETURNING FROM THE EDITOR

Once the entered text is displayed, properly aligned and analyzed, blank lines will be absorbed and not re-displayed after the operation. The appearance is thus cleaned up.

The “label” part is truncated to 8 characters. The “operation code” part is produced over 4 characters. The “operand” part is left as-is, with surrounding spaces removed. This rule applies only to lines that are not comments.

II-4 THE INPUT/OUTPUT CONTROL MENU

This allows communication with the external module in the expected manner. Reading and writing take place on the designated unit.

This menu lets you load another program into the program buffer. If you do not wish to mix programs, the previous program must first be cleared.

This menu also allows saving the current program and the general program. The latter can be done in two ways:

- Normal save: creation of a listed file containing the text. The generated file is organized between the minimum and maximum addresses implicitly defined by your program.

- Save to a specified block: in this case you indicate yourself the number of the block to use for saving. This number must be a value between 0 and FFFFH. The system will reject it if the file size is too large for that block.

The program is then loaded again from the last character saved.

III-2 THE DIFFERENT OPERANDS

"Byte" operands: these represent a value between 0 and 255. When used in an operation, only the low-order byte of the result is kept.

"Word" operands: these are 16-bit numeric expressions, ranging from 0 to 65535. Their value is inserted as-is into the program.

"Expression" operands: these are arithmetic formulas using constants, labels, and the + or - operators. They can be used as numeric values.

Examples:
- Decimal constants: values written directly
- Hexadecimal constants: begin with "E"
- Binary constants: begin with "B"

Instructions are composed of three fields:

1) The label field: up to 8 characters.
A label is a name that designates an address.
It is followed by a colon (:).

2) The operation code field: up to 4 characters.
It receives the Z80 operations, pseudo-instructions, etc.

3) The operand field: contains either a value, an expression, or a label.

Labels are global in the program and allow addresses to be referenced wherever they appear.

IV THE LANGUAGE

The language consists of three types of instructions:
1) Pseudo-instructions
2) Z80 assembler instructions
3) Calls to macro-instructions defined by the programmer

Assembler instructions generate machine code directly, while pseudo-instructions serve to control the flow of the assembly process.

IV-1 PSEUDO-INSTRUCTIONS

- ASC : syntax LABEL ASC "TEXT"
  Generates the ASCII codes of the text placed between quotes.

- CHG : syntax CHG CHARACTER,STRING
  Allows modification of the &6 variable by translation, replacing the first character with the given string.

If the result exceeds 6 characters, the string is truncated.
Example: if &6 = ABCABC

```
CHG A,E  → &6 = EBC EBC
CHG A,C  → &6 = CBC CBC
CHG F,A  → &6 = ABCABC
CHG A,EF → &6 = EFB CEF
```

- DATA : syntax LABEL DATA OCT1,OCT2,...
Generates the one-byte representation of each number given.

- DO : syntax DO N
Assembles the text N times, from the DO line until
the pseudo-instruction ENDO.
It is possible to nest loops, and the parser will
not reject them.
Each use of DO initializes a specific variable
which decrements upon encountering an ENDO,
resuming analysis after the last DO encountered.

Example:

```
DO 8
  INC HL
  INC HL
  DEC D
DO 3
  DEC D
  DEC D
ENDO
```

The first DO is not taken into account.

- DS : syntax DS OCT1,OCT2
Generates OCT1 bytes initialized to the value OCT2.
OCT1 must be determined from the start...

IV-1 (continued) PSEUDO-INSTRUCTIONS

- EQU : syntax LABEL EQU EXP
Associates a numeric value with a label.
Example:

```
SETL EQU 3
INC HL
```

- IF : syntax IF EXP
Equivalent to a conditional test: if the
expression is true, the following lines are

processed.
Example:

```
IF 12
PUSH DE
LD HL,12
ENDIF
```

- IFGT : syntax IFGT WORD
  Same as IF but tests if the value is greater.

- IFEQ : syntax IFEQ WORD
  Same as IF but tests for equality.

Note: values are treated as positive 16-bit numbers.

- MAC : syntax LABEL MACRO
  Defines a macro-instruction.
  Must end with ENDM.

- MOVE : syntax MOVE VAL1,VAL2,STRING
  Initializes the &6 variable.
  VAL2 must be less than or equal to VAL1.
  &6 takes the value of the string of length VAL2,
  placed into a space of length VAL1.
  If the string is shorter, blanks are generated.

Examples:

```
MOVE 2,7 "ABCDEF"
MOVE 4,2 "DE"
```

- MUTE : syntax MUTE
  Marks the end of a macro-instruction.
  Causes a return to the calling line with any
  parameters if applicable.

- PAGE : syntax PAGE
  Marks the start of a new assembly.
  Everything before is considered finalized.
  PAGE resets the assembler state.

- SET : syntax SET VAL
  Initializes the given variable to VAL.

IV-2 THE Z80 ASSEMBLY LANGUAGE

Calling a program means that the address of the
instruction following the call is stored on the

system stack. This allows the processor to
resume execution of the program when returning.

The instructions generate the machine operations
directly. Here are some examples:

```
ADD A, BYTE        ; add the byte to A
ADD HL, BC         ; add BC to HL
ADC A, BYTE        ; add with carry
AND A, BYTE        ; logical AND
OR  A, BYTE        ; logical OR
XOR A, BYTE        ; exclusive OR
CP  A, BYTE        ; compare A with the byte
INC REGISTER       ; increment a register
DEC REGISTER       ; decrement a register
JP  ADDRESS        ; unconditional jump
JR  OFFSET         ; relative jump
CALL ADDRESS       ; call subroutine
RET                ; return from subroutine
PUSH REGISTER      ; push register onto stack
POP  REGISTER      ; pop register from stack
EI                 ; enable interrupts
DI                 ; disable interrupts
EX DE, HL          ; exchange DE with HL
EX AF, AF'         ; exchange AF with alternate AF
```

IV-2 (continued) Z80 ASSEMBLY LANGUAGE INSTRUCTIONS

```
PUSH REG      ; push register onto the system stack
POP REG       ; pop register from the stack

CALL ADR      ; call subroutine
RET           ; return from subroutine
RETI          ; return from maskable interrupt
RETN          ; return from non-maskable interrupt

RLC REG       ; rotate left with carry
RRC REG       ; rotate right with carry
RL  REG       ; rotate left through carry
RR  REG       ; rotate right through carry

SLA REG       ; shift left arithmetic (bit 0 cleared, bit 7 into Carry)
SRA REG       ; shift right arithmetic (bit 7 preserved)
SRL REG       ; shift right logical (bit 7 cleared, bit 0 into Carry)
```

```
LD A,(HL)     ; load A with value pointed by HL
LD (HL),A     ; store A into memory pointed by HL
LD REG,BYTE   ; load register with immediate value

NEG           ; A = -A (two's complement)

IN A,(PORT)   ; read input from port into A
OUT (PORT),A  ; output contents of A to port

IN (C)        ; read from port specified in C
OUT (C),REG   ; write register to port specified in C

INI / INIR    ; input with HL increment, repeat until BC = 0
IND / INDR    ; input with HL decrement, repeat until BC = 0
OUTI / OTIR   ; output with HL increment, repeat until BC = 0
OUTD / OTDR   ; output with HL decrement, repeat until BC = 0
```

V MACRO-INSTRUCTION MANAGEMENT

Using macro-instructions

A macro-instruction is a model of an instruction, composed of sequences of instructions that are generated each time the programmer calls them in the main program. They are not subroutines: with each call, the instructions are generated anew.

The purpose is to avoid repetition by providing the programmer with pseudo-instructions that make program writing easier. The usefulness of these macro-instructions depends on how they are defined.

The programmer may be content to use the assembler as a simple assembler, or instead create his own language. This is possible because the tool allows control over the assembly process.

Without strict boundaries, one can distinguish several levels of macro usage:

1) A level for handling repetitions.
2) A level for parameter and option handling.
3) A level for more advanced autogeneration.

The examples that follow belong to the first two levels. Programmers approaching the third

level no longer need guidance, but should
study compiler techniques.

The sub-assembler you have offers the following
possibilities:

- When generating a macro-instruction, the
  number of parameters it expects is specified.
  This number must be between 0 and 5.
  Within the body of the macro-instruction,
  these parameters are referenced as &1, &2, &3, etc.

V-2-3 THE PSEUDO-PARAMETER &0

Calling a macro, for example:
   PUSH BC, PUSH DE, PUSH HL, PUSH IX
generates stacking of the registers on the stack,
which is 4 × 2 = 8 bytes.

You can define a macro-instruction that
automatically performs this:

```
  MACRO PUSHALL &0
    PUSH BC
    PUSH DE
    PUSH HL
    PUSH IX
  ENDM
```

The call:
   PUSHALL
generates the instructions PUSH BC, PUSH DE,
PUSH HL, PUSH IX without having to type them
every time.

The pseudo-parameter &0 indicates the number
of parameters in the call. Its index is stored
in &0. You can consider &0 as a static variable.

Example:

```
   SET &0 = 4
   MACRO MULTIPUSH &0
     PUSH &1
     PUSH &2
     PUSH &3
     PUSH &4
   ENDM
```

The call:
```
   MULTIPUSH BC,DE,HL,AF
```
generates:
```
   PUSH BC
   PUSH DE
   PUSH HL
   PUSH AF
```

V-2-4 THE AUXILIARY CHARACTER VARIABLE &6

Like parameters, this variable is a string of
up to 6 characters. But unlike parameters, it
is a static variable.

Two pseudo-instructions operate on it: MOVE
and CHG.

Example:
```
   MOVE 6,3 "ABC"
```
results in &6 = "ABC   "

```
   CHG A,Z
```
replaces the first character A with Z,
so &6 = "ZBC   "

These manipulations allow you to write macros
more compactly. For example:

```
   MACRO PUSHPOP
     PUSH &6
     POP  &6
   ENDM
```

V-3 USAGE EXAMPLES

V-3-2 USING MACROS TO HANDLE REPETITIONS

The technique is to use a macro-instruction
whose name is intended to replace an instruction.
This can range from simply making an instruction
clearer to creating completely new instructions.

Example:
```
   JP HL
   MACRO JPHL
```

```
    JP (HL)
  ENDM
```

V-3-3 USING MACROS TO HANDLE SUBROUTINE CALLS

Example:

```
  CALL MACRO &1
```

generates CALL followed by the parameter.

VI ERROR MESSAGES

Error messages are linked to the way the analyzer works.
The analysis is performed in two stages: each stage is triggered only if the previous one did not produce an error.

First stage: the analyzer searches for the macro-instruction and the start of the program.
This can lead to several error messages.

Examples of possible errors:
- "INVALID NUMBER": you declared a macro with more than 5 parameters.
- "MULTIPLE MACRO": you declared a macro with a name already in use.
- "INVALID NAME": the name is invalid or too long (names must be at most 4 characters).

VI (continued) ERROR MESSAGES

- STRING INVALID: a character string was used without delimiters.
- LABEL DOUBLE: the label was already defined.
- STACK FULL: an operation caused a stack overflow.

Third stage: syntactic analysis of the code.
It scans all the lines and implements the program. It stops by indicating the exact command in error.
This can lead to the detection of new errors.

- INVALID REFERENCE: a reference to an instruction whose name is mandatory at the previous level.

- INVALID OPERAND: the operand does not match
  the expected form.

VII PARTICULARITIES OF THE ADAM SYSTEM

Without covering all of ADAM's possibilities,
here are some key points to better use the language.

1) Memory sharing:
   ADAM's EOS (Elementary Operating System) manages
   the various peripherals.
   Address $0200 is used by a management table
   to locate the interrupt vectors.
   These addresses are available to the programmer
   and initialized for returns.
   Note: a non-maskable interrupt is triggered by DF.
   The keyboard: address $F020.
   The system returns in A the value of the key pressed.
   CALL $F020 reads a key.

2) The VDP (Video Display Processor).
   This is the video processor used…

VII (continued) PARTICULARITIES OF THE ADAM SYSTEM

2) The VDP (Video Display Processor)
   The VDP is accessed through system routines.
   Example:
     CALL F020 → read a key
     CALL F030 → display to screen
     CALL F040 → set cursor position
     CALL F050 → change background color
     CALL F060 → change text color
     CALL F070 → invert colors
     CALL F080 → initialize the global screen
     CALL F0C0 → access the printer

   For the printer, simply send the ASCII code
   to print. Upon return from F0C0, if everything
   is fine, A contains $0E. Otherwise, another
   value indicates an error.

Example of printing:
```
   TEMP DS 1,0
   PRT  LD (TEMP),8
```

```
REL  LD A,(TEMP)
CALL F0C0
```

3) File particularities

BASIC uses the ALDOS function to load coded programs. However, the system requires certain conventions to properly load them.

As a result, the file must contain an identification block before the generated program.

Example:

```
PGM ADR = xxxx
DATA 1,0,2
DY  L&D
```

Specifying load and execution addresses.

A RESET will automatically reload the block at the specified TSH address.

French Version

SOMMAIRE

Il suffit de mettre la cassette dans le lecteur puis de faire "RESET". Le système se charge et lance le programme.

Dès la première image écran affichée, en appuyant sur les touches "Flèche haute" et "Flèche basse", vous modifierez la couleur du fond et du texte.

II-1 LE MENU DE SÉLECTION

Le premier menu est simplement un menu de sélection d'autres menus :

1 - le mode édition
2 - le menu de contrôle des entrées/sorties
3 - le menu d'assemblage
4 - le menu d'initialisation
5 - le menu des paramètres d'environnement

Pour accéder à l'un de ces menus, il vous suffit de taper soit 1, 2, 3, 4 ou 5.

Ce menu vous permet de choisir la configuration dans laquelle vous allez travailler.

Ce choix permet de sélectionner pour la mémoire des pages, les unités de lecture et d'écriture, ainsi que pour l'imprimante : le choix du papier, le nombre de lignes imprimées et le nombre de lignes dans la page.

Il vous permet aussi d'introduire un titre à afficher au début de vos listes. Pour modifier, positionnez-vous à l'aide des flèches sur l'option. Tapez ensuite la valeur de votre option. Tapez sur "RETOUR" pour revenir au menu principal.

Quand vous introduisez un item, vous pouvez utiliser les caractères alphabétiques ainsi que les touches "droite" et "gauche".

II-2 LE MENU D'INITIALISATION

C'est le plus simple : il demande seulement la confirmation de la destruction du buffer de travail. Son but est d'éviter une destruction du programme suite à une fausse manœuvre.

II-3 LE MENU D'ÉDITION

II-3-1 LE FONCTIONNEMENT

Le principe est simple.

L'écran est une fenêtre qui permet d'introduire, modifier ou supprimer du texte à assembler. Tout texte est écrit dans la zone de travail et affiché à l'écran. Le curseur indique la position d'introduction du caractère suivant.

II-3-2 LES CARACTÈRES SPÉCIAUX

- La barre d'espace : espace
- La touche RETOUR : retour de ligne
- La touche TABULATION : tabulation
- La touche "Flèche haute" : déplacement du curseur vers le haut
- La touche "Flèche basse" : déplacement du curseur vers le bas
- La touche "Flèche gauche" : déplacement du curseur à gauche
- La touche "Flèche droite" : déplacement du curseur à droite
- La touche "DEL" : suppression du caractère
- La touche "INS" : insertion d'un espace
- La touche "CTRL" combinée avec d'autres touches : fonctions spéciales
- La touche "RETURN" : fin de saisie de la ligne et passage à la suivante

Lorsque vous introduisez un texte, vous pouvez utiliser les touches alphabétiques du clavier ainsi que les touches spéciales ci-dessus.
"RETURN" marque la fin d'une ligne.
"CTRL-F" provoque une recherche de texte dans le fichier courant.
"CTRL-L" permet de reformater la ligne.

Tous les caractères spéciaux du système sont regroupés et utilisables depuis le menu principal.

II-3-3 LE RETOUR DE L'ÉDITEUR

Une fois affiché le texte saisi, bien aligné et analysé en conformité, les lignes blanches seront absorbées et non réaffichées après l'opération. L'apparence est donc nettoyée.

La partie "label" est tronquée à 8 caractères. La partie "code opération" est produite sur 4 caractères. La partie "opérande" est laissée telle quelle, avec suppression des blancs qui l'entourent. Ceci n'est valable que pour les lignes qui ne sont pas des commentaires.

II-4 LE MENU DE CONTRÔLE DES ENTRÉES/SORTIES

Il permet de communiquer avec le module extérieur de la même manière que vous imaginez. La lecture et l'écriture se font sur l'unité que vous avez désignée.

Ce menu permet de charger à la suite du buffer programme un autre programme. Si l'on ne veut pas mélanger les programmes, il est nécessaire de détruire le programme précédent.

Ce menu permet aussi la sauvegarde du programme courant et du programme général. Ce dernier peut s'opérer de deux manières :

- Sauvegarde normale : création d'un fichier répertorié contenant le texte. Le fichier généré est organisé entre l'adresse minimum et l'adresse maximum implicitement définies par votre programme.

- Sauvegarde sur un bloc attribué : dans ce cas vous précisez vous-même le numéro du bloc où sauvegarder. Ce numéro doit être une valeur comprise entre 0 et FFFFH. Le système ne l'accepte pas si la taille du fichier est trop longue pour ce bloc.

Le programme est ensuite chargé à partir de ce dernier caractère sauvegardé.

III-2 LES DIFFÉRENTS OPÉRANDES

Les opérandes "octet" : ils représentent une valeur comprise entre 0 et 255. Lorsqu'ils sont utilisés dans une opération, seul l'octet de poids faible du résultat est gardé.

Les opérandes "mot" : ce sont des expressions numériques sur 16 bits, donc comprises entre 0 et 65535. Leur valeur est insérée telle quelle dans le programme.

Les opérandes "expression" : ce sont des formules arithmétiques utilisant des constantes, des étiquettes, et les opérateurs + ou -. Ils peuvent être utilisés comme valeurs numériques.

Exemples :
- constantes décimales : valeurs écrites directement
- constantes hexadécimales : commencent par "E"
- constantes binaires : commencent par "B"

Les instructions sont composées de trois champs :

1) Le champ étiquette : formé de 8 caractères max. Une étiquette est un nom permettant de repérer une adresse. Elle est suivie de deux points (:).

2) Le champ code opération : formé de 4 caractères. Il est destiné à recevoir les opérations Z80, les pseudo-instructions, etc.

3) Le champ opérande : contient soit une valeur, soit une expression, soit une étiquette.

Les étiquettes sont globales dans le programme et permettent de référencer les adresses partout où elles apparaissent.

IV LE LANGAGE

Le langage est composé de trois types d'instructions :
1) Les pseudo-instructions
2) Le langage assembleur Z80
3) L'appel des macro-instructions définies par le programmeur

Les instructions assembleur génèrent directement du langage machine tandis que les pseudo-instructions servent à contrôler le déroulement de l'assemblage.

IV-1 LES PSEUDO-INSTRUCTIONS

- ASC : syntaxe LABEL ASC "TEXTE"
  Génère les codes ASCII du texte mis entre guillemets.

- CHG : syntaxe CHG CARACTERE,CHAINE
  Permet de modifier la variable &6 par traduction en remplaçant le premier caractère par la chaîne.
  Si le résultat dépasse 6 caractères, la chaîne est tronquée.
  Ex : si &6 = ABCABC
    CHG A,E → &6 = EBC EBC
    CHG A,C → &6 = CBC CBC
    CHG F,A → &6 = ABCABC
    CHG A,EF → &6 = EFB CEF

- DATA : syntaxe LABEL DATA OCT1,OCT2,...
  Génère la représentation sur un octet de chaque nombre indiqué.

- DO : syntaxe DO N
  Assemble N fois le texte situé après la ligne DO et avant la pseudo-instruction ENDO.
  Il est possible d'impliquer des boucles imbriquées, bien que l'analyseur ne provoque pas de rejet.
  Chaque utilisation de DO provoque l'initialisation d'une variable spécifique qui se décrémente à la rencontre d'un ENDO, et provoque l'analyse après le dernier DO rencontré.

Exemple :

```
DO 8
  INC HL
  INC HL
  DEC D
DO 3
  DEC D
  DEC D
ENDO
```

Le premier DO n'est pas pris en considération.

- DS : syntaxe DS OCT1,OCT2
  Permet de générer OCT1 octets initialisés à la valeur OCT2.
  OCT1 doit être déterminé dès sa ...

IV-1 (suite) LES PSEUDO-INSTRUCTIONS

- EQU : syntaxe LABEL EQU EXP
  Associe une valeur numérique à un label.

Exemple :
```
SETL EQU 3
INC HL
```

- IF : syntaxe IF EXP
  Équivaut à un test : si l'expression est vraie,
  les lignes qui suivent sont prises en compte.
  Exemple :
```
IF 12
PUSH DE
LD HL,12
ENDIF
```

- IFGT : syntaxe IFGT MOT
  Identique à IF sauf qu'on teste si la valeur
  est supérieure.

- IFEQ : syntaxe IFEQ MOT
  Identique à IF sauf qu'on teste l'égalité.

Remarque : les valeurs sont considérées comme
des nombres positifs de 16 bits.

- MAC : syntaxe LABEL MACRO
  Permet de définir une macro-instruction.
  Elle doit être close par ENDM.

- MOVE : syntaxe MOVE VAL1,VAL2,CHAINE
  Permet d'initialiser la variable &6.
  On doit avoir que VAL2 ≤ VAL1.
  La variable &6 prend la valeur de la chaîne de
  caractères, de longueur VAL2, placée dans une
  zone de longueur VAL1.
  Si la chaîne est plus courte, des blancs sont générés.

Exemples :
```
MOVE 2,7 "ABCDEF"
MOVE 4,2 "DE"
```

- MUTE : syntaxe MUTE
  Marque la fin d'une macro-instruction.
  Cela provoque un retour à la ligne d'appel
  avec, si besoin, des paramètres.

- PAGE : syntaxe PAGE
  Marque le début d'un nouvel assemblage.
  Tout ce qui précède est considéré comme

totalement généré et figé. PAGE marque
l'état initial de l'assemblage.

- SET : syntaxe SET VAL
Initialise la variable donnée à VAL.

IV-2 LE LANGAGE ASSEMBLEUR Z80

L'appel d'un programme signifie que l'adresse de l'instruction suivante à l'appel est rangée dans la pile système. Cela permet au processeur de reprendre l'exécution du programme au retour.

Les instructions réalisent directement les opérations machine. Voici quelques exemples :

```
ADD A, OCTET        ; additionne l'octet à A
ADD HL, BC          ; additionne BC à HL
ADC A, OCTET        ; addition avec retenue
AND A, OCTET        ; ET logique
OR  A, OCTET        ; OU logique
XOR A, OCTET        ; OU exclusif
CP  A, OCTET        ; compare A avec l'octet
INC REGISTRE        ; incrémente un registre
DEC REGISTRE        ; décrémente un registre
JP  ADRESSE         ; saut inconditionnel
JR  DEPLACEMENT     ; saut relatif
CALL ADRESSE        ; appel de sous-programme
RET                 ; retour de sous-programme
PUSH REGISTRE       ; empile le registre
POP  REGISTRE       ; dépile le registre
EI                  ; active les interruptions
DI                  ; désactive les interruptions
EX DE, HL           ; échange DE et HL
EX AF, AF'          ; échange des registres AF
```

IV-2 (suite) INSTRUCTIONS DU LANGAGE ASSEMBLEUR Z80

```
PUSH REG      ; empile le registre dans la pile système
POP REG       ; dépile le registre

CALL ADR      ; appel de sous-programme
RET           ; retour de sous-programme
RETI          ; retour d'interruption masquable
RETN          ; retour d'interruption non masquable
```

```
RLC REG       ; rotation gauche du registre avec report dans Carry
RRC REG       ; rotation droite du registre avec report dans Carry
RL  REG       ; rotation gauche du registre à travers Carry
RR  REG       ; rotation droite du registre à travers Carry

SLA REG       ; décalage logique gauche (bit 0 = 0, Carry reçoit bit 7)
SRA REG       ; décalage arithmétique droite (bit 7 conservé)
SRL REG       ; décalage logique droite (bit 7 = 0, Carry reçoit bit 0)

LD A,(HL)     ; charge A avec le contenu pointé par HL
LD (HL),A     ; stocke A à l'adresse pointée par HL
LD REG,OCTET  ; charge un registre avec une valeur immédiate

NEG           ; fait A = -A (complément à deux)

IN A,(PORT)   ; lit le contenu du port d'entrée dans A
OUT (PORT),A  ; envoie le contenu de A vers un port de sortie

IN (C)        ; lit un port indiqué par C
OUT (C),REG   ; écrit un registre vers un port indiqué par C

INI / INIR    ; lit avec incrémentation HL, répète jusqu'à BC = 0
IND / INDR    ; lit avec décrémentation HL, répète jusqu'à BC = 0
OUTI / OTIR   ; écrit avec incrémentation HL, répète jusqu'à BC = 0
OUTD / OTDR   ; écrit avec décrémentation HL, répète jusqu'à BC = 0
```

V GESTION DES MACRO-INSTRUCTIONS

L'usage des macro-instructions

Une macro-instruction est un modèle d'instruction composé de séquences d'instructions qui seront générées à chaque fois que, dans son programme principal, le programmeur fait appel à elles. Ce ne sont donc pas des sous-programmes : à chaque appel, les instructions sont générées.

L'intérêt est d'éviter des répétitions en fournissant au programmeur des pseudo-instructions qui facilitent l'écriture des programmes. La puissance de ces macro-instructions dépendra de vos définitions.

Le programmeur peut se contenter d'utiliser l'assembleur comme un simple assembleur, ou au contraire créer son propre langage. Cette possibilité est d'autant plus grande que l'outil

permet de contrôler le déroulement de
l'assemblage.

Sans qu'il y ait réellement de frontières,
on peut distinguer plusieurs niveaux
d'utilisation des macros :

1) Un niveau de gestion des répétitions.
2) Un niveau de paramétrage des paramètres
   et des options.
3) Un niveau d'autogénération évoluée.

Les exemples qui suivent appartiennent aux
deux premiers niveaux. Les programmeurs
abordant le troisième niveau n'ont plus besoin
de conseils, mais doivent se documenter
sur les techniques de compilation avancées.

Le sous-assembleur que vous avez dispose des
possibilités suivantes :

- La génération de la macro-instruction
  implique le nombre de paramètres qu'elle
  précède. Ce nombre doit être compris entre
  0 et 5. Dans le corps de la macro-instruction,
  ces paramètres seront désignés par &1, &2, &3, etc.

V-2-3 LE PSEUDO-PARAMÈTRE &0

L'appel d'une macro, par exemple :
   PUSH BC, PUSH DE, PUSH HL, PUSH IX
génère l'empilement des registres sur la pile,
soit 4 × 2 = 8 octets.

On peut définir une macro-instruction qui
réalise cela automatiquement :

```
  MACRO PUSHALL &0
    PUSH BC
    PUSH DE
    PUSH HL
    PUSH IX
  ENDM
```

L'appel :
   PUSHALL
génère les instructions PUSH BC, PUSH DE,

PUSH HL, PUSH IX sans que vous ayez à les taper à chaque fois.

Le pseudo-paramètre &0 indique le nombre de paramètres de l'appel. L'indice de ce paramètre est contenu dans &0. On peut considérer que &0 est une variable statique.

Exemple :

```
  SET &0 = 4
  MACRO MULTIPUSH &0
    PUSH &1
    PUSH &2
    PUSH &3
    PUSH &4
  ENDM
```

L'appel :

```
  MULTIPUSH BC,DE,HL,AF
```

génère :

```
  PUSH BC
  PUSH DE
  PUSH HL
  PUSH AF
```

V-2-4 LA VARIABLE CARACTÈRE AUXILIAIRE &6

Comme les paramètres, cette variable est une chaîne de 6 caractères maximum. Mais contrairement aux paramètres, il s'agit d'une variable statique.

Deux pseudo-instructions permettent d'agir sur elle : MOVE et CHG.

Exemple :

```
  MOVE 6,3 "ABC"
```

donne &6 = "ABC "

```
  CHG A,Z
```

remplace le premier caractère A par Z, soit &6 = "ZBC "

Ces manipulations permettent d'écrire des macros plus compactes. Par exemple :

```
  MACRO PUSHPOP
    PUSH &6
    POP  &6
  ENDM
```

V-3 EXEMPLES D'UTILISATION

V-3-2 UTILISATION POUR GÉRER DES RÉPÉTITIONS

La technique consiste à employer une
macro-instruction dont le nom est destiné à
remplacer une instruction. Cela peut aller du
simple remplacement (rendre une instruction
plus claire) à la création d'instructions
spécifiques.

Exemple :

```
  JP HL
  MACRO JPHL
    JP (HL)
  ENDM
```

V-3-3 USAGE POUR LA GESTION D'APPEL DE SOUS-PROGRAMMES

Exemple :

```
  CALL MACRO &1
```

génère CALL suivi du paramètre.

VI LES MESSAGES D'ERREUR

Les messages d'erreur sont liés à la manière
dont le programme d'analyse travaille.
L'analyse se fait en deux temps : chaque étape
est enclenchée seulement si la précédente
n'a pas donné lieu à une erreur.

Première étape : l'analyseur recherche la
macro-instruction et le début du programme.
Cela peut conduire à plusieurs messages d'erreur.

Exemples d'erreurs possibles :
- "INVALID NUMBER" : vous avez déclaré une macro
  avec plus de 5 paramètres.
- "MULTIPLE MACRO" : vous avez déclaré une macro
  dont le nom était déjà attribué.

- "INVALID NAME" : le nom est incorrect ou trop long
  (le nom doit être formé de 4 caractères maximum).

VI (suite) LES MESSAGES D'ERREUR

- STRING INVALID : une chaîne de caractères est
  utilisée sans être délimitée.
- LABEL DOUBLE : le label est déjà défini.
- STACK FULL : une opération a conduit à une
  saturation de la pile.

Troisième étape : l'analyse syntaxique du code.
Elle parcourt toutes les lignes et implante
le programme. Elle s'interrompt en précisant
exactement la commande en erreur.
Cela peut conduire à la détection de nouvelles erreurs.

- INVALID REFERENCE : il s'agit d'une référence
  à une instruction dont le nom est obligatoire
  au niveau précédent.
- INVALID OPERAND : l'opérande n'est pas de
  la forme attendue.

VII LES PARTICULARITÉS DU SYSTÈME ADAM

Sans pouvoir aborder toutes les possibilités
d'ADAM, il convient de connaître quelques
points pour mieux utiliser le langage.

1) Le partage mémoire :
   L'exploitation (EOS) d'ADAM permet de gérer
   les différents périphériques.
   L'adressage de $0200 par une table de gestion
   situe les vecteurs d'interruptions.
   Les adresses sont à disposition du programmeur.
   Elles sont initialisées pour le retour.
   À noter : une interruption non masquable est
   déclenchée par DF.
   Le clavier : l'adresse $F020.
   Le système renvoie dans A la valeur de la
   touche appuyée.
   CALL $F020 lit une touche.

2) Le VDP (Video Display Processor).
   Il s'agit du processeur vidéo utilisé…

VII (suite) PARTICULARITÉS DU SYSTÈME ADAM

2) Le VDP (Video Display Processor)

Le VDP est accessible par des routines système.

Exemple :

```
CALL F020   → lecture d'une touche
CALL F030   → affichage à l'écran
CALL F040   → positionnement du curseur
CALL F050   → changement de couleur de fond
CALL F060   → changement de couleur de texte
CALL F070   → inversion des couleurs
CALL F080   → initialisation de l'écran global
CALL F0C0   → accès à l'imprimante
```

Pour l'imprimante, il suffit d'envoyer le code ASCII à imprimer. Au retour de F0C0, si tout est OK, A contient $0E. Sinon, une autre valeur signale une erreur.

Exemple d'impression :

```
TEMP DS 1,0
PRT  LD (TEMP),8
REL  LD A,(TEMP)
CALL F0C0
```

3) Particularités des fichiers

Le BASIC gère avec la fonction ALDOS le chargement des programmes codés. Mais il faut savoir que le système a besoin de certains aménagements pour pouvoir charger ces codes.

Il en résulte que le fichier doit contenir avant le programme généré une zone d'identification.

Exemple :

```
PGM ADR = xxxx
DATA 1,0,2
DY  L&D
```

Adresse de chargement et d'exécution.

Un RESET produit le rechargement automatique du bloc à l'adresse TSH.