Macro-Assembleur Alcyane
Version 1.1

Table des matières

Révisions

Le tableau suivant présente les révisions du présent document et les améliorations du macro-assembleur Alcyane.

Version	Modifications
1.0	Version initiale
1.1	Ajout de la directive `RADIX` Correction d'un bug de création d'une étiquette publique Correction d'un bug de gestion des étiquettes des macro-instructions Modification de l'affichage des types de variables

Pourquoi un assembleur Alcyane ?

En général, un assembleur a besoin de deux passes pour compiler : une première passe pour enregistrer toutes les étiquettes et les références, une seconde passe pour générer le code et localiser les erreurs.

Les toutes premières machines Alcyane, à l'époque où elles étaient vendues en kit, ne possédaient qu'une interface pour magnétophone à cassettes. Il était compliqué voire impossible de demander à l'utilisateur de rembobiner entre les deux passes d'assemblage. Pour cela, MBC avait développé un assembleur à une seule passe qui employait un astucieux système de chaînage en mémoire pour éviter de lire deux fois les fichiers sources. Le défaut était qu'aucune étiquette ne pouvait être renseignée dans le listing, ce qui ne simplifiait pas la mise au point.

L'assemblage en une seule passe a perduré jusqu'à la dernière version de l'assembleur Alcyane. Entre temps, MBC avait ajouté la possibilité de précompiler les modules puis de les lier. Le résultat était un gain de temps extrêmement appréciable (une minute au lieu de vingt pour assembler un basic).

Malheureusement, l'assembleur Alcyane n'a que peu évolué et il a gardé des capacités limitées. Pour contourner ces limitations et simplifier le développement, j'ai décidé d'écrire un macro-assembleur pour Windows un peu plus performant tout en restant globalement compatible avec Alcyane et les quelques fichiers sources que j'ai pu reconstituer au moyen d'un désassembleur.

Ce macro-assembleur est destiné à Alcyane et, pour cela, ne génère que des fichiers binaires linéaires, c'est-à-dire destinés à être implantés en mémoire à une adresse unique. Il ne gère pas de sections séparées pour le code et les données ni plusieurs sections de code.

Il possède, par contre, certains avantages comme la gestion des macro-instructions, la compilation conditionnelle, la génération automatique de données en virgule flottante, la traduction des minuscules accentuées en codes Alcyane, la gestion des structures de données, la génération de fichiers HEX et la génération d'une liste de symboles compatible avec l'émulateur Alcyane. Il permet également de générer des fichiers binaires qui peuvent être importés par l'émulateur Alcyane ou ayant exactement la taille d'une eprom de type 2708, 2716 ou 2732 (1, 2 ou 4 kilo-octets).

Le programme de l'assembleur

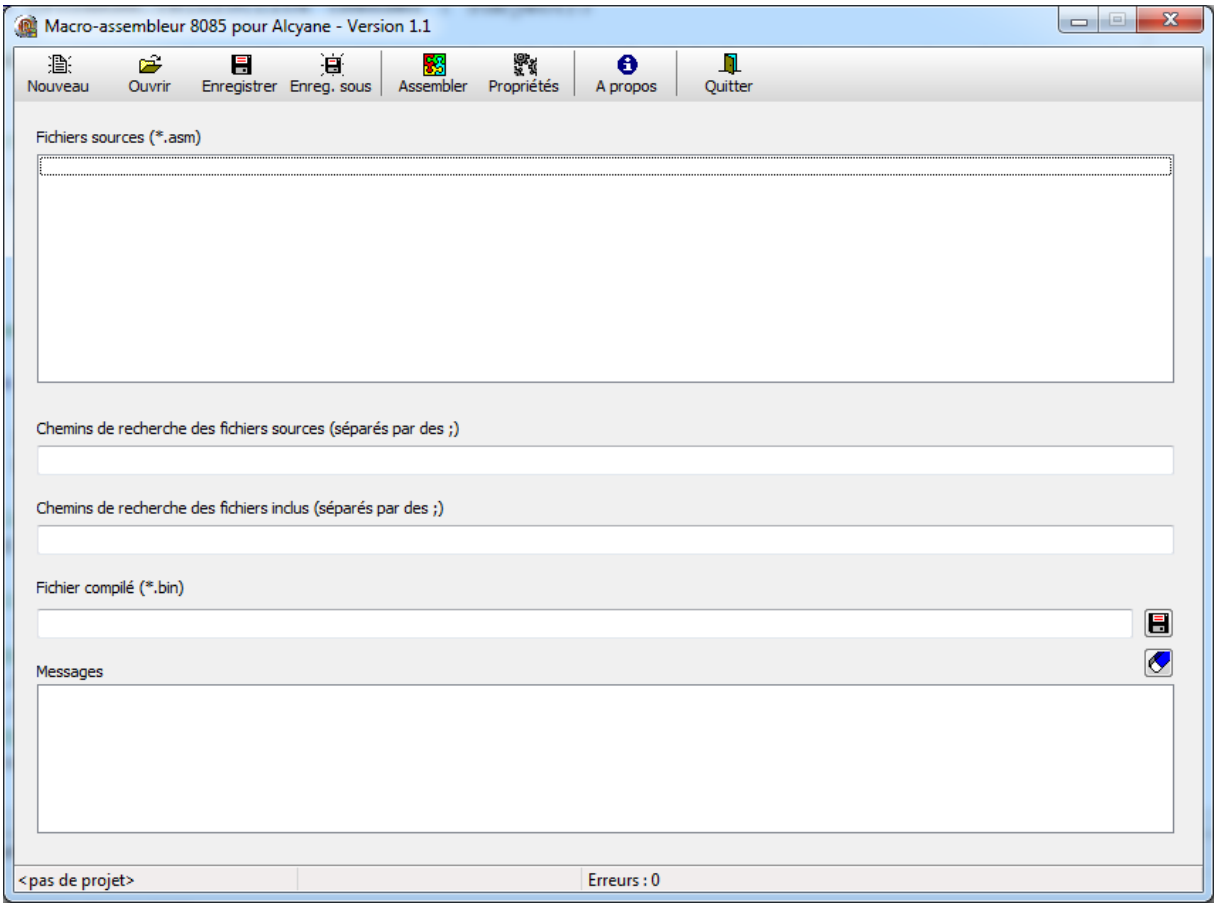
L'assembleur fonctionne en mode « fenêtre » et se compose d'un seul fichier `AsmAlcy.exe`. Pour les plus curieux, cet assembleur est écrit en Delphi, le Pascal étant mon langage de prédilection.

Par rapport à l'assembleur Alcyane, il propose quelques améliorations notables :

- Un fonctionnement par projets au moyen de la directive `USE`.
- Une compilation en deux passes qui génère des listings entièrement exploitables.
- La gestion de la compilation conditionnelle (`IF` / `ELSE` / `ELSEIF` / `ENDIF`).
- La gestion des macro-instructions (`MACRO` / `ENDM`).
- La gestion des structures de données (`STRUCT` / `ENDS`).
- La gestion de fichiers inclus (`INCLUDE`).
- La définition de variables exportées (`PUBLIC`).
- La définition simplifiée de valeurs en virgule flottante (`DF`).
- Des messages d'erreur explicites.
- La gestion d'expressions complexes dans les instructions.
- Une possible compatibilité avec les anciens fichiers sources d'Alcyane.

Exécution de l'assembleur

L'exécution du fichier `AsmAlcy.exe` affiche la fenêtre principale de l'assembleur :



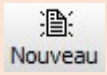
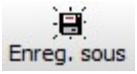
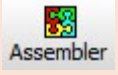
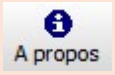
Sur cette fenêtre apparaissent :

- Une barre de boutons pour piloter l'assembleur.
- Une zone de saisie des fichiers à assembler.
- Une zone de saisie des chemins d'accès aux fichiers sources ajoutés par la directive `USE`.
- Une zone de saisie des chemins d'accès aux fichiers inclus par la directive `INCLUDE`.
- Une zone de saisie du nom du fichier binaire généré.
- Une zone d'affichage des messages générés par l'assembleur (informations ou erreurs).
- Une zone indiquant le nom du projet, son état modifié ou non et le nombre d'erreurs.

Utilisation de l'assembleur

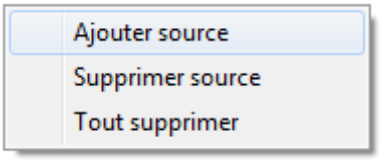
La barre de boutons

Tout en haut de la fenêtre de l'assembleur se trouve une barre de boutons permettant de créer et gérer un projet et de lancer l'assemblage.

Bouton	Fonction
	Ce bouton permet de créer un nouveau projet vierge. Si un projet est déjà présent, l'assembleur propose de l'enregistrer.
	Ce bouton permet d'ouvrir un projet déjà créé afin de le modifier ou de lancer l'assemblage.
	Ce bouton permet d'enregistrer le projet présent en mémoire.
	Ce bouton permet d'enregistrer le projet présent en mémoire sous un nom différent.
	Ce bouton permet de lancer l'assemblage du projet en mémoire.
	Ce bouton permet d'afficher la fenêtre des propriétés du projet présent en mémoire.
	Ce bouton permet d'afficher les informations de version du programme.
	Ce bouton permet de quitter le programme.

La zone de saisie des fichiers sources

La zone intitulée « Fichiers sources (`*.asm`) » permet de créer et modifier la liste des fichiers assembleur constituant un projet. Elle utilise un menu pop-up qui est affiché en cliquant sur le bouton de droite de la souris :



- Cliquez sur « Ajouter source » pour faire apparaître le dialogue de sélection de fichiers. La validation d'un nom de fichier l'ajoute à la fin de la liste.

- Sélectionnez la ligne contenant le nom de fichier à supprimer et cliquez sur « Supprimer source » pour le faire disparaître de la liste.

- Cliquez sur « Tout supprimer » pour effacer tous les noms de fichiers.

Les chemins de recherche des fichiers sources

La zone intitulée « Chemins de recherche des fichiers sources (séparés par des ;) » contient la liste des chemins d'accès aux fichiers du projet et aux fichiers définis par la directive `USE` (voir plus loin). Si plusieurs noms sont présents dans la liste, ils doivent être séparés par des points-virgules.

Cette liste est mise à jour automatiquement à chaque ajout de fichier source. Elle peut être modifiée si nécessaire.

Les chemins de recherche des fichiers inclus

La zone intitulée « Chemins de recherche des fichiers inclus (séparés par des ;) » contient la liste des chemins d'accès aux fichiers inclus par la directive `INCLUDE` (voir plus loin). Si plusieurs noms sont présents dans la liste, ils doivent être séparés par des points-virgules.

Par défaut, cette liste est identique à celle des fichiers sources. Elle peut être modifiée si nécessaire.

Le nom du fichier compilé

La zone intitulée « Fichier compilé (`*.bin`) » permet de définir le nom du fichier résultant de l'assemblage du projet. Par défaut, le nom du binaire est celui du projet avec une extension « `.bin` ». Il est possible de choisir un autre nom de fichier en cliquant sur le bouton  situé à droite de la zone d'affichage.

Si l'utilisateur demande la génération d'un fichier de symboles pour l'émulateur, il sera créé dans le même répertoire que le fichier compilé. Le nom du fichier des symboles est basé sur le nom du projet et se termine par « `_sym.txt` ».

Si l'utilisateur demande l'enregistrement de la liste des messages, le fichier sera créé dans le même répertoire que le fichier compilé. Le nom du fichier des messages est basé sur le nom du projet et se termine par « `_msg.txt` ».

Si la génération d'un fichier au format HEX est demandée, le fichier sera créé dans le même répertoire que le fichier compilé. Le nom du fichier au format HEX est basé sur le nom du projet avec une extension « `.hex` ».

Les messages de fonctionnement et d'erreurs

La zone d'affichage des messages se présente ainsi :

La zone défile au fur et à mesure de l'affichage de nouveaux messages. Un ascenseur permet de voir les messages les plus anciens. Le bouton « gomme » situé en haut à droite permet d'effacer tous les messages.

Les messages d'erreur sont affichés en rouge pour les repérer plus facilement.

Travail avec un projet

Un projet est constitué des éléments suivants :

- Une liste de fichiers sources (`*.asm`) à assembler l'un après l'autre.
- Des chemins de recherche pour les fichiers.
- Un nom de fichier binaire généré.
- Diverses propriétés.

Un fichier projet contient tous ces éléments dans un fichier de texte aisément éditable si nécessaire. La structure d'un fichier projet est décrite plus loin.

Créer un projet

Pour créer un projet vierge, cliquez sur le bouton « Nouveau ». Ensuite, ajoutez un ou plusieurs fichiers sources (`*.asm`) en utilisant le clic droit dans la zone de saisie, vérifiez que les chemins d'accès aux fichiers permettent à l'assembleur de lire tous les fichiers, définissez les propriétés du projet puis sauvegardez-le en cliquant sur le bouton « Enregistrer ».

Ouvrir un projet

Cliquez sur le bouton « Ouvrir » pour charger tous les éléments d'un projet.

Modifier un projet

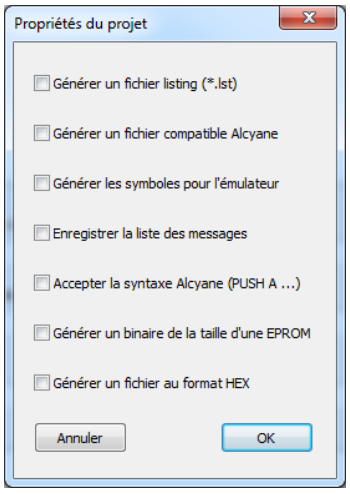
Quand est projet est chargé, vous pouvez éditer tous les champs et les propriétés du projet : ajout ou suppression de fichiers sources, modification des chemins d'accès, propriétés d'assemblage, etc.

Enregistrer un projet

Après création ou modification d'un projet, cliquez sur le bouton « Enregistrer ». Si le projet est nouveau et n'a pas encore de nom, l'assembleur vous proposera d'en créer un.

Propriétés d'un projet

Cliquer sur le bouton « Propriétés » fait apparaître la liste suivante :



Générer un fichier listing (.lst)*

Cocher cette case permet de générer un fichier de texte contenant le résultat complet de l'assemblage. Le fichier généré possède le nom du projet et l'extension « `.lst` ».

Générer un fichier compatible Alcyane

Cocher cette case permet de générer un fichier binaire ayant une taille multiple de 253 octets qui peut être importé par l'émulateur.

Générer les symboles pour l'émulateur

Cocher cette case permet de générer une liste des symboles du projet qui peut être importé par l'émulateur, ce qui simplifie la mise au point.

Enregistrer la liste des messages

Cocher cette case permet d'enregistrer la liste des messages générés par l'assembleur. Ceci permet d'analyser plus facilement les éventuelles erreurs.

Accepter la syntaxe Alcyane (PUSH A …)

L'assembleur Alcyane avait quelques bizarreries comme par exemple le remplacement des instructions `PUSH PSW` et `POP PSW` respectivement par `PUSH A` et `POP A`. De plus, l'assembleur Alcyane comprenait les directives `IND`, `RES` et `DATA` détaillées plus loin.

Cocher cette case permet de compiler sans problème d'anciens fichiers sources Alcyane.

Générer un binaire de la taille d'une EPROM

Cocher cette case permet, quand c'est possible, de générer un fichier binaire ayant exactement 1024, 2048 ou 4096 octets pour la programmation d'une EPROM de type 2708, 2716 ou 2732.

Générer un fichier au format HEX

Cocher cette case permet de générer un fichier au format HEX Intel. Le fichier généré possède une extension « `.hex` ».

Assembler un projet

Après avoir défini les fichiers sources, les chemins d'accès aux fichiers et les propriétés d'un projet, l'assemblage du projet peut être lancé en cliquant sur le bouton « Assembler ». Le résultat de l'assemblage est résumé dans la zone d'affichage des messages.

Contenu d'un fichier de projet

Un fichier de projet est constitué, comme un fichier de type « `.ini` », de sections dont les noms apparaissent entre crochets [ ] et de données.

Voici un exemple de fichier de projet :

```
[Produit]
Produit=Assembleur
[Sources]
K:\Projets\Alcyane\ASM\Sources Alcyane\PATB\PATB.asm
[Binaire]
K:\Projets\Alcyane\ASM\Sources Alcyane\PATB\PATB.bin
[Options]
GenereList=1
GenereAlcy=0
GenereSymEmu=0
EnregMsg=0
SyntaxeAlcy=0
GenereEprom=0
GenereHex=0
[Chemins]
Sources=K:\Projets\Alcyane\ASM\Sources Alcyane\PATB\
Inclus=
```

La section [Produit]

La section `[Produit]` contient le nom du produit. Ceci permet de différencier les fichiers de projet des divers logiciels que j'ai écrits : Assembleur, Émulateur, etc.

La section [Sources]

La section `[Sources]` contient les noms et chemins d'accès aux fichiers sources constituant le projet. Ici, il n'y a qu'un seul fichier source : `PATB.asm`. Les différents fichiers constituant un projet apparaissent sur des lignes différentes.

La section [Binaire]

La section `[Binaire]` contient le nom du fichier binaire généré par l'assembleur, ici : `PATB.bin`.

La section [Options]

La section `[Options]` contient les cinq propriétés associées à un projet :

- `GenereList` : génération d'un fichier listing.
- `GenereAlcy` : génération d'un fichier compatible Alcyane.
- `GenereSymEmu` : génération de la liste des symboles pour l'émulateur.
- `EnregMsg` : génération de la liste des messages.
- `SyntaxeAlcy` : support de la syntaxe de l'assembleur Alcyane.
- `GenereEprom` : génération d'un binaire pour EPROM de 1024, 2048 ou 4096 octets.
- `GenereHex` : génération d'un fichier au format HEX Intel.

La valeur qui suit le nom de la propriété indique si elle est activée (=1) ou désactivée (=0).

La section [Chemins]

La section `[Chemins]` définit les chemins d'accès aux fichiers sources (variable `Sources`) ou aux fichiers inclus (variable `Inclus`). Le nom de la variable est suivi du ou des chemins d'accès séparés par des « ; ».

Syntaxe des fichiers assembleur

Ce paragraphe n'est pas un cours de programmation en assembleur ni une liste des instructions du 8085. Vous trouverez la description complète du 8085 ici :
http://www.bitsavers.org/components/intel/MCS80/MCS80_85_Users_Manual_Jan83.pdf

Les fichiers à assembler sont des fichiers de texte constitués de suites de lignes indépendantes. Dans une ligne, les minuscules sont équivalentes aux majuscules. La syntaxe générale d'une ligne est :

```
[Étiquette[:]] [Instruction [Paramètres]] [Commentaire]
```

Tous ces champs sont optionnels.

Le champ Étiquette

Ce champ est toujours placé en début de ligne. Pour être pris en compte, il ne doit pas être précédé d'un espace ou d'une tabulation. Il doit obligatoirement commencer par une lettre ou un « _ ». Les caractères suivants peuvent être des lettres, des chiffres, « . » ou « _ ».

Le champ étiquette peut se terminer par le caractère « : » afin d'améliorer la compatibilité avec d'autres assembleurs.

Par exemple :

```
AFFICHE  LXI  H,1000H   ;adresse de début de l'écran
DEBUT:   LXI  SP,0000H  ;pointeur de pile
```

Une étiquette peut contenir un ou plusieurs points. Ceci n'est réellement utile que pour faire référence aux champs d'une structure (voir plus loin). Je conseille de réserver l'usage des points aux structures pour ne pas rendre le programme incompréhensible.

Par exemple :

```
DECR55  MVI  A,55H      ;syntaxe valide mais à éviter
A.2     DCR  A
        JNZ  A.2
```

Le champ Instruction

Ce champ ne doit pas se trouver en début de ligne. Si une étiquette est présente sur la ligne, il doit en être séparé par au moins un espace ou une tabulation.

Le champ instruction peut contenir une instruction du microprocesseur comme dans l'exemple ci-dessus, un appel de macro ou une directive de compilation :

```
        MVI   A,'3'
CONV    MACRO P1,P2
        IF  DISQUETTE=0
CR      EQU   0DH
        CONV 41H,1200H
```

Le champ Paramètres

Ce champ contient les paramètres de l'instruction assembleur ou de la directive :

```
AFFICHE LXI  H,1000H
        CALL CLAVIER
PI      DF   3.1415926535897
```

Le champ Paramètres doit être séparé du champ Instruction par au moins un espace ou une tabulation.

Le champ Commentaire

Ce champ débute par un « . » ou un « ; ». Son contenu est ignoré par l'assembleur. Par exemple :

```
AFFICHE LXI H,1000H  ;adresse de début de la RAM vidéo
```

Pour des questions de lisibilité, il est conseillé de laisser quelques espaces entre les paramètres de l'instruction et le commentaire. La plupart des éditeurs de texte récents permettent d'aligner les champs d'une ligne source, ce qui en améliore la lisibilité. Je suggère d'écrire les commentaires en minuscules pour en améliorer la lisibilité.

Compilation conditionnelle

Le macro-assembleur pour Alcyane permet d'assembler certaines portions de code de façon conditionnelle. Cela peut être utile si, par exemple, le code doit gérer certains périphériques et pas d'autres.

La compilation conditionnelle emploie quatre directives : `IF`, `ELSEIF`, `ELSE` et `ENDIF`. Un bloc de compilation conditionnelle débute obligatoirement par `IF` et se termine par `ENDIF`. Les directives `ELSEIF` et `ELSE` doivent être placées entre `IF` et `ENDIF`.

Plusieurs blocs `IF` / `ENDIF` peuvent être imbriqués.

La directive `IF` doit être suivie d'une expression dont la valeur doit être connue au moment où la directive est rencontrée. Le code qui suit la directive `IF` sera traité si le résultat de l'expression est différent de 0.

Exemple :

```
A6E   EQU 1
      IF A6E
        MVI  A,81H   ;commutation RAM entre 0 et 1FFFH
        OUT  8FH
      ENDIF
```

La directive `ELSE` permet de traiter le cas où la directive `IF` donne un résultat nul. Par exemple :

```
INVERSE EQU 1
        IF INVERSE
          ORI  80H
        ELSE
          ANI  7FH
        ENDIF
```

La directive `ELSEIF` permet de gérer de multiples cas à la façon de l'instruction « *case* » des langages évolués. Par exemple :

```
A6    EQU 0
A6E   EQU 1
A10   EQU 0
      IF A10
        LXI  H,2000H
      ELSEIF A6
        LXI  H,0000H
      ELSE
        MVI  A,20H
        OUT  7FH
        LXI  H,0000H
      ENDIF
```

Macro-instructions

Les macro-instructions permettent de créer un code plus clair et plus lisible en évitant les erreurs dues à des copier-coller mal réalisés.

La définition d'une macro-instruction débute par le mot-clé `MACRO` et se termine par le mot-clé `ENDM`.

Le mot-clé `MACRO` doit être précédé d'une étiquette et optionnellement suivi d'un ou plusieurs paramètres. Par exemple, cette macro-instruction porte le nom `CONV` et accepte un seul paramètre dont le nom est `VALEUR` :

```
CONV    MACRO VALEUR
```

À l'intérieur d'une macro-instruction, les paramètres doivent être précédés de « `&` » pour être reconnus. Lors de l'assemblage, les paramètres réels remplacent les paramètres formels.

Exemple complet de macro-instruction :

```
;convertit un octet 00..0F en son code ASCII
CONV    MACRO VALEUR
        MVI   A,&VALEUR
        ANI   0FH
        CPI   0AH
        CMC
        ACI   30H
        DAA
        ENDM
```

Pour utiliser cette macro-instruction, il suffit de placer son nom suivi du paramètre dans le code assembleur :

```
        CONV 0AH
```

Si une macro-instruction admet plusieurs paramètres, ils doivent être séparés par des virgules. Par exemple :

```
;convertit un octet 00..0F en code ASCII et le stocke en mémoire
CONV    MACRO VALEUR, ADRESSE
        MVI   A,&VALEUR
        ANI   0FH
        CPI   0AH
        CMC
        ACI   30H
        DAA
        STA   &ADRESSE
        ENDM
```

Son appel se fera de cette façon, par exemple :

```
        CONV  0BH,BUFFER
```

Si la macro-instruction nécessite un paramètre contenant une virgule, il est possible de placer le paramètre entre guillemets ou entre apostrophes :

```
INST    MACRO INSTRUCTION
        &INSTRUCTION
        ENDM
```

L'appel de la macro-instruction se fera ainsi :

```
        INST "MVI  A,33H"
```

Si une étiquette est présente dans une macro-instruction, celle-ci est nécessairement locale et ne peut pas apparaître dans une expression hors de la macro-instruction. Par exemple :

```
;Renvoie CY=1 si le caractère dans A est 0AH ou 0DH
CRLF    MACRO
        CPI  0AH
        JZ   VU
        CPI  0DH
        JZ   VU
        ORA  A    ;CY=0
        RET
VU      STC       ;CY=1
        RET
        ENDM
```

L'étiquette `VU` n'est pas accessible à l'extérieur de la macro-instruction `CRLF`.

Lors de l'assemblage, chaque ligne du listing d'une macro-instruction est précédée d'un ou plusieurs signes « + » selon la profondeur de la macro-instruction. Par exemple :

```
                    CONV    MACRO
                            CPI  0AH
                            JC   A000
                            ADI  07H
                    A000    ADI  30H
                            STAX D
                            INX  D
                            ENDM


                    BIN2ASC MACRO P1,P2
                            PUSH H
                            PUSH D
                            LHLD &P2
                            XCHG
                            LHLD &P1
                            MOV  A,M
                            ANI  0F0H
                            RRC
                            RRC
                            RRC
                            RRC
                            CONV
                            MOV  A,M
                            ANI  0FH
                            CONV
                            POP  D
                            POP  H
                            ENDM
```

```
                                BIN2ASC 3412H,6000H
+ 0000H  E5                       PUSH H
+ 0001H  D5                       PUSH D
+ 0002H  2A 00 60                 LHLD 6000H
+ 0005H  EB                       XCHG
+ 0006H  2A 12 34                 LHLD 3412H
+ 0009H  7E                       MOV  A,M
+ 000AH  E6 F0                    ANI  0F0H
+ 000CH  0F                       RRC
+ 000DH  0F                       RRC
+ 000EH  0F                       RRC
+ 000FH  0F                       RRC
+                                 CONV
++ 0010H  FE 0A                    CPI  0AH
++ 0012H  DA 17 00                 JC   A000
++ 0015H  C6 07                    ADI  07H
++ 0017H  C6 30           A000     ADI  30H
++ 0019H  12                       STAX D
++ 001AH  13                       INX  D
+
+ 001BH  7E                       MOV  A,M
+ 001CH  E6 0F                    ANI  0FH
+                                 CONV
++ 001EH  FE 0A                    CPI  0AH
++ 0020H  DA 25 00                 JC   A000
++ 0023H  C6 07                    ADI  07H
++ 0025H  C6 30           A000     ADI  30H
++ 0027H  12                       STAX D
++ 0028H  13                       INX  D
+
+ 0029H  D1                       POP  D
+ 002AH  E1                       POP  H

                                END
```

Le nom d'une macro-instruction se comporte comme toutes les étiquettes. Seule une macro-instruction définie comme publique (soit par une directive `PUBLIC`, soit par absence de `PUBLIC` dans le module où elle est définie) est visible depuis tous les modules.

Structures

Les structures permettent de grouper des données afin d'en simplifier la gestion et de permettre des évolutions sans modifications profondes du code.

La définition d'une structure débute par le mot-clé `STRUCT` et se termine par le mot-clé `ENDS`. Le mot-clé `STRUCT` doit être précédé par une étiquette. Par exemple :

```
ENTETE STRUCT
FMT    DS  1   ;indicateur de disque formaté
NODSQ  DS  2   ;numéro de disque
DERSEC DS  2   ;numéro de secteur libre
       ENDS
```

Une structure ne peut contenir que des définitions de données ou de zones de stockage (directives `DB`, `DW`, `DS`, `DF`, `IND`, `DATA`, `RES`, `SYSDATE` et `SYSTIME`) ou une sous-structure.

Il n'est pas obligatoire de nommer chacun des champs d'une structure. Par exemple :

```
STR    STRUCT
NUM    DB  01H
       DB  00H
LGR    DW  0002H
ADR    DW  1234H
       ENDS
```

Une structure peut contenir une ou plusieurs sous-structures. Par exemple :

```
DSQ    STRUCT
DRIVE  DS 1
HDR    ENTETE
NLA    STR
       ENDS
```

La déclaration d'une structure ne génère aucun code. Pour cela, il faut « instancier » la structure. Par exemple :

```
;structures pour les disques 1 et 2
DSQ1   DSQ   ;DSQ1 et DSQ2 sont des structures de type DSQ
DSQ2   DSQ
```

Les champs des structures sont accessibles en utilisant des étiquettes particulières où les noms des champs sont séparés par des « . ». Par exemple, en utilisant le code ci-dessus :

```
   LHLD DSQ1.HDR.DERSEC
   LXI  D,DSQ2.DRIVE
```

Si une sous-structure ne possède pas de nom, tous ses champs sont considérés comme appartenant à la structure mère. Par exemple :

```
VARIAB STRUCT
TYPE   DS 1
VALEUR DS 2
       ENDS

VALX   STRUCT
LGR    DB  2
       VARIAB
       ENDS

X1     VALX
```

Pour référencer les champs de `X1`, par exemple, il faut utiliser `X1.LGR`, `X1.TYPE` et `X1.VALEUR`.

Dans le cas où une structure est référencée par un pointeur, il est possible d'obtenir un « offset » par rapport au début de la structure en utilisant non pas le nom de la structure instanciée mais le nom de sa déclaration. Par exemple :

```
   LHLD ADRDSQ           ;HL = pointeur sur une structure de type DSQ
   LXI  B,ENTETE.DERSEC  ;BC = 0003 = offset du champ DERSEC
   DAD  B                ;HL = adresse du champ DERSEC
```

Ceci permet de modifier le format d'une structure sans avoir à modifier le programme ni rechercher tous les endroits où la position d'un champ est utilisée.

Si une déclaration ou l'instanciation d'une structure sont déclarées publiques, tous les champs de cette structure le sont également.

Lors de l'instanciation d'une structure, chaque ligne est précédée du signe « > » dans le listing. Par exemple :

```
3000H                       ORG  3000H

                    ENTETE  STRUCT
                    FMT     DS  1
                    NODSQ   DS  2
                    SCTLIB  DS  2
                            ENDS

                    DSQ     STRUCT
                    NUMERO  DS  1
                    INIT    DS  1
                    ;en-tête
                    ENTDSQ  ENTETE
                    ;descripteur fichier
                    DESFIC  DESCR
                            ENDS


                    DSQ1    DSQ
> 3000H               DSQ1.NUMERO  DS  1
> 3001H               DSQ1.INIT    DS  1
>                     ;en-tête
> 3002H               DSQ1.ENTDSQ
> 3002H               DSQ1.ENTDSQ.FMT     DS  1
> 3003H               DSQ1.ENTDSQ.NODSQ   DS  2
> 3005H               DSQ1.ENTDSQ.SCTLIB  DS  2
>                     ;descripteur fichier
> 3007H               DSQ1.DESFIC.NOM     DS  6
> 300DH               DSQ1.DESFIC.C1      DS  2
> 300FH               DSQ1.DESFIC.C2      DS  2
> 3011H               DSQ1.DESFIC.C3      DS  3
```

Notez que l'étiquette `DSQ1.ENTDSQ` est ajoutée automatiquement par l'assembleur de façon à créer une étiquette contenant son adresse et la taille de la sous-structure (voir opérateur `SIZE`).

Autres directives

La directive EQU

Cette directive permet d'affecter une valeur numérique à une étiquette. Elle doit donc être précédée d'une étiquette. La valeur qui suit la directive est une expression dont la valeur doit être connue au moment où la directive est rencontrée. Par exemple :

```
CR      EQU  0DH
LF      EQU  0AH
FINMEM EQU  0FFFFH
BUFFER EQU  FINMEM-1000H
```

La directive ORG

Cette directive permet de définir l'adresse à laquelle le code assemblé sera exécutable. Elle ne doit pas être précédée d'une étiquette. La valeur qui suit la directive est une expression numérique dont la valeur doit être connue au moment où la directive est rencontrée. Par exemple :

```
       ORG  2020H
DEBUT EQU  4000H
       ORG  DEBUT+0100H
```

Il ne doit y avoir qu'une seule directive `ORG` dans un programme. L'emploi de plusieurs directives `ORG` peut donner des résultats inattendus voire inexploitables.

La directive RADIX

Cette directive permet de définir la façon dont le macro-assembleur traite les valeurs numériques. Elle ne doit pas être précédée d'une étiquette.

La directive doit être suivie d'une lettre majuscule ou minuscule parmi les valeurs suivantes :

Lettre	Base
N ou n	Décimal
H ou h	Hexadécimal
Q ou q	Octal
X ou x	Binaire

En l'absence de directive `RADIX`, toutes les valeurs numériques dont le type n'est pas explicité par une lettre N, H, Q ou X sont considérées comme décimales.

La directive `RADIX` a une action globale. Elle reste valide jusqu'à la fin de l'assemblage ou bien jusqu'à une nouvelle directive `RADIX`. Par exemple :

```
                    RADIX N
0000H   64    A     DB   100
                    RADIX Q
0001H   FF    B     DB   377
                    RADIX H
0002H   33    C     DB   33
                    RADIX X
0003H   55    D     DB   01010101
```

Voir également le paragraphe Les valeurs numériques plus loin.

La directive DB

Cette directive permet de définir des données numériques sur un octet ou des textes ou bien une combinaison des deux. La directive `DB` est suivie d'une liste d'expressions séparées par des virgules.

Dans ces expressions, les textes doivent être encadrés par des apostrophes ou des guillemets. Par exemple :

```
MSG   DB 'Système Alcyane prêt',0DH
CR    EQU 0DH
LF    EQU 0AH
CRLF  DB CR, LF
AINV  DB "A" OR 80H  ;A en inversion vidéo
```

La directive DW

Cette directive permet de définir des mots de 16 bits. Elle est particulièrement utile pour créer des tables d'indirection. Par exemple :

```
TABLE DW LECT12  ;lecture secteur disque 1
      DW LECT12  ;lecture secteur disque 2
      DW LECT34  ;lecture secteur disque 3
      DW LECT34  ;lecture secteur disque 4
      DW LECT56  ;lecture secteur disque 5
      DW LECT56  ;lecture secteur disque 6
```

La directive `DW` peut être suivie d'une ou plusieurs expressions dont les résultats numériques sont codés sur 16 bits. Par exemple :

```
BAUDS DW 110,300,600,1200,2400,4800,9600
```

La fonction `SIZE` peut alors être utilisée pour déterminer le nombre d'entrées dans la table :

```
NBBAUDS EQU SIZE (BAUDS) / 2
```

La directive DF

Cette directive permet de définir une valeur en virgule flottante codée sur 8 octets au format IEEE 754 sans avoir à effectuer de conversion. Cela rend les programmes sources plus lisibles. Par exemple :

```
PI   DF 3.14159265358979323 ;valeur de Pi
```

Le séparateur décimal doit être un point. Si un commentaire existe après une directive `DF`, il doit obligatoirement être placé après un point-virgule.

La directive DS

Cette directive permet de réserver une zone mémoire qui sera utilisée pour stocker des données. Par exemple :

```
BUFCLA  DS 192    ;Buffer clavier longueur 192 octets
PTRECR  DS 2      ;Pointeur sur la mémoire écran
```

La directive SYSDATE

Cette directive permet de générer une chaîne de 10 caractères ASCII contenant la date de compilation au format JJ/MM/AAAA. Par exemple :

```
3000H               ORG   3000H
3000H  32 38 2F 31  DATE  SYSDATE
3004H  30 2F 32 30
```

La directive SYSTIME

Cette directive permet de générer une chaîne de 8 caractères ASCII contenant l'heure de compilation au format HH:MM:SS. Par exemple :

```
4000H                ORG   4000H
4000H  31 34 3A 33   TIME  SYSTIME
4004H  36 3A 31 30
```

La directive PUBLIC

Cette directive permet d'indiquer à l'assembleur qu'une étiquette est publique et peut être « vue » par d'autres fichiers sources. En l'absence de directive `PUBLIC`, toutes les étiquettes du module sont publiques. Dès que la directive `PUBLIC` est rencontrée, toutes les étiquettes qui suivent sont locales au module.

La directive `PUBLIC` peut être suivie d'une ou plusieurs étiquettes séparées par des virgules. Par exemple :

```
        PUBLIC ADDITION
        PUBLIC SIN, COS, TAN
```

Remarque : pour des questions de lisibilité et pour éviter des erreurs de compilation, il est conseillé de placer les directives `PUBLIC` au début de chaque module.

La directive USE

Cette directive est utilisée pour créer des listes de fichiers assemblés dans un projet. Par exemple :

```
;fichier MATH.asm
        USE SIN
        USE COS
        USE TAN
```

Les fichiers `SIN.asm`, `COS.asm` et `TAN.asm` seront traités comme s'ils se trouvaient dans la liste des fichiers du projet. Il n'est pas indispensable d'ajouter « `.asm` » après un nom de module, l'assembleur le fait automatiquement si aucune extension n'est présente.

Afin d'améliorer la lisibilité, il est conseillé d'employer un seul fichier `.asm` dans le projet et de placer tous les noms des fichiers à assembler dans ce fichier au moyen de directives `USE`. L'utilisation des directives `IF`, `ELSE`, `ELSEIF` et `ENDIF` permettent de gérer différentes versions du programme assemblé.

La directive INCLUDE

Cette directive permet d'inclure un fichier source dans un module. Cela peut s'avérer utile, par exemple, pour ajouter des définitions à un module. Si la directive `INCLUDE` se trouve après une directive `PUBLIC`, tous les éléments inclus seront locaux au module. Par exemple :

```
;fichier inclus DEFCRLF.asm
CR    EQU 0DH
LF    EQU 0AH
APOS  EQU 27H
GUILL EQU 22H

;fichier CLAVIER.asm
         PUBLIC  LECTCLAV ;LECTCLAV est publique
         INCLUDE DEFCRLF  ;CR, LF, APOS et GUILL sont locales
LECTCLAV IN  0E2H
         RLC
         JNC LECTCLAV
         IN  0C2H
         OUT 0E2H
         CPI CR
         JZ  TRAITECR
         CPI LF
         JZ  TRAITELF
         CPI APOS
         JZ  TRAITEAPOS
         CPI GUILL
         JZ  TRAITEGUILL
         JMP LECTCLAV
```

Il n'est pas indispensable d'ajouter « `.asm` » après un nom de fichier inclus, l'assembleur le fait automatiquement si aucune extension n'est présente.

Différences entre INCLUDE et USE

La directive `USE` ajoute les fichiers à assembler à la fin de la liste des fichiers du projet tandis que la directive `INCLUDE` insère un fichier source à l'intérieur d'un module.

Il est déconseillé de constituer un fichier d'assemblage général au moyen de directives `INCLUDE` car, dès lors qu'une directive `PUBLIC` sera rencontrée, toutes les étiquettes seront considérées comme locales au module et cela risque de poser des problèmes lors de l'assemblage (étiquettes dupliquées ou inaccessibles, par exemple).

La directive RES

Cette directive n'est utilisée que dans les fichiers sources originaux d'Alcyane. Elle est strictement équivalente à la directive `DS`. Elle n'est reconnue que si la compatibilité avec la syntaxe Alcyane est activée dans les paramètres du projet.

La directive IND

Cette directive n'est utilisée que dans les fichiers sources originaux d'Alcyane. Elle permet de créer une valeur à deux octets de la même façon que la directive `DW`. Elle n'accepte cependant qu'un seul paramètre. Elle n'est reconnue que si la compatibilité avec la syntaxe Alcyane est activée dans les paramètres du projet.

Les directives DATA

Ces directives ne sont utilisées que dans les fichiers sources originaux d'Alcyane. Elles sont constituées du mot-clé `DATA` suivi d'une lettre puis de paramètres. Les trois lettres possibles sont :

A : définition d'un texte, par exemple `DATA A Système Alcyane prêt`. Un caractère 0DH est ajouté à la fin du texte.

B : définition d'un texte, comme `DATA A` mais sans ajout du 0DH final.

H : définition de données binaires sous forme de paires de caractères hexadécimaux, par exemple `DATA H 01 7F 45 9B`.

Exemples :

```
DATA B Système   ;pas de CR ajouté
DATA A Alcyane   ;CR ajouté en fin de texte
DATA H 21 00 10 36 41 C9
```

Les directives `DATA` ne sont reconnues que si la compatibilité avec la syntaxe Alcyane est activée dans les paramètres du projet.

La directive END

Cette directive est optionnelle. Elle marque la fin de la partie compilable d'un fichier. Elle peut être utilisée pour masquer une partie de source en cours de développement, par exemple.

Les expressions

Dans l'assembleur, toutes les valeurs numériques peuvent être des expressions constituées d'étiquettes, de constantes numériques ou alphanumériques et d'opérateurs.

Les opérateurs

Les opérateurs reconnus sont les suivants :

Opérateur	Fonction	Priorité
+	Addition	2
-	Soustraction ou négation	2
*	Multiplication	1
/	Division	1
MOD	Reste de la division	1
SHR	Décalage à droite d'un bit	1
SHL	Décalage à gauche d'un bit	1
NOT ()	Inversion logique de tous les bits	3
AND	Et logique	2
OR	Ou logique	2
XOR	Ou exclusif logique	2
HIGH ()	Octet de poids fort	3
LOW ()	Octet de poids faible	3
SIZE ()	Taille d'un élément	3
>	Supérieur	2
<	Inférieur	2
>=	Supérieur ou égal	2
<=	Inférieur ou égal	2
=	Égal	2
<>	Différent	2
( )	Parenthèses pour grouper des calculs	
$	Pointeur PC courant	

Les opérateurs de priorité 1 sont évalués avant les opérateurs de priorité 2, etc. Par exemple :

```
A   EQU   55H
B   EQU   NOT (A) + 3    ;B = FFADH
```

Il est cependant recommandé d'utiliser des parenthèses pour grouper les termes d'une expression afin de rendre les fichiers sources plus lisibles et sans ambiguïté.

Important : pour les fonctions `NOT`, `HIGH`, `LOW` et `SIZE`, les parenthèses entourant l'argument sont obligatoires.

Les valeurs numériques

Dans une expression, les valeurs numériques peuvent être exprimées en quatre bases différentes : décimal, hexadécimal, binaire ou octal. Par défaut, la base décimale est utilisée.

Pour spécifier la base d'une valeur numérique, il faut ajouter une lettre à la fin de la déclaration de la valeur :

- N ou n : décimal.
- H ou h : hexadécimal.
- X ou x : binaire.
- Q ou q : octal.

Par exemple :

```
MVI  A,10h
LXI  D,1000H+(20N*128N)
LXI  H,0A451H
ADI  01011101X
XRI  377q
```

Il est important de noter qu'une valeur numérique doit toujours débuter par un chiffre. En hexadécimal, il faut donc ajouter un « 0 » d'en-tête si la valeur est supérieure à 9FFFH, sans quoi l'assembleur interprétera la valeur comme une étiquette (`0A000H` est une constante numérique, `A000H` est un symbole).

La directive RADIX permet de modifier la base par défaut (voir plus haut).

Les valeurs alphanumériques

Dans une expression, les valeurs alphanumériques doivent être placées entre guillemets ou entre apostrophes. Par exemple :

```
DB "Ceci est une constante alphanumérique"
DB 'Pour représenter l''apostrophe, il faut la doubler'
```

Les instructions du microprocesseur peuvent également recevoir des constantes alphanumériques. Par exemple :

```
MVI  A, "9"   ;équivalent à MVI  A, 39H
```

Une constante alphanumérique peut également intervenir dans une expression dont le résultat est un octet. Par exemple :

```
MVI  M, "A" OR 80H ;inversion vidéo
```

Une constante alphanumérique constituée de deux caractères exactement est compatible avec une valeur numérique sur 16 bits. Il est ainsi possible d'écrire :

```
LXI  H,"AB"
IF  "AB" < "CD"
```

Les opérateurs relationnels

Les opérateurs relationnels >, <, >=, <=, = et <> réalisent une comparaison des termes à droite et à gauche de l'opérateur. Un opérateur relationnel génère 0 (faux) ou 1 (vrai) selon le résultat de la comparaison. Par exemple :

```
A   EQU 1 < 2         ;génère 1
B   EQU 1 > 2         ;génère 0
C   EQU 1 >= 2        ;génère 0
D   EQU 1 <= 2        ;génère 1
E   EQU 1 = 2         ;génère 0
F   EQU 1 <> 2        ;génère 1
G   EQU "AB" < "BC" ;génère 1
```

Utilisation des fonctions

Fonction MOD

La fonction `MOD` calcule le modulo (reste de la division) de son paramètre. Par exemple :

```
A   EQU   33N
B   EQU   5N
C   EQU   A MOD B ;C = 33 modulo 5 = 3
```

Fonctions SHR et SHL

Les fonctions `SHR` et `SHL` réalisent respectivement un décalage à gauche et à droite de tous les bits de la valeur. Par exemple :

```
A   EQU   1 SHL 3    ;1 SHL 3 = 08H
B   EQU   1 SHL 6    ;1 SHL 6 = 40H
C   EQU   A OR B     ;C = 48H
```

Fonctions AND, OR, XOR et NOT

Les fonctions `AND`, `OR`, `XOR` et `NOT` réalisent respectivement les opérations logiques et, ou, ou exclusif et inversion logique. Par exemple :

```
A   EQU   7CH
B   EQU   A AND 0FH    ;B = 0CH
C   EQU   A OR 80H     ;C = 0FCH
D   EQU   A XOR 0CH    ;D = 70H
E   EQU   NOT A        ;E = 83H
```

Fonctions HIGH et LOW

Les fonctions `HIGH` et `LOW` renvoient respectivement l'octet de poids fort et l'octet de poids faible du résultat d'une expression. Par exemple :

```
A   EQU 7C45H
B   EQU LOW (A)    ;B = 45H
C   EQU HIGH (A)   ;C = 7CH
```

Fonction SIZE

La fonction `SIZE` renvoie la taille d'un élément. Ceci est particulièrement utile pour les champs des structures mais peut également être employé sur un élément de type `DB`, `DW`, `DF`, `DS`, `IND`, `RES` ou `DATA` ou bien une sous-structure. Par exemple :

```
;Déclaration de la structure
DESCR  STRUCT
NDV    DS 1
SCTLIB DS 2
NOM    DB 'DISQUE 1',0DH
       ENDS

;Instanciation de la structure
DSQ1   DESCR

;Taille du champ NOM obtenue de deux façons différentes
       LXI  B,SIZE (DESCR.NOM)
       LXI  D,SIZE (DSQ1.NOM)

;Centrage de titre
TITRE  DB  'Titre de la fenêtre', 0DH
MARGE  EQU (80N-SIZE(TITRE))/2
```

Fonction $

La fonction `$` renvoie le pointeur courant lors d'un assemblage. Par exemple :

```
FATAL  JMP $   ;erreur fatale -> boucle infinie
```

Le symbole _END_

Le symbole spécial `_END_` est généré automatiquement par l'assembleur. Il contient l'adresse qui suit immédiatement la dernière instruction assemblée. Ce symbole peut être utilisé, par exemple, pour recopier un logiciel chargé en 2020H à l'adresse 0000H (comme le fait le basic).

Le symbole _ORG_

Le symbole spécial `_ORG_` est généré automatiquement par l'assembleur. Il contient l'adresse de la dernière directive `ORG`. En général, une seule directive `ORG` est autorisée, sans quoi le résultat de l'assemblage est imprévisible voire inexploitable.

Le listing d'assemblage

L'assembleur peut fournir un listing complet de l'assemblage si l'option « Générer un fichier listing (*.lst) » est activée.

Le listing d'assemblage contient toutes les lignes assemblées ainsi que diverses informations comme les adresses des instructions, les codes hexadécimaux générés et les erreurs éventuelles. Par exemple :

```
                    ;Test de chargement des offsets
307DH  21 00 30     OFFSET  LXI   H,DSQ1
3080H  01 14 00             LXI   B,DSQ.DESFIC.C2
3083H  09                   DAD   B
3084H  C9                   RET

                    ;Tests de Size
3085H  06 05                MVI   B,SIZE (ENTETE)
3087H  0E 0D                MVI   C,SIZE (DESCR)
3089H  16 14                MVI   D,SIZE (DSQ)

308BH  11 01 00             LXI   D,SIZE (DSQ1.NUMERO)
308EH  01 05 00             LXI   B,SIZE (DSQ.ENTDSQ)
3091H  21 05 00             LXI   H,SIZE (DSQ1.ENTDSQ)
3094H  3E 0A                MVI   A,SIZE (NOMS.NOM1)
3096H  26 08                MVI   H,SIZE (NOMS.NOM3)
```

Le listing de chaque module est suivi de la liste des étiquettes locales à ce module. En fin d'assemblage, la liste des étiquettes du projet est ajoutée. Par exemple :

```
3076H  P DEBUT                  3000H  s DESCR
0006H    DESCR.C1               0008H    DESCR.C2
000AH    DESCR.C3               0000H    DESCR.NOM
3000H  S DSQ                    0019H PC DSQ.DESFIC
0012H  C DSQ.DESFIC.C1          0014H  C DSQ.DESFIC.C2
0016H  C DSQ.DESFIC.C3          000CH PC DSQ.DESFIC.NOM
0007H  C DSQ.ENTDSQ             0002H PC DSQ.ENTDSQ.FMT
0003H PC DSQ.ENTDSQ.NODSQ       0005H PC DSQ.ENTDSQ.SCTLIB
0001H PC DSQ.INIT               0000H PC DSQ.NUMERO
3000H    DSQ1                   3007H    DSQ1.DESFIC
300DH    DSQ1.DESFIC.C1         300FH    DSQ1.DESFIC.C2
3011H    DSQ1.DESFIC.C3         3007H    DSQ1.DESFIC.NOM
3002H    DSQ1.ENTDSQ            3002H    DSQ1.ENTDSQ.FMT
3003H    DSQ1.ENTDSQ.NODSQ      3005H    DSQ1.ENTDSQ.SCTLIB
3001H    DSQ1.INIT              3000H    DSQ1.NUMERO
3014H    DSQ2                   301BH    DSQ2.DESFIC
3021H    DSQ2.DESFIC.C1         3023H    DSQ2.DESFIC.C2
3025H    DSQ2.DESFIC.C3         301BH    DSQ2.DESFIC.NOM
3016H    DSQ2.ENTDSQ            3016H    DSQ2.ENTDSQ.FMT
3017H    DSQ2.ENTDSQ.NODSQ      3019H    DSQ2.ENTDSQ.SCTLIB
3015H    DSQ2.INIT              3014H    DSQ2.NUMERO
3000H  s ENTETE                 0000H    ENTETE.FMT
0001H    ENTETE.NODSQ           0003H    ENTETE.SCTLIB
3000H    ESSAI                  3028H    NOMS
3063H    NOMS.C1                3065H    NOMS.C2
3067H    NOMS.C3                305DH    NOMS.NOM
3028H    NOMS.NOM1              3032H    NOMS.NOM2
3034H    NOMS.NOM3              303CH    NOMS.VAL1
306AH    NOMS.VAL2              307DH  P OFFSET
3000H    TSTNOM                 003BH    TSTNOM.C1
003DH    TSTNOM.C2              003FH    TSTNOM.C3
0035H    TSTNOM.NOM             0000H    TSTNOM.NOM1
000AH    TSTNOM.NOM2            000CH    TSTNOM.NOM3
0014H    TSTNOM.VAL1            0042H    TSTNOM.VAL2
```

Les étiquettes peuvent être précédées de leur type codé par une lettre. Les différents types sont les suivants :

Type d'étiquette	Marquage
Structure publique	S
Champ d'une structure publique	C
Macro-instruction publique	M
Structure locale à un module	s
Champ d'une structure locale	c
Macro-instruction locale	m
Aucun type particulier	Pas de marque

La lettre P qui peut précéder un type indique une variable publique comme c'est le cas pour la variable `DEBUT` de l'exemple.

Les messages d'erreur et d'information

Il existe deux types de messages d'erreur et d'information :

- Les messages généraux qui apparaissent dans la zone « Messages » en bas de la fenêtre principale ou dans une fenêtre spéciale au centre de l'écran.
- Les messages liés à l'assemblage qui apparaissent dans le listing d'assemblage.

Les messages généraux

<nom de fichier> n'est pas un projet pour l'assembleur

Le fichier projet n'est pas destiné à l'assembleur Alcyane. Vérifiez que le champ `Produit` dans la section `[Produit]` est égal à « `Assembleur` ».

Aucun fichier source à assembler

Le projet ne contient aucun fichier source.

Fichier source absent <nom du fichier>

Le fichier source <nom du fichier> n'a pas été trouvé dans les répertoires du projet.

Aucun fichier destination

Le fichier résultat de l'assemblage (.bin) n'a pas été spécifié dans le projet.

Lignes assemblées : <x>

Ce message indique le nombre de lignes assemblées.

Octets générés : <x>

Ce message indique le nombre d'octets utiles dans le fichier assemblé.

Erreurs : <x>

Ce message indique le nombre total d'erreurs trouvées durant l'assemblage.

Temps écoulé : <x> ms

Ce message indique le temps écoulé (en millisecondes) pendant l'assemblage.

Fin d'assemblage

Ce message indique la fin de l'assemblage.

Les messages liés à l'assemblage

Tous les messages d'erreur ou d'information liés à l'assemblage indiquent la passe d'assemblage entre crochets, le nom du fichier incriminé et la ligne où l'erreur a été détectée. Par exemple :

```
[2] Fichier testrel.asm, Ligne 9, mauvais type de variable
```

Les erreurs marquées « `Erreur fatale` » mettent fin immédiatement à l'assemblage.

Binaire généré trop grand pour une Eprom de 4 k-octets

L'option « Générer un binaire de la taille d'une EPROM » a été activée, mais le résultat de l'assemblage dépasse 4 kilo-octets.

Génération d'un fichier binaire de <taille> octets pour eprom

Ce message indique que le binaire généré tient dans une EPROM et spécifie sa taille.

Échec de l'écriture du fichier binaire

Le fichier résultat de l'assemblage n'a pas pu être écrit. Vérifiez qu'il n'est pas ouvert dans un autre programme.

Parenthèse ouvrante manquante

Il manque une parenthèse ouvrante dans une expression.

Parenthèse fermante manquante

Il manque une parenthèse fermante dans une expression.

Erreur dans une constante numérique

La valeur numérique est invalide ou son spécificateur de format est inconnu.

Erreur dans une constante alphanumérique

Il manque une apostrophe ou un guillemet à la fin d'une expression alphanumérique.

Variable <nom> inconnue

Dans une expression, la variable dont le nom est indiqué n'est pas définie.

Erreur dans la valeur numérique <valeur>

Les chiffres d'une valeur binaire, octale, décimale ou hexadécimale ne sont pas corrects.

Expression incomplète ou manquante

Une expression est attendue mais aucune n'est présente.

Mauvais paramètre pour SIZE

L'expression entre parenthèses après la fonction `SIZE` n'est pas une étiquette valide.

Erreur indéterminée

Il s'agit d'une erreur interne de l'assembleur. Merci de me la rapporter avec le fichier source fautif.

Mauvais type de variable

L'expression mélange des données numériques et alphanumériques incompatibles.

Division par zéro

La division ou la fonction `MOD` provoquent une division par zéro.

Erreur interne : variable inconnue

Il s'agit d'une erreur interne de l'assembleur. Merci de me la rapporter avec le fichier source fautif.

Valeur supérieure à 255 donc tronquée

L'assembleur attend une expression dont le résultat est inférieur ou égal à 255. Ce message indique que la valeur a été réduite pour ne pas arrêter l'assemblage. Il peut s'agir d'une réduction volontaire, mais je suggère de l'éviter.

Valeur supérieure à 65535 donc tronquée

L'assembleur attend une expression dont le résultat est inférieur ou égal à 65535. Ce message indique que la valeur a été réduite pour ne pas arrêter l'assemblage. Il peut s'agir d'une réduction volontaire, mais je suggère de l'éviter.

Erreur de syntaxe dans une expression

L'expression est mal écrite. Vérifiez les parenthèses et la compatibilité des opérandes.

Zone réservée trop grande (<taille> octets)

Dans une directive `DS` ou `RES`, la zone réservée dépasse 65536 octets.

Mauvaise valeur en virgule flottante

Dans une directive `DF`, la valeur en virgule flottante est mal écrite.

Erreur dans une directive DATA

Dans une directive `DATA A`, `DATA B` ou `DATA H`, les données ne sont pas correctes.

Étiquette ignorée avant INCLUDE

Une étiquette est présente avant la directive `INCLUDE`, ce qui est interdit.

Étiquette ignorée avant RADIX

Une étiquette est présente avant la directive `RADIX`, ce qui est interdit.

Directive INCLUDE : fichier <nom> non trouvé

Le fichier spécifié dans la directive `INCLUDE` n'a pas été trouvé dans les chemins de recherche des fichiers inclus.

La structure <nom> n'existe pas

La structure dont le nom est indiqué n'existe pas ou n'est pas accessible (si son nom n'est pas public, par exemple).

Instruction <instruction> inconnue dans une structure

Le mot-clé ou l'instruction n'est pas admis dans une structure. Seuls sont acceptés `DB`, `DW`, `DF`, `DS`, `IND`, `RES` et `DATA` ainsi qu'un nom de structure.

Ligne mal terminée

La ligne contient des caractères au-delà de la fin de l'instruction ou de la directive.

Erreur fatale : valeur de IF indéfinie

L'assembleur ne peut pas évaluer l'expression située après une directive `IF`. Cette expression doit obligatoirement être calculable au moment où la directive est rencontrée.

Erreur fatale : ELSEIF sans IF

L'assembleur a rencontré une directive `ELSEIF` qui n'est pas précédée d'une directive `IF`.

Erreur fatale : ELSEIF interdit après ELSE

L'assembleur a rencontré une directive `ELSEIF` située après une directive `ELSE`.

Erreur fatale : ELSE sans IF

L'assembleur a rencontré une directive `ELSE` qui n'est pas précédée d'une directive `IF`.

Erreur fatale : ENDIF sans IF

L'assembleur a rencontré une directive `ENDIF` qui n'est pas précédée d'une directive `IF`.

Étiquette ignorée avant IF, ELSE, ELSEIF ou ENDIF

L'étiquette présente avant une directive de compilation conditionnelle est ignorée.

Étiquette <nom> redéfinie dans la macro <macro>

La macro-instruction dont le nom est indiqué contient une étiquette redéfinie.

Étiquette <nom> redéfinie

L'étiquette dont le nom est indiqué a déjà été définie auparavant.

Erreur fatale : valeur de EQU indéfinie

L'assembleur ne peut pas évaluer l'expression située après une directive `EQU`. Cette expression doit obligatoirement être calculable au moment où la directive est rencontrée.

Étiquette ignorée avant ORG

L'étiquette présente avant la directive `ORG` est ignorée.

Erreur fatale : adresse de ORG indéfinie

L'assembleur ne peut pas évaluer l'expression située après une directive `ORG`. Cette expression doit obligatoirement être calculable au moment où la directive est rencontrée.

Instruction inconnue

L'assembleur n'a pas reconnu l'instruction. Il s'agit probablement d'une faute de frappe ou d'une directive réservée de l'assembleur Alcyane alors que son utilisation n'a pas été validée.

Étiquette ignorée avant PUBLIC

L'étiquette présente avant la directive `PUBLIC` est ignorée.

Étiquette PUBLIC <nom> redéfinie

L'étiquette indiquée a déjà été définie comme publique.

Étiquette PUBLIC <nom> invalide

Le nom indiqué n'est pas valide en tant qu'étiquette.

Étiquette ignorée avant USE

L'étiquette présente avant la directive `USE` est ignorée.

Directive USE : module <nom> non trouvé

Le nom de fichier indiqué n'a pas été trouvé dans la liste des chemins de recherche des fichiers sources.

Directive MACRO : absence du nom de la macro

Le nom de la macro-instruction est absent après une directive `MACRO`.

Mauvais paramètre pour la macro <nom>

Le nom du paramètre ne respecte pas la syntaxe d'un nom de paramètre (débute par une lettre, contient des lettres, des chiffres et le symbole « _ »).

Directive MACRO interdite dans une macro

La directive `MACRO` ne peut pas se trouver à l'intérieur d'une définition de macro-instruction.

Directive PUBLIC interdite dans une macro

La directive `PUBLIC` ne peut pas se trouver à l'intérieur d'une définition de macro-instruction.

Directive ENDM mal placée

La directive `ENDM` n'est pas précédée d'une directive `MACRO`.

Directive STRUCT : absence de nom de la structure

Le nom de la structure n'a pas été spécifié.

Directive ENDS mal placée

La directive `ENDS` n'est pas appairée à une directive `STRUCT`.

Macro <nom> inexistante

La macro-instruction dont le nom est indiqué n'a pas été déclarée préalablement.

Mauvais nombre de paramètres pour la macro <nom>

Le nombre de paramètres formels ne correspond pas au nombre de paramètres réels dans l'appel de macro-instruction.

Mauvais nom de registre (<registre>)

L'instruction ne reconnaît pas le registre dont le nom est indiqué.

Virgule manquante

Il manque une virgule dans l'instruction.

Utiliser HLT plutôt que MOV M,M

Bien que l'instruction `MOV M,M` génère le code `76H` (code de l'instruction `HLT`), je recommande l'usage de l'instruction `HLT`.

Mauvais paramètre pour RST (<valeur>)

L'instruction `RST` doit être suivie d'une valeur numérique comprise entre 0 et 7.

Erreur fatale : déséquilibre IF / ELSE / ELSEIF / ENDIF

Il existe un déséquilibre entre les instructions de compilation conditionnelle.

Trucs et astuces

Ce paragraphe regroupe quelques astuces de programmation qui peuvent aider à développer du code simple et performant.

Conversion d'un quartet en code ASCII

Il est fréquent d'avoir besoin d'afficher le code ASCII d'une valeur numérique. La conversion d'un octet en son équivalent sous forme de texte hexadécimal à deux caractères peut être effectuée avec le code suivant :

```
        LXI  H,1000H    ;emplacement d'affichage sur l'écran
        MVI  A,0D5H     ;valeur à convertir
        PUSH PSW        ;sauvegarde de la valeur
        RRC             ;0D5H -> 05DH
        RRC
        RRC
        RRC
        CALL CONV       ;conversion du quartet de poids faible
        POP  PSW        ;restaure la valeur
CONV    ANI  0FH        ;conserve les 4 bits faibles
        CPI  0AH        ;CY = 1 si A < 0AH
        CMC             ;CY = 1 si A >= 0AH
        ACI  30H        ;ajoute 30H ou 31H selon CY
        DAA             ;convertit en décimal
                        ;00H..09H -> 30H..39H
                        ;0AH..0FH -> 41H..46H
        MOV  M,A        ;stocke le caractère ASCII
        INX  H          ;adresse suivante
        RET
```

Ce sous-programme peut aisément être adapté pour convertir une valeur de 16 bits en ASCII.

Recherche d'un mot-clé dans une table

Ce genre de sous-programme est utile pour créer un embryon de langage. Tout d'abord, créons la table des mots-clés à reconnaître. Dans cette table, les mots-clés sont séparés par le caractère « : » et le code `0FFH` marque la fin de la table.

```
TBLCMD  DB  'LOAD:SAVE:VARS:ERASE:'
        DB  'LIST:NEW:COPY:'
        DB  'PRINT:EDIT:DIGITS'
        DB  0FFH
```

En entrée, la zone de mémoire contenant le texte à analyser est pointée par la paire de registres DE. En sortie, l'accumulateur A vaut 0 si aucun mot-clé n'a été reconnu. Sinon, A contient une valeur de 1 à 10 selon le mot-clé reconnu.

```
ANABUF  PUSH H              ;sauvegarde les registres
        PUSH D
        PUSH B
        LXI  H,TBLCMD       ;HL pointe la table des commandes
        DCX  H              ;HL pointe juste avant le début de la table
        MVI  B,00H          ;B = index du premier mot de la table
        PUSH D              ;sauvegarde l'adresse du texte à analyser
ANA000  POP  D              ;DE pointe le début du texte à analyser
        PUSH D
BCLCMP  INX  H              ;cherche le début d'un mot-clé dans la table
        MOV  A,M            ;A = caractère de la table
        CPI  0FFH
        JZ   FINTAB         ;saut si fin de table
```

```
        CPI   ':'           ;caractère séparateur ?
        JNZ   BCLCMP        ;saut si pas caractère séparateur
        INX   H             ;HL pointe après le séparateur
        INR   B             ;index de la commande suivante
        DCX   H             ;compense INX H suivant
ANA001  INX   H
        MOV   A,M
        CPI   ':'           ;caractère séparateur ?
        JZ    TROUVE        ;fin de comparaison sur séparateur
        LDAX  D             ;caractère du texte à analyser
        INX   D
        CMP   M             ;compare à un caractère de la table
        JNZ   ANA000        ;saut si pas d'égalité
        JMP   ANA001        ;suite de la comparaison
TROUVE  MOV   A,B           ;A = index de la commande trouvée (1..n)
        POP   D             ;restaure les registres
        POP   B
        POP   H             ;2 x POP H pour préserver DE
        POP   H             ;qui pointe après le mot-clé trouvé
        RET
FINTAB  XRA   A             ;mot-clé pas trouvé (A = 0)
        POP   D
        POP   B
        POP   D
        POP   H
        RET
```

Selon le mot-clé reconnu, le programme peut se brancher sur l'un des sous-programmes listés dans la table suivante :

```
EXECMD  DW   EXLOAD         ;commande LOAD
        DW   EXSAVE         ;commande SAVE
        DW   EXVARS         ;commande VARS
        DW   EXERASE        ;commande ERASE
        DW   EXLIST         ;commande LIST
        DW   EXNEW          ;commande NEW
        DW   EXCOPY         ;commande COPY
        DW   EXPRINT        ;commande PRINT
        DW   EXEDIT         ;commande EDIT
        DW   EXDIGIT        ;commande DIGIT
```

Pour cela, il est possible d'utiliser la valeur renvoyée par le sous-programme `ANABUF` ci-dessus :

```
SAUTCMD ORA   A             ;test si A = 0
        RZ                  ;retour si commande inconnue
        LXI   H,EXECMD      ;HL = adresse de la table des sous-programmes
        DCR   A             ;ramène A entre 0 et n
        ADD   A             ;A = 2 x A
        ADD   L             ;A = A + L
        MOV   L,A           ;L = A + L
        ADC   H             ;A = A + L + H + CY
        SUB   L             ;A = A + H + CY
        MOV   H,A           ;H = A + H + CY
        MOV   A,M           ;A = poids faible d'une entrée de la table
        INX   H             ;pointe le poids fort
        MOV   H,M           ;H = poids fort
        MOV   L,A           ;L = poids faible
        PCHL                ;branche sur le sous-programme
```