MANUEL
DE REFERENCE
ASSEMBLEUR
8080

TABLE DES MATIERES

1. INTRODUCTION

L'assembleur du CP/M lit des fichiers source en langage Assembleur sur la disquette, et produit un langage machine 8080 en format Hex Intel. L'assembleur du CP/M est lancé en frappant :

ASM nomfichier

ou

ASM nomfichier.parms

Dans les deux cas, l'assembleur suppose qu'il y a sur la disquette un fichier portant le nom :

nomfichier.ASM

qui contient un fichier source langage assembleur 8080. La première et la deuxième forme décrites ci-dessus, diffèrent seulement en ce que la seconde forme permet de passer des paramètres à l'assembleur pour contrôler l'accès au fichier source et les destinations des fichiers Hex et d'impression.
Dans tous les cas, l'assembleur du CP/M est chargé, et il imprime le message :

VP/M ASSEMBLER VER n.n

où n.n est le numéro de la version actuelle. Dans le cas de la première commande, l'assembleur lit le fichier source supposé être de type "ASM" et crée deux fichiers de sortie :

nomfichier.HEX

et

nomfichier.PRN

Le fichier "HEX" contient le code machine correspondant au programme d'origine en format hex Intel, et le fichier "PRN" contient un listing avec des annotations montrant le code machine généré, les drapeaux d'erreur, et les lignes source. Si des erreurs apparaissent pendant la translation, elles seront données dans le fichier PRN ainsi qu'à la console. La seconde forme de commande peut être utilisée pour spécifier les fichiers d'entrée et de sortie au lieu de les laisser prendre des valeurs par défaut. Dans ce cas, la portion "parms" de la commande est un groupe de trois lettres qui spécifie l'origine du fichier source, la destination du fichier hex et la destination du fichier d'impression.

La forme est la suivante :

nomfichier.p1p2p3

où p1,p2,p3 sont 3 lettres

p1: A,B,...,Y désigne le nom du disque qui contient le fichier source.

p2: A,B,...,Y désigne le nom du disque qui recevra le fichier hex.

Z saute la création du fichier hex.

p3: A,B,...,Y désigne le nom du disque qui recevra le fichier d'impression

X sort le listing sur l'écran

Z saute la création du fichier d'impression

Ainsi la commande :

ASM X.AAA

indique que le fichier source (X.ASM) doit être pris dans le disque A, et que les fichiers hex (X.HEX) et d'impression (X.PRN) doivent être créés aussi sur le disque A. Cette forme de commande est aussi prise si l'assembleur tourne à partir du disque A. Autrement dit, la commande ci-dessus, si l'opérateur a adressé le disque A, est équivalente à :

ASM X

La commande :

ASM X.ABX

indique que le fichier source doit être pris dans le disque A, que le fichier hex est placé sur le disque B, et que le fichier de listing doit être envoyé sur la console. La commande :

ASM X.BZZ

prend le fichier source sur le disque B, et saute la création des fichiers hex et prn (cette commande est utile pour une exécution rapide de l'assembleur pour vérifier la syntaxe du programme).

Le format du programme source est compatible avec l'assembleur 8080 d'Intel (les macros ne sont cependant pas implémentés dans l'assembleur du CP/M actuellement), et avec l'assembleur du Processor Technogy Software No 1. Autrement dit, l'assembleur du CP/M accepte des programmes source écrits dans n'importe quel format. Il y a certaines extensions dans l'assembleur du CP/M qui le rendent plus facile à utiliser. Ces extensions sont décrites ci-dessous.

2. FORMAT D'UN PROGRAME

Un programme en assembleur acceptable en entrée par l'assembleur se compose d'une suite d'instructions de la forme :

ligne No étiquette opération opérande ;commentaire

où chaque champ peut être présent dans n'importe quel cas. Chaque instruction en assembleur se termine par un retour chariot et un line-feed (le line-feed est inséré automatiquement par le programme ED), ou par le caractère "!" qui est traité comme une fin de ligne par l'assembleur (ainsi plusieurs instructions en assembleur peuvent être écrites sur la même ligne physique si elles sont séparées par des points d'exclamation).

Le numéro de ligne est un nombre en base dix représentant le numéro de ligne dans le programme source, qui est autorisé sur n'importe quelle ligne source pour maintenir la compatibilité avec le format Processor Technology. En général, ces numéros de ligne sont insérés si un éditeur orienté-ligne est utilisé pour construire le programme d'origine, et ASM ne tient pas compte de ce champ s'il est présent.

Le champ étiquette a la forme suivante :

identificateur

ou

identificateur:

et est optionnel, sauf dans certains types d'instruction. L'identificateur est une suite de caractères alphanumériques (alphabétiques et numériques), le premier caractère devant être alphabétique. Les identificateurs peuvent être utilisés librement par le programmeur pour étiquetter des éléments comme des pas de programmes et des ordres d'assemblage, mais ne peuvent pas dépasser 16 caractères de longueur. Tous les caractères d'un identificateur sont significatifs, sauf le symbole dollar ($) qui peut être utilisé pour rendre le nom plus lisible. De plus, toutes les lettres minuscules sont traitées comme si elles étaient des majuscules.
Remarquez que les deux points ":" qui suivent l'identificateur sont optionnels (pour garder la compatibilité entre Intel et Processor Technology). Ainsi, vous pouvez avoir les types d'étiquettes suivants :

```
x          xy         nom$long
x:         yx1:       nom$plus$long:
X1Y2       X1x2       x234$5678$9012$3456:
```

Le champ opération contient soit une instruction assembleur ou pseudo opération, soit un code opération machine 8080. Les pseudo opérations et les codes opération machine sont décrits ci-dessous.

Le champ opérande de l'instruction, en général, contient une expression constituée de constantes et d'étiquettes, et d'opérations arithmétiques ou logiques sur ces éléments. Des détails plus complets sur ces expressions sont donnés ci-dessous.

Le champ commentaire contient des caractères arbitraires après le symbole ";" jusqu'à la prochaine fin de ligne réelle ou logique. Les caractères sont lus et imprimés, sinon ils ne sont pas pris en compte par l'assembleur. Afin de conserver la compatibilité avec l'assembleur de Processor Technology, l'assembleur du CP/M traite aussi les instructions qui commencent par un "*" dans la colonne 1 comme des commentaires, qui sont imprimés mais ne sont pas pris en compte lors du processus d'assemblage.
Remarquez que l'assembleur de Processor Technology a aussi pour effet de ne pas tenir compte dans cette opération de caractères qui se trouvent après le champ opérande qui a été examiné. Ceci entraine une situation ambigüe lorsqu'on essaie d'être compatible avec le langage d'Intel, étant donné que n'importe quelles expressions sont autorisées dans ce cas. Désormais, les programmes qui utilisent cet effet pour introduire des commentaires devront être édités avec un ";" avant ces champs pour être assemblés correctement.

Le programme en assembleur est constitué d'une suite d'instructions de la forme décrite ci-dessus, et terminé éventuellement par une instruction END. Toutes les instructions qui suivent le END ne sont pas prises en compte par l'assembleur.

3. FORMATION DES OPERANDES

Pour écrire complètement les codes opération et les pseudo opérations, il faut tout d'abord présenter la forme du champ opérande, étant donné qu'il est utilisé dans presque toutes les instructions Les expressions qui se trouvent dans le champ opérande se composent d'opérandes simples (étiquettes, constantes, et mots réservés), combinés en expressions correctement formées grâce à des opérateurs arithmétiques et logiques. Le calcul de l'expression est fait lors de l'assemblage. Chaque expression doit donner une valeur de 16 bits lors de l'assemblage. Le nombre de chiffres significatifs dans le résultat ne doit pas dépasser l'utilisation attendue. Autrement dit, si une expression doit être utilisée dans une instruction de déplacement immédiat d'octet, les huit bits les plus significatifs doivent être nuls. Les restrictions sur les significations des expressions sont données avec chaque instruction.

3.1 LES ETIQUETTES

Comme nous vous l'avons dit plus haut, une étiquette est un identificateur qui apparaît dans une instruction particulière.
En général, l'étiquette a une valeur déterminée par le type de l'instruction qu'elle précède. Si l'étiquette apparaît dans une instruction qui génère un code machine ou réserve de l'espace mémoire (par exemple une instruction MOV ou une pseudo opération DS) l'étiquette a la valeur de l'adresse du programme auquel elle renvoie. Si l'étiquette précède un EQU ou un SET, l'étiquette prend la valeur qui résulte de l'évaluation du champ opérande. A part pour l'instruction SET, un identificateur peut "étiquetter" une seule instruction.

Quand une étiquette apparaît dans un champ opérande, sa valeur est remplacée par l'assembleur. Cette valeur peut ensuite être combinée avec d'autres opérandes et opérateurs pour former le champ opérande d'une instruction.

3.2 LES CONSTANTES NUMERIQUES

Une constante numérique est une valeur de 16 bits dans l'une des bases. La base, appelée racine de la constante, est indiquée par un indicateur de racine. Les indicateurs de racine sont les suivants :

B	constante binaire (base 2)
O	constante octale (base 8)
Q	constante octale (base 8)
D	constante décimale (base 10)
H	constante hexadécimale (base 16)

Q est un indicateur de racine de remplacement pour les nombres octaux étant donné que la lettre O est souvent confondue avec le chiffre 0. Toute constante numérique qui n'est pas suivie d'un indicateur de racine est supposée être en **base 10** .

Une constante est ainsi composée d'une suite de chiffres, suivie éventuellement d'un indicateur de racine, les chiffres devant être compatibles avec la racine. Autrement dit, les constantes binaires doivent être composées des chiffres 0 et 1, les constantes octales de chiffres compris entre 0 et 7, et les constantes décimales des chiffres décimaux 0 à 9. Les constantes hexadécimales contiennent les chiffres décimaux, plus les "chiffres" hexadécimaux A (10D), B (11D), C (12D), D(13D), E (14D) et F (15D). Remarquez que le premier chiffre d'un nombre hexadécimal doit être un chiffre décimal (0 à 9) pour éviter de confondre une constante hexadécimale avec un identificateur (un zéro en tête est toujours suffisant). Une constante composée de cette manière (octale, décimale ou hexadécimale) doit correspondre à un nombre binaire composé de 16 bits au plus, sinon elle est tronquée à droite par l'assembleur. De la même manière que pour les identificateurs, on peut mettre des "$" à l'intérieur des constantes numériques pour les rendre plus lisibles. Enfin, l'indicateur de racine devient une majuscule si c'est une minuscule. Voici quelques exemples de constantes numériques correctes :

```
1234     1234D     1100B      1111$0000$1111$0000B
1234H    0FFEH     33770      00$77$22Q
3377a    0fe3H     1234d      0ffffh
```

3.3 LES MOTS RESERVES

Il existe plusieurs suites de caractères réservés qui ont des significations prédéfinies dans le champ opérande d'une instruction. Les noms des registres du 8080 sont donnés ci-dessous et, si on les rencontre, ils donnent les valeurs montrées à droite

```
A        7
B        0
C        1
D        2
E        3
H        4
L        5
M        6
SP       6
PSW      6
```

(là-aussi, les noms en minuscules ont les mêmes valeurs que leurs équivalents majuscules). Les instructions machine peuvent aussi être utilisées dans les champs opérandes, et prennent la valeur de leur code interne. Dans le cas d'instructions qui nécessitent des opérandes, l'opérande spécifique devenant une partie de la valeur en bit binaire de l'instruction, (par exemple MOV A,B), la valeur de l'instruction (dans ce cas MOV) est la valeur en bit de l'instruction avec des zéros dans les champs optionnels (par exemple MOV donne 40H).

Lorsque le symbole "$" apparaît dans le champ opérande (pas à l'intérieur de constantes numériques ou d'identificateurs), sa valeur devient l'adresse de la prochaine instruction à générer, sans compter l'instruction contenue dans la ligne logique courante.

3.4 LES CONSTANTES CHAINES

Les constantes chaînes représentent des suites de caractères ASCII, et sont représentées en entourant ces caractères d'apostrophes (').
Toutes les chaînes doivent être entièrement contenues dans la ligne physique courante (ce qui autorise les symboles "!" à l'intérieur des chaînes) et ne doivent pas dépasser 64 caractères. Le caractère apostrophe peut être contenu dans une chaîne en le représentant avec une double apostrophe (''), qui deviendra une apostrophe simple lorsqu'elle sera lue par l'assembleur. Dans la plupart des cas, la longueur de la chaîne est limitée à un ou deux caractères (exception faite pour la pseudo opération DB), et la chaîne devient une valeur sur 8 ou 16 bits, respectivement. La chaîne de deux caractères devient une constante de 16 bits, le second caractère étant l'octet le moins significatif et le premier le plus significatif.

La valeur d'un caractère est son code ASCII. Il n'y a pas de conversion à l'intérieur des chaînes, les majuscules et les minuscules peuvent être représentées. Remarquez, cependant, que seuls les caractères ASCII graphiques (d'impression) sont autorisés dans les chaînes. Voici des exemples de chaînes correctes :

```
'A'     'AB'      'ab'     'c'
''''    ' '''     ''''''   ''' '
          a,                 "

' Coucou Coucou.'
' Coucou c'est moi.'
' Je dis "coucou".'
```

3.5 LES OPERATEURS ARITHMETIQUES ET LOGIQUES

Les opérandes décrits ci-dessus peuvent être combinés en notation algébrique normale utilisant n'importe quelle combinaison correctement formée d'opérandes, d'opérateurs, et d'expressions entre parenthèses. Les opérateurs reconnus dans le champ opérande sont les suivants :

a + b	somme arithmétique sans signe de a et b
a - b	différence arithmétique sans signe entre a et b
+ b	plus unaire (donne b)
- b	moins unaire (équivalent à 0-b)
a * b	multiplication sans signe de a par b
a / b	division sans signe de a par b
a MOD b	reste de la division a/b
NOT b	complément logique de b (tous les 0 deviennent des 1 et les 1 des 0), b étant considéré comme une valeur de 16 bits.
a AND b	et logique bit par bit de a et b
a OR b	ou logique bit par bit de a et b
a XOR b	ou exclusif logique bit par bit de a et b
a SHL b	valeur résultant du décalage à gauche sur a d'une quantité b, complétée par des zéros
a SHR b	valeur résultant du décalage à droite sur a d'une quantité b, complétée par des zéros

Dans tous les cas, a et b représentent des opérandes simples (étiquettes constantes numériques, mots réservés, chaîne de un ou deux caractères), ou des expressions entièrement entre parenthèses. En voici des exemples :

```
10+20     10h+37Q          L1/3          (L2+4)    SHR 3
('a' and 5fh) + '0'        ('B' +b) OU (PSW + M)
(1+(2+c) ) shr  (A- (B+1) )
```

Remarquez que tous les calculs sont faits au moment de l'assemblage en tant qu'opérations sur 16 bits sans signe. Ainsi -1 est calculé comme 0-1 ce que donne la valeur 0ffffh. L'expression du résultat doit correspondre au code opération qui est utilisé. Si, par exemple, l'expression est utilisée dans une instruction ADI (addition immédiate), les huits bits les plus significatifs doivent être nuls. Ainsi, l'opération "ADI-1" donne un message d'erreur (-1 donne 0fffh qui ne peut être représenté sur 8 bits), alors que "ADI (-1) AND 0FFH" est accepté par l'assembleur car l'opération "AND" met dans ce cas les huits bits les plus significatifs à zéro.

3.6 PRIORITES DES OPERATEURS

Pour être plus pratique pour l'opérateur, l'ASM suppose que les opérateurs ont des priorités relatives d'application qui permettent à l'opérateur d'écrire des opérations sans mettre certains niveaux de parenthèses. L'expression du résultat a des parenthèses imaginaires définies par les priorités relatives. L'ordre d'application des opérateurs dans des expressions sans parenthèses est donné ci-dessous. Les opérateurs situés sur la même ligne ont la même priorité, et sont appliqués de gauche à droite lorsqu'ils sont rencontrés dans une expression :

```
* / MOD SHL SHR
  - +
  NOT
  AND
 OR XOR
```

Voici des exemples d'expressions dont l'interprétation donnée par l'assembleur avec des parenthèses est donnée à droite :

```
a * b + c                    (a * b) + c
a + b * c                    a + (b * c)
a MOD b * c SHL d            ( (a MOD b) * c) SHL d
a OR b AND NOT c + d SHL e a OR (b AND (NOT (c + (d SHL e))))
```

Des sous-expressions parenthésées équilibrées peuvent toujours être utilisées pour prendre la priorité sur des parenthèses supposées, et ainsi la dernière expression ci-dessus pourrait être réécrite pour forcer l'application des opérateurs dans un ordre différent :

```
(a OR b) AND (NOT c) + d SHL e
```

Ce qui donne avec les parenthèses induites par les priorités :

```
(a OR b) AND ((NOT c) + (d SHL e) )
```

Remarquez qu'une expression non parenthèsée est correctement formée seulement si l'expression que donne l'insertion des parenthèses induites est bien formée.

4. ORDRES EN ASSEMBLEUR

Les ordres en assembleur sont utilisés pour affecter des étiquettes aux valeurs spécifiques pendant l'assemblage, effectuer des assemblages conditionnels, définir des zones de mémoire et spécifier les adresses de départ du programme. Chaque ordre d'assembleur est indiqué par une "pseudo opération" qui apparaît dans le champ opération de la ligne. Les pseudo opérations sont les suivantes :

ORG	affecte l'origine du programme ou des données
END	termine le programme, avec éventuellement une adresse de début
EQU	"égalité" numérique
SET	"affectation" numérique
IF	commence l'assemblage conditionnel
ENDIF	termine l'assemblage conditionnel
DB	définit les données en octets
DW	définit les données en mots
DS	définit la zone de stockage des données

Les pseudo-opérations sont décrites en détail individuellement ci-dessous.

4.1 L'INSTRUCTION ORG

L'instruction ORG a la forme :

étiquette ORG expression

où "étiquette" est une étiquette de programme optionnel et "expression" une expression de 16 bits se composant d'opérandes définis avant l'instruction ORG. L'assembleur commence à générer un code machine à l'emplacement spécifié dans l'expression. Il peut y avoir n'importe quel nombre d'instruction ORG dans un programme et il n'y a aucune vérification sur les éventuels chevauchements des zones de mémoire que cela pourrait entraîner. Remarquez que les programmes écrits pour le système CP/M commencent par une instruction ORG de la forme :

```
ORG   100H
```

ce qui fait que la génération du code machine commence à la base de la zone des programmes transitoires (TPA) du CP/M. Si une étiquette est donnée dans une instruction ORG, l'étiquette prend la valeur de l'expression (cette étiquette peut être utilisée dans le champ opérande d'autres instructions pour représenter cette expression).

4.2 L'INSTRUCTION END

L'instruction END est optionnelle dans un programme en assembleur, mais si elle est présente, elle doit être la dernière instruction toutes les instructions qui la suivent ne seront pas prises en compte lors de l'assemblage. Les deux formes de l'instruction END sont les suivantes :

```
étiquette END
étiquette END expression
```

où, là encore, l'étiquette est optionnelle. Si la première forme est utilisée le processus d'assemblage s'arrête et l'adresse de début du programme prise par défaut est 0000. Sinon, l'expression est calculée et devient l'adresse de début du programme (cette adresse de début est contenue dans le dernier enregistrement du fichier "hex" en code machine formaté Intel qui est obtenu après l'assemblage). Ainsi, la plupart des programmes en assembleur du CP/M se termineront par l'instruction :

```
END 100H
```

qui donne l'adresse de début par défaut de 100H (début de la zone des programmes transitoires : TPA).

4.3 L'INSTRUCTION EQU

L'instruction EQU (égal) est utilisée pour affecter des "équivalences" à des valeurs numériques. La forme est la suivante :

```
étiquette   EQU   expression
```

l'étiquette devant être présente et ne devant pas servir d'étiquette à d'autres instructions. L'assembleur évalue l'expression et affecte cette valeur à l'identificateur donné dans le champ étiquette. L'identificateur est généralement un nom qui décrit la valeur d'une manière plus compréhensible pour l'homme. Par la suite, ce nom est utilisé dans le programme pour "paramétrer" certaines fonctions. Supposons par exemple que la donnée reçue à partir d'une télétype apparaisse sur un port d'entrée donné et que cette donnée soit envoyée à la télétype par le port de sortie suivant (dans l'ordre). La série d'instructions EQU ci-dessous pourrait être utilisée pour définir ces ports dans une certaine configuration hardware.

```
TTYBASE     EQU 10H          ;numéro du port de base de TTY
TTYIN       EQU TTYBASE      ;entrée des données sur TTY
TTYOUT      EQU TTYBASE+1    ;sortie des données de TTY
```

A un endroit situé plus loin dans le programme, les instructions ayant accès à la télétype pourraient être :

```
        IN      TTYIN           ;lit la donnée de TTY dans le registre A
        ...
        OUT     TTYOUT          ;écrit sur TTY la donnée du registre A
```

ce qui rend le programme plus lisible que si les port d'E/S absolus avaient été utilisés. Par la suite, si la configuration hardware est redéfinie pour faire commencer les ports de communication avec la télétype à 7FH au lieu de 10H, il suffirait simplement de changer la première instruction et de mettre à la place :

```
        TTYBASE     EQU     TFH  ;numéro du port de base de TTY
```

et le programme pourrait être réassemblé sans changer les autres instructions.

4.4 L'INSTRUCTION SET

L'instruction SET est analogue à l'instruction EQU et a la forme :

```
        étiquette     SET     expression
```

La seule différence est que l'étiquette peut apparaître dans d'autres instructions SET du programme. L'expression est évaluée et devient la valeur actuelle associée à l'étiquette. Ainsi, l'instruction EQU donne à une étiquette une valeur unique dans tout le programme, alors que l'instruction SET définit une valeur qui est valable à partir de cette instruction SET jusqu'au moment où l'étiquette apparaît dans une autre instruction SET située après. L'utilisation de SET est analogue à celle de EQU, mais est utilisée le plus souvent en assemblage conditionnel de contrôle.

4.5 LES INSTRUCTIONS IF ET ENDIF

Les instructions IF et ENDIF définissent un intervalle d'instructions en assembleur qui doivent être comprises ou non dans le processus d'assemblage. Leur forme est la suivante :

```
IF expression
instruction No 1
instruction No 2
    ...
instruction No n
ENDIF
```

Après avoir rencontré une instruction IF, l'assembleur évalue l'expression qui suit le IF (toutes les opérandes de l'expression doivent avoir été définies précédemment). Si l'expression donne une valeur non nulle, les instructions 1 à n sont assemblées ; si l'expression donne un zéro, les instructions sont listées mais non assemblées. L'assemblage conditionnel est souvent utilisé pour écrire un programme "générique" unique qui contient un certain nombre de configurations possibles, quelques portions spécifiques du programme seulement étant choisies pour chaque assemblage particulier. Les segments de programme suivants, par exemple, pourraient faire partie d'un programme qui communique soit avec une télétype, soit avec une console CRT (mais pas avec les deux à la fois) en choisissant une valeur particulière de TTY avant le début de l'assemblage.

```
VRAI        EQU         0FFFFH          ;définit la valeur de VRAI
FAUX        EQU         NOT VRAI        ;définit la valeur de FAUX
;
TTY         EQU         VRAI            ;VRAI est TTY, FAUX est CRT
;
TTYBASE     EQU         10H             ;base des ports d'E/S de TTY
CRTBASE     EQU         20H             ;base des ports d'E/S de CRT
            IF          TTY             ;assemblage relatif à TTY
CONIN       EQU         TTYBASE         ;entrée console
CONOUT      EQU         TTYBASE +1      ;sortie console
            ENDIF
;
            IF          NOT TTY         ;assemblage relatif à CRTBASE
CONIN       EQU         CRTBASE         ;entrée console
CONOUT      EQU         CRTBASE +1      ;sortie console
            ENDIF

            ...
            IN          CONIN           ;lit la donnée sur console
            ...
            OUT         CONOUT          ;écrit la donnée sur console
```

Dans ce cas, le programme s'assemblerait pour une configuration où une télétype serait connectée et basée au port 10H.
L'instruction qui définit TTY pourrait être remplacé par :

```
TTY  EQU  FAUX
```

et, dans ce cas, le programme s'assemblerait pour un CRT basé au port 20H.

4.6 L'INSTRUCTION DB

L'instruction DB permet au programmeur de définir des zones de stockage initialisées en format simple précision (en octet). La forme de l'instruction est la suivante :

étiquette DB e1,e2,...,en

où e1,...en sont, soit des expressions qui donnent des valeurs sur 8 bits (les huit bits les plus significatifs doivent être nuls), soit des chaînes ASCII de longueur inférieure ou égale à 64 caractères. Il n'y a pas de restrictions pratiques sur le nombre d'expressions comprises dans une seule ligne source. Les expressions sont évaluées et placées dans leur ordre dans le fichier de code machine après la dernière adresse du programme générée par l'assembleur, Les caractères des chaines sont placés de manière identique en mémoire en commençant par le premier caractère et en finissant par le dernier. Les chaînes de longueur supérieure à deux caractères ne peuvent pas être utilisées comme opérandes dans des expressions plus compliquées (elles doivent se trouver seules entre les virgules).
Remarquez que les caractères ASCII sont toujours mis en mémoire avec leur bit de parité remis à zéro. De plus, souvenez-vous qu'il n'y a pas de conversion de minuscules en majuscules dans les chaînes.

L'étiquette optionnelle peut être utilisée pour référencer la zone de données pour le reste du programme. Voici des exemples d'instructions DB correctes :

```
data:   DB      0,1,2,3,4,5
        DB      data and 0ffh,5,377Q,1+2+3+4
signon  DB      'veuillez donner votre nom', cr,lf,0
        DB      'AB' SHR 8, 'C', 'DE' AND 7FH
```

4.7 L'INSTRUCTION DW

L'instruction DW est analogue à l'instruction DB à la différence près que c'est une zone de stockage en double précision (deux octets) qui est initialisée. La forme en est la suivante :

```
étiquette  DW  e1,e2,...,en
```

où e1,...,en sont des expressions qui donnent des valeurs sur 16 bits. Remarquez que les chaînes ASCII de un ou deux caractères sont autorisées, mais pas les chaînes de plus de deux carctères. Dans tous les cas, le stockage des données est compatible avec le processeur 8080 : l'octet le moins significatif de l'expression est stocké en premier en mémoire, suivi de l'octet le plus significatif . Voici des exemples :

```
doub:  DW      0ffefh,doub+4,signon-$,255+255
       DW      'a',5,'ab','CD',6 shl 8 or 11b
```

4.8 L'INSTRUCTION DS

L'instruction DS est utilisée pour réserver une zone de mémoire non initialisée, et a la forme suivante :

```
étiquette   DS   expression
```

où l'étiquette est optionnelle. L'assembleur commence la génération du code après la zone réservée par les DS. Ainsi, l'instruction DS donnée ci-dessus a exactement le même effet que les instructions :

```
étiquette:      EQU $          ; la valeur de l'étiquette est
                               l'emplacement actuel du code"
                ORG $+expression     ; passe la zone réservée
```

5. CODES OPERATIONS

Les codes opération d'assembleur forment la partie principale des programmes en assembleur et constituent le champ opération de l'instruction. En général, ASM accepte tous les mnémoniques standards du micro ordinateur 8080 d'Intel, et qui sont donnés en détail dans le manuel Intel "8080 Assembly Language Programming Manual".

Les étiquettes sont optionnelles sur chaque ligne d'entrée, et, si elles sont présentes, prennent la valeur de l'instruction juste avant qu'elle ne soit sortie. Les opérateurs sont donnés brièvement dans les sections suivantes comme complément, mais vous devez vous reporter aux manuels d'Intel pour des détails plus complets sur les opérateurs.

Dans tous les cas,

e3 représente une valeur sur 3 bits de l'intervalle 0-7 qui peut être un des registres prédéfinis A,B,C,D,E,H,L,M,SP ou PSW

e8 représente une valeur sur 8 bits de l'intervalle 0-255

e16 représente une valeur sur 16 bits de l'intervalle 0-65535

et peuvent eux-mêmes être constitués de combinaisons arbitraires d'opérandes et d'opérateurs. Dans certains cas, les opérandes sont réduits à certaines valeurs à l'intérieur de l'intervalle permis, comme dans le cas de l'instruction PUSH. Ces cas seront indiqués lorsqu'ils seront rencontrés.

Dans les sections qui suivent, chaque code opération est donné dans sa forme la plus générale, avec un exemple spécifique, une courte explication et les restrictions spéciales.

5.1 SAUTS, APPELS ET RETOURS

Les instructions de saut, d'appel et de retour permettent plusieurs formes différentes qui testent les drapeaux de condition mis dans le CPU du micro-ordinateur 8080. Ces formes sont :

JMP	e16	JMP L1	saut inconditionnel à l'étiquette
JNZ	e16	JMPL2	saut si la condition est non nulle à l'étiquette
JZ	e16	JMP 100H	saut à l'étiquette si la condition est nulle
JNC	e16	JNC L1+4	saut si pas de retenue
JC	e16	JC L3	saut si retenue
JPO	e16	JPO $+8	saut à l'étiquette en cas de parité impaire
JPE	e16	JPE L4	saut à l'étiquette en cas de parité paire
JP	e16	JP GAMMA	saut à l'étiquette en cas de résultat positif
JM	e16	JM a1	saut à l'étiquette
CALL	e16	CALL S1	appel inconditionnel au sous-programme
CNZ	e16	CNZ S2	appel au sous-programme si le drapeau est non nul
CZ	e16	CZ 100H	appel au sous-programme si le drapeau est nul
CNC	e16	CNC S1+4	appel de sous-programme si pas de retenue
CC	e16	CC S3	appel de sous-programme si retenue existe
CPO	e16	CPO $+8	appel au sous-programme en cas de parité impaire
CPE	e16	CPE S4	appel au sous-programme en cas de parité paire
CP	e16	CP GAMMA	appel au sous-programme en cas de résultat positif
CM	e16	CM b1$c2	appel au sous-programme
RST	e3	RST 0	"redémarrage" du programme équivalent à CALL 8*e3, sauf que c'est un appel sur un octet
RET			retour depuis le sous-programme
RNZ			retour si un drapeau non nul est mis
RZ			retour si un drapeau nul est mis
RNC			retour si pas de retenue
RC			retour si retenue
RPO			retour si la parité est impaire
RPE			retour si la parité est paire
RP			retour si le résultat est positif
RM			retour si signe -

5.2 INSTRUCTIONS D'OPERANDES IMMEDIATES

Plusieurs instructions disponibles chargent des registres simple ou double précision, ou des cellules de mémoire simple précision, avec des valeurs constantes, et il y a aussi des instructions qui effectuent des opérations arithmétiques ou logiques immédiates sur l'accumulateur (registre A).

MVI	e3,e8	MVI B,255	met la donnée immédiate dans le registre A,B,C,D,H,L ou M (mémoire)
ADI	e8	ADI 1	ajoute l'opérande immédiate à A sans retenue.
ACI	e8	ACI OFFH	ajoute l'opérande immédiate à A avec retenue
SUI	e8	SUI L+3	soustrait de A sans retenue
SBI	e8	SBI L AND 11B	soustrait de A avec retenue
ANI	e8	ANI $ AND 7FH	"et" logique immédiat de A et de la donnée
XRI	e8	XRI 11110000B	"ou exclusif" immédiat de A et de la donnée
ORI	e8	ORI L AND 1+1	"ou" logique immédiat de A et de la donnée
CPI	e8	CPI 'a'	compare A et la donnée immédiate (identique à SUI mais A n'est pas modifié)
LXI	e3,e16	LXI B,100H	chargement étendu immédiat dans la paire de registres (e3 doit être égal à B,D,H ou SP)

5.3 INSTRUCTIONS D'INCREMENTATION ET DE DECREMENTATION

Le répertoire d'instructions du 8080 comprend les instructions permettant d'incrémenter ou de décrémenter des registres simple ou double précision :

INR	e3	INR E	incrémentation registre simple précision (e3 doit amener A,B,C,D,E,H,L ou M)
DCR	e3	DCR A	décrémentation registre simple précision (e3 doit amener A,B,C,D,E,H,L ou M)
INX	e3	INX SP	incrémentation de la paire de registres double précision (e3 doit donner B,D,H ou SP)
DCX	e3	DCX B	décrémentation de la paire de registres double précision (e3 doit donner B,D,H ou SP)

5.4 INSTRUCTIONS DE DEPLACEMENT DES DONNEES

Les instructions qui font passer les données de la mémoire au CPU en mémoire sont données ci-dessous :

MOV	e3,e3	MOV A,B	met la valeur de l'élément de gauche dans l'élément de droite (e3 peut valoir A,B,C,D,E,H,L ou M) MOV M,M est interdit
LDX	e3	LDAX B	charge le registre A avec l'adresse calculée (e3 est B ou D)
STAX	e3	STAX D	stocke le registre A à l'adresse calculée (e3 est égal à B ou D)
LHLD	e16	LHLD L1	charge le registre HL directement de l'adresse e16 (double précision pour H et L)
SHLD	e16	SHLD L5+x	stocke HL directement à l'adresse e16 (double précision de H et L à la mémoire)
LDA	e16	LDA Gamma	charge le registre A en provenance de l'adresse e16
STA	e16	STA X3-5	stocke le registre A en mémoire à l'adresse e16
POP	e3	POP PSW	charge une paire de registres en provenance de la pile, place le pointeur (SP). e3 doit donner soit B,D,H ou encore PSW
PUSH	e3	PUSH B	stocke une paire de registres dans la pile place le pointeur (SP). e3 doit donner B,D,H ou PSW
IN	e8	IN 0	chasse le registre A avec les données en provenance du port e8.
OUT	e8	OUT 255	envoie les données du registre A a port e8
XTHL			échange les données du haut de la pile avec HL
PCHL			met dans le compteur original (PC) les données en provenance de HL
SPHL			emplit le pointeur de pile (SP) avec les données de HL
XCHG			échange la paire DE avec la paire HL

5.5 OPERATIONS DE L'UNITE ARITHMETIQUE ET LOGIQUE

Instructions agissant sur l'accumulateur simple précision pour réaliser des opérations arithmétiques et logiques.

ADD	e3	ADD	B	ajouter le registre donné par e3 à l'accumulateur sans retenue (e3 doit donner soit A,B,C,D,E,H ou L)
ADC	e3	ADC	L	ajouter le registre à A avec retenue, e3 comme ci-dessus
SUB	e3	SUB	H	soustraction du registre e3 à A sans retenue, e3 défini comme ci-dessus
SBB	e3	SBB	2b	soustraction du registre e3 à A avec retenue, e3 défini comme ci-dessus
ANA	e3	ANA	1+1	"ET" logique du registre avec A, e3 comme ci-dessus
XRA	e3	XRA	A	"OU" exclusif avec A, e3 comme ci-dessus
ORA	e3	ORA	B	"OU" logique avec A, e3 comme ci-dessus
CMP	e3	CMP	H	comparer le registre à A, e3 comme ci-dessus
DAA				ajustement décimal du registre A selon la dernière opération de l'unité arithmétique et logique
CMA				complémentation des bits du registre A
STC				positionnement du flag de retenue à 1
CMC				complémentation du bit de retenue
RLC				rotation à gauche des bits ; effet secondaire : bit de retenue affecté (le bit de plus grand poids de A devient la retenue)
RRC				rotation à droite des bits ; effet secondaire : bit de retenue affecté (le bit de moindre poids de A devient la retenue)
RAL				rotation de (retenue/A) à gauche (la retenue est impliquée dans la rotation)
RAR				rotation de (retenue/A) à droite (la retenue est impliquée dans la rotation)
DAD	e3	DAD	B	addition en double précision de la paire de registre e3 à HL (e3 doit donner B,D,H ou SP)

5.6 INSTRUCTIONS DE CONTROLE

Les quatre instructions suivantes sont considérées comme instructions de contrôle :

HLT	stopper le processeur 8080
DI	désactiver le système d'interruptions
EI	autoriser le système d'interruptions
NOP	pas d'opération