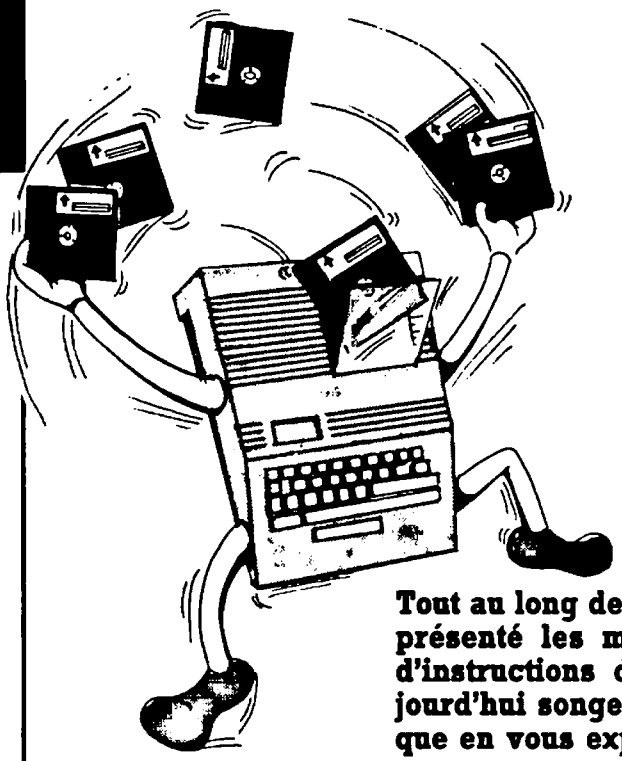
L'ABC DE LA MICRO-INFORMATIQUE



ASSEMBLEUR ET DIRECTIVES D'ASSEMBLAGE

Tout au long de cette série, nous vous avons présenté les modes d'adressage et le jeu d'instructions du 8088, et il nous faut aujourd'hui songer à mettre tout cela en pratique en vous expliquant comment écrire des programmes en assembleur. Pour cela, deux solutions existent sur les PC : la plus simple consiste à faire appel au mini-assembleur contenu dans DEBUG ; la plus efficace consiste à utiliser un assembleur digne de ce nom ou, mieux, un macro-assembleur.

L'utilisation de DEBUG présente un avantage, sur le plan du prix de revient ; en effet, cet utilitaire étant fourni avec le DOS, il est possible de s'initier à l'assembleur à bon compte. Les possibilités offertes sont, par contre, très réduites (ce qui est normal, les concepteurs de DEBUG l'ayant muni d'un « mini » assembleur seulement) et ne permettent pas de mener à bien l'écriture d'un programme un tant soit peu important. De plus, un certain nombre de fonctions essentielles de l'assembleur, telles celles liées à l'utilisation des étiquettes, ne sont pas accessibles.

Un véritable assembleur ou macro-assembleur, tel MASM de Microsoft par exemple, est évidemment plus coûteux mais permet de disposer de fonctions intéressantes, dignes des langages évolués et, surtout, permettant réellement d'écrire des programmes aussi longs que nécessaire. Nous allons donc présenter, dans les lignes qui suivent, les directives essentielles de MASM. Sous DEBUG, vous n'aurez accès qu'à un très petit nombre d'entre elles, bien sûr ; mais, au moins, vous saurez ce que peut faire un véritable assembleur et ne serez pas dépaysés lorsque vous en rencontrerez un. Précisons que les directives présentées ci-après sont conformes à celles préconisées par Intel (qui est le fabricant du microprocesseur 8088, rappelons-le) et se retrouvent donc identiques à elles-mêmes, à quelques détails mineurs près éventuellement, sur tous les assembleurs 8088.

DIRECTIVES ET PSEUDO INSTRUCTIONS

L'assembleur étant capable de gérer certaines informations non directement liées aux instructions ou aux modes d'adressage du microprocesseur, telles que noms de variables, affectations d'étiquettes, etc., il est capable de reconnaître un certain nombre de mots clés appelés directives d'assemblage ou encore pseudo-instructions. Ce sont ces directives que nous allons voir maintenant car elles ont des fonctions bien définies et doivent respecter des règles de syntaxe particulières.

LES VARIABLES

Tout programme qui se respecte ne peut se concevoir sans un certain nombre de variables. Tout comme en langage évolué, il est possible, avec un assembleur adéquat, de définir des variables, de leur donner une place en mémoire et de leur affecter un nom. Compte tenu des grandeurs que peuvent manipuler les processeurs de la famille 8088, les variables peuvent être de quatre types différents : Byte (octet), Word (mot de 16 bits), Double word (mot de 32 bits) et Quad word (mot de 64 bits).

Pour définir une variable, il faut donc indiquer à l'assembleur de quel type elle est, ce qui se fait très facilement au moyen de deux lettres : D comme définition et B, W, D ou Q selon la taille de la variable. Ainsi DD définit-il une variable double mot, soit 32 bits. Ces directives s'utilisent de la façon suivante :

nom de variable DX valeur ; commentaire.

Nom de variable est une chaîne de caractères définissant le nom que vous souhai-

tez donner à la variable : sa taille est variable selon les assembleurs, mais est en général limitée à 8 caractères. L'espace n'est pas autorisé car c'est le caractère séparateur entre les différents « champs » de la ligne.

DX est DB, DW, DD ou DQ selon la taille de la variable, tandis que valeur est la valeur initiale que vous souhaitez voir affecter à la variable. Si celle-ci est quelconque, un point d'interrogation doit être utilisé. Les valeurs numériques sont écrites sans signe distinctif particulier si elles sont exprimées en décimal. Elles sont suivies d'un H si elles sont en hexadécimal et sont comprises entre deux apostrophes s'il s'agit du code ASCII d'un caractère.

Il est possible, avec ce type de syntaxe, de définir plusieurs valeurs consécutives en partant d'un unique nom de variable. Il suffit de faire suivre le DX par plusieurs données séparées par des virgules, ainsi : table DB 2,5,8,10 réserve-t-il 4 octets (à cause du B) dont le premier s'appelle table et vaut 2, et est suivi par un octet initialisé à 5, un à 8 et un à 10 (décimal).

On peut de la sorte définir une chaîne de caractères par un : chaîne DB « BONJOUR », qui aura pour effet de placer en mémoire, à partir de l'adresse repérée par le nom chaîne, et les uns à la suite des autres, les codes ASCII de B, O, N, etc.

Les valeurs qui suivent DX doivent évidemment être en rapport avec la taille de la variable définie. Si la valeur est trop petite, tout va bien (il est possible de coder 3 sur 32 bits !), mais si la valeur est trop grande, elle sera nécessairement tronquée (essayez de coder 12423 sur 8 bits !).

Si une valeur doit être reproduite identique à elle-même plusieurs fois, il est inutile de l'écrire autant de fois que nécessaire ; il suffit de faire appel à la directive DUP sous la forme : plusieurs DB 23 DUP (5) qui réserve 23 octets initialisés à 5 dont le premier a pour nom de variable « plusieurs ».

Il est également possible de définir des constantes grâce à la directive EQU qui s'utilise de la façon suivante : nom EQU valeur, et qui affecte au nom précisé la valeur qui suit la directive EQU. Attention, contrairement à ce qui se passe avec DX, il n'y a pas ici de réservation de place mémoire mais uniquement définition d'une valeur constante qu'il sera impossible de modifier dans la suite du programme.

Ces définitions de constantes par EQU sont très souples et peuvent s'appeler les unes les autres. Ainsi, par exemple, si un circuit comportant plusieurs registres internes doit être utilisé par un programme, pourra-t-on écrire :

circuit EQU 1000H (adresse « de base » du circuit)
reg1 EQU circuit+1 (définition de l'adresse du registre 1 qui est donc 1000+1 soit 1001)
reg2 EQU circuit+2 (définition de l'adresse du registre 2 qui est donc 1000+2 soit 1002), etc.

De même, les notations suivantes sont-elles tout à fait correctes :

DOUZE EQU 2 * 6
QUATORZE EQU DOUZE+2

puisque la directive EQU affecte vraiment la valeur numérique précisée au nom de constante.

DEFINITIONS DES SEGMENTS D'UN PROGRAMME

Un programme assembleur est composé de plusieurs segments qu'il faut obligatoirement, décrire à l'assembleur pour qu'il puisse travailler correctement. Cette définition se fait avec la directive SEGMENT qui s'utilise de la façon suivante :

nom SEGMENT alignement type classe.

Les trois paramètres qui suivent SEGMENT sont optionnels.

Nom n'est rien d'autre que le nom que vous souhaitez donner au segment ansi défini.

Alignement définit à partir de quel type d'adresse va commencer le segment ; quatre valeurs principales sont admises :

- PARA (diminutif de paragraphe) qui spécifie une adresse divisible par 16.
- WORD qui signifie que le segment commence à une adresse de mot, c'est-à-dire à une adresse paire.
- BYTE qui signifie que le segment commence à une adresse d'octet, c'est-à-dire en fait n'importe où.
- PAGE qui signifie que le segment commence à une adresse divisible par 256.

Le paramètre type précise le type de segment ainsi défini, parmi les choix principaux suivants :

- rien (qui est la valeur par défaut) définit un segment local au module de programme. Si des segments de même nom figurent dans d'autres modules, ils seront considérés comme différents.
- PUBLIC aura pour effet de concaténer ce segment avec d'autres de même nom lors de l'édition de liens qui suit la phase d'assemblage.
- STACK précise que le segment est utilisé comme pile accessible selon un mécanisme LIFO (Last In First Out ou dernier entré premier sorti).

Le paramètre classe enfin permet de préciser à quelle « famille » appartient le segment. C'est une définition purement formelle qui n'est pas exploitée par l'assembleur mais permet simplement de rendre les programmes plus lisibles.

Lorsqu'un segment est créé, l'assembleur lui affecte un compteur qui lui est propre, appelé le compteur d'adresse. Ce dernier ne doit pas être confondu avec un registre interne du microprocesseur ; il n'a, en effet, rien à voir avec ce dernier et est uniquement un « point de repère » au sein du segment considéré. Ce compteur d'adresse peut être modifié par des directives qui sont :

- ORG, suivi d'une expression, qui a pour effet de forcer le compteur à la valeur de cette expression.
- EVEN qui force le compteur d'adresses à la valeur paire qui suit immédiatement la valeur courante.

Pour terminer la définition d'un segment, il faut utiliser la directive ENDS, sans paramètre particulier.

LES ASSOCIATIONS DE SEGMENTS

Plusieurs directives permettent d'associer, de façon plus ou mois intime, des segments différents. La plus simple d'emploi est GROUP, qui permet de préciser à l'assembleur que plusieurs segments de noms différents sont à placer dans le même bloc de 64 K-octets afin qu'ils soient accessibles par un même registre de segmentation du microprocesseur. Elle s'utilise tout simplement sous la forme :

nom du groupe GROUP nom1, nom2, nom3, etc., où nom1, nom2, etc., sont les noms des différents segments à grouper.

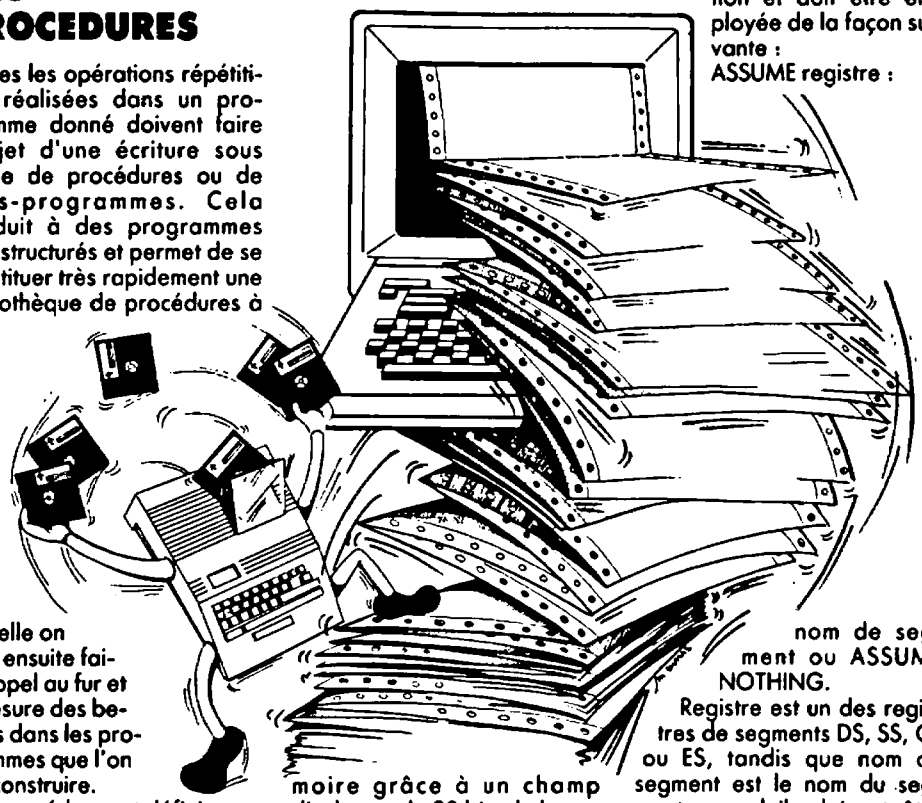
Un autre type d'association importante est caractérisé par la possibilité de faire appel, d'un module de programme à un autre, à des constantes ou des variables définies une fois pour toutes. Pour cela, trois directives sont disponibles :

- NAME suivie d'un nom qui a pour effet de donner un nom à un module.
- PUBLIC qui doit précéder le nom de toute variable ou constante définie dans un mo-

dule et qui doit être accessible à d'autres modules. Si tel n'est pas le cas, les définitions de variables et de constantes sont considérées comme locales et n'ont de signification que dans le module où elles ont été définies.
– EXTRN qui doit précéder le nom des constantes ou variables utilisées dans un module donné mais définies dans un autre module. C'est en fait la « réciproque » de PUBLIC.

LA DEFINITION DES PROCEDURES

Toutes les opérations répétitives réalisées dans un programme donné doivent faire l'objet d'une écriture sous forme de procédures ou de sous-programmes. Cela conduit à des programmes plus structurés et permet de se constituer très rapidement une bibliothèque de procédures à laquelle on peut ensuite faire appel au fur et à mesure des besoins dans les programmes que l'on a à construire.



Une procédure est définie tout simplement grâce à deux directives : PROC, placée au début, et ENDP placée à la fin, de la façon suivante :
nom de la procédure PROC (FAR)
...instructions de la procédure...
nom de la procédure ENDP
La directive FAR est optionelle et permet des appels de procédures inter-segments. Par défaut, une procédure est du type NEAR et ne peut donc être appelée que du même segment ou groupe de segments (revoir le descriptif de l'instruction CALL si nécessaire).

L'ASSEMBLEUR ET L'ADRESSAGE SEGMENTE

Comme nous l'avons vu lors de la présentation matérielle du 8088, ce circuit est capable d'adresser 1 M-octet de mémoire grâce à un champ d'adresse de 20 bits de large. Ce champ est constitué en deux fois grâce à des registres 16 bits dont l'un contient un déplacement par rapport au début d'un segment et dont l'autre contient une base qui est multipliée par 16 pour obtenir l'adresse réelle.
Pour le programmeur, et donc pour l'assembleur, les choses sont un peu plus compliquées car, pour une donnée placée à une adresse quelconque en mémoire, il existe une multitude de couples « déplacement-base » qui permettent d'y accéder. Il est donc nécessaire de fournir des informations relatives aux segments à utiliser à l'assembleur, afin d'arriver à un adressage correct. Cette notion, propre aux microprocesseurs de la famille 8088, est source de nombreuses erreurs et est d'un archaïsme difficilement admissible ; malheureusement il faut faire avec !
La directive ASSUME est prévue pour cette utilisation et doit être employée de la façon suivante :
ASSUME registre : nom de segment ou ASSUME NOTHING.
Registre est un des registres de segments DS, SS, CS ou ES, tandis que nom de segment est le nom du segment auquel ils doivent être associés. Ainsi :
ASSUME DS:données fait rechercher les variables d'un programme (traditionnellement liées au registre de segment DS) dans le segment de nom « données ».
Une directive ASSUME reste valide tant qu'aucune nouvelle directive du même type ne vient agir sur le même registre. De ce fait, il est possible d'annuler volontairement le rôle d'une directive ASSUME avec ASSUME NOTHING (nothing signifiant rien en américain).

LES MACRO INSTRUCTIONS

L'intérêt d'un assembleur évolué, ou plus exactement d'un macro-assembleur, est de permettre à l'utilisateur de définir des macro-instructions.
Une macro-instruction est un groupe d'instructions quelconques, auquel on affecte un nom, et que l'on peut ensuite utiliser autant de fois que nécessaire dans un programme comme s'il faisait partie des instructions de base du microprocesseur. Contrairement à un sous-programme ou à une procédure, une macro-instruction n'est donc pas un morceau de programme appelé lorsque c'est utile et écrit seulement une fois pour toutes. Une macro-instruction reproduit les instructions qui la composent autant de fois qu'elle est utilisée dans le programme. Cela consomme de la place par rapport à un sous-programme mais est plus rapide en vitesse d'exécution.
La définition d'une macro-instruction se passe de la façon suivante :
nom de la macro MACRO liste de paramètres
...suite d'instructions 8088...
ENDM
Nom de la macro est le nom que vous souhaitez donner à la macro-instruction ; nom que vous employerez ensuite dans votre programme pour utiliser cette dernière. Liste de paramètres est une suite de noms de variables séparés les uns des autres par des virgules. Ces noms correspondent à des variables utilisées dans la macro-instruction, et il leur est substitué les données numériques correspondantes lors de l'appel de la macro, comme nous allons le voir ci-après. La directive ENDM quant à elle

signale tout simplement à l'assembleur la fin de la macro.
Voici un exemple pratique avec une macro-instruction d'addition :

```
addition MACRO oper1, oper2, result
MOV BX, oper2
ADD BX, oper1
MOV result, BX
ENDM
```

Lorsque nous voudrons utiliser cette macro-instruction dans un programme, il suffira d'écrire :

```
addition X,Y,Z
```

et l'assembleur traduira cela automatiquement par la séquence :

```
MOV BX,Y
ADD BX,X
MOV Z,BX
```

où X, Y et Z sont évidemment quelconques et choisis par vos soins.

Toutes les instructions du 8088 sont utilisables au sein d'une macro-instruction, ainsi que les structures classiques telles que des boucles par exemple. De ce fait, des étiquettes peuvent être définies dans des macro (adresses de retour de boucles par exemple) mais, dans ce cas, il faut signaler à l'assembleur que ces étiquettes sont locales ; en effet, si tel n'était pas le cas, il générerait un message d'erreur dès le deuxième appel de la macro, car il trouverait alors une étiquette déjà définie (lors du premier appel). Pour ce faire, la directive LOCAL est proposée et s'utilise tout simplement devant le nom de l'étiquette concernée. Ainsi :

```
tempo MACRO temps
LOCAL boucle
MOV CX, temps
boucle : LOOP
ENDM
```

sera parfaitement correct puisque l'étiquette boucle est définie comme étiquette locale à la macro.
Enfin, lorsque l'on réalise des macro comportant des instructions de tests suivies de branchements conditionnels, on se trouve confronté au problème de macro à plusieurs sorties. Comme il ne peut y avoir qu'un ENDM en fin de définition de macro, ces sorties, on ne peut plus normales bien sûr, doivent être matérialisées par des directives EXITM, qui s'utilisent seules. Ces directives font continuer le programme avec l'instruction qui suit immédiatement la macro-instruction.

CONCLUSION

Nous en resterons là pour cette présentation des directives du macro-assembleur classique du 8088. Nous n'avons pas la prétention d'avoir été exhaustif, car de nombreuses autres directives sont disponibles. Nous les avons volontairement passées sous silence en essayant de nous restreindre à celles les plus fréquemment utilisées. Lorsque vous serez devenu un programmeur confirmé, il vous sera facile de les étudier, et de les utiliser alors si nécessaire.

C. TAVERNIER